ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση δεδομένων σε κατανεμημένα συστήματα πραγματικού χρόνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΤΖΙΜΑ ΣΟΦΙΑ

Επιβλέπων : Βαρβαρίγου Θεοδώρα
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΈΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
& ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση δεδομένων σε κατανεμημένα συστήματα πραγματικού χρόνου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΤΖΙΜΑ ΣΟΦΙΑ

Επιβλέπων : Βαρβαρίγου Θεοδώρα
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Μαρτίου 2015.

.....	.....	.....
Βαρβαρίγου Θεοδώρα Καθηγητής Ε.Μ.Π.	Καγιάφας Ελευθέριος Καθηγητής Ε.Μ.Π.	Λούμος Βασίλειος Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2015

.....

ΤΖΙΜΑ ΣΟΦΙΑ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής εργασίας ήταν η μελέτη και η ανάπτυξη συστημάτων πραγματικού χρόνου, τα οποία επεξεργάζονται συλλογές κειμένων και εξάγουν συμπεράσματα για το περιεχόμενο και το συναίσθημα τους. Μελετήθηκαν αλγόριθμοι διαφορετικής φύσης με στόχο να διαπιστωθεί τόσο η απόδοση του χρησιμοποιούμενου εργαλείου, όσο και η λειτουργικότητα του αλγορίθμου σε πραγματικό χρόνο, με μετρικές όπως η χρησιμοποίηση μνήμης, το πλήθος δεδομένων επεξεργασίας ανά δευτερόλεπτο και η αποκρισιμότητα του συστήματος με αυστηρούς χρονικούς περιορισμούς.

Η χρησιμότητα των συστημάτων πραγματικού χρόνου που επεξεργάζονται συλλογές κειμένων, με σκοπό την εξαγωγή συμπερασμάτων για το περιεχόμενό τους, γίνεται φανερή αν αναλογιστούμε την ανάπτυξη των κοινωνικών δικτύων ως σύγχρονες μορφές επικοινωνίας και έκφρασης. Η συγκέντρωση και η ανάλυση περιεχομένου που έχει δημιουργηθεί από χρήστες των κοινωνικών δικτύων, μας επιτρέπει να υπολογίσουμε χρήσιμα στατιστικά για την αίσθηση που επικρατεί στην κοινή γνώμη γύρω από ένα συγκεκριμένο θέμα.

Στα πλαίσια της μελέτης, αναλύθηκαν αλγόριθμοι από τα ευρύτερα πεδία των πιθανολογικών θεματικών μοντέλων και της ανάλυσης συναισθήματος. Υλοποιήθηκαν, με τη χρήση του εργαλείου Storm, και ειδικότερα του Trident, τοπολογίες που εκτελούνται ατέρμονα σε συστοιχία υπολογιστών. Αυτές οι τοπολογίες δέχονται δεδομένα σε μορφή ροής γεγονότων και εξάγουν σε πραγματικό χρόνο πληροφορίες γύρω από αυτά. Με χρήση ανάλογων συστημάτων είναι ικανή η επίβλεψη γεγονότων και η άμεση λήψη αποφάσεων με βάση αυτά.

Λέξεις Κλειδιά: Συστήματα πραγματικού χρόνου, Apache Storm, Ανάλυση συναισθήματος, Πιθανολογικά θεματικά μοντέλα, Μοντέλο Λανθάνουσας Κατανομής Dirichlet, Συστοιχία υπολογιστών, Παράλληλη επεξεργασία, Συστήματα επεξεργασίας μεγάλου όγκου δεδομένων, Κοινωνικά δίκτυα

Abstract

The scope of this thesis is the study and development of real-time systems, which process large collections of documents and draw conclusions about their content and emotion. We studied different nature algorithms in order to determine both the performance of the used tools and the real-time response of the algorithm, using metrics such as memory usage, the amount of data units processed per second, as well as the responsiveness of the system under severe time constraints.

The usefulness of real-time systems that process document collections, in order to draw conclusions about their content, becomes obvious if we consider the raise of social networks as modern forms of communication and expression. The analysis of user created content in social networks, allows us to compute useful statistics about the feeling that prevails in public opinion around a particular theme.

During the study, algorithms from the areas of probabilistic topic modelling and sentiment analysis were analyzed. We implemented those algorithms using Apache Storm, and created topologies that run endlessly in a Storm cluster. Those topologies accept data as a stream of events and export real-time information about them. Such systems are capable of monitoring events and making immediate decisions, based on them.

Keywords: real-time systems, Apache Storm, sentiment analysis, probabilistic topic models, Latent Dirichlet Allocation, Cluster computing, parallel processing, big data systems, social networks

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια της φοίτησής μου στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα να ευχαριστήσω την καθηγήτρια κα. Θεοδώρα Βαρβαρίγου και τον κο. Τσέρπε Κων/νο, του Εργαστηρίου Κατανεμημένης Γνώσης και Συστημάτων Πληροφορικής, για την εμπιστοσύνη που μου δείξανε και την δυνατότητα που μου δώσανε να εκπονήσω την διπλωματική μου στο συγκεκριμένο θέμα.

Παράλληλα, τους καθηγητές κο. Καγιάφα Ελευθέριο και Λούμο Βασίλειο που συμμετέχουν στην επιτροπή αξιολόγησης της συγκεκριμένης διπλωματικής.

Θα ήθελα επίσης, να ευχαριστήσω πολύ τον Υποψήφιο Διδάκτορα Θάνο Παπαοικονόμου, για την σημαντικότατη βοήθειά και καθοδήγησή του καθ' όλη την εξέλιξη της διπλωματικής μου. Με την εμπειρία και τις συμβουλές του με βοήθησε να ορίσω την κατεύθυνση της έρευνάς μου και να την ολοκληρώσω με επιτυχία.

Για την διεκπεραίωσή του θέματός μου, πολύ σημαντική ήταν και η προσφορά του Εργαστηρίου Υπολογιστικών Συστημάτων της σχολής, με την παροχή εικονικών μηχανημάτων για την εκτέλεση των πειραμάτων.

Καταλήγοντας, θα ήθελα να ευχαριστήσω τους συνεργάτες μου σε όλη την πορεία της σχολής, και μετέπειτα, για όσα μάθαμε μαζί και για τη βοήθεια τους σε κρίσιμες στιγμές. Η συνεργασία και η ομαδικότητα που επιτύχαμε είναι το πιο σημαντικό πράγμα που έμαθα στη σχολή. Εξίσου μεγάλα ευχαριστώ θα ήθελα να αποδώσω στην οικογένεια μου και τους φίλους μου για την συμπαράστασή τους όλα αυτά τα χρόνια.

Τζίμα Σοφία

Table of Contents

1 Εισαγωγή

1.1 Διατύπωση προβλήματος

Τα κοινωνικά δίκτυα αποτελούν έναν από τους πιο δημοφιλείς σύγχρονους τρόπους επικοινωνίας, μέσω διαδικτύου. Η ανάπτυξη τους έχει προκύψει ως φυσικό επακόλουθο της ανάπτυξης του διαδικτύου και των έξυπνων συσκευών, που παρέχουν στους χρήστες τη δυνατότητα της εύκολης και γρήγορης πρόσβασης στο λογαριασμό τους από παντού. Ανάλογα με το περιεχόμενο και τη λειτουργία τους, τα κοινωνικά δίκτυα, μπορούν να χωριστούν σε κατηγορίες με κυριότερες τις:

- Βασισμένα στην κοινωνική δικτύωση (Facebook, MySpace, LinkedIn, Ello, Google+, ask.fm)
- Ιστολόγια (Blogger, Wordpress)
- Μικροιστολόγια (Twitter, Tumblr)
- Wikis (Wikipedia)
- Πολυμεσικό περιεχόμενο (Instagram, flickr, deviantArt, YouTube, Vimeo, Pinterest...)

Το συνεχώς αυξανόμενο περιεχόμενο που δημιουργείται από τους χρήστες των δικτύων, αποκτά όλο και περισσότερη σημασία, εφόσον εκφράζει τις απόψεις και τα συναισθήματά τους γύρω από κάποιο θέμα. Συνεπώς, για να αξιοποιηθεί αυτό το περιεχόμενο, η σύγχρονη έρευνα επικεντρώνεται στην ανάπτυξη συστημάτων και

αλγορίθμων ικανών να επεξεργαστούν μεγάλο όγκο δεδομένων και να εξάγουν συμπεράσματα. Συστήματα που είναι ικανά να πραγματοποιήσουν αυτή την επεξεργασία είναι γνωστά ως Συστήματα Ανάλυσης Μεγάλου Όγκου Δεδομένων. Τα αποτελέσματα που προκύπτουν έχουν ενδιαφέρον τόσο για τον ακαδημαϊκό, όσο και για τον επιχειρησιακό κόσμο.

Επιπρόσθετα, ο ρυθμός με τον οποίο οι χρήστες δημοσιεύουν περιεχόμενο έχει σημαντικό αντίκτυπο στη διάδοση της πληροφορίας και των ειδήσεων. Η πληροφορία έχει χρηστικότητα στο χρόνο που δημοσιεύεται και μετά από σύντομο χρονικό διάστημα μπορεί να είναι ξεπερασμένη. Στην περίπτωση αυτή, η λήψη στατιστικών δεδομένων σχετικά με το περιεχόμενο εμπεριέχει την έννοια του χρόνου. Τα δεδομένα πρέπει να υπόκεινται επεξεργασία στο χρόνο της δημιουργίας τους και τα εξαγόμενα συμπεράσματα, και κατά συνέπεια οι αποφάσεις που μπορεί να ληφθούν με βάση αυτά, να αναφέρονται σε συγκεκριμένο χρονικό διάστημα. Τα συστήματα που κινούνται προς αυτή την κατεύθυνση, δηλαδή την επεξεργασία δεδομένων σε μορφή γεγονότων με αυστηρούς χρονικούς περιορισμούς και στιγμιαία απόκριση, λέγονται συστήματα πραγματικού χρόνου.

1.2 Συνεισφορά

Η παρούσα διπλωματική εργασία ασχολείται με την υλοποίηση και την μελέτη δύο αλγορίθμων ανάλυσης κειμένων για εξόρυξη θεματολογίας σε πραγματικό χρόνο: τον αλγόριθμο *Λανθάνουσας Κατανομής Dirichlet*, για την ανάδειξη των θεμάτων μιας συλλογής κειμένων και τον αλγόριθμο *Ανάλυσης Συναισθήματος με χρήση γράφων ν-γραμμάτων*, για την κατηγοριοποίηση μιας συλλογής κειμένων με βάση το συναίσθημα.

Ο πρώτος αλγόριθμος έχει αρχικά προταθεί από τους *David Blei et al* [4] και είναι ένα παραγωγικό μοντέλο που στοχεύει στην ανάδειξη της δομής μιας συλλογής κειμένων. Έχουν υλοποιηθεί πολλές παραλλαγές και επεκτάσεις του αλγορίθμου, οι οποίες έχουν σκοπό να ελαχιστοποιήσουν κάποιες αυστηρές υποθέσεις του μοντέλου και να αυξήσουν την ακρίβεια των αποτελεσμάτων υπό συγκεκριμένες προϋποθέσεις.

Ο δεύτερος αλγόριθμος προτάθηκε από τους *Aisopos et al* [1] και ανήκει στο ευρύτερο πεδίο της ανάλυσης συναισθήματος. Επιδιώκει να ανιχνεύσει την πολικότητα του συναισθήματος μικρών κειμένων, αντιμετωπίζοντας γνωστούς περιορισμούς ανάλογων συστημάτων, όταν αναφέρονται σε περιεχόμενο από χρήστες κοινωνικών δικτύων. Τέτοιοι περιορισμοί είναι ο θόρυβος, η εξάρτηση από τη γλώσσα και η άγνοια για το περιεχόμενο του κειμένου εκ των προτέρων.

Στα πλαίσια της εργασίας αναπτύχθηκαν οι παραπάνω αλγόριθμοι σε συστήματα πραγματικού χρόνου. Στόχος ήταν η δημιουργία συστημάτων που λαμβάνουν συνεχόμενη είσοδο από κάποια πηγή, και παράγουν αποτελέσματα σχετικά με το περιεχόμενο της σε πραγματικό χρόνο. Τα κείμενα εισόδου προέρχονται από χρήστες του Twitter, ενός κοινωνικού δικτύου που εντάσσεται στην κατηγορία των μικροιστολογίων. Οι λόγοι που επιλέχθηκε το Twitter ως πηγή κειμένων έχουν να κάνουν με το πλήθος και το κοινωνικό ενδιαφέρον των παραγόμενων κειμένων. Οι χρήστες του προέρχονται από πολλές χώρες και ανήκουν σε διαφορετικές ομάδες: συμμετέχουν σε αυτό επιστήμονες, διάσημοι, εκπρόσωποι χωρών, και επιχειρήσεων. Έτσι υπάρχει πληθώρα παραγόμενου περιεχομένου, από ένα μεγάλο εύρος θεμάτων.

Από την επεξεργασία των κειμένων, μελετήθηκε η συμπεριφορά των αλγορίθμων, η σύγκλιση των αποτελεσμάτων και η αποδοτικότητα των συστημάτων, δηλαδή το πλήθος των κειμένων που μπορούν να επεξεργαστούν σε μικρό χρονικό διάστημα.

Για την ανάπτυξη των συστημάτων, χρησιμοποιήθηκε το Apache Storm και συγκεκριμένα το πλαίσιο Trident του Storm. Το Storm είναι ένα κατανεμημένο υπολογιστικό σύστημα ανοιχτού κώδικα που παρέχει ένα σύνολο αρχών για δημιουργία συστημάτων πραγματικού χρόνου.

1.3 Οργάνωση κειμένου

Η παρούσα διπλωματική εργασία χωρίζεται σε δύο μέρη. Στο πρώτο μέρος παρουσιάζεται το θεωρητικό υπόβαθρο των χρησιμοποιούμενων εργαλείων και αλγορίθμων και περιλαμβάνονται όλες τις απαιτούμενες πληροφορίες για την κατανόηση των εννοιών που θα χρησιμοποιηθούν. Το δεύτερο μέρος περιέχει την περιγραφή των υλοποιήσεων που πραγματοποιήθηκαν με τη χρήση του Apache Storm και τα αποτελέσματα που προέκυψαν κατά την εκτέλεση των τοπολογιών σε συστοιχία.

Συγκεκριμένα, στο κεφάλαιο 3 παρουσιάζονται οι Αλγόριθμοι που υλοποιήθηκαν στα πλαίσια της εργασίας. Αρχικά, γίνεται μια γενική αναφορά στους ευρύτερους κλάδους των πιθανολογικών θεματικών μοντέλων και της ανάλυσης συναισθήματος. Έπειτα, παρουσιάζονται οι αρχές και τα χαρακτηριστικά των μοντέλων που επιλέχθηκαν για την υλοποίηση.

Στο κεφάλαιο 4 γίνεται μια παρουσίαση των χρησιμοποιούμενων τεχνολογιών, με έμφαση στις βασικές αρχές και τον τρόπο λειτουργίας του Storm. Επίσης, γίνεται μια

σύντομη αναφορά στη μη-σχεσιακή βάση δεδομένων Redis που χρησιμοποιήθηκε για την αποθήκευση των ενδιάμεσων δεδομένων των προγραμμάτων.

Στα κεφάλαια 5 και 6, αναπτύσσεται αναλυτικά η λογική πίσω από την υλοποίηση των αλγορίθμων. Παρουσιάζονται οι τεχνικές που επιλέχθηκαν για τα διάφορα στάδια της υλοποίησης καθώς και τα αποτελέσματα της εκτέλεσης στη συστοιχία Storm.

Τα συμπεράσματα που εξάγονται για την αποδοτικότητα του Storm στην ανάπτυξη συστημάτων επεξεργασίας περιεχομένου από χρήστες κοινωνικών δικτύων, όπως και οι ιδιαιτερότητες που εισάγονται στην υλοποίηση ανάλογα με τη φύση του κάθε αλγορίθμου αναλύονται στο 7^{ο} κεφάλαιο.

2 *Σχετικές εργασίες*

Ως αποτέλεσμα της ταχύτατης ανάπτυξης των κοινωνικών δικτύων, έχει αυξηθεί η ανάγκη για την επεξεργασία των πληροφοριών που δημιουργούνται από αυτά και την εξαγωγή συμπερασμάτων για το περιεχόμενό τους. Έτσι, η μελέτη ανάλογων συστημάτων και αλγορίθμων αποτελεί σημαντικό κλάδο της σύγχρονης έρευνας με ιδιαίτερο ενδιαφέρον, γύρω από την οποία βρίσκουμε πολλές σχετικές εργασίες.

Αρχικά, ο αλγόριθμος της Λανθάνουσας Κατανομής Dirichlet (LDA), που μελετήθηκε εκτενώς στην συγκεκριμένη εργασία, θεωρείται ένα καθιερωμένο εργαλείο στο χώρο των θεματικών μοντέλων. Η χρήση του στην εξαγωγή θεμάτων από διαδικτυακό περιεχόμενο έχει μελετηθεί σε σημαντικό βαθμό τα τελευταία χρόνια. Οι Krestel et al [22] πραγματοποίησαν μια ανάλυση σχετικά με την χρησιμοποίηση του σε εξαγωγή ετικετών για διαδικτυακό περιεχόμενο. Οι Hong et al [16] μελέτησαν την χρήση του απλού μοντέλου LDA και της επέκτασής του "Μοντέλο Συγγραφέα - Θέματος" (Author - Topic Model) σε περιεχόμενο από το Twitter, καταλήγοντας στο ότι η χρήση του απλού μοντέλου είναι πιο ακριβής σε κείμενα μικρού μεγέθους. Υποστήριξαν επίσης, ότι η αποτελεσματικότητα της εκπαίδευσης των θεματικών μοντέλων επηρεάζεται σημαντικά από το μέγεθος των κειμένων. Οι Paul el al [18] χρησιμοποίησαν ως πρότυπο τον αλγόριθμο LDA για να εξάγουν πληροφορίες από το Twitter σχετικές με την δημόσια υγεία. Επίσης, μια υλοποίηση σχετική με την εκτέλεση του αλγορίθμου LDA με χρήση του Storm βρίσκουμε και από τους Yogesh et

al [25], οι οποίοι χρησιμοποίησαν το Storm και την υλοποίηση του Mallet[1] για το LDA, για τη δημιουργία της τοπολογίας τους.

Όσον αφορά τον αλγόριθμο της Ανάλυσης συναισθήματος, βρίσκουμε μια υλοποίηση του για ανάλυση περιεχομένου του Twitter με χρήση του Storm από τους Raina et al [14]. Μια εκτενής ανάλυση σχετικά με τα χαρακτηριστικά του μοντέλου της ανάλυσης συναισθήματος και την αποδοτικότητα διαφορετικών προσεγγίσεων σχετικά με την εκπαίδευση του μοντέλου, πραγματοποίησαν οι Hassan et al [21]. Οι συγκεκριμένοι μάλιστα υποστήριξαν, σε αντίθεση με τα περισσότερα μοντέλα, ότι η αφαίρεση των StopWords κατά τη διαδικασία εκπαίδευσης μειώνει, αντί να αυξάνει την ακρίβεια των μοντέλων.

Τέλος, μια αναλυτική παρουσίαση της λειτουργίας του Storm πάνω στο Twitter έχει γίνει από τους Toshniwal et al [2], στην οποία παρουσιάζεται το εργαλείο και τα κυριότερα χαρακτηριστικά του, ενώ παράλληλα μελετάται η συμπεριφορά μιας συστοιχίας Storm σε περιπτώσεις αποτυχίας.

[1] http://mallet.cs.umass.edu/

3 *Θεωρητικό υπόβαθρο*

3.1 Αλγόριθμοι Πιθανολογικών Θεματικών Μοντέλων

3.1.1 Πιθανολογικά Μοντέλα

Ως μοντέλο ορίζουμε το σύνολο των εννοιών και δεδομένων που περιγράφουν με μαθηματικό τρόπο ένα σύστημα. Τα **αιτιοκρατικά** μοντέλα είναι αυτά στα οποία οι γνωστές μεταβλητές αρκούν για ακριβή πρόβλεψη των αποτελεσμάτων τους. Τα πιθανολογικά ή **στοχαστικά** μοντέλα είναι εκείνα τα οποία προσδιορίζονται τόσο από παρατηρήσιμες όσο και από μη ελεγχόμενες (άγνωστες) μεταβλητές. Οι διαφορετικές καταστάσεις στις οποίες μπορεί να βρεθούν οι μεταβλητές του συστήματος χαρακτηρίζονται από κάποια από κοινού κατανομή πιθανότητας πάνω στις παρατηρήσιμες και άγνωστες μεταβλητές, η οποία αντικατοπτρίζει τον παράγοντα της τύχης που συνδέεται με τα αποτελέσματα. Τα στοχαστικά μοντέλα αποτελούν ένα στυλοβάτη της σύγχρονης έρευνας τεχνητής νοημοσύνης, καθώς παρέχουν όλα τα απαραίτητα εργαλεία για την ανάλυση του τεράστιου όγκου των δεδομένων που παράγονται τόσο στην επιστήμη, όσο και την καθημερινή ζωή.

3.1.2 Πιθανολογικά Θεματικά Μοντέλα

Όσο η ποσότητα της πληροφορίας που ψηφιοποιείται και αποθηκεύεται αυξάνεται εκθετικά, η ανάγκη για να την ταυτοποιήσουμε και να τη διακριτοποιήσουμε, ώστε να μπορέσουμε να τη διαχειριστούμε αποτελεσματικά γίνεται μεγαλύτερη. Είναι συνεπώς απαραίτητη η ύπαρξη κατάλληλων συστημάτων που αυτοματοποιούν διαδικασίες, οι οποίες κάνουν πιο εύκολη την αναζήτηση αυτού που μας ενδιαφέρει μέσα από την πληθώρα των παρεχόμενων πληροφοριών στα ειδησεογραφικά, επιστημονικά, κοινωνικά και ενημερωτικά δίκτυα. Ένα εργαλείο που μπορεί να αναλύσει μεγάλη ποσότητα πληροφορίας, ώστε να ομαδοποιήσει κοινά σημεία και να την ταυτοποιήσει με βάση το περιεχόμενό της είναι τα πιθανολογικά θεματικά μοντέλα.

Στους τομείς της μηχανικής μάθησης και της επεξεργασία φυσικής γλώσσας, με τον όρο **πιθανολογικά θεματικά μοντέλα** αναφερόμαστε στους αλγόριθμους που χρησιμοποιούνται για την ανάλυση πολύ μεγάλων αρχείων κειμένων, την ανίχνευση της θεματικής δομής τους και των τρόπων με τους οποίους τα αποτελέσματα συνδέονται μεταξύ τους. Αυτά τα αποτελέσματα ονομάζονται **θέματα**. Ένα κείμενο ή μια συλλογή κειμένων θεωρείται ότι αναφέρονται σε κάποιο θέμα όταν συγκεκριμένες λέξεις ή μοτίβα λέξεων εμφανίζονται σε σημαντικές συχνότητες. Οι αλγόριθμοι θεματικών μοντέλων είναι ικανοί να κάνουν αυτή την αξιολόγηση χωρίς να μεσολαβεί ανθρώπινος παράγοντας σε κάποιο από τα στάδια της διαδικασίας. Για να το πετύχουν αυτό, αντιστρέφουν την διαδικασία παραγωγής των κειμένων έτσι ώστε να καταλήξουν στην αρχική δομή τους: τα επιμέρους θέματα. Χαρακτηρίζονται λοιπόν και ως παραγωγικά πιθανολογικά μοντέλα με άγνωστες μεταβλητές τα επιμέρους θέματα. Κατά τη διαδικασία παραγωγής των κειμένων υπολογίζεται η από κοινού κατανομή πιθανότητας πάνω στις άγνωστες και τις γνωστές μεταβλητές, η οποία χρησιμοποιείται έπειτα κατά την ανάλυση δεδομένων ώστε να υπολογιστεί η δεσμευμένη κατανομή πιθανότητας των θεμάτων.

Το μέγεθος της πληροφορίας που μπορούν να επεξεργαστούν τα πιθανολογικά θεματικά μοντέλα είναι τέτοιας κλίμακας που είναι αδύνατον να γίνει από τον άνθρωπο. Λόγω του ευρέος πεδίου εφαρμογής τους σε τομείς όπως βιοπληροφορική, αναγνώριση προτύπων, εξόρυξη πληροφορίας από δικτυακό περιεχόμενο και όραση υπολογιστών, έχουν αναπτυχθεί πολλά θεματικά μοντέλα τις τελευταίες δεκαετίες.

Κάποια γνωστά μοντέλα αλγορίθμων είναι:

- Latent Dirichlet Allocation (LDA) model
- Online Latent Dirichlet Allocation

- Hierarchical Dirichlet Process (HDP) model
- Probabilistic latent semantic indexing
- Pachinko Allocation
- Author- Topic Model

3.2 Λανθάνουσα Κατανομή Dirichlet

3.2.1 Περιγραφή Αλγορίθμου

Ένα από τα πιο απλά στατιστικά μοντέλα για την ανίχνευση θεμάτων σε σύνολα κειμένων είναι το μοντέλο **Λανθάνουσας Κατανομής Dirichlet**. Αυτό το μοντέλο αναπτύχθηκε από τους David Blei et al [4]. Όσον αφορά την πρακτική εφαρμογή του, αποτελεί έναν από τους σημαντικότερους αλγόριθμους που έχουν αναπτυχθεί και αναλυθεί για την εξαγωγή θεμάτων από μεγάλα αρχεία κειμένων. Συνεπώς, το πεδίο εφαρμογής του είναι αρκετά ευρύ, αφού δοθέντος ενός κατάλληλου λεξιλογίου, μπορεί να οδηγήσει σε ακριβή αποτελέσματα. Χαρακτηριστικά, προτείνεται από τους Krestel Ralf et al [15], η χρήση του μοντέλου για την αυτόματη επιλογή ετικετών για διαδικτυακό περιεχόμενο. Οι ετικέτες χρησιμοποιούνται στα κοινωνικά δίκτυα για την οργάνωση του περιεχομένου του χρήστη και την εύρεση σχετικού περιεχομένου από άλλους χρήστες.

Η ιδέα πίσω από αυτό το μοντέλο είναι απλή: το κάθε κείμενο αποτελείται από ένα σύνολο θεμάτων, τα οποία προσδιορίζονται από συγκεκριμένες λέξεις συνδεδεμένες με κάποια πιθανότητα. Μια συλλογή κειμένων μοιράζεται κοινά θέματα περιεχομένου, αλλά το κάθε κείμενο εκφράζει τα θέματα σε διαφορετικές αναλογίες. Ο τρόπος που κατανέμονται τα θέματα σε μια συλλογή κειμένων, σε συνδυασμό με την αναλογία των θεμάτων και την κατανομή των λέξεων στο κάθε κείμενο αποτελούν την “κρυμμένη δομή” της συλλογής. Για να αναγνωριστεί η θεματολογία των κειμένων, η οποία αποτελεί την άγνωστη μεταβλητή του μοντέλου, το υπολογιστικό πρόβλημα ανάγεται στην αντιστροφή της διαδικασίας παραγωγής τους: αν υπολογιστεί η δομή με την οποία δημιουργήθηκαν αρχικά τα κείμενα, έχει επιτευχθεί η ανίχνευση των θεμάτων που την αποτελούν.

Σύμφωνα με τον David Blei [7], το μοντέλο Λανθάνουσας Κατανομής Dirichlet προσπαθεί να αυτοματοποιήσει την ανίχνευση των θεμάτων με βάση ένα υπάρχον λεξικό που αποκλείει τις λέξεις που δεν εισάγουν θεματολογία στο κείμενο (όπως άρθρα και συνδέσμους). Ορίζουμε ως “θέμα” την κατανομή γύρω από ένα συγκεκριμένο λεξιλόγιο.

Τεχνικά, το λεξιλόγιο και η κατανομή του σε θέματα πρέπει να έχει δημιουργηθεί πριν από την επεξεργασία των κειμένων. Εφόσον έχουμε το λεξικό, η διαδικασία παραγωγής των λέξεων *για κάθε κείμενο* γίνεται σε 2 στάδια:

1. Επιλέγουμε τυχαία μια κατανομή από τα υπάρχοντα θέματα
2. Για κάθε λέξη στο κείμενο:
    - Επιλέγουμε τυχαία ένα θέμα απ' την κατανομή των θεμάτων του σταδίου 1
    - Επιλέγουμε τυχαία κάποια λέξη από την αντίστοιχη κατανομή του θέματος

Έτσι, εκφράζεται η διαισθητική προσέγγιση ότι το κείμενο αποτελείται από κάποια θέματα σε διαφορετικές αναλογίες (στάδιο 1) και η κάθε λέξη του ανήκει σε κάποιο από αυτά τα θέματα (στάδιο 2). Το πλήθος των θεμάτων θεωρείται γνωστό για τον αλγόριθμο (σταθερά K), αλλά δεν έχει προηγηθεί κανενός είδους χαρακτηρισμός των κειμένων. Στο παραγωγικό αυτό μοντέλο, οι άγνωστες μεταβλητές είναι αυτές που περιγράφουν τη δομή των κειμένων και οι γνωστές μεταβλητές είναι οι λέξεις των κειμένων.

Η “τυχαία” επιλογή των θεμάτων στο στάδιο 1 γίνεται με χρήση της κατανομής Dirichlet. Η διαδικασία που επαναλαμβάνεται έχει στόχο να κατανείμει τις λέξεις του κειμένου στα διάφορα θέματα που αντιπροσωπεύουν τη συλλογή. Αυτά τα θέματα παρότι αποτελούν τη δομή του κειμένου είναι δύσκολο να υπολογιστούν – λεγόμενα και κρυμμένη δομή, για αυτό λέγονται λανθάνοντα. Συνεπώς έτσι προκύπτει το όνομα του μοντέλου: “Λανθάνουσα Κατανομή Dirichlet”.

Το υπολογιστικό πρόβλημα της ανάδειξης της δομής των κειμένων ισοδυναμεί με τον προσδιορισμό της δεσμευμένης κατανομής πιθανότητας των άγνωστων μεταβλητών, δεδομένων των κειμένων. Για να υπολογίσουμε την δεσμευμένη κατανομή πιθανότητας των αγνώστων μεταβλητών του μοντέλου ορίζουμε κάποιες βασικές έννοιες:

- λεξικό: το σύνολο των λέξεων που ορίζουν τα θέματα {1, 2, ... , V }
- λέξη: η βασική δομή των διακριτών δεδομένων. Είναι ένα αντικείμενο του λεξικού, για αυτό αναπαρίσταται ως ένα μονοδιάστατο διάνυσμα w με $w^v = 1$ και $w^u = 0$ για $u \neq v$
- κείμενο: μια αλληλουχία από N λέξεις που συμβολίζονται ως $\mathbf{w} = (w_1, w_2, \dots, w_N)$
- συλλογή: μια συλλογή από M κείμενα που συμβολίζονται ως $D = \{\mathbf{w_1}, \mathbf{w_2}, \dots, \mathbf{w_M}\}$

Ορίζουμε επίσης:

- β_k : ένα θέμα, ως μια κατανομή πάνω στο λεξικό
- $\theta_{d,k}$: η αναλογία του θέματος k στο κείμενο d
- $z_{d,n}$: η επιλογή του θέματος που ανήκει η λέξη n στο κείμενο d
- $w_{d,n}$: η λέξη n στο κείμενο d

Η παραγωγική διαδικασία που περιγράψαμε νωρίτερα, καθορίζει την από κοινού κατανομή πιθανότητας για τις γνωστές και άγνωστες μεταβλητές:

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \prod_{n=1}^{N} [\, p(z_{(d,n)}|\theta_d)\, p(w_{(d,n)}|\beta_{(1:K)}, z_{(d,n)})] \quad (1)$$

Οι εξαρτήσεις που περιγράφονται στην παραπάνω εξίσωση, ορίζονται από την διαδικασία παραγωγής των θεμάτων:

- Η επιλογή του θέματος της κάθε λέξης του κειμένου εξαρτάται από την αναλογία του θέματος στο κείμενο (Στάδιο 2α)
- Η επιλογή της λέξης για κάθε κείμενο εξαρτάται από την επιλογή του θέματος και την κατανομή του θέματος πάνω στο λεξικό (Στάδιο 2β)

Αυτές οι εξαρτήσεις ορίζουν μαθηματικά το μοντέλο της Λανθάνουσας Κατανομής Dirichlet.

Η δεσμευμένη κατανομή πιθανότητας της δομής των κειμένων, δοθέντων των γνωστών κειμένων υπολογίζεται ως:

$$p(\beta_{(1:K)}, \theta_{(1:D)}, z_{(1:D)}|w_{(1:D)}) = \frac{p(\beta_{(1:K)}, \theta_{(1:D)}, z_{(1:D)}, w_{(1:D)})}{p(w_{(1:D)})} \quad (2)$$

Ο αριθμητής είναι η κοινή κατανομή πιθανότητας για όλες τις γνωστές και άγνωστες μεταβλητές, η οποία είναι εύκολο να υπολογιστεί. Ο παρονομαστής όμως είναι η οριακή πιθανότητα των γνωστών μεταβλητών, δηλαδή η πιθανότητα η συγκεκριμένη συλλογή να περιλαμβάνει οποιοδήποτε θέμα. Αυτό στην πράξη ισοδυναμεί με τον υπολογισμό του αθροίσματος όλων των ικανών τρόπων να αντιστοιχίσουμε κάθε λέξη της συλλογής σε ένα θέμα, ο οποίος είναι εκθετικού μεγέθους, άρα πρακτικά αδύνατο να υπολογιστεί.

Οι αλγόριθμοι θεματικών μοντέλων εστιάζουν λοιπόν στην προσέγγιση της εξίσωσης 2 με μεθόδους που εμπίπτουν σε 2 κυρίως κατηγορίες – στους δειγματοληπτικούς και

αλγόριθμους βελτιστοποίησης. Οι δειγματοληπτικοί αλγόριθμοι συλλέγουν δείγματα από την δεσμευμένη κατανομή με σκοπό να την προσεγγίσουν εμπειρικά. Ένα τέτοιο παράδειγμα είναι η δειγματοληψία κατά Gibbs. Αντίθετα, οι αλγόριθμοι βελτιστοποίησης είναι ντετερμινιστικοί. Δημιουργούν μια οικογένεια κατανομών πάνω στην κρυφή δομή της συλλογής και υπολογίζουν το μέλος της οικογένειας που είναι πιο κοντά στην δεσμευμένη κατανομή. Έτσι το πρόβλημα της προσέγγισης μεταφράζεται σε πρόβλημα βελτιστοποίησης. Οι αλγόριθμοι βελτιστοποίησης συνήθως στηρίζονται στο ιεραρχικό μοντέλο του Bayes, και λέγονται και αλγόριθμοι συμπερασματολογίας κατά Bayes.

Η έρευνα στο χώρο των θεματικών μοντέλων εστιάζει και στα δύο είδη αλγορίθμων, τόσο στη βελτιστοποίηση τους, όσο και στην αποδοτικότητα της χρήσης τους για κάθε θεματικό μοντέλο. Έχουν αναπτυχθεί πολλοί διαφορετικοί αλγόριθμοι, με πλεονεκτήματα και μειονεκτήματα που τους καθιστούν ένα επιστημονικό πεδίο με προοπτικές μελλοντικής έρευνας.

3.2.2 Συνδεδεμένη μεταβολική μέθοδος Bayes για την Λανθάνουσα Κατανομή Dirichlet

Το πρόβλημα του υπολογισμού των θεμάτων μιας συλλογής κειμένων, όπως αυτό παρουσιάστηκε στην προηγούμενη ενότητα, ανάγεται στον υπολογισμό της δεσμευμένης πιθανότητας της κρυμμένης δομής της συλλογής, δοθέντων των κειμένων. Υπολογιστικά είναι ένα πρόβλημα εκθετικής πολυπλοκότητας και οι αλγόριθμοι εστιάζουν κυρίως στην αποτελεσματική και ακριβή προσέγγισή του με διάφορες μεθόδους. Οι αλγόριθμοι αυτοί εμπίπτουν γενικά σε δύο κατηγορίες – τους δειγματοληπτικούς αλγόριθμους και τους αλγόριθμους βελτιστοποίησης.

Για το μοντέλο της Λανθάνουσας Κατανομής Dirichlet φαίνεται εμπειρικά ότι η δεύτερη κατηγορία είναι αποτελεσματικότερη και δίνει ακριβέστερα αποτελέσματα. Παρόλα αυτά οι καθιερωμένοι μεταβολικοί αλγόριθμοι επεξεργασίας κατά δεσμίδες είναι υπολογιστικά ακριβοί, αφού σε κάθε επανάληψη εκτελούν μια ανάλυση για κάθε λέξη της συλλογής και έπειτα ενημερώνουν μεταβλητές εξαρτώμενες από όλα τα προηγούμενα δεδομένα. Συνεπώς το μέγεθος των δεδομένων που πρέπει να αποθηκεύουν και το κόστος επεξεργασίας για κάθε επανάληψη τους καθιστά ασύμφορους για μεγάλο όγκο δεδομένων και ανάλυση σε πραγματικό χρόνο – χαρακτηριστικά απαραίτητα για αλγόριθμους θεματικών μοντέλων.

Γι' αυτό το λόγο υπήρξε η ανάγκη να αναπτυχθεί ένας αλγόριθμος, ο οποίος μπορεί να επεξεργαστεί μεγάλο όγκο δεδομένων συνεχόμενα, χωρίς να τα αποθηκεύει παράλληλα. Με αυτό τον τρόπο μπορεί να ξεπεραστεί το κύριο πρόβλημα της επεξεργασίας κατά δεσμίδες, αν τα δεδομένα έρθουν σε μορφή ροής, χρησιμοποιηθούν μια φορά από τον αλγόριθμο και έπειτα απορριφθούν.

Ο αλγόριθμος της συνδεδεμένης μεταβολικής μεθόδου Bayes για την λανθάνουσα κατανομή Dirichlet είναι βασισμένος σε μια στοχαστική βελτιστοποίηση που τον καθιστά ικανό να επεξεργαστεί πολύ μεγάλο πλήθος δεδομένων σε πραγματικό χρόνο. Μάλιστα, έχει αποδειχθεί ότι παράγει ικανοποιητικές προσεγγίσεις και συγκλίνει γρηγορότερα από ότι ο συμβατικός αλγόριθμος επεξεργασίας σε δεσμίδες. Αυτό τον καθιστά μια πρακτική μέθοδο για τον υπολογισμό της δεσμευμένης κατανομής των περίπλοκων ιεραρχικών κατά Bayes μοντέλων, όπως αυτό της Λανθάνουσας Κατανομής Dirichlet. Αναπτύχθηκε μεταγενέστερα του κλασικού μοντέλου (2010), από τους David M. Blei et al [6].

ΜΕΤΑΒΟΛΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ ΕΠΕΞΕΡΓΑΣΙΑΣ ΚΑΤΑ ΔΕΣΜΙΔΕΣ

Οι συμβατικοί μεταβολικοί αλγόριθμοι επεξεργασίας κατά δεσμίδες χρησιμοποιούν μια απλούστερη κατανομή για να προσεγγίσουν την πραγματική δεσμευμένη κατανομή των άγνωστων μεταβλητών. Αυτή η κατανομή χρησιμοποιεί 3 παραμέτρους $q(z,\theta,\beta)$, οι οποίοι βελτιστοποιούνται για να μεγιστοποιήσουν τη σύγκλιση στην πραγματική κατανομή, καταλήγοντας σε μια κατανομή της μορφής

$$q(z_{di}=k)=\Phi_{dw_{di}k}; q(\theta_d)=Dirichlet(\theta_d;\gamma_d); q(\beta_k)=Dirichlet(\beta_k;\lambda_k)$$

όπου:

- φ: η δεσμευμένη κατανομή πιθανότητας της επιλογής των θεμάτων ανά λέξη z
- θ: η δεσμευμένη κατανομή πιθανότητας της αναλογίας των θεμάτων σε κάθε κείμενο
- β: η δεσμευμένη κατανομή των θεμάτων
- λ: τα θέματα

Για να υπολογίσουμε τη συνολική σύγκλιση του αλγορίθμου, υπολογίζουμε τις επιμέρους συγκλίσεις $l(n_d,\varphi_d,\gamma_d,\lambda)$ και αθροίζουμε στο σύνολο των κειμένων. Εν αντιθέσει με τους δειγματοληπτικούς αλγόριθμους, τα κείμενα μπορούν να θεωρηθούν ως το

άθροισμα των λέξεων τους. Συνεπώς, η συνολική σύγκλιση του αλγορίθμου είναι

$$L=\sum_{D} l(n_d, \varphi_d, \gamma_d, \lambda)$$.

Ο ψευδοκώδικας του αλγορίθμου για επεξεργασία κατά δεσμίδες δίνεται στην εικόνα 1.

Algorithm 1 Batch variational Bayes for LDA

Initialize $\boldsymbol{\lambda}$ randomly.
while relative improvement in $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\phi}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) > 0.00001$ **do**
  E step:
  for $d = 1$ to D **do**
    Initialize $\gamma_{dk} = 1$. (The constant 1 is arbitrary.)
    repeat
      Set $\phi_{dwk} \propto \exp\{\mathbb{E}_q[\log \theta_{dk}] + \mathbb{E}_q[\log \beta_{kw}]\}$
      Set $\gamma_{dk} = \alpha + \sum_w \phi_{dwk} n_{dw}$
    until $\frac{1}{K} \sum_k |\text{change in} \gamma_{dk}| < 0.00001$
  end for
  M step:
  Set $\lambda_{kw} = \eta + \sum_d n_{dw} \phi_{dwk}$
end while

Εικόνα 3.2.1. Αλγόριθμος μεταβολικής μεθόδου για επεξεργασία κατά δεσμίδες, όπως παρουσιάζεται από τους Blei et al [6]

ΣΥΝΕΧΟΜΕΝΟΙ ΜΕΤΑΒΟΛΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ ΕΠΕΞΕΡΓΑΣΙΑΣ

Ο αντίστοιχος συνεχόμενος αλγόριθμος επεξεργασίας είναι ικανός να επεξεργαστεί τεράστιο όγκο δεδομένων που φτάνουν σε μορφή ροής και έχει πολύ μικρότερες απαιτήσεις μνήμης, εφόσον δεν χρειάζεται να επεξεργαστεί ολόκληρη τη συλλογή σε μια επανάληψη. Επιπλέον, συγκλίνει γρηγορότερα από τον αλγόριθμο επεξεργασίας κατά δεσμίδες για μεγάλο όγκο δεδομένων.

Η λειτουργία του βασίζεται στον αλγόριθμο της επεξεργασίας κατά βαθμίδες και έχει ως στόχο να βελτιστοποιήσει το άθροισμα $L(n, \lambda)=\sum_{D} l(n_d, \varphi(n_d, \lambda), \gamma(n_d, \lambda), \lambda)$ κατά το βήμα Ε, όπως αυτό περιγράφεται στον αλγόριθμο της εικόνας 1. Η μεταβλητή που

θέλουμε να βελτιστοποιήσουμε είναι η λ. Για να επιτευχθεί αυτό, κάθε φορά που φτάνει ένα κείμενο n_t , εκτελούμε ένα βήμα Ε, έτσι ώστε να βρεθούν οι τοπικά βέλτιστες τιμές για τα γ_t, φ_t , με σταθερό λ. Έπειτα υπολογίζουμε το $\tilde{\lambda}$, την βέλτιστη τιμή του λ. Η βέλτιστη τιμή, ισοδυναμεί με την περίπτωση που θα είχαμε συλλογή με μόνο ένα κείμενο το οποίο θα επαναλαμβανόταν D φορές. Στην πραγματικότητα, το $D \rightarrow \infty$ αφού τα κείμενα θα φτάνουν με μια συνεχόμενη, ατέρμονη ροή.

Τέλος, για να ενημερώσουμε το λ χρησιμοποιούμε το $\tilde{\lambda}$, και την προηγούμενη τιμή του πολλαπλασιασμένα με ένα συντελεστή. Αναλυτικά, ο αλγόριθμος παρουσιάζεται στην Εικόνα 3.2.2.

Algorithm 2 Online variational Bayes for LDA

Define $\rho_t \triangleq (\tau_0 + t)^{-\kappa}$
Initialize $\boldsymbol{\lambda}$ randomly.
for $t = 0$ to ∞ **do**
  E step:
  Initialize $\gamma_{tk} = 1$. (The constant 1 is arbitrary.)
  repeat
    Set $\phi_{twk} \propto \exp\{\mathbb{E}_q[\log \theta_{tk}] + \mathbb{E}_q[\log \beta_{kw}]\}$
    Set $\gamma_{tk} = \alpha + \sum_w \phi_{twk} n_{tw}$
  until $\frac{1}{K} \sum_k |\text{change in} \gamma_{tk}| < 0.00001$
  M step:
  Compute $\tilde{\lambda}_{kw} = \eta + D n_{tw} \phi_{twk}$
  Set $\boldsymbol{\lambda} = (1 - \rho_t)\boldsymbol{\lambda} + \rho_t \tilde{\boldsymbol{\lambda}}$.
end for

Εικόνα 3.2.2. Συνεχόμενος αλγόριθμος μεταβολικής μεθόδου, όπως παρουσιάζεται από τους Blei et al [6]

3.3 Ανάλυση Συναισθήματος

Το πρόβλημα της ανάλυσης συναισθήματος, ανήκει στην ευρύτερη κατηγορία των μοντέλων που επιδιώκουν να ανιχνεύσουν το περιεχόμενο κειμένων. Διαφέρει παρόλα αυτά, από τα πιθανολογικά θεματικά μοντέλα, γιατί τα δεύτερα αντιστοιχίζουν τα κείμενα σε θεματικές κατηγορίες, που ανήκουν σε κάποιο συγκεκριμένο σύνολο. Επιπλέον, τα κείμενα μπορεί να ανήκουν σε μια ή περισσότερες από τις κατηγορίες που έχουν οριστεί για το συγκεκριμένο πρόβλημα. Αντίθετα, η ανάλυση συναισθήματος επιδιώκει να κατηγοριοποιήσει τα κείμενα σε μικρό σύνολο κατηγοριών, οι οποίες είναι αμοιβαία αποκλειόμενες, όπως “αρνητικό” - “θετικό” - “ουδέτερο”. Πλέον, το πρόβλημα της ανίχνευσης των θεμάτων απλοποιείται σε αυτό της ανίχνευσης της πολικότητας του θέματος του κειμένου, δηλαδή στο κατά πόσον το περιεχόμενό έχει θετική, αρνητική ή ουδέτερη φόρτιση.

Η χρησιμότητα της παραπάνω ανάλυσης, όταν αυτή γίνεται σε περιεχόμενο που έχει δημιουργηθεί από χρήστες του διαδικτύου, έγκειται στην ικανότητα να μετρηθεί η συνολική εντύπωση που επικρατεί για το συγκεκριμένο θέμα. Αναλύοντας κείμενα με χρήση φίλτρων, είναι δυνατή η εξαγωγή χρήσιμων συμπερασμάτων για την ευρύτερη γνώμη, γύρω από το θέμα ενδιαφέροντος. Γι αυτό το λόγο, η ανάλυση συναισθήματος αποκαλείται επίσης και εξόρυξη απόψεων.

3.3.1 Ανάλυση συναισθήματος σε μικρο-ιστολόγια

Με την εξάπλωση των κοινωνικών δικτύων και του αυξανόμενου περιεχόμενου που δημιουργείται καθημερινά από τους χρήστες τους, η ανάγκη για ανάπτυξη αλγορίθμων που επεξεργάζονται αυτά τα δεδομένα και εξάγουν συμπεράσματα γίνεται μεγαλύτερη. Μια ολόκληρη κατηγορία δεδομένων προς ανάλυση, είναι όσα έχουν προέλθει από μικρο-ιστολόγια, δηλαδή κοινωνικά δίκτυα με περιορισμένο μέγεθος δημοσιευόμενων κείμενων. Τα εν λόγω κείμενα, λόγω της ιδιομορφίας του περιεχόμενού τους, προσθέτουν επιπλέον δυσκολία στους αλγόριθμους ανάλυσης και απαιτούν ανθεκτικότητα σε υψηλά επίπεδα θορύβου:

1. Μήκος κειμένου – στα περισσότερα μικρο-ιστολόγια τα κείμενα είναι περιορισμένου μεγέθους (για παράδειγμα το twitter προσφέρει μέγιστο όριο 140 χαρακτήρες), με

αποτέλεσμα το περιεχόμενο να συμπτύσσεται σε βαθμό που να δυσκολεύει την αναγνώριση της πολικότητας του.

2. Λεξιλόγιο – οι χρήστες των κοινωνικών δικτύων συνήθως χρησιμοποιούν απλό καθημερινό λεξιλόγιο, συντομεύσεις και κομμένες λέξεις που μπορεί να διαφέρουν αρκετά από την πρωτότυπη και δυσκολεύουν την ανίχνευση του νοήματος της.
3. Υψηλά επίπεδα θορύβου – η χρήση των κοινωνικών δικτύων, που είναι πλέον καθημερινή και προσβάσιμη από παντού για τους περισσότερους χρήστες, έχει απαλλαχθεί από την αυστηρότητα στη σύνταξη και την ορθογραφία. Κατ επέκτασιν, η ευκολία δημοσίευσης αυξάνει την πιθανότητα ύπαρξης ορθογραφικών ή συντακτικών λαθών που εισάγουν υψηλό επίπεδο θορύβου, ο οποίος πρέπει να αναγνωριστεί και να αναλυθεί ανάλογα.
4. Πολυγλωσσία – οι χρήστες των κοινωνικών δικτύων, και κατά συνέπεια τα κείμενά που δημοσιεύουν μπορεί να είναι σε οποιαδήποτε γλώσσα. Αυτό αποτελεί έναν παράγοντα που πρέπει να ληφθεί υπόψιν από τους αλγόριθμους ανάλυσης.

3.3.2 Ανάλυση συναισθήματος με χρήση γράφων ν-γραμμάτων

Μια τεχνική που χρησιμοποιείται για ανάλυση συναισθήματος σε δεδομένα από το μικρο-ιστολόγια, είναι η ανάλυση με χρήση γράφων ν-γραμμάτων. Είναι μια τεχνική επιβλεπόμενης μηχανικής μάθησης, στην οποία τα δεδομένα αναπαριστώνται με τη χρήση γράφων. Ένας ταξινομητής αναλαμβάνει να συγκρίνει τα διανύσματα χαρακτηριστικών αυτών των γράφων με αντίστοιχα της κάθε κατηγορίας πολικότητας, για να εντοπίσει ομοιότητες. Με βάση τις ομοιότητες κατατάσσεται το κείμενο στην ανάλογη κατηγορία. Προηγείται ένα στάδιο εκπαίδευσης, στο οποίο συγκεντρώνονται δεδομένα που αντιπροσωπεύουν τις επιμέρους κατηγορίες και η εκπαίδευση του αλγορίθμου ταξινόμησης.

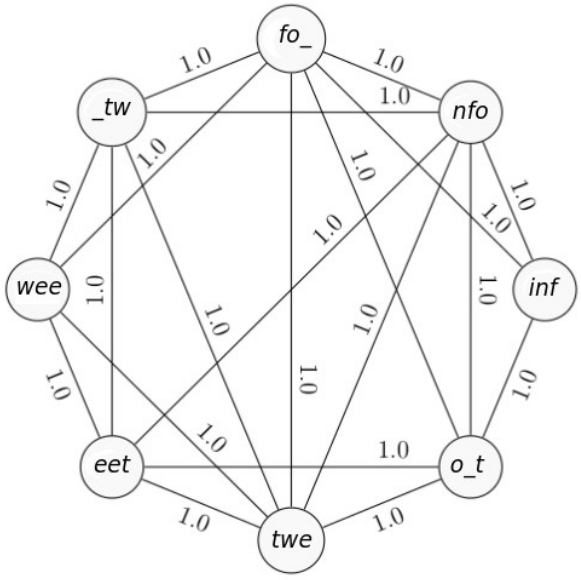
Πιο συγκεκριμένα, τα δεδομένα στην αναπαρίστανται με γράφους ν-γραμμάτων:

Ορισμός 3.3.1. Ένας γράφος ν-γραμμάτων είναι ένας γράφος $G=V^G, E^G, W$, όπου V^G είναι το σύνολο κορυφών (σημειωμένες με τους χαρακτήρες του αντίστοιχου ν-γράμματος), E_G είναι το σύνολο των μη κατευθυνόμενων ακμών (σημειωμένες με βάση την αλληλουχία των ετικετών των παρακείμενων κορυφών τους με αλφαβητική σειρά), και W είναι η συνάρτηση αντιστοίχισης βάρους στην κάθε ακμή.

Ουσιαστικά, ένα ν-γραμμα είναι μια ακολουθία ν χαρακτήρων. Μια φράση απεικονίζεται με ένα γράφο, του οποίου οι κορυφές είναι το σύνολο ν-γραμμάτων από όλες τις δυνατές υπακολουθίες μεγέθους ν. Για παράδειγμα, οι κορυφές του ν-γράφου για τη φράση “info tweet”, με ν=3 είναι {“inf”, “nfo”, “fo_”, “o_t”, “_tw”, “twe”, “wee”, “eet”}. Οι συνηθέστερες τιμές του ν είναι 2 (2-γράμματα), 3 (3-γράμματα), 4 (-γράμματα).

Οι ακμές μεταξύ των κορυφών προστίθενται βάσει της απόστασης μεταξύ των δυο υποακολουθιών μέσα στη φράση. Αν η απόσταση είναι μικρότερη από ν, τότε δημιουργείται ακμή μεταξύ των κορυφών. Τα βάρη στις ακμές υπολογίζονται από την συνάρτηση αντιστοίχισης W.

Στην εικόνα 3.3.1 παρουσιάζεται ένα παράδειγμα γράφου ν-γραμμάτων.



Εικόνα 3.3.1. Παράδειγμα γράφου ν-γραμμάτων

Για την κατηγοριοποίηση της φράσης, που απεικονίζεται με τον γράφο G_{t_i} ο ταξινομητής χρησιμοποιεί ένα διάνυσμα χαρακτηριστικών που την προσδιορίζει. Με την εκπαίδευση που προηγήθηκε, έχουν δημιουργηθεί οι γράφοι πολικότητας. Το διάνυσμα που χρησιμοποιείται από τον ταξινομητή στη συγκεκριμένη τεχνική προκύπτει από τον υπολογισμό των μετρικών ομοιότητας:

- Ομοιότητα Συνοχής (Containment Similarity – **CS**): εκφράζει το ποσοστό των ακμών του γράφου G_{t_i} που περιέχονται στον γράφο πολικότητας G_{t_p} . Αν e είναι μια ακμή του γράφου ν-γραμμάτων και με $|G|$ συμβολίσουμε το πλήθος των ακμών ενός γράφου, ορίζουμε την συνάρτηση μ ως: $\mu(e,G)=1, e \in G$, $\mu(e,G)=0, e \notin G$ και ο τελικός τύπος για την ομοιότητα συνοχής (CS)

  υπολογίζεται από τη σχέση: $CS(G_{t_i}, G_{T_p})=\sum_{e \in G_{t_i}} \frac{\mu(e, G_{T_p})}{min(|G_{t_i}|, |G_{T_p}|)}$.

- Ομοιότητα τιμής (Value Similarity – **VS**): εκφράζει το ποσοστό των ακμών του γράφου G_{t_i} που περιέχονται στο γράφο πολικότητας G_{t_p} , λαμβάνοντας υπόψη και τα βάρη τους. Κάθε κοινή ακμή e έχει βάρη $w^{t_i}(e)$, $w^{t_p}(e)$ στους γράφους G_{t_i} και G_{t_p} συνεισφέροντας $\frac{VR(e)}{max(|G_{t_i}|, |G_{T_p}|)}$ στο δείκτη VS. Ο όρος VR (Value Ratio) είναι ένας συμμετρικός συντελεστής κλίμακας $VR:[0,1] \rightarrow [0,1]$ με τύπο: $VR(e)=\frac{min(w^{t_i}(e), w^{T_p}(e))}{max(w^{t_i}(e), w^{T_p}(e))}$, οπότε η ομοιότητα τιμής (VS) τελικά υπολογίζεται από τη σχέση:

$$VS(G_{t_i}, G_{T_p})=\frac{\sum_{e \in G_{t_i}} \frac{min(w^{t_i}(e), w^{T_p}(e))}{max(w^{t_i}(e), w^{T_p}(e))}}{max(|G_{t_i}|, |G_{T_p}|)}$$

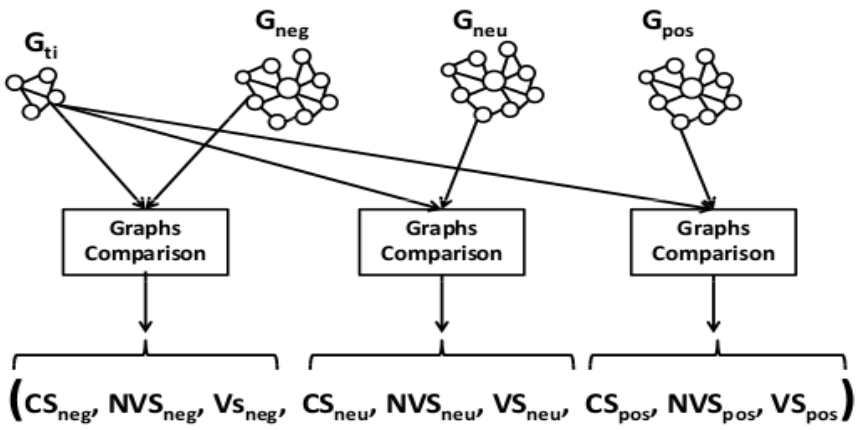
  Ο δείκτης συγκλίνει στο 1 για γράφους G_{t_i} και G_{t_p} που μοιράζονται ακμές με παρόμοια βάρη. Η τιμή VS = 1 δηλώνει το τέλειο ταίριασμα μεταξύ των συγκρινόμενων γράφων.

- Κανονικοποιημένη Ομοιότητα Τιμής (Normalized Value Similarity – **NVS**): αποσυνδέει το δείκτη Ομοιότητας Τιμής VS από την επίδραση του μεγέθους του μεγαλύτερου γράφου. Αυτό γίνεται διαιρώντας με το δείκτη Ομοιότητας Μεγέθους (Size Similarity - SS) ο οποίος ορίζεται ως : $SS(G_{t_i}, G_{T_p})=\frac{min(|G_{t_i}|, |G_{T_p}|)}{max(|G_{t_i}|, |G_{T_p}|)}$.

Ο τύπος για την κανονικοποιημένη ομοιότητα τιμής (NVS) περιγράφεται από τη σχέση: $NVS(G_{t_i}, G_{T_p}) = \frac{VS(G_{t_i}, G_{T_p})}{SS(G_{t_i}, G_{T_p})}$.

Με βάση τους 3 παραπάνω ορισμούς, υπολογίζεται το διάνυσμα χαρακτηριστικών του γράφου ν-γραμμάτων για την φράση, συγκρίνοντας με κάθε έναν από τους γράφους πολικότητας. Το διάνυσμα χαρακτηριστικών που προκύπτει έχει τις τιμές CS, VS και NVS. Έπειτα, ο ταξινομητής δέχεται ως είσοδο το διάνυσμα και αποφαίνεται για την πιθανότερη κατηγορία στην οποία ανήκει το tweet.

Ένα πλεονέκτημα της τεχνικής αναπαράστασης με γράφους ν-γραμμάτων είναι η ανεξαρτησία από την γλώσσα του κειμένου, η οποία θα εισήγαγε ένα επίπεδο δυσκολίας στον αλγόριθμο ανάλυσης. Επίσης, με αυτή την προσέγγιση το διάνυσμα χαρακτηριστικών της φράσης είναι σταθερό και το μέγεθός του είναι ίσο με τον αριθμό των γράφων πολικότητας.



Εικόνα 3.3.2. Σύγκριση γράφου ν-γραμμάτων του tweet i, με τους γράφους πολικότητας, για τον υπολογισμό του διανύσματος χαρακτηριστικών, όπως παρουσιάζεται από τους Aisopos et al [1]

4 Τεχνολογικό Υπόβαθρο

4.1 Συστήματα επεξεργασίας πραγματικού χρόνου

Με τον όρο υπολογισμό σε πραγματικό χρόνο, αναφερόμαστε στη μελέτη των συστημάτων που αφορούν μια συνεχή είσοδο, επεξεργασία και συστηματική έξοδο των δεδομένων. Τέτοιου είδους συστήματα γίνονται όλο και περισσότερο αναγκαία για την κάλυψη συγκεκριμένων αναγκών, όπως επίβλεψη συστήματος, ηλεκτρονικό εμπόριο, ανίχνευση απάτης και λήψη αποφάσεων, τόσο για επιστημονικά όσο και εταιρικά περιβάλλοντα. Είναι απαραίτητο να επεξεργάζονται υψηλό ποσοστό δεδομένων σε μικρό χρονικό διάστημα και κατά συνέπεια πρέπει να εγγυώνται απόκριση μέσα σε αυστηρούς χρονικούς περιορισμούς (της τάξης των milliseconds ή ακόμη και nanoseconds).

Σύμφωνα με την κρισιμότητα της άμεσης απάντησης, τα συστήματα πραγματικού χρόνου χωρίζονται σε 3 κατηγορίες:

- αυστηρά – η απώλεια του χρονικού ορίου για την επεξεργασία μιας μονάδας δεδομένων ισοδυναμεί με αποτυχία του συστήματος
- σταθερά – η σπάνια απώλεια χρονικού ορίου για την επεξεργασία μας μονάδας δεδομένων είναι ανεκτή αλλά μπορεί να υποβαθμίσει την ποιότητα υπηρεσιών του συστήματος

- χαλαρά – η χρησιμότητα ενός αποτελέσματος μειώνεται με το πέρας του χρονικού ορίου, υποβαθμίζοντας την ποιότητα υπηρεσιών του συστήματος

Σύμφωνα με τους *Stonebraker et al* [23], ένα επιτυχημένο σύστημα πραγματικού χρόνου πρέπει να είναι ικανό να επεξεργαστεί πολύ μεγάλο αριθμό δεδομένων συνεχόμενα. Επίσης, επιβάλλεται να παρέχει ένα μηχανισμό ερωτήσεων προς τη ροή, για τα τρέχοντα ή τα προγενέστερα δεδομένα. Η ακεραιότητα αυτών των δεδομένων, η αποθήκευση και η ανάκτησή τους πρέπει να εξασφαλίζεται ανά πάσα στιγμή, ανεξάρτητα από οποιαδήποτε αποτυχία του συστήματος. Συνεπώς το σύστημα είναι απαραίτητο να διαχειρίζεται ατέλειες, όπως καθυστέρηση στην άφιξη, άφιξη εκτός σειράς, ή ακόμη και απώλεια των δεδομένων. Τέλος, η παραγόμενη έξοδος συνηθίζεται να είναι προβλέψιμη και διαθέσιμη για επίβλεψη, έτσι ώστε να επιβεβαιώνεται ο ντετερμινισμός και η επαναληπτικότητα του συστήματος.

Για την κατασκευή μιας εφαρμογής πραγματικού χρόνου, ο μηχανικός πρέπει να χρησιμοποιήσει σύγχρονες γλώσσες προγραμματισμού και κατάλληλα λειτουργικά συστήματα ή δίκτυα. Τα παραπάνω παρέχουν τα απαιτούμενα framework για την ανάπτυξη και την εγκατάσταση της εφαρμογής.

4.2 APACHE STORM

4.2.1 Βασικές έννοιες και αρχές

Το Apache Storm είναι ένα κατανεμημένο υπολογιστικό σύστημα ανοιχτού κώδικα, μια μηχανή επεξεργασίας δεδομένων βασισμένη στα γεγονότα που παρέχει ένα σύνολο γενικών αρχών για υπολογισμούς πραγματικού χρόνου. Δημιουργήθηκε από τον Nathan Marz το 2011 και πρόσφατα μετέβει κάτω από την άδεια του Apache. Χρησιμοποιείται για την αξιόπιστη επεξεργασία απεριόριστου όγκου δεδομένων σε πολλαπλές περιπτώσεις, όπως στην ανάλυση συστήματος σε πραγματικό χρόνο, σε online συστήματα μηχανικής μάθησης, στο συνεχόμενο υπολογισμό και σε κατανεμημένο RPC. Ουσιαστικά το Storm είναι η εκδοχή του Hadoop για συστήματα πραγματικού χρόνου, αφού είναι ικανό να διαχειριστεί τεράστιο ποσοστό δεδομένων σε συστοιχία υπολογιστών και να παρέχει αποτελέσματα. Αυτό που το διαφοροποιεί παρόλα αυτά, από ένα συμβατικό σύστημα μεγάλου όγκου δεδομένων, είναι ο προσανατολισμός στην ανίχνευση γεγονότων. Αυτό σημαίνει ότι μπορεί να αναγνωρίσει τα πιο σημαντικά γεγονότα από μια συνεχή ροή και να ανταποκριθεί άμεσα.

Το Twitter[1] είναι η πρώτη εταιρία που χρησιμοποίησε το Storm, μέσω της Backtype. Εν έτη 2014 περίπου 500 εκατομμύρια tweets δημιουργούνται κάθε μέρα[2]. Συνεπώς, η τάση των σύγχρονων τεχνολογιών είναι η δημιουργία συστημάτων που είναι ικανά να επεξεργαστούν αυτόν τον μεγάλο όγκο δεδομένων και να εξάγουν συμπεράσματα για το περιεχόμενό τους. Ένα τέτοιο παράδειγμα είναι ένα σύστημα που ανιχνεύει τα θέματα που γίνονται πολύ δημοφιλή. Το Storm παρέχει όλη την λειτουργικότητα που είναι απαραίτητη για να δημιουργηθεί ένα τέτοιο σύστημα, μέσω αλγορίθμων που θα καθορίσει ο χρήστης. Επί του παρόντος, περισσότερες από 70 εταιρίες χρησιμοποιούν και δηλώνουν την υποστήριξή τους για το Storm, όπως φαίνεται και στην σελίδα του[3].

Το Storm είναι ιδανικό για επεξεργασία σε πραγματικό χρόνο, καθώς είναι:

- **γρήγορο** – αξιολογήθηκε ως ικανό να επεξεργαστεί έως και 1 εκατομμύριο μηνύματα μεγέθους 100 bytes το δευτερόλεπτο ανά κόμβο.
- **κλιμακώσιμο**.
- **ανθεκτικό σε σφάλματα** – παρέχει μηχανισμό αυτόματης επανεκκίνησης των “πεθαμένων” κόμβων- εργατών.
- **αξιόπιστο** – παρέχει μηχανισμούς εξασφάλισης της επεξεργασίας κάθε μονάδας δεδομένων ακριβώς μια φορά.
- **εύκολο στη χρήση**.
- **εύκολο στον προγραμματισμό –** μειώνει σημαντικά την περιπλοκότητα της επεξεργασίας σε πραγματικό χρόνο και υποστηρίζει σχεδόν όλες τις γλώσσες εφόσον ο προγραμματιστής κατασκευάσει μια μικρή ενδιάμεση βιβλιοθήκη.
- **εύκολο στην επίβλεψη –** παρέχει ένα γραφικό τρόπο αναπαράστασης του παραγόμενου συστήματος, για την ανίχνευση λαθών και την εξαγωγή στατιστικών εκτέλεσης.

[1] https://twitter.com

[2] https://about.twitter.com/company

[3] https://storm.apache.org

Υπάρχουν δύο είδη κόμβων σε μια συστοιχία υπολογιστών Storm: ο κόμβος αφέντης και οι κόμβοι εργάτες. Ο κόμβος αφέντης (λεγόμενος και **Nimbus**) είναι υπεύθυνος για την επίβλεψη της συστοιχίας και αναθέτει μέρη της δουλειάς στους εργάτες. Οι κόμβοι εργάτες είναι υπεύθυνοι για την εκτέλεση των εργασιών που τους έχουν ανατεθεί. Σύμφωνα με τις εντολές του Nimbus, τρέχουν έναν daemon επίβλεψης (Supervisor), ο οποίος διαχειρίζεται τις επιμέρους εργασίες. Η επικοινωνία μεταξύ του Nimbus και των εργατών επιτυγχάνεται μέσω μιας συστοιχίας Zookeeper που συντονίζει τους κόμβους.

4.2.2 Συνιστώσες μιας συστοιχίας Storm

Για να υλοποιηθεί ένα σύστημα Storm χρησιμοποιούνται μια σειρά από συνιστώσες που εισάγουν την επιθυμητή λειτουργικότητα για την είσοδο, αναπαράσταση, επεξεργασία και έξοδο δεδομένων στο σύστημα. Οι κυριότερες συνιστώσες που συναντάμε σ' ένα σύστημα είναι η τοπολογία, οι τούπλες, τα spouts, τα bolts και ο κατανεμημένος RPC εξυπηρετητής.

Η κύρια οντότητα που τρέχει σε ένα μια συστοιχία Storm είναι μια **τοπολογία**. Η τοπολογία είναι ένας γράφος σε υψηλό επίπεδο αφαίρεσης, ο οποίος περιγράφει το συνολικό υπολογισμό παραστώντας τις συνδέσεις μεταξύ των διαφορετικών κόμβων και τη λογική σε κάθε κόμβο. Μπορεί να περιγραφεί σαν ένα δίκτυο από οντότητες μετασχηματισμού όπου τα δεδομένα ρέουν συνεχόμενα. Η τοπολογία αποτελείται από spouts και bolts, τα οποία υλοποιούν τις επιμέρους διαδικασίες. Τα δεδομένα υφίστανται επεξεργασία σαν ροή, το οποίο σημαίνει ότι ο υπολογισμός είναι άπειρος, εφόσον υπάρχει διαθέσιμη είσοδος.

Ένα **spout** αποτελεί το κύριο σημείο εισαγωγής δεδομένων σε μια τοπολογία Storm. Είναι ένα στοιχείο που δημιουργεί τη ροή εισόδου, εκπέμποντας μια λίστα προκαθορισμένης δομής. Τυπικά, ένα spout διαβάζει από ένα σύστημα ουράς, όπως το RabbitMQ[1] ή το Kafka[2], το οποίο μπορεί να είναι ένας διαμεσολαβητής ή μια διεπαφή προγραμματισμού εφαρμογών για ροές δεδομένων. Επιπλέον έχει την ικανότητα να παράγει τη δική του ροή δεδομένων. Αυτή η αρχιτεκτονική επιτρέπει να δημιουργηθούν διαφορετικά είδη bolts, που να λαμβάνουν είσοδο από ένα μοναδικό spout.

[1] http://www.rabbitmq.com

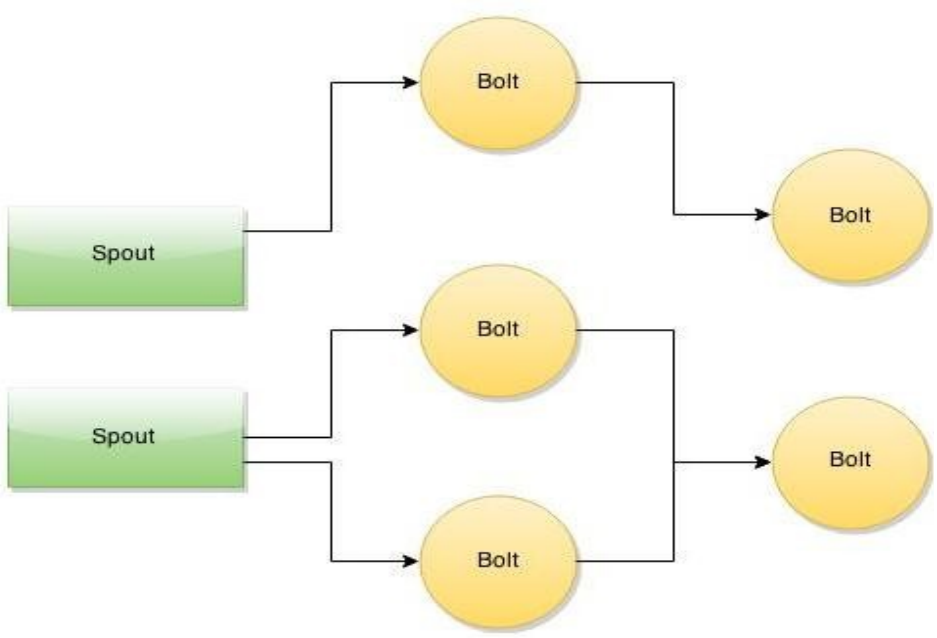
[2] http://kafka.apache.org

Η λίστα με τα προκαθορισμένα πεδία που εκπέμπει το spout αναφέρεται ως "**τούπλα**", που είναι μια στοιχειώδης δομή σε μια συστοιχία Storm. Μια τούπλα είναι ένα σειριοποιήσιμο αντικείμενο που εκπέμπεται μεταξύ των κόμβων εργατών, για να λάβει μέρος σε μια σειρά από συγκεκριμένους υπολογισμούς. Ο λόγος που οι τούπλες είναι σειριοποιήσιμα αντικείμενα, είναι η ευελιξία που απαιτείται σε ένα κατανεμημένο σύστημα. Εφόσον δεν δηλώνονται οι τύποι των πεδίων της τούπλας, τα bolt πρέπει να έχουν την δυνατότητα να μετατρέπουν τα αντικείμενα από την σειριοποιημένη μορφή στην κανονική και αντίστροφα, με έναν γενικό τρόπο. Για την σειριοποίηση των απλών τύπων, όπως List, HashMap και Array, το Storm χρησιμοποιεί τη βιβλιοθήκη Kryo[1], που παράγει γρήγορες και απλές σειριοποιήσεις.

Για κάθε μια από τις πιο σύνθετες δομές, προτείνεται να δημιουργηθεί ένας ειδικός σειριοποιητής Kryo, διαφορετικά το Storm θα επιδιώξει να χρησιμοποιήσει τον γενικό της σειριοποιητή της Java, ο οποίος είναι υπολογιστικά ακριβός. Στην περίπτωση που το αντικείμενο προς μεταφορά δεν είναι σειριοποιήσιμο, το Storm θα επιστρέψει ένα μήνυμα λάθους και θα τερματίσει την τοπολογία. Η ατέρμονη σειρά από τις τούπλες που μεταφέρονται μεταξύ των κόμβων και μετασχηματίζονται, αποτελεί την ροή των δεδομένων.

Οι συνιστώσες που υλοποιούν τους υπολογισμούς πάνω στη ροή από τις τούπλες αποκαλούνται **bolts**. Ένα bolt μπορεί να επεξεργαστεί τούπλες εισόδου με πολλαπλούς τρόπους: μπορεί να εκτελέσει συναρτήσεις, να εφαρμόσει φίλτρα και αθροίσεις ή ενοποιήσεις. Τέλος μπορεί να αποθηκεύσει και να ανακτήσει τα δεδομένα χρησιμοποιώντας μια βάση δεδομένων. Για να πραγματοποιήσει πιο περίπλοκους υπολογισμούς απαιτούνται πολλαπλά επίπεδα από bolts, έτσι ώστε να φτιαχτεί το δίκτυο των μετασχηματισμών. Τα bolts μπορούν να λάβουν δεδομένα που εκπέμπονται από spouts, ή από άλλα bolts και να εκπέμψουν τούπλες για το επόμενο επίπεδο αν χρειάζεται.

[1] https://code.google.com/p/kryo/



Εικόνα 4.2.1. Παράδειγμα μιας τοπολογίας Storm

Μια τελευταία συνιστώσα τους Storm είναι ο **κατανεμημένος RPC** εξυπηρετητής. Από μια οπτική υψηλού επιπέδου, ο DRPC εξυπηρετητής είναι υπεύθυνος για να εκτελεί απομακρυσμένες κλήσεις συναρτήσεων στην τρέχουσα τοπολογία για να λάβει αποτελέσματα στη στιγμή. Ένα πρόγραμμα-πελάτης στέλνει στον DRPC εξυπηρετητή το όνομα της συνάρτησης προς κλήση και τις παραμέτρους της συνάρτησης. Ο εξυπηρετητής είναι ικανός να λάβει τα αποτελέσματα από την τοπολογία, να εκτελέσει έντονους υπολογισμούς και να στείλει πίσω τα αποτελέσματα.

Όταν εκτελείται μια τοπολογία Storm, 4 εμπλεκόμενοι παράγοντες επηρεάζουν τον επιτευχθέντα παραλληλισμό:

- **κόμβος**: οι μηχανές που ρυθμίζονται ως συμμετέχοντες της συστοιχίας Storm. Τίθεται κατά την εγκατάσταση της συστοιχίας.
- **εργάτης**: η ανεξάρτητη διεργασία που εκτελεί ένα υποσύνολο της τοπολογίας και τρέχει σε δικό της εικονικό μηχάνημα Java. Μπορεί να τρέξει έναν ή περισσότερους εκτελεστές για τις συνιστώσες της (spouts ή bolts). Ρυθμίζεται κατά την δημιουργία της τοπολογίας.
- **εκτελεστής**: το νήμα που τρέχει μέσα σε μια διεργασία εργάτη. Μπορεί να εκτελεί μια ή περισσότερες υποεργασίες της ίδιας συνιστώσας. Οι υποεργασίες τρέχουν με σειριακό

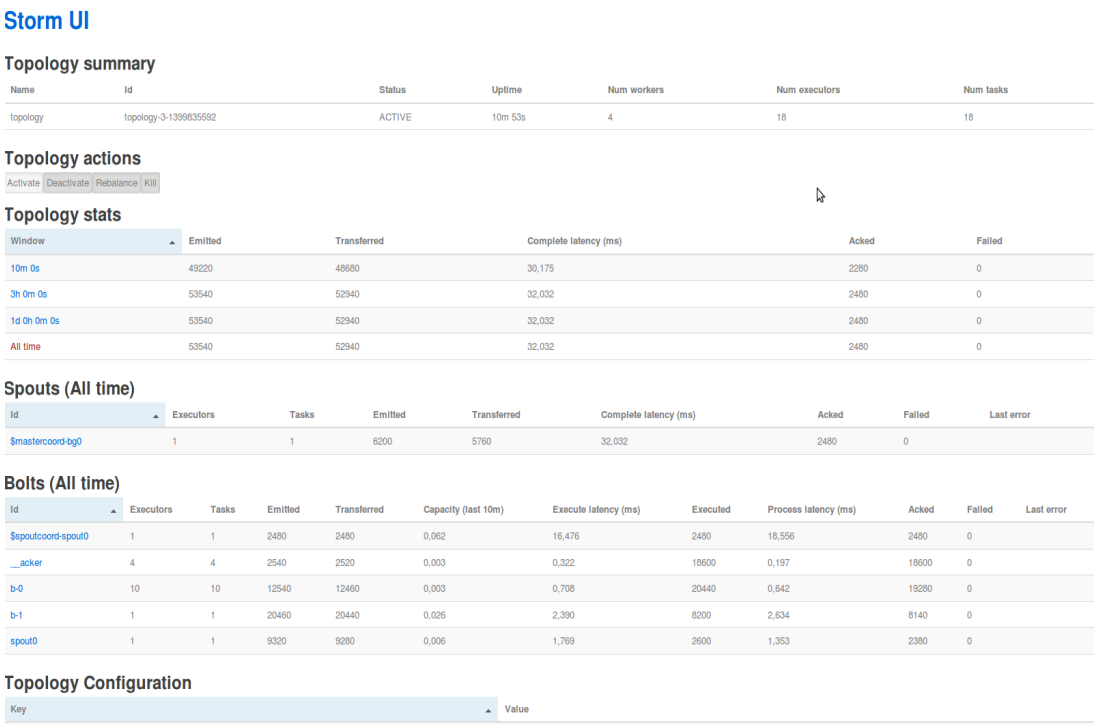
τρόπο σε έναν εκτελεστή. Ο αριθμός των εκτελεστών για κάθε συνιστώσα μπορεί να οριστεί θέτοντας τον παράγοντα *parallelism hint*.

- **υποεργασία**: η ουσιαστική επεξεργασία των δεδομένων, σαν τμήμα ενός spout, ή bolt που τρέχει από έναν εκτελεστή. Από προεπιλογή, αν δεν προσδιοριστεί ο αριθμός των υποεργασιών είναι ίδιος με τον αριθμό των εκτελεστών. Μπορεί να τεθεί για κάθε μια συνιστώσα (spout ή bolt), μέσω της ιδιότητας *NumTasks*.

Συνδυάζοντας τους παραπάνω υποπαράγοντες, μπορεί να υπολογιστεί ο συνολικός παράγοντας παραλληλισμού του συστήματος. Το Storm παρέχει μια εντολή για να αλλάζει ο παραλληλισμός της τοπολογίας κατά της εκτέλεση: την εντολή *rebalance*.

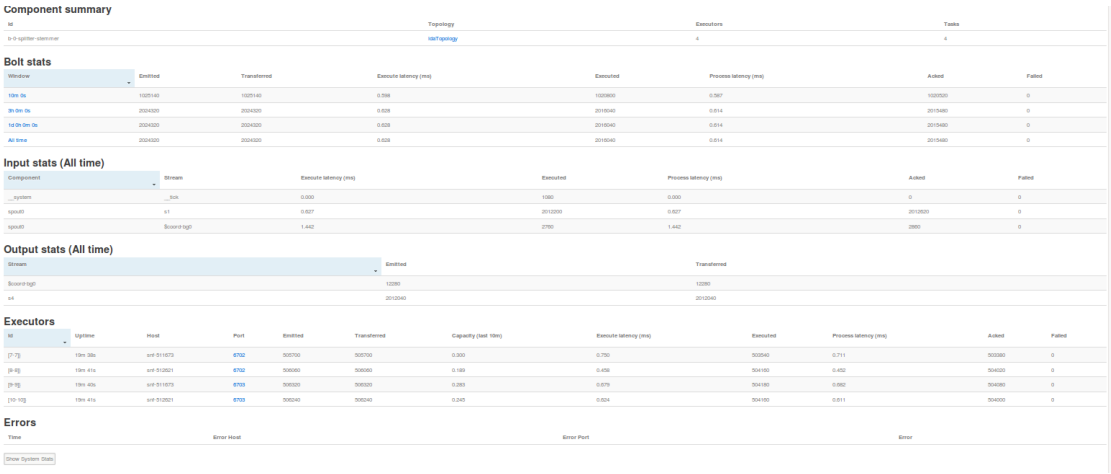
4.2.3 Περιβάλλον χρήστη Storm

Το Storm παρέχει επίσης ένα γραφικό περιβάλλον χρήστη, που μπορεί να χρησιμοποιηθεί για την επίβλεψη των τρεχόντων τοπολογιών, την παρακολούθηση των στατιστικών των εργατών, και την πραγματοποίηση συγκεκριμένων ενεργειών πάνω τους (Ενεργοποίηση – Απενεργοποίηση, Εξισορρόπηση, Σκότωμα).



Εικόνα 4.2.2. Γραφικό περιβάλλον χρήστη Storm

Οι πληροφορίες που είναι διαθέσιμες στο γραφικό περιβάλλον αφορούν τον αριθμό των τουπλών που έχουν εκπεμφθεί, εκτελεστεί και επιβεβαιωθεί από όλα τα bolts. Ως *$spoutcoord-spout0* αναφέρεται μια γενική συνιστώσα που υπάρχει σε όλες τις τοπολογίες και λειτουργεί ως συντονιστής, ανάλογα με το είδος του spout που χρησιμοποιείται. Κάτω από αυτή, εμφανίζονται τα spout και bolt της τοπολογίας, με τα αναλυτικά στατιστικά τους. Για το κάθε μία από τις συνιστώσες της τοπολογίας μπορούμε να λάβουμε αναλυτικότερες πληροφορίες, για την είσοδο και την έξοδο των τουπλών και τους εκτελεστές στους οποίους εκτελείται, ανοίγοντας τη σελίδα για τις πληροφορίες του. Ένα τέτοιο παράδειγμα φαίνεται στην εικόνα 4.2.3.



Εικόνα 4.2.3. Γραφικό περιβάλλον για τις πληροφορίες συγκεκριμένης συνιστώσας τοπολογίας Storm

Τέλος, στις νεότερες εκδόσεις του Storm υπάρχει η δυνατότητα γραφικής αναπαράστασης της τοπολογίας. Όλες οι πληροφορίες που απεικονίζονται στο περιβάλλον χρήστη λαμβάνονται μέσω REST API από τον Nimbus.

4.2.4 Κύκλος ζωής μιας τοπολογίας

Μια τοπολογία Storm υποβάλλεται στην συστοιχία από τον Nimbus με την εντολή *storm jar filename.jar mainClass topologyName.* Αναλυτικές οδηγίες για την εγκατάσταση της συστοιχίας που πρέπει να έχει προηγηθεί δίνονται στο Παράρτημα. Αυτή η εντολή θέτει

τις μεταβλητές περιβάλλοντος που θα χρησιμοποιηθούν αργότερα από την κλάση που αναλαμβάνει να υποβάλλει την τοπολογία (*StormSubmitter*).

Ο *StormSubmitter* ανεβάζει το jar αρχείο στον client του Nimbus, που θα αναλάβει να το υποβάλλει στη συστοιχία. Το αρχείο ανεβαίνει σε κομμάτια των 15kB. Έπειτα, καλεί την συνάρτηση *submitTopology*, η οποία σειριοποιεί τις ρυθμίσεις της τοπολογίας και εκτελεί την υποβολή της. Μόλις λάβει την επιβεβαίωση ότι η τοπολογία υποβλήθηκε επιτυχώς, επιβεβαιώνει ότι οι ρυθμίσεις είναι ίδιες σε όλους τους κόμβους της τοπολογίας, ώστε να ξεκινήσει να αναθέτει υποεργασίες σε αυτούς. Για να αναθέσεις τις υποεργασίες πρέπει να δημιουργήσει μια αντιστοίχηση μεταξύ των υποεργασιών και των κόμβων που θα τις αναλάβουν. Αυτά, σε συνδυασμό με τις ρυθμίσεις και το αρχείο .jar ορίζουν την στατική κατάσταση της τοπολογίας.

Με την εντολή *mk-assignment* ο Nimbus αναθέτει στους εργάτες τις υποεργασίες που θα εκτελέσουν. Αρχικά, η τοπολογία είναι σε αδρανή μορφή και περιμένει από τον Nimbus το σήμα για να ξεκινήσει να εκπέμπει τούπλες από τα spouts. Μόλις ξεκινήσει, δύο διεργασίες Zookeeper του *Supervisor*, αναλαμβάνουν να κατεβάζουν τον απαιτούμενο κώδικα στους κόμβους εργάτες και να ενημερώνουν τις αναθέσεις των εκτελούμενων υποεργασιών όταν αυτές αλλάζουν. Επίσης, οι εργάτες ξεκινάνε μια διεργασία που είναι υπεύθυνη για τον δικό τους συντονισμό. Αυτή η διεργασία αρχικοποιεί τις συνδέσεις των εργατών, για να ξεκινήσει η επικοινωνία μεταξύ τους, και εκκινεί μια διεργασία- νήμα, η οποία ελέγχει για αλλαγές. Για παράδειγμα, αν κάποιος εργάτης σκοτωθεί και ανατεθεί σε άλλο μηχάνημα, πρέπει να γίνει σύνδεση με την νέα τοποθεσία του μηχανήματος. Παράλληλα, ενημερώνει την κατάσταση της τοπολογίας στη μεταβλητή *storm-active-atom*, που χρησιμοποιείται από τα spout, ώστε να καθορίσουν αν θα εκπέμψουν επόμενη τούπλα ή όχι. Τέλος, εκτελεί τις υποεργασίες τρέχοντας τον κώδικα που της αντιστοιχεί.

Κατά τη διάρκεια της ζωής της τοπολογίας, ο Nimbus την επιβλέπει κάνοντας ελέγχους ανά τακτά χρονικά διαστήματα. Σύμφωνα με τη μεταβλητή "*nimbus.monitor.freq.secs*", που έχει τεθεί κατά την ρύθμιση της τοπολογίας, καλείται η συνάρτηση που ελέγχει αν οι κόμβοι είναι ζωντανοί και επαναλαμβάνει τις αναθέσεις των υποεργασιών, αν είναι απαραίτητο.

Τέλος, με την εντολή *kill* σκοτώνεται η τοπολογία. Όταν ο Nimbus λάβει αυτή την εντολή, την στέλνει στους κόμβους και δίνει ένα διάστημα αναμονής, ώστε να ολοκληρώσουν τις τρέχουσες διεργασίες. Σε αυτό το διάστημα η κατάσταση της τοπολογίας είναι “σκοτωμένη”, έτσι ώστε να διασφαλιστεί ότι σε περίπτωση αποτυχίας του Nimbus, όταν επανεκκινηθεί η συστοιχία, ο Nimbus θα αναγνωρίσει την σκοτωμένη τοπολογία και θα

εκτελέσει τη συνάρτηση αναμονής, πριν τη σκοτώσει οριστικά. Ένα παράλληλο νήμα αναλαμβάνει να καθαρίσει τους φακέλους που χρησιμοποιούνται από τους εργάτες για να δηλώσουν την παρουσία τους καθώς και τους φακέλους στους οποίους αποθηκεύονται τα τμήματα κώδικα και οι ρυθμίσεις.

4.3 Trident: υπολογισμός στην κορυφή του Storm

Το **Trident** είναι μια επέκταση του Storm, που το καθιστά ένα εύκολο στη χρήση, κατανεμημένο, framework πραγματικού χρόνου για μεγάλο όγκο δεδομένων. Κατά κύριο λόγο υλοποιεί μια αφαίρεση υψηλού επιπέδου για τη σχεδίαση υπολογισμών πραγματικού χρόνου πάνω στο Storm, προσφέροντας μια διεπαφή προγραμματισμού εφαρμογών, η οποία αυτοματοποιεί τις πιο συνήθεις διαδικασίες. Το Trident χειρίζεται τις τούπλες σαν μια ροή δεδομένων, στο οποίο εφαρμόζονται πολλές διαδικασίες σαν συναρτήσεις. Συνεπώς η αλληλουχία του τουπλών είναι για το Trident τα πεδία εισόδου και εξόδου για τις σχετικές συναρτήσεις. Κάθε επίπεδο υπολογισμού καταναλώνει ένα πεδίο εισόδου για να παράξει αν χρειάζεται ένα πεδίο εξόδου.

Για να μετασχηματίσει τη ροή, το Trident τη χωρίζει σε τμήματα κατά μήκος των κόμβων της συστοιχίας. Οι επιμέρους υποροές τρέχουν παράλληλα στους διαφορετικούς κόμβους της συστοιχίας. Οι διαδικασίες που λαμβάνουν χώρα, μπορεί να είναι τοπικές ή κατά μήκος ολόκληρης της συστοιχίας. Στην πρώτη περίπτωση δεν απαιτούν μεταφορά μέσω του δικτύου και μπορούν να εκτελεστούν ανεξάρτητα σε κάθε τμήμα. Στην δεύτερη περίπτωση απαιτείται δικτυακή μεταφορά δεδομένων μεταξύ των κόμβων, γιατί τα δεδομένα δεν είναι λογικά ανεξάρτητα μεταξύ τους. Ένα καλό παράδειγμα για να φανεί η διαφορά μεταξύ των τοπικών και των μη-τοπικών διαδικασιών είναι οι συναρτήσεις άθροισης: η συνάρτηση *partitionAggregate* αθροίζει όλες τις τούπλες σε ένα συγκεκριμένο τμήμα, ενώ η συνάρτηση *persistentAggregate* αθροίζει όλες τις τούπλες, από όλη τη ροή, και αποθηκεύει το αποτέλεσμα σε μια TridentState. Συνεπώς, ο συνολικός υπολογισμός μπορεί να γίνει με έναν κομψό και αδιαφανή τρόπο, χρησιμοποιώντας την διεπαφή προγραμματισμού εφαρμογών του Trident, η οποίο παρέχει:

- διεργασίες που εκτελούνται τοπικά σε κάθε τμήμα της ροής
- διεργασίες που ανακατανέμουν τη ροή στους κόμβους, με βάση κάποια λογικά κριτήρια

- διεργασίες άθροισης που επιτελούν δικτυακή μεταφορά δεδομένων, σαν κομμάτι της διεργασίας
- διεργασίες σε ομαδοποιημένες ροές δεδομένων
- συγχωνεύσεις και ενοποιήσεις των ροών

Ένα ακόμη κύριο πλεονέκτημα του Trident είναι ότι αντίθετα με το Storm, οι υπολογισμοί διατηρούν μια κατάσταση. Αυτή την κατάσταση τη διαχειρίζεται το Trident με έναν μηχανισμό που εγγυάται την σωστή ενημέρωσή της ώστε το σύστημα να είναι συνεπές. Ο μηχανισμός αυτός λειτουργεί ως εξής:

Οι τούπλες ομαδοποιούνται σε μικρές παρτίδες, και σε κάθε μια από αυτές δίνεται ένα μοναδικό αυξανόμενο αναγνωριστικό κωδικό (tx-id). Όταν μια παρτίδα φτάσει σε ένα bolt για επεξεργασία, προτού γίνει ενημέρωση στην συνολική κατάσταση ελέγχεται αν όλες οι προηγούμενες παρτίδες έχουν ολοκληρωθεί. Με αυτό τον τρόπο εξασφαλίζεται η επεξεργασία της κάθε τούπλας μόνο μια φορά. Το Trident παρέχει όλες τις απαραίτητες κλάσεις για να υλοποιηθούν εύκολα αυτού του είδους οι καταστάσεις με βέλτιστο τρόπο. Παρέχει επίσης ενσωματωμένους μηχανισμούς κρυφής μνήμης για ακόμη μεγαλύτερη ταχύτητα στην ενημέρωση της κατάστασης.

Η κατάσταση μπορεί να είναι είτε εσωτερική στην τοπολογία (αποθηκευμένη στη μνήμη) ή εξωτερική, αποθηκευμένη σε μια βάση δεδομένων όπως η Memcached[1] , η Cassandra[2] ή η Redis[3].

Μόλις μια τοπολογία Trident εγκατασταθεί, αυτόματα γίνεται σύνθεσή της σε Storm spouts και bolts με έναν αποδοτικό τρόπο. Όλες οι διαδικασίες που περιγράφηκαν παραπάνω είναι βελτιστοποιημένες σε βαθμό που να επιτυγχάνεται η μέγιστη επίδοση. Συνεπώς, η ανάπτυξη ενός περίπλοκου, υψηλής επίδοσης και αξιόπιστου συστήματος πραγματικού χρόνου είναι εννοιολογικά απλή και αποτελεσματική με τη χρήση του Trident.

[1] http://memcached.org

[2] http://cassandra.apache.org

[3] http://redis.io

4.4 Βάση Δεδομένων Redis

Η βάση δεδομένων Redis ανήκει στην κατηγορία των μη-σχεσιακών βάσεων. Οι μη-σχεσιακές βάσεις δεδομένων, όπως φαίνεται από το όνομά τους, προσφέρουν μηχανισμούς αποθήκευσης των δεδομένων με τρόπους διαφορετικούς από τον σχεσιακό. Τα δεδομένα δεν αποθηκεύονται σε πίνακες, αλλά σε άλλου είδους δομές, όπως σύνολα και λίστες. Η Redis είναι μια ελαφριά βάση δεδομένων που προσφέρεται για τη αποθήκευση δομών, μεταξύ των οποίων σύνολα, λίστες, ταξινομημένα σύνολα και ζευγάρια κλειδιών- τιμής. Πετυχαίνει εξαιρετικές επιδόσεις, αποθηκεύοντας τα δεδομένα στη μνήμη. Τα δεδομένα μπορούν να γραφτούν και στο δίσκο για μόνιμη διατήρηση. Παρόλα αυτά, αν απενεργοποιηθεί η διατήρηση των δεδομένων, τότε λειτουργεί σαν μια πολύ γρήγορη βάση δεδομένων πάνω στη μνήμη.

Εκτός από την επίδοσή της, η Redis, προσφέρει πολλά χαρακτηριστικά που την καθιστούν μια εύχρηστη επιλογή:

- προγραμματισμό στη γλώσσα σεναρίων LUA
- μηχανισμό αντικατάστασης των κλειδιών με χρήση του αλγορίθμου LRU
- εκτέλεση εντολών με σωλήνωση
- κλειδιά με πεπερασμένο χρόνο ύπαρξης
- μηχανισμό ειδοποίησης για γεγονότα

Μπορεί να χρησιμοποιηθεί με τις περισσότερες γλώσσες προγραμματισμού, όπως για παράδειγμα C, C#, C++, Erlang, Java, Prolog, Python και Haskell, χρησιμοποιώντας τον αντίστοιχο client. Ένας από τους επικρατέστερους clients για Java είναι ο Jedis.[1]

ΕΛΕΓΧΟΣ ΕΠΙΔΟΣΗΣ

Η Redis παρέχει ένα εργαλείο για μέτρηση της επίδοσης της, το οποίο προσομοιώνει την εκτέλεση M ταυτόχρονων εντολών συγκεκριμένου τύπου (πχ SET σε σύνολο, PUSH σε λίστα) από N πελάτες στον εξυπηρετητή. Χρησιμοποιώντας αυτό το εργαλείο μπορούμε να υπολογίσουμε πόσες εντολές μπορούν να εκτελεστούν σε 1 δευτερόλεπτο στο κάθε μηχάνημα.

[1] https://github.com/xetorthio/jedis

Εκτελούμε την εντολή *redis-benchmark* με τις επιθυμητές ρυθμίσεις:

- -t: οι εντολές που θέλουμε να προσομοιάσουμε
- -n: ο αριθμός των ταυτόχρονων εντολών που στέλνει ο κάθε πελάτης (M)
- -c: ο αριθμός των ταυτόχρονων πελατών (N)
- -P: αριθμός σταδίων σωλήνωσης

Ενδεικτικά, παρουσιάζονται τα αποτελέσματα που προέκυψαν από την εκτέλεση των εντολών σε μηχάνημα της συστοιχίας Storm:

```
redis-benchmark -t set,lpush -n 100000 -c 50 -q
SET: 74404.77 requests per second
LPUSH: 74404.77 requests per second

redis-benchmark -t set,lpush -n 1000000 -c 50 -P 16 -q
SET: 270416.44 requests per second
GET: 243249.81 requests per second
```

Προκύπτει ότι κατά μέσο όρο παραπάνω από 70000 εντολές μπορούν να εκτελεστούν από τον εξυπηρετητή σε 1 δευτερόλεπτο. Η χρήση της σωλήνωσης αύξησε στο τριπλάσιο την επίδοση για την ίδια εντολή. Συνεπώς, αποτελεί ένα σημαντικό χαρακτηριστικό της Redis για τις σύγχρονες εφαρμογές που μπορούν να το εκμεταλλευτούν. Ουσιαστικά, με τη χρήση σωλήνωσης, η κάθε εντολή στέλνεται από τους πελάτες πριν τελειώσει η εκτέλεση της προηγούμενης εντολής. Έτσι με μια κλήση ανάγνωσης ο εξυπηρετητής μπορεί να διαβάσει παραπάνω από μια εντολές από τον κάθε πελάτη. Ο χρόνος εκτέλεσης της κάθε εντολής συνολικά μειώνεται δραματικά, αφού τα στάδια εκτελούνται παράλληλα.

5 *Υλοποίηση του αλγορίθμου Λανθάνουσας κατανομής Dirichlet*

Ο στόχος αυτής της υλοποίησης είναι η δημιουργία μιας Τοπολογίας που επεξεργάζεται μια ατέρμονη συλλογή κειμένων με σκοπό την ανάδειξη των θεμάτων της σε πραγματικό χρόνο.

5.1 Χρησιμοποιούμενες Τεχνολογίες

Για την υλοποίηση του αλγορίθμου της Λανθάνουσας Κατανομής Dirichlet στο περιβάλλον του Storm, σύμφωνα με τη θεωρητική ανάλυση, επιλέγουμε το μοντέλο της συνδεδεμένης μεταβολικής μεθόδου κατά Bayes. Αυτό το μοντέλο είναι ιδανικό για υλοποίηση στο Storm, λόγω της ικανότητάς του να εξάγει αποτελέσματα με μοναδική επεξεργασία των κειμένων εισόδου. Η φύση του αλγορίθμου είναι σύμφωνη με τη επεξεργασία σε πραγματικό χρόνο του Storm.

Για τον σχεδιασμό της τοπολογίας επιλέγουμε τη χρήση του Trident, λόγω της απλότητας και της αποτελεσματικότητας των συναρτήσεων που παρέχει. Στηριζόμαστε για την υλοποίησή μας σε μια υπάρχουσα υλοποίηση του αλγορίθμου Συνδεδεμένης Λανθάνουσας Κατανομής Dirichlet σε Java[1].

[1] https://github.com/miberk/jolda

Επιλέξαμε τη βάση Redis για την αποθήκευση των δεδομένων. Η ταχύτατη αποθήκευση και ανάκτηση δεδομένων που παρέχει, είναι απαραίτητη για μια εφαρμογή πραγματικού χρόνου. Η επικοινωνία με τη βάση επιτυγχάνεται με τη χρήση του Jedis client.

Τέλος, για τη διαχείριση του έργου χρησιμοποιήθηκε ο Apache Maven[1]. Ο Apache Maven είναι ένα εργαλείο που οργανώνει την δομή και τις εξαρτήσεις του κώδικα και αυτοματοποιεί τις διαδικασίες στα στάδια ζωής του έργου (ανάπτυξη, έλεγχος, πακετάρισμα, εγκατάσταση).

5.2 Υλοποίηση Τοπολογίας

Η Τοπολογία περιλαμβάνει τα εξής στάδια:

- είσοδος δεδομένων σε μορφή μικρών προτάσεων (tweets) από spout
- ανάλυση και επεξεργασία των προτάσεων και εφαρμογή φίλτρων σε bolts
- εφαρμογή του αλγορίθμου Λανθάνουσας Κατανομής Dirichlet σε bolt

5.2.1 Είσοδος δεδομένων

Χρησιμοποιήθηκαν 3 είδη spout για την εισαγωγή των δεδομένων στην Τοπολογία:

- TextFileSpout : για είσοδο των δεδομένων από μεγάλα αρχεία κειμένου.
- TwitterSampleSpout: για είσοδο των δεδομένων από τη δημόσια ροή tweets του Twitter. Για να αποκτήσουμε πρόσβαση στη ροή χρησιμοποιούμε την προγραμματιστική διεπαφή εφαρμογών του Twitter. Αρχικά, πρέπει να χρησιμοποιήσουμε τον μηχανισμό ταυτοποίησης Oauth 2.0. Με την χρήση αυτού του μηχανισμού, επιβάλλεται να παρέχουμε τα απαραίτητα consumerKey, consumerSecret, oauthAccessToken και oauthAccessSecret για την ταυτοποίηση της εφαρμογής μας, έτσι ώστε να έχουμε πρόσβαση σε συγκεκριμένες λειτουργίες. Για να είναι ασφαλής η διαδικασία της ταυτοποίησης, κάθε ανάλογη αίτηση πρέπει να γίνεται μέσω SSL, με χρήση απολητκικών σημείων https. Εφόσον η ταυτοποίηση είναι επιτυχής, μια από τις διαθέσιμες λειτουργίες είναι και η προσπέλαση (GET) της δημόσιας ροής[2] μέσω του REST API που παρέχει το Twitter. Η δημόσια ροή είναι ένα τυχαίο υποσύνολο των δημόσιων tweets που είναι διαθέσιμη με τον ίδιο τρόπο για κάθε συνδεδεμένο πελάτη.

[1] http://maven.apache.org/

[2] https://stream.twitter.com/1.1/statuses/sample.json

- SocketSpout: για είσοδο των δεδομένων μέσω ενός εξυπηρετητή, ο οποίος δέχεται συνδέσεις πελατών σε κάποια θύρα. Όταν ένας πελάτης συνδεθεί στον εξυπηρετητή, δημιουργείται μια TCP σύνδεση μεταξύ τους. Το τελικό σημείο αυτής της σύνδεσης ταυτοποιείται από την διεύθυνση IP του εξυπηρετητή και τη θύρα, στην οποία συνδέονται οι πελάτες. Ο εξυπηρετητής δέχεται τα μηνύματα του πελάτη και τα εισάγει σε μια ουρά, από την οποία διαβάζει το spout.

5.2.2 Ανάλυση των κειμένων

Το επόμενο στάδιο είναι να αναλύσουμε τα εισερχόμενα tweets, ώστε να κρατήσουμε την απαραίτητη πληροφορία – το καθαρό περιεχόμενο του κειμένου. Για να το πετύχουμε αυτό εφαρμόζουμε μια σειρά από μεθόδους μετασχηματισμού στη ροή. Χρησιμοποιούμε τη συνάρτηση *.each* του Trident, η οποία εφαρμόζει την μέθοδο που δίνεται ως παράμετρος σε κάθε εισερχόμενη δεσμίδα από τούπλες. Όπως έχει αναφερθεί στην περιγραφή του Trident, για λόγους βελτιστοποίησης οι τούπλες “πακετάρονται” σε μικρές δεσμίδες και περνάνε από τα bolts.

Έτσι, σε κάθε επιμέρους συνάρτηση που αναφέρεται παρακάτω, διαχειριζόμαστε την είσοδο σαν μικρή δεσμίδα από τούπλες. Με την σειρά, εφαρμόζονται οι μέθοδοι Split, Stem και Remove StopWords για να έχουμε ως αποτέλεσμα τις ρίζες των λέξεων από τα tweets, παραβλέποντας τις κοινές λέξεις. Ως κοινές λέξεις, θεωρούμε μια λίστα από stopWords, που περιλαμβάνει άρθρα, συνδέσμους και συνήθεις λέξεις που δεν εισάγουν περιεχόμενο. Επίσης, αποκλείουμε τις συνήθεις λέξεις του Twitter, όπως via, RT και όσες ξεκινάνε από @. Κάθε μια από αυτές τις μεθόδους εκπέμπει το αποτέλεσμα της, για να καταναλωθεί από το επόμενο στάδιο επεξεργασίας. Έπειτα, οι τούπλες που αποτελούν την κάθε δεσμίδα αθροίζονται με έναν *ListAggregator*, ώστε να δοθούν σαν μια είσοδος στο τελευταίο στάδιο, για λόγους βελτιστοποίησης. Με τον παράγοντα *parallelismHint*, ορίζουμε το επίπεδο παραλληλίας για τα πρώτα στάδια της επεξεργασίας, τον οποίο μπορούμε να θέσουμε κατά βούληση, ανάλογα με τον αριθμό των διαθέσιμων μηχανημάτων που έχουμε για επεξεργασία. Έτσι, οι συναρτήσεις *.each* και *.aggregate* εκτελούνται παράλληλα σε N υποροές που κατανέμονται σαν εκτελεστές στους κόμβους επεξεργασίας.

5.2.3 Εκτέλεση του αλγορίθμου

Στο τελευταίο στάδιο, εφαρμόζεται ο αλγόριθμος με τη χρήση της συνάρτησης *persistentAggregate*. Αυτή η συνάρτηση είναι μη-τοπική και εφαρμόζεται σε όλες τις τούπλες κατά μήκος όλων των υποροών στους κόμβους. Ουσιαστικά, οι τούπλες από τις N υπορροές συγκεντρώνονται σε έναν κόμβο, ώστε να αθροιστούν σε μια κοινή *TridentState*. Αυτός ο περιορισμός εισάγεται λόγω της φύσης του αλγορίθμου και επιτυγχάνεται με τη χρήση της συνάρτησης *persistentAggregate*, η οποία συγκεντρώνει όλες τις τούπλες σε έναν κόμβο για να πραγματοποιήσει την άθροιση.

Για την άθροιση χρησιμοποιείται ένας αθροιστής που υλοποιεί τη διαπροσωπία *ReducerAggregator*. Ένας αθροιστής τέτοιου τύπου δημιουργεί ένα αντικείμενο, που αποτελεί την κατάσταση *TridentState*, στη συνάρτηση αρχικοποίησης *init*. Το αντικείμενο της άθροισης ανανεώνεται κάθε φορά που φτάνει μια τούπλα στον τελικό αθροιστή με τη συνάρτηση *reduce*. Οι αθροιστές τύπου *ReducerAggregator* είναι βελτιστοποιημένοι σε επίπεδο δεσμίδας. Αυτό σημαίνει ότι για κάθε δεσμίδα μεγέθους N που θα φτάσει στον αθροιστή, η συνάρτηση reduce θα εκτελεστεί N φορές πάνω στο αντικείμενο, αλλά θα αποθηκευτεί στην κατάσταση *TridentState* μια φορά: στο τέλος της δεσμίδας. Έτσι ελαχιστοποιείται η επικοινωνία με τη βάση, στην οποία αποθηκεύεται η κατάσταση.

Όλα τα παραπάνω αυτοματοποιούνται σε μεγάλο βαθμό με τη χρήση του Trident, αφού για να υλοποιηθούν οι προαναφερόμενες διαδικασίες στη ροή, το μόνο που χρειάζεται να παρέχουμε είναι τις υλοποιήσεις τους. Το Trident ασχολείται “κρυφά” από τον προγραμματιστή με το πως αυτές οι διαδικασίες θα εκτελεστούν σε bolts και πως τα bolts θα συνδεθούν μεταξύ τους, τοπικά ή και κατά μήκος της συστοιχίας.

5.2.4 Αποθήκευση των ενδιάμεσων καταστάσεων Trident

Για την αποθήκευση των καταστάσεων χρησιμοποιούμε τη βάση Redis. Στην κλάση *RedisMapState* έχει υλοποιηθεί όλη η απαραίτητη επικοινωνία με τον Redis εξυπηρετητή και η λογική για την αποθήκευση και την ανάκτηση των δεδομένων. Η συνάρτηση *persistentAggregate* δέχεται ως παράμετρο ένα αντικείμενο τύπου RedisMapState. Ως αποτέλεσμα, ο αθροιστής χρησιμοποιεί τις συναρτήσεις *multiPut* και *multiGet* του αντικειμένου για να αποθηκεύσει και να ανακτήσει την κατάσταση, αντίστοιχα. Έτσι, επιτυγχάνεται η ενημέρωση της κατάστασης. Το αντικείμενο που αποθηκεύεται, δηλαδή η ενδιάμεση κατάσταση είναι τύπου *OnlineLDA* και σειριοποιείται για να αποθηκευτεί στο Redis.

5.2.5 Χτίσιμο Τοπολογίας

Η συνάρτηση που χτίζει την συνολική τοπολογία είναι η *buildTopology*. Τα πεδία που αναφέρονται ως είσοδοι και έξοδοι στις συναρτήσεις του Trident είναι οι ενδιάμεσες καταστάσεις της ροής, ενώσω αυτή μετασχηματίζεται από το ένα στάδιο στο άλλο. Η συνάρτηση .name() χρησιμοποιήθηκε για την ονοματοδοσία των bolts, ώστε να είναι πιο εύκολη η παρατήρηση τους από το γραφικό περιβάλλον του Storm.

```
public static StormTopology buildTopology(StateFactory state)
                    throws IOException {

        // spout from Twitter random stream
        TwitterSampleSpout spout = new TwitterSampleSpout(username,passwd);
        TridentTopology topology = new TridentTopology();

        TridentState ldaState =
        topology
                .newStream("spout", spout)
                .parallelismHint(1)
                .shuffle()
                .each(new Fields("sentence"), new Splitter(),
                      new Fields("wordList")).name(“splitter”)
                .parallelismHint(4)
                .each(new Fields("wordList"), new Stemmer(),
                      new Fields("stemwordList")).name(“stemmer”)
                .each(new Fields("stemwordList"), new FilterApplier(),
                      new Fields("filteredList")).name(“filter”)
                .aggregate(new Fields("filteredList"), new ListAggregator(),
                      new Fields("aggregated"))
                .parallelismHint(4)
                .persistentAggregate(state, new Fields("aggregated"),
                      new LDAAggregator(), new Fields("output"));

        return topology.build();
}
```

5.3 Περιγραφή βασικών κλάσεων

Για να υλοποιηθούν οι συναρτήσεις και οι αθροιστές που απαρτίζουν την τοπολογία, χρησιμοποιήθηκαν οι κλάσεις που περιγράφονται σε αυτή την ενότητα. Για συντομία θα αναφερθεί η περιγραφή των βασικότερων κλάσεων και συναρτήσεων τους.

Κλάση	**Περιγραφή**
RedisMapState<T>	Η κατάσταση Trident που χρησιμοποιείται για την αποθήκευση των ενδιάμεσων καταστάσεων στην βάση δεδομένων
RSerializer<T>	Η κλάση που χρησιμοποιείται για σειριοποίηση των αντικειμένων, ώστε να αποθηκευτούν στην βάση
QueryRedis	Η κλάση που χρησιμοποιείται για ανάκληση δεδομένων από την βάση, με τη χρήση του drpc μηχανισμού

TwitterSampleSpout	Spout για εισαγωγή δεδομένων στην τοπολογία από τη δημόσια ροή του Twitter
TextFileSpout	Spout για εισαγωγή δεδομένων στην τοπολογία από αρχείο
SocketSpout	Spout για εισαγωγή δεδομένων στην τοπολογία από εξυπηρετητή socket

Splitter	Υλοποίηση της μεθόδου Split, που διαχωρίζει τις προτάσεις (tweets) στις επιμέρους λέξεις τους
Stemmer	Υλοποίηση της μεθόδου Stem, που υπολογίζει την ρίζα της κάθε λέξης
StopWordRemover	Υλοποίηση της μεθόδου RemoveStopWords που φιλτράρει τις κοινές λέξεις
ListAggregator	Αθροιστής της κάθε δεσμίδας σε λίστα από τούπλες
LDAAggregator	Αθροιστής όλων των λιστών από τούπλες, σε ένα αντικείμενο τύπου OnlineLDA μέσω της εφαρμογής του αλγορίθμου
OnlineLDA	Ο κύριος αλγόριθμος της Συνδεδεμένης Λανθάνουσας Κατανομής Dirichlet που εκτελείται για τα εισερχόμενα κείμενα
Result	Το αποτέλεσμα της εκτέλεσης του αλγορίθμου, που περιγράφει τα θέματα

	συνδεδεμένα με κάποια πιθανότητα
LDAVocabulary	Το χρησιμοποιούμενο λεξιλόγιο του αλγορίθμου

Πίνακας 5.3.1. Περιγραφή των βασικών κλάσεων της υλοποίησης του Αλγορίθμου Συνεχόμενης Λανθάνουσας Κατανομής Dirichlet

Κλάση	**Κύριες Μεθόδοι**
RedisMapState<	StateFactory nonTransactional(InetSocketAddress server)
T>	List<T> multiGet(List<List<Object>> keys)
	void multiPut(List<List<Object>> keys, List<T> vals)
RSerializer<T>	String serialize(T obj)
	T deserialize(String val)

TwitterSampleS	void open()
pout	void declareOutputFields(OutputFieldsDeclarer declarer)
	void nextTuple()
TextFileSpout	void open(Map conf, TopologyContext context, SpoutOutputCollector collector)
	void declareOutputFields(OutputFieldsDeclarer declarer)
	void nextTuple()
SocketSpout	void open(Map conf, TopologyContext context, SpoutOutputCollector collector)
	void declareOutputFields(OutputFieldsDeclarer declarer)
	void nextTuple()

Splitter	void execute(TridentTuple tuple, TridentCollector collector)
Stemmer	void execute(TridentTuple tuple, TridentCollector collector)
StopWordsRem	void prepare(Map conf, TridentOperationContext context)
over	void execute(TridentTuple tuple, TridentCollector collector)
ListAggregator	List<String> init(Object batchId, TridentCollector collector)

	void aggregate(List<String> val, TridentTuple tuple)
	void complete(List<String> val, TridentCollector collector)
LDAAggregator	OnlineLDA init()
	OnlineLDA reduce(OnlineLDA curr, TridentTuple tuple)
OnlineLDA	Result workOn(Documents docs)
Result	String toString()
LDAVocabulary	addWordsToVocabulary(String[] stringsToBeAdded)
	contains(String token)

Πίνακας 5.3.2. Σημαντικότερες μέθοδοι των κλάσεων της υλοποίησης του Αλγορίθμου Συνεχόμενης Λανθάνουσας Κατανομής Dirichlet

5.4 Εκτέλεση της Τοπολογίας

Για να ξεκινήσουμε την τοπολογία, πρέπει να έχουμε εγκαταστήσει νωρίτερα μια συστοιχία Storm. Το Storm παρέχει τη δυνατότητα να εκτελεστεί η τοπολογία τοπικά σε LocalCluster, για λόγους δοκιμής, ή στη συστοιχία. Η συνάρτηση που υποβάλλει την τοπολογία στη συστοιχία είναι η main:

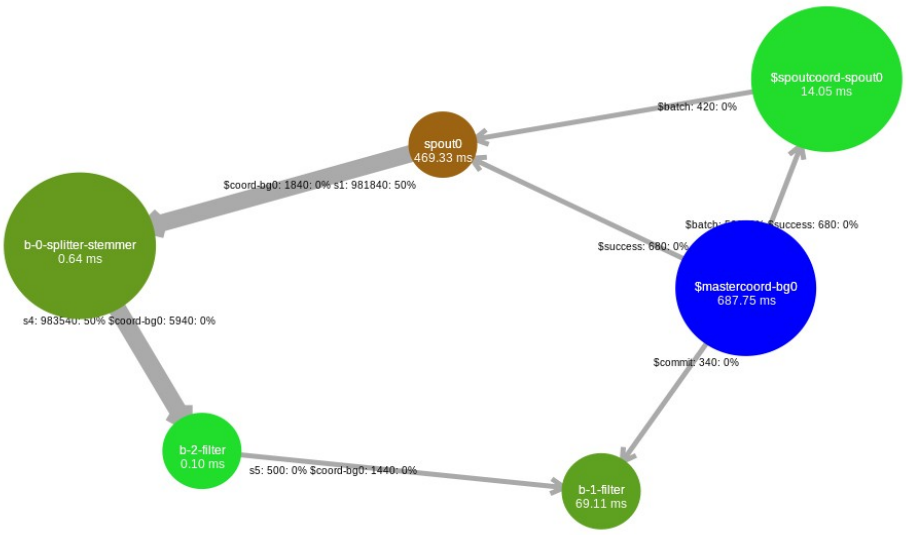
```
public static void main(String[] args) throws Exception {

        Config conf = new Config();
        conf.put(RichSpoutBatchExecutor.MAX_BATCH_SIZE_CONF, 2500);
        conf.setNumWorkers(4);

        //TridentState on Redis
        StateFactory redis = new RedisMapState.Factory(new
                InetSocketAddress("83.212.121.170", 6379),"online_lda");

        StormTopology topology = buildTopology(redis);
        //running topology on cluster
        StormSubmitter.submitTopologyWithProgressBar(args[0], conf,
        topology);
}
```

Στις ρυθμίσεις της τοπολογίας θέτουμε των αριθμό των εργατών και το μέγεθος της δεσμίδας τουπλών, μέσω της μεταβλητής RichSpoutBatchExecutor.MAX_BATCH_SIZE_CONF. Κατά την δημιουργία της κατάστασης Trident ορίζουμε το όνομα της μεταβλητής, με το οποίο αυτή θα αποθηκεύεται στη βάση (“online_lda”).

Αρχικά, πρέπει να παράξουμε το .jar αρχείο του προγράμματος. Με την χρήση της εντολής mvn package του maven δημιουργούμε το .jar. Το αρχείο που παράγεται έχει ενσωματωμένες όλες τις εξαρτήσεις σε εξωτερικές βιβλιοθήκες και είναι έτοιμο για εκτέλεση. Το υποβάλλουμε στη συστοιχία Storm, εκτελώντας την εντολή *storm jar filename.jar mainClass topologyName* στον Nimbus.

Η εκτέλεση της τοπολογίας ξεκινάει άμεσα. Οι εργάτες ξεκινούν και αρχίζουν να εκτελούν τις υποεργασίες που τους έχουν ανατεθεί. Αυτό μπορούμε να το διαπιστώσουμε και από το γραφικό περιβάλλον. Στην επόμενη εικόνα φαίνεται η γραφική αναπαράσταση της τοπολογίας, όπως αυτή αναλύθηκε σε bolt και spout από το Trident. Παρατηρούμε ότι οι συναρτήσεις Split, Stem και RemoveStopWords έχουν ενοποιηθεί σε ένα κοινό bolt (b-0), καθώς εσωκλείονται σε αυτό οι δύο ενδιάμεσες καταστάσεις (splitter και stemmer) και εξάγονται οι φιλτραρισμένες τούπλες (filter) για εκτέλεση από τα επόμενα στάδια. Αυτή η ενοποίηση σε κοινό bolt γίνεται αυτόματα από το Trident, για λόγους βελτιστοποίησης.

Στο bolt b-2 γίνεται η άθροιση των δεσμίδων σε μια κοινή τούπλα, ενώ στο b-1 εκτελείται ο αλγόριθμος LDA και αποθηκεύεται η κατάσταση Trident στη βάση.



Εικόνα 5.4.1. Γραφική αναπαράσταση της τοπολογίας “lda topology”

Τα παραπάνω μπορούμε να τα επιβεβαιώσουμε από τα log αρχεία των εργατών και από την επίβλεψη της βάσης. Για να διαπιστώσουμε ότι τα δεδομένα αποθηκεύονται επιτυχημένα στη βάση μπορούμε να εκτελέσουμε την εντολή MONITOR σε ένα redis-client.

5.5 *Εξαγωγή Αποτελεσμάτων*

5.5.1 Χαρακτηριστικά Μηχανημάτων Συστοιχίας

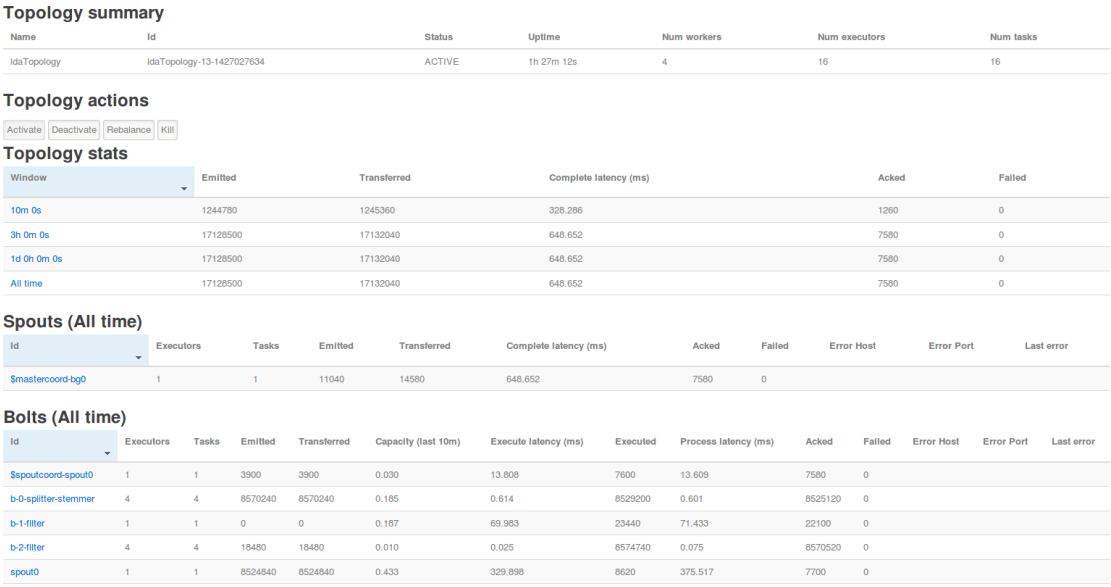
Τα μηχανήματα της συστοιχίας έχουν τα χαρακτηριστικά που παρουσιάζονται στον παρακάτω πίνακα. Παρατηρούμε ότι το μηχάνημα του Nimbus είναι το λιγότερο δυνατό, καθώς δεν θα χρειαστεί να εκτελέσει διεργασίες της τοπολογίας, αλλά μόνο τις απαραίτητες διαδικασίες συγχρονισμού.

	CPU	RAM	Disk	OS
Nimbus	1	2048	40GB	Ubuntu
Worker 1	2	4096	40GB	Ubuntu
Worker 2	2	4096	40GB	Ubuntu

Πίνακας 5.5.1. Χαρακτηριστικά Μηχανημάτων Συστοιχίας

5.5.2 Αποτελέσματα Εκτέλεσης

Από το γραφικό περιβάλλον λαμβάνουμε τα στοιχεία για την εκτέλεση του αλγορίθμου. Σε χρόνο περίπου 1h 27m εκτελέστηκαν 8.524.840 tweets, χρησιμοποιώντας ως μέγεθος δεσμίδας 2500. Για τις μεταβλητές του αλγορίθμου Online LDA χρησιμοποιήσαμε D= 8.524.840 και K= 3.



Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
ldaTopology	ldaTopology-13-1427027634	ACTIVE	1h 27m 12s	4	16	16

Topology actions

Activate | Deactivate | Rebalance | Kill

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	1244780	1245360	328.286	1260	0
3h 0m 0s	17128500	17132040	648.652	7580	0
1d 0h 0m 0s	17128500	17132040	648.652	7580	0
All time	17128500	17132040	648.652	7580	0

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
$mastercoord-bg0	1	1	11040	14580	648.652	7580	0			

Bolts (All time)

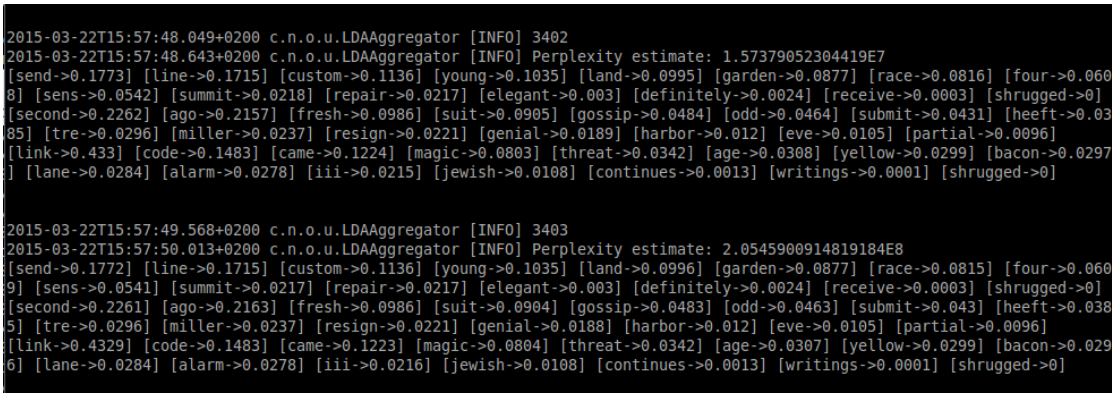
Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
$spoutcoord-spout0	1	1	3900	3900	0.030	13.808	7600	13.609	7580	0			
b-0-splitter-stemmer	4	4	8570240	8570240	0.185	0.614	8529200	0.601	8525120	0			
b-1-filter	1	1	0	0	0.187	69.983	23440	71.433	22100	0			
b-2-filter	4	4	18480	18480	0.010	0.025	8574740	0.075	8570520	0			
spout0	1	1	8524840	8524840	0.433	329.898	8620	375.517	7700	0			

Εικόνα 5.5.1. Αποτελέσματα εκτέλεσης της τοπολογίας “lda topology” για μέγεθος δεσμίδας 2500, D= 8.524.840 και K= 3.

Παρατηρούμε από τα στατιστικά των bolts, ότι οι τούπλες ακολουθούν την ροή: *spout0 -> b0 -> b2-> b1*. Ανάλογα με τις διεργασίες που εκτελούνται σε κάθε bolt, εκπέμπεται ένας αριθμός από τούπλες στο επόμενο στάδιο. Οι τούπλες αυτές εκτελούνται και επιβεβαιώνονται από το επόμενο στάδιο. Στο b-2, όπου γίνεται άθροιση της δεσμίδας σε μια ενιαία τούπλα, παρατηρούμε ότι έχουμε πολύ λιγότερες εκπεμπόμενες τούπλες σε σχέση με τις εισερχόμενες. Παρατηρούμε επίσης ότι οι εκτελεστές για τα στάδια b-0 και b-1 είναι τέσσερις, όπως ορίσαμε με την χρήση του .paralellismHint, ενώ οι εκτελεστές για το spout και τον τελικό αθροιστή είναι ένας.

Τέλος, βλέπουμε ότι το b-1 έχει μεγάλο χρόνο εκκίνησης επεξεργασίας (Execute Latency), λόγω του χρόνου εκτέλεσης του αλγορίθμου online lda. Το αποτέλεσμα του χρόνου εκτέλεσης του τελευταίου σταδίου είναι, ότι κάποιες τούπλες εκπέμπονται πάνω από 1 φορά για επεξεργασία, καθώς λήγει το χρονικό περιθώριο του χρόνου εκτέλεσης τους. Αυτό αποτελεί άνω όριο στην ταχύτητα επεξεργασίας των τουπλών καθώς το συγκεκριμένο στάδιο δεν είναι δυνατόν να παραλληλοποιηθεί.

Από το log του εργάτη που εκτελεί το b-1 λαμβάνουμε το τελικό αποτέλεσμα για την δεσμίδα 3403, επιβεβαιώνοντας ότι εκτελέστηκαν οι προβλεπόμενες τούπλες που δόθηκαν ως είσοδο στην τοπολογία. Συνολικά, οι 3403 τούπλες περιλαμβάνουν 3403 * 2500 (μέγεθος δεσμίδας) = 8.507.500 tweets, όσα δηλαδή tweets εισήχθηκαν στην τοπολογία.

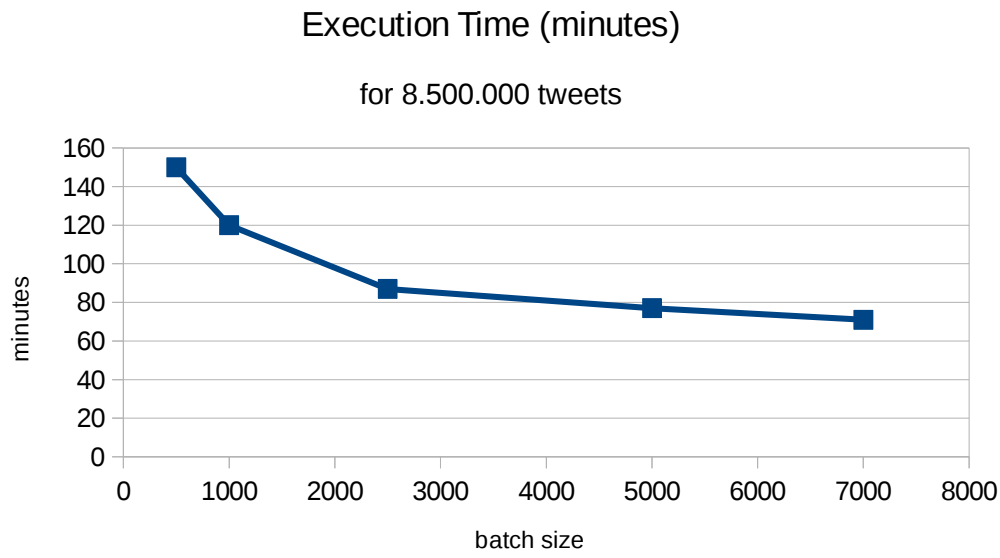


Εικόνα 5.5.2. Αρχείο log εργάτη που εκτελεί το b-1

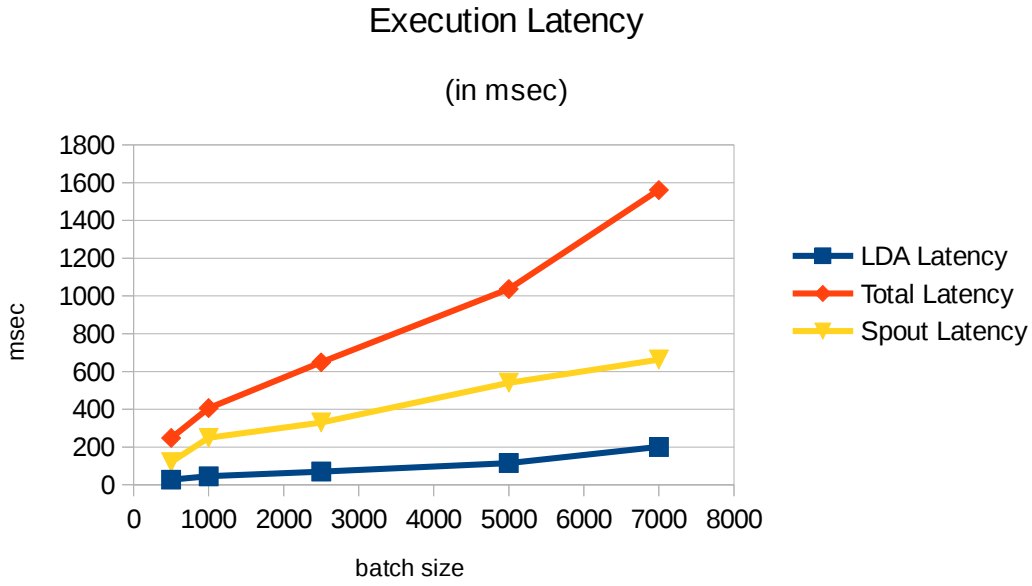
Για τις ίδιες μεταβλητές του αλγορίθμου και τον ίδιο αριθμό tweets (~8.500.000), εκτελέσαμε τον αλγόριθμο για διαφορετικά μεγέθη δεσμίδας (#τούπλων) και τα αποτελέσματα συνοψίζονται στον παρακάτω πίνακα:

#τούπλων	Χρόνος Εκτέλεσης	Καθυστέρηση Εκτέλεσης Αλγορίθμου LDA	Συνολική Καθυστέρηση	Καθυστέρηση Spout
500	2h 30m	27.235 ms	248.211 ms	119.81 ms
1000	2h 0m	45.033 ms	406.212 ms	249.01 ms
2500	1h 27m	69.983 ms	648.652 ms	329.9 ms
5000	1h 17m	115.023 ms	1036.600 ms	540.3 ms
7000	1h 11m	200.500 ms	1561.211 ms	663.44 ms

Πίνακας 5.5.1. Χρόνοι εκτέλεσης του αλγορίθμου για διαφορετικό μέγεθος τούπλας



Εικόνα 5.5.3. Χρόνος εκτέλεσης αλγορίθμου για 8.500.000 tweets σε συνάρτηση με το μέγεθος της δεσμίδας

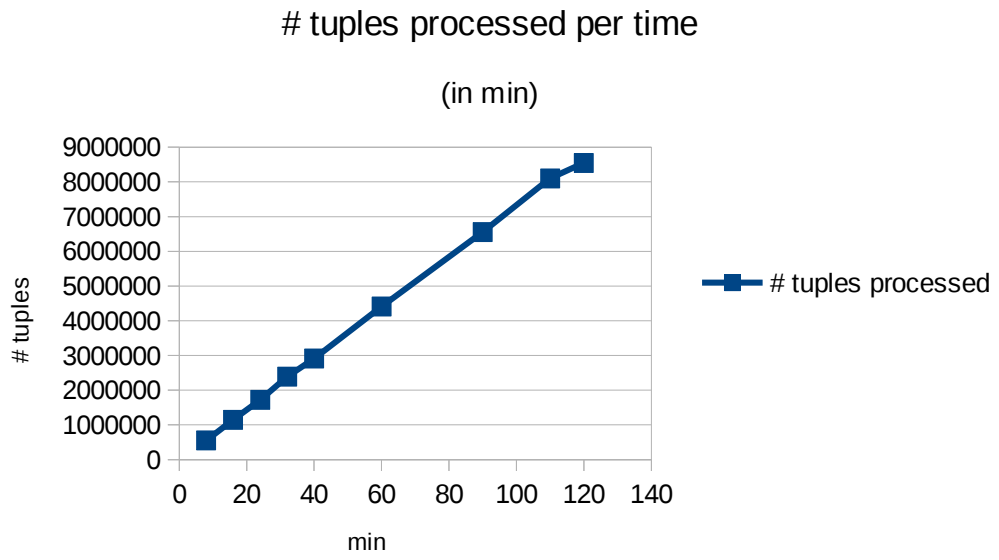


Εικόνα 5.5.4. Καθυστέρηση εκτέλεσης αλγορίθμου για 8.500.000 tweets σε συνάρτηση με το μέγεθος της δεσμίδας

Για μέγεθος δεσμίδας 1000 τούπλες εκτελέσαμε τον αλγόριθμο, μετρώντας τον αριθμό των τουπλών που εκτελέστηκαν σε συγκεκριμένο χρονικό διάστημα, καταλήγοντας στον παρακάτω πίνακα:

Χρόνος μέτρησης	# επεξεργασμένων τουπλών
8	551100
16	1146320
24	1721880
32	2388340
40	2912260
60	4409520
90	6555480
120	8545100

Πίνακας 5.5.2. Αριθμός επεξεργασμένων τουπλών ανά μονάδα χρόνου



Εικόνα 5.5.5. Αριθμός επεξεργασμένων τουπλών ανά μονάδα χρόνου

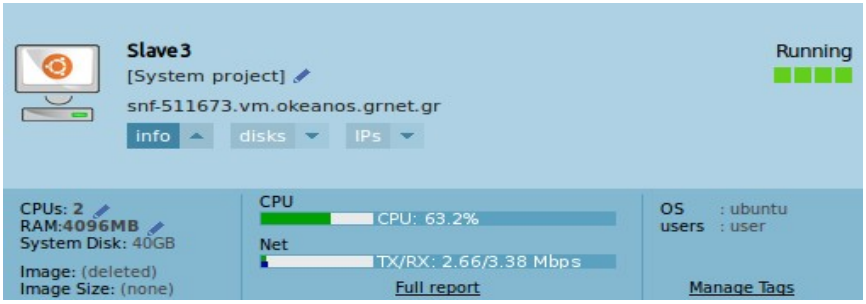
Από την παραπάνω ανάλυση παρατηρούμε ότι το μέγεθος της δεσμίδας παίζει σημαντικό ρόλο στο χρόνο εκτέλεσης του αλγορίθμου. Τα μικρά μεγέθη δεσμίδας (500) δεν συμφέρουν υπολογιστικά, καθώς μεταφέρουν μικρή ποσότητα πληροφορίας. Κατ' επέκτασιν, το χρονοβόρο στάδιο της εκτέλεσης του αλγορίθμου επαναλαμβάνεται πολύ περισσότερες φορές και αυξάνεται ο συνολικός χρόνος εκτέλεσης. Συμπεραίνουμε λοιπόν, ότι προτιμάται γενικά η χρήση μεγαλύτερου μεγέθους για τις δεσμίδες.

Παρόλα αυτά παρατηρούμε, ότι από ένα μέγεθος και μετά (~2500) η κλίση της καμπύλης είναι πιο ομαλή, δηλαδή δεν μειώνεται σημαντικά ο χρόνος εκτέλεσης με την αύξηση του μεγέθους της δεσμίδας. Υπάρχει όμως δυσανάλογη αύξηση στην καθυστέρηση εκτέλεσης του προγράμματος, δηλαδή στο χρόνο απόκρισης. Αυτό οφείλεται, σύμφωνα με την εικόνα 5.5.4, εν μέρει στην εκτέλεση του αλγορίθμου LDA, όμως περισσότερο στην διαδικασία “γεμίσματος” των μεγαλύτερων δεσμίδων από το spout. Μεγαλύτερο μέγεθος δεσμίδων συνεπάγεται εισαγωγή μεγάλης καθυστέρησης.

Συνεπώς, λαμβάνοντας υπόψιν ότι η καθυστέρηση απόκρισης είναι μια σημαντική παράμετρος για ένα σύστημα πραγματικού χρόνου, θεωρούμε ότι το ιδανικό μέγεθος δεσμίδας κυμαίνεται κοντά στις 2000-2500 τούπλες. Σε αυτό το μέγεθος, ο χρόνος εκτέλεσης είναι αποδεκτός, ενώ ο χρόνος καθυστέρησης δεν έχει αυξηθεί σε απαγορευτικό βαθμό. Σημειώνεται εδώ, ότι για την επιλογή του μεγέθους της δεσμίδας σε πραγματικό περιβάλλον, θα πρέπει να ληφθεί υπόψιν και η σύγκλιση του αλγορίθμου, καθώς είναι πιθανόν να μεταβάλλεται για διαφορετικά μεγέθη δεσμίδων. Παρόλα αυτά, η συγκεκριμένη παράμετρος είναι έξω από τη σκοπιά της συγκεκριμένης εργασίας και δεν μελετάται εδώ.

Επίσης, παρατηρείται από το διάγραμμα 5.5.5, ότι η εκτέλεση των τουπλών είναι γραμμική με το πέρασμα του χρόνου. Αυτό οφείλεται στο ότι η κατάσταση Trident που αποθηκεύεται στη βάση, δεν μεγαλώνει ανάλογα με την είσοδο, κάτι το οποίο είναι χαρακτηριστικό του αλγορίθμου της Συνεχόμενης Λανθάνουσας Κατανομής Dirichlet.

Τέλος, σημειώνεται ότι ο χρόνος εκτέλεσης δεν μπορεί να μειωθεί παραπάνω λόγω της φύσης του αλγορίθμου, και συγκεκριμένα λόγω του τελευταίου σταδίου της τοπολογίας, όπου γίνεται η άθροιση των δεσμίδων στον LDAAggregator. Αυτό επιβεβαιώνεται και από το γραφικό περιβάλλον του μηχανήματος (Εικόνα 5.5.4), όπου παρατηρούμε ότι η χρήση του επεξεργαστή είναι κατά μέσο όρο κοντά στο 60%. Αν μπορούσαμε να εκμεταλλευτούμε κάποιο βαθμό παραλληλίας σ' αυτό το επίπεδο, θα διαθέταμε επιπλέον πόρους για επεξεργασία.



Εικόνα 5.5.4. Γραφικό περιβάλλον εικονικού μηχανήματος εκτέλεσης.

Σε επόμενο στάδιο εκτελέσαμε τον αλγόριθμο για μέγεθος τούπλας 2000 και μεταβλητές αλγορίθμου Online LDA, D= 8.524.840 και K= 50, με τα αποτελέσματα:

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
storm-online-lda	storm-online-lda-15-1427037685	ACTIVE	3h 13m 3s	4	16	16

Topology actions

Activate Deactivate Rebalance Kill

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	419960	420040	1937.091	220	0
3h 0m 0s	7524540	7526440	2840.984	3740	0
1d 0h 0m 0s	8084800	8086860	2870.891	4020	0
All time	8084800	8086860	2870.891	4020	0

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
$mastercoord-bg0	1	1	5880	7940	2870.891	4020	0			

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
$spoutcoord-spout0	1	1	1680	1680	0.004	14.995	4020	14.313	4020	0			
b-0-splitter-stemmer	4	4	4045900	4045900	0.069	0.674	4031020	0.660	4021640	0			
b-1-filter	1	1	0	0	0.825	743.763	14340	682.058	11980	0			
b-2-filter	4	4	9340	9340	0.003	0.028	4055080	0.122	4045820	0			
spout0	1	1	4022000	4022000	0.076	181.095	6320	360.941	4100	0			

Εικόνα 5.5.3. Αποτελέσματα εκτέλεσης της τοπολογίας “lda topology” για μέγεθος δεσμίδας 2000, D= 8.524.840 και K= 50.

Παρατηρούμε μεγάλη αύξηση στον χρόνο εκτέλεσης του τελευταίου σταδίου, λόγω της αύξησης του αριθμού των θεμάτων σε 50. Πλέον, σε διπλάσιο χρόνο (3h 10m) εκτελέστηκαν οι μισές τούπλες (~4.000.000). Συμπεραίνουμε ότι ο χρόνος εκτέλεσης του τελευταίου σταδίου λειτουργεί σαν σημείο συμφόρησης για την τοπολογία. Παρόλα αυτά, το Storm χειρίζεται αυτές τις καθυστερήσεις, επαναλαμβάνοντας τις τούπλες, έτσι ώστε να εξασφαλίσει ότι δε θα υπάρχουν απώλειες δεδομένων, όπως ορίζουν οι αρχές ενός συστήματος πραγματικού χρόνου.

6 Υλοποίηση Αλγορίθμου Υπολογισμού Ομοιότητας

Ο στόχος αυτής της υλοποίησης είναι η δημιουργία μιας τοπολογίας η οποία δέχεται δύο ροές από μικρά κείμενα (tweets) και υπολογίζει την ομοιότητά τους σε πραγματικό χρόνο. Η συνολική ομοιότητα των ροών, μπορεί να ληφθεί σε οποιοδήποτε στάδιο της εκτέλεσης της τοπολογίας και αφορά στην συγκεκριμένη χρονική στιγμή που λαμβάνουμε την μέτρηση. Όταν οι δύο ροές ολοκληρωθούν μπορούμε να συμπεράνουμε την ομοιότητά τους για τον σύνολο των tweets. Αυτή η τοπολογία μπορεί να χρησιμοποιηθεί ορίζοντας μια από τις δύο ροές ως ροή που εκφράζει κάποια πολικότητα, έτσι ώστε να υπολογιστεί η ομοιότητα της δεύτερης με αυτή. Η ομοιότητα της ροής των tweets με τον γράφο πολικότητας είναι η ανάλυση συναισθήματος της.

6.1 Χρησιμοποιούμενες Τεχνολογίες

Για τον υπολογισμό των γράφων ν-γραμμάτων των κειμένων χρησιμοποιήθηκε ο αλγόριθμος υπολογισμού, που παρουσιάζεται στο κεφάλαιο 3.3. Η τοπολογία σχεδιάστηκε με τη χρήση του Trident.

Η βάση δεδομένων που χρησιμοποιείται για τα ενδιάμεσα δεδομένα της εκτέλεσης, είναι η Redis. Η σύνδεση με τη βάση επιτυγχάνεται με χρήση του Jedis.

Για τη διαχείριση του έργου χρησιμοποιήθηκε ο Apache Maven.

6.2 Υλοποίηση Αλγορίθμου

Η τοπολογία περιλαμβάνει τα εξής στάδια:

- Είσοδος δεδομένων (tweets) από spout
- Υπολογισμός του συνόλου ακμών ν-γραμμάτων για κάθε tweet
- Υπολογισμός της ομοιότητας μεταξύ των δύο ροών ν-γραμμάτων

6.2.1 Είσοδος δεδομένων

Η είσοδος των δεδομένων στην τοπολογία γίνεται μέσω των spout που αναφέρθηκαν στο κεφάλαιο 5. Για τη συγκεκριμένη τοπολογία, εκκινούμε δύο ροές, επομένως 2 spout αντίστοιχα, που εισάγουν στην τοπολογία τα tweets των οποίων την ομοιότητα θέλουμε να μετρήσουμε. Οι δύο ροές είναι πλήρως ανεξάρτητες και εκτελούνται παράλληλα.

6.2.2 Υπολογισμός γράφου ν-γραμμάτων

Ο υπολογισμός των ν-γραμμάτων για κάθε tweet εισόδου γίνεται σε bolt, που καταναλώνει τα εισαγόμενα στην τοπολογία δεδομένα και υλοποιείται στην κλάση NGramsCalculator. Σύμφωνα με τη θεωρητική ανάλυση, υπολογίζονται οι κορυφές του γράφου, δηλαδή όλες οι πιθανές υποακολουθίες μεγέθους ν. Έπειτα, με μέγεθος παραθύρου w υπολογίζονται οι ακμές του γράφου, ως σύνδεσμοι μεταξύ των υπακολουθιών με απόσταση στο κείμενο μικρότερη του w. Τέλος, υπολογίζεται το βάρος κάθε ακμής. Κάθε ακμή εκπέμπεται στο επόμενο στάδιο επεξεργασίας. Η δομή της ακμής που εκπέμπεται είναι απλή: μια συμβολοακολουθία που αποτελείται από τα ν-γράμματα που αποτελούν τους 2 κόμβους της ακμής και ένα πεδίο που ορίζει το βάρος της. Αυτή η δομή βολεύει, λόγω του τρόπου με του οποίου θα αποθηκευτεί αργότερα στη βάση Redis.

Η εκπεμπόμενη ακμή αυτού του σταδίου, εισέρχεται σε ένα bolt το οποίο ταξινομεί λεξικογραφικά τις δύο κορυφές της ακμής. Αυτό εξασφαλίζει, ότι στο επόμενο στάδιο της σύγκρισης, οι ακμές “αβγ κλμ” “κλμ αβγ”, δε θα θεωρηθούν διαφορετικές.

Τα κείμενα είναι ανεξάρτητα, επομένως ο βαθμός παραλληλίας αυτών των δύο σταδίων μπορεί να επιλεχθεί ανάλογα με τον αριθμό των εκτελεστών που έχουμε στην τοπολογία.

6.2.3 Υπολογισμός ομοιότητας

Για τον υπολογισμό ομοιότητας των ροών, αρχικά τις διαχωρίζουμε ορίζοντας τη μια ροή, ως ΡΟΗ1, και τη δεύτερη ως ΡΟΗ2. Αρχικά, όλες οι ακμές αποθηκεύονται στη βάση Redis, σαν αντικείμενα τύπου Hashes. Ένα αντικείμενο τύπου Hash στην βάση Redis, ορίζεται ως ένα προσδιοριστικό κλειδί, και ζεύγη ιδιοτήτων- τιμών που το προσδιορίζουν. Εν προκειμένω, οι ακμές αποθηκεύονται με κλειδί τη συμβολοακολουθία που ορίζουν οι δύο κορυφές τους και πρόθεμα τον γράφο στον οποίο ανήκουν. Για παράδειγμα, η ακμή μεταξύ των ν-γραμμάτων “αβγ” και “κλμ”, στον γράφο 1, αποθηκεύεται με κλειδί “1_αβγ κλμ”. Τα βάρη των ακμών, αποθηκεύονται σαν πεδίο “weight” πάνω στο κάθε Hash.

Έπειτα, πάνω στις ροές των tweets εφαρμόζεται μια συνάρτηση, η οποία είναι υπεύθυνη για την διατήρηση της κατάστασης Trident, η *.partitionPersist*(). Αυτή η συνάρτηση, όπως φαίνεται από το όνομά της, εφαρμόζεται επάνω σε κάθε τμήμα (partition) ροής ξεχωριστά και ενημερώνει την κατάσταση. Αν εκτελεστεί σε διαφορετικά τμήματα της ροής κάθε γράφου ταυτόχρονα, τίθεται ζήτημα συγχρονισμού για την ενημέρωση των μετρητών. Αυτό σημαίνει ότι πρέπει να συγκεντρώσουμε αρχικά όλες τις τούπλες σε ένα ενιαίο τμήμα από όπου θα ενημερώνεται συνολικά η κατάσταση της ροής. Αυτό γίνεται με τη συνάρτηση *.global*().

Η συνάρτηση *.partitionPersist*() εφαρμόζει την συνάρτηση update σε κάθε τούπλα ξεχωριστά. Η συνάρτηση update ορίζεται σε μια κλάση που υλοποιεί τη διαπροσωπία *StateUpdater*. Για τις ακμές της κάθε ροής, ελέγχεται αν υπάρχει ίδια ακμή στην άλλη ροή, και ενημερώνονται οι αντίστοιχες τιμές στη βάση. Όλη η λογική για την υλοποίηση της παραπάνω λειτουργίας, την αποθήκευση των ακμών και την ενημέρωση των μετρητών περιέχεται στην κλάση *RedisMapState*. Οι τιμές που αποθηκεύονται στη βάση, είναι το σύνολο τον ακμών για κάθε ροή, (count1, count2), το σύνολο των μεταξύ τους κοινών ακμών και το άθροισμα των βαρών των κοινών ακμών. Με βάση αυτές τις τιμές μπορούμε σε κάθε στιγμή να υπολογίσουμε την ομοιότητα των δύο γράφων.

6.2.4 Χτίσιμο Τοπολογίας

Συνολικά, η τοπολογία χτίζεται στην συνάρτηση buildTopology:

```
public static StormTopology buildTopology(StateFactory state)
                        throws IOException {

        // spout from Twitter random stream
        TwitterSampleSpout spout = new TwitterSampleSpout(username,passwd);
        TridentTopology topology = new TridentTopology();

        TridentState state =
        topology
                .newStream("spout1", spout)
                .parallelismHint(1)
                .shuffle()
                .each(new Fields("sentence"), new NGramsCalculator(),
                      new Fields("ngrams")).parallelismHint(10).name(“calc”)
                .each(new Fields("ngrams"), new OrderEdges(),
                      new Fields("ordered")).parallelismHint(10).name(“order”)
                .global()
                .partitionPersist(state, new Fields("ordered"),
                       new SetUpdaterA());

        topology
                .newStream("spout2", spout2)
                .parallelismHint(1)
                .shuffle()
                .each(new Fields("sentence"), new NGramsCalculator(),
                      new Fields("ngrams")).parallelismHint(10).name(“calc”)
                .each(new Fields("ngrams"), new OrderEdges(),
                      new Fields("ordered")).parallelismHint(10).name(“order”)
                .global()
                .partitionPersist(state, new Fields("ordered"),
                       new SetUpdaterB());


        return topology.build();
}
```

6.3 Περιγραφή βασικών κλάσεων

Για να υλοποιηθούν οι συναρτήσεις και οι αθροιστές που απαρτίζουν την τοπολογία, χρησιμοποιήθηκαν οι κλάσεις που περιγράφονται σε αυτή την ενότητα. Για συντομία θα αναφερθεί η περιγραφή των βασικότερων κλάσεων μαζί με τις βασικότερες συναρτήσεις τους.

Κλάση	**Περιγραφή**
RedisMapState<T>	Η κατάσταση Trident που χρησιμοποιείται για την αποθήκευση των ενδιάμεσων καταστάσεων στην βάση δεδομένων
RSerializer<T>	Η κλάση που χρησιμοποιείται για σειριοποίηση των αντικειμένων, ώστε να αποθηκευτούν στην βάση
QueryRedis	Η κλάση που χρησιμοποιείται για ανάκληση δεδομένων από την βάση, με τη χρήση του drpc μηχανισμού

TwitterSampleSpout	Spout για εισαγωγή δεδομένων στην τοπολογία από τη δημόσια ροή του Twitter
TextFileSpout	Spout για εισαγωγή δεδομένων στην τοπολογία από αρχείο
SocketSpout	Spout για εισαγωγή δεδομένων στην τοπολογία από εξυπηρετητή socket

Edge	Αντικείμενο που περιγράφει την ακμή ενός γράφου
NGramsCalculator	Η κλάση που χρησιμοποιείται για υπολογισμό του γράφου ν-γραμμάτων για κάθε tweet
OrderEdges	Η κλάση που υπολογίζει την αλφαβητική σειρά των υποακολουθιών της ακμής
UpdaterA	Η κλάση που ενημερώνει την κατάσταση Trident για τον γράφο 1.
UpdaterB	Η κλάση που ενημερώνει την κατάσταση Trident για τον γράφο 2.

Πίνακας 6.3.1. Περιγραφή των βασικών κλάσεων της υλοποίησης του Αλγορίθμου Ανάλυσης Συναισθήματος

Κλάση	**Κύριες Μεθόδοι**
RedisMapState<	StateFactory nonTransactional(InetSocketAddress server)
T>	List<T> multiGet(List<List<Object>> keys)
	void multiPut(List<List<Object>> keys, List<T> vals)
RSerializer<T>	String serialize(T obj)
	T deserialize(String val)

TwitterSampleS	void open()
pout	void declareOutputFields(OutputFieldsDeclarer declarer)
	void nextTuple()
TextFileSpout	void open(Map conf, TopologyContext context, SpoutOutputCollector collector)
	void declareOutputFields(OutputFieldsDeclarer declarer)
	void nextTuple()
SocketSpout	void open(Map conf, TopologyContext context, SpoutOutputCollector collector)
	void declareOutputFields(OutputFieldsDeclarer declarer)
	void nextTuple()

Edge	double getWeight()
	String getValue
NGramsCalculat or	void execute(TridentTuple tuple, TridentCollector collector)
OrderEdges	void execute(TridentTuple tuple, TridentCollector collector)
UpdaterA	public void updateState(State state, List tuples, TridentCollector collector)
UpdaterB	public void updateState(State state, List tuples, TridentCollector collector)

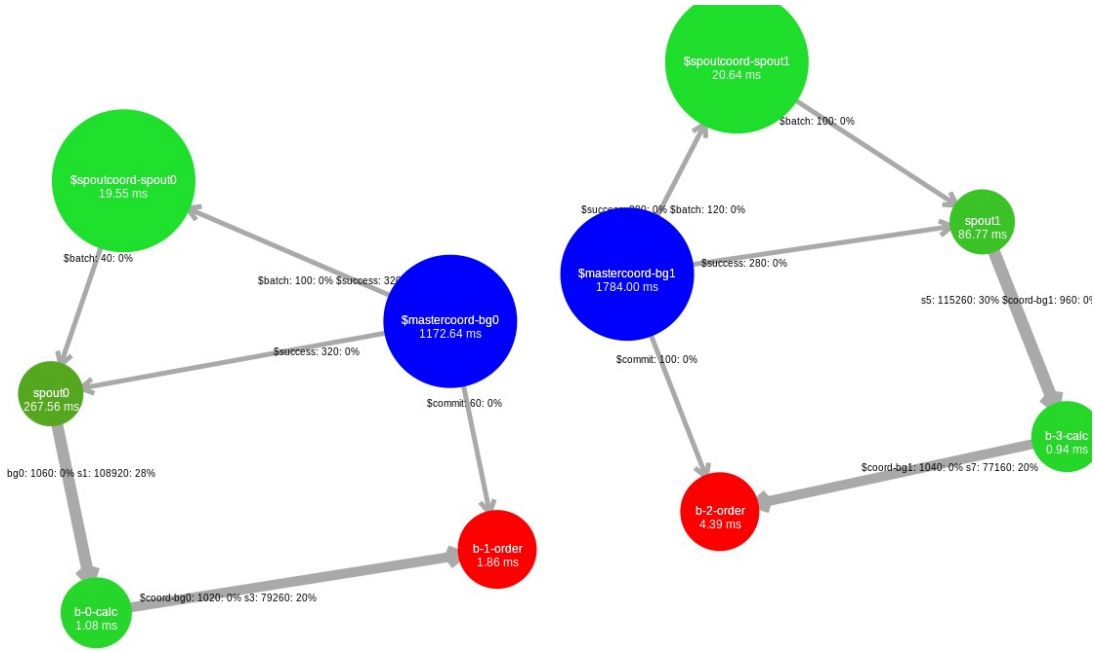
Πίνακας 6.3.2. Σημαντικότερες μέθοδοι των κλάσεων της υλοποίησης του Αλγόριθμου Ανάλυσης Συναισθήματος

6.4 Εκτέλεση της Τοπολογίας

Για να ξεκινήσουμε την τοπολογία, πρέπει να έχουμε εγκαταστήσει νωρίτερα μια συστοιχία Storm. Η συνάρτηση που υποβάλλει την τοπολογία στη συστοιχία είναι η main:

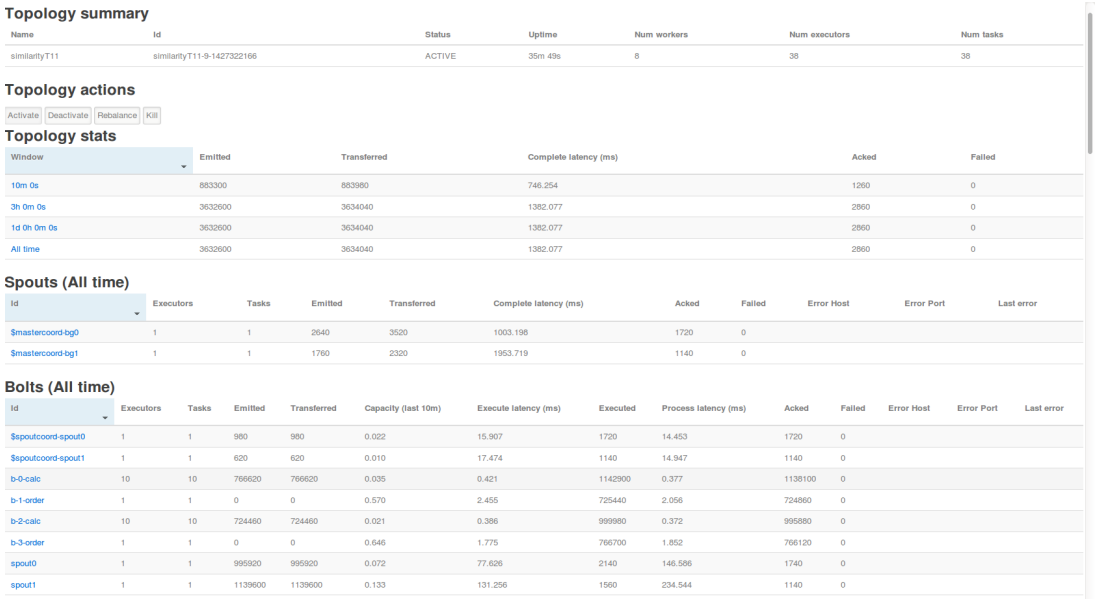
```
public static void main(String[] args) throws Exception {
        Config conf = new Config();
        conf.put(RichSpoutBatchExecutor.MAX_BATCH_SIZE_CONF, 1000);
        conf.setNumWorkers(10)

        StateFactory redis = new RedisMapState.Factory(
                new InetSocketAddress("83.212.121.170", 6379),"");

        StormSubmitter.submitTopologyWithProgressBar(args[0], conf,
                buildTopology(redis))
}
```

Δημιουργούμε το αρχείο .jar της τοπολογίας με την χρήση του maven και το υποβάλλουμε στη συστοιχία Storm. Η τοπολογία ξεκινάει, όπως διαπιστώνουμε και από το γραφικό περιβάλλον:



Εικόνα 6.4.1. Γραφική αναπαράσταση τοπολογίας υπολογισμού ομοιότητας δύο γράφων.

Παρατηρούμε ότι έχουμε δύο ροές δεδομένων, σύμφωνα με το σχεδιασμό της τοπολογίας. Οι ροές δεδομένων είναι ανεξάρτητες και εκτελούνται παράλληλα. Επίσης, στο γραφικό περιβάλλον φαίνεται, ότι τα bolt που υπολογίζουν τις ακμές εκτελούνται παράλληλα σε 10 νήματα.



Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks
similarityT11	similarityT11-9-1427322166	ACTIVE	35m 49s	8	38	38

Topology actions

Activate | Deactivate | Rebalance | Kill

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	883300	883980	746.254	1260	0
3h 0m 0s	3632600	3634040	1382.077	2860	0
1d 0h 0m 0s	3632600	3634040	1382.077	2860	0
All time	3632600	3634040	1382.077	2860	0

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error
$mastercoord-bg0	1	1	2640	3520	1003.198	1720	0			
$mastercoord-bg1	1	1	1760	2320	1953.719	1140	0			

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
$spoutcoord-spout0	1	1	980	980	0.022	15.907	1720	14.453	1720	0			
$spoutcoord-spout1	1	1	620	620	0.010	17.474	1140	14.947	1140	0			
b-0-calc	10	10	766620	766620	0.035	0.421	1142900	0.377	1138100	0			
b-1-order	1	1	0	0	0.570	2.455	725440	2.056	724860	0			
b-2-calc	10	10	724460	724460	0.021	0.386	999980	0.372	995880	0			
b-3-order	1	1	0	0	0.646	1.775	766700	1.852	766120	0			
spout0	1	1	995920	995920	0.072	77.626	2140	146.586	1740	0			
spout1	1	1	1139600	1139600	0.133	131.256	1560	234.544	1140	0			

Εικόνα 6.4.2. Εκτέλεση τοπολογίας για μέγεθος δεσμίδας 2000

6.5 Εξαγωγή Αποτελεσμάτων

6.5.1 Χαρακτηριστικά Μηχανημάτων Συστοιχίας

Τα χαρακτηριστικά των μηχανημάτων της συστοιχίας, που χρησιμοποιήθηκαν για την εκτέλεση της τοπολογίας, είναι ίδια με αυτά που παρουσιάζονται στο κεφάλαιο 5.5.1.

6.5.2 Αποτελέσματα Εκτέλεσης

Εκτελώντας την τοπολογία για είσοδο tweets 1.000.000 ακμών λάβαμε τα αποτελέσματα που συνοψίζονται στον παρακάτω πίνακα.

Μετρική	**Τιμή**
Σύνολο ακμών ροής 1	715071
Σύνολο ακμών ροής 2	789012
Σύνολο κοινών ακμών	298392
Άθροισμα βαρών	276519.6

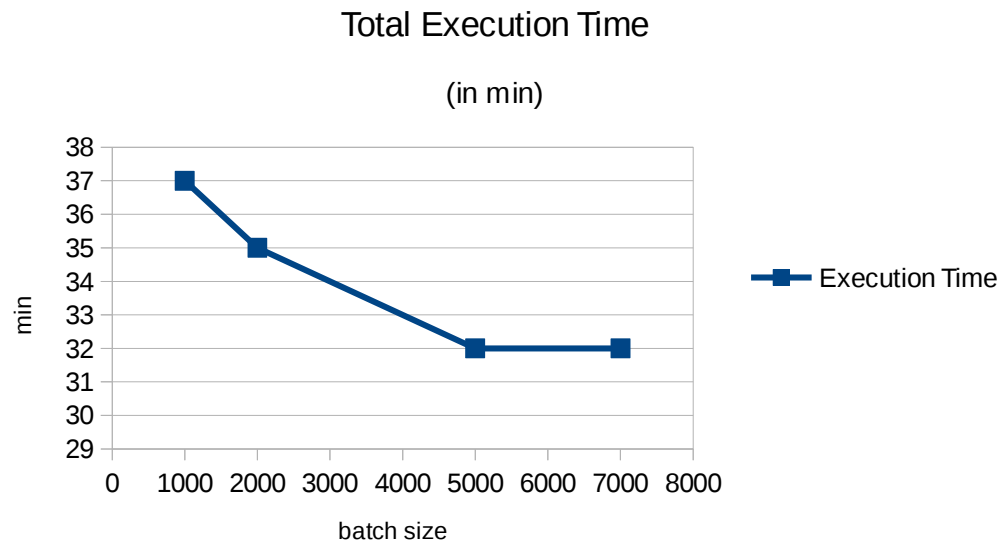
Σύμφωνα με αυτά υπολογίζουμε για τους δύο γράφους εισόδου:

CS = 0,417, VS = 0,350, NVS = 0,3861

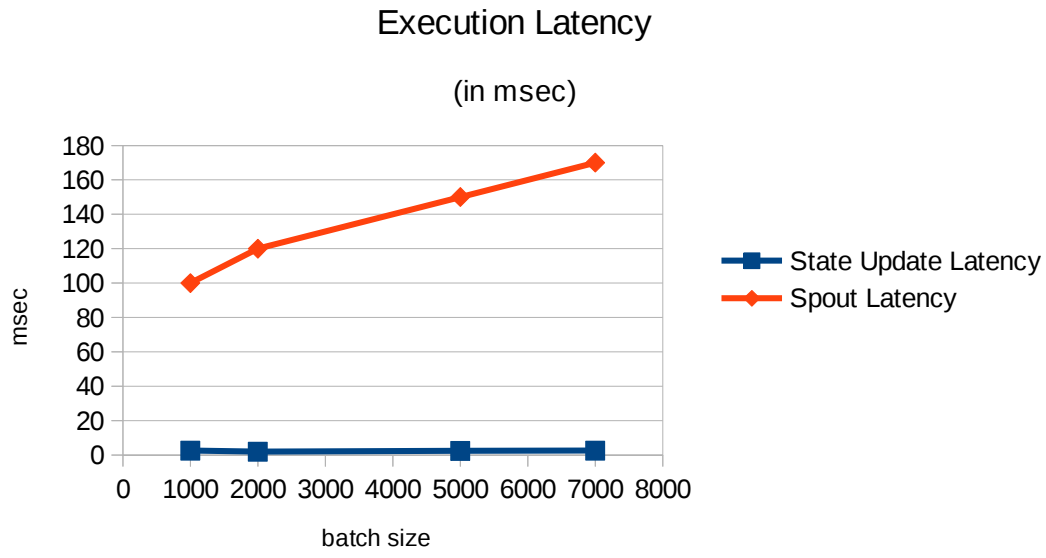
Οι χρόνοι εκτέλεσης και καθυστέρησης παρουσιάζονται στον επόμενο πίνακα. Σημειώνεται, ότι οι χρόνοι καθυστέρησης για την ενημέρωση της κατάστασης και του spout αποτελούν το μέσο όρο της καθυστέρησης των δύο ροών.

# τούπλων	Χρόνος Εκτέλεσης	Καθυστέρηση Ενημέρωσης Κατάστασης	Καθυστέρηση Spout
1000	37 m	2.6 ms	100 ms
2000	35 m	2 ms	120 ms
5000	32 m	2.5 ms	150 ms
7000	32 m	2.6 ms	170 ms

Πίνακας 6.5.1. Αποτελέσματα εκτέλεσης για 1.000.000 ακμές εισόδου



Εικόνα 6.5.1. Συνολικός χρόνος εκτέλεσης για διαφορετικό μέγεθος δεσμίδας



Εικόνα 6.5.2. Συνολικός χρόνος καθυστέρησης για διαφορετικό μέγεθος δεσμίδας

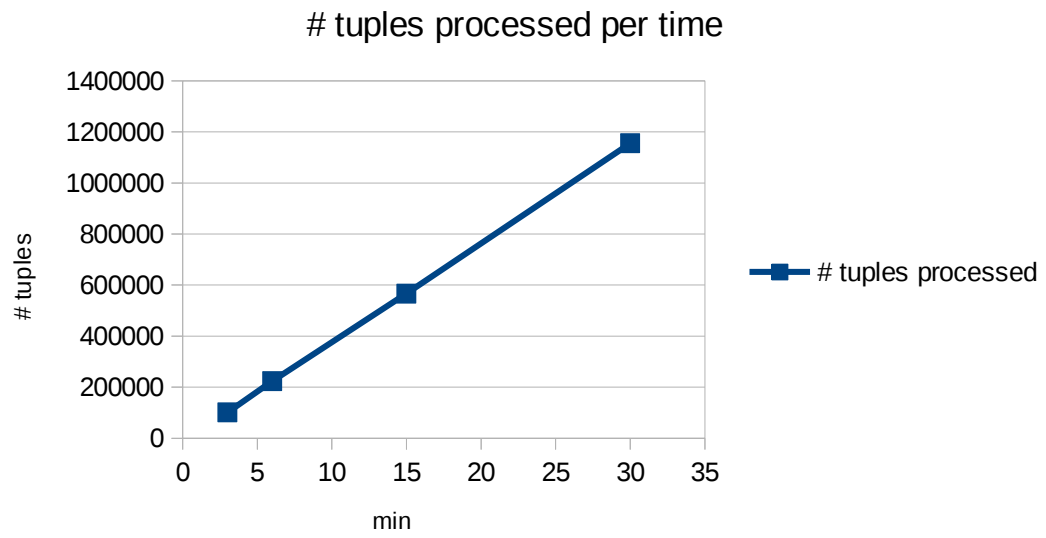
Παρατηρούμε από τα διαγράμματα ότι το μέγεθος της δεσμίδας δεν επηρεάζει σημαντικά το χρόνο εκτέλεσης, παρά μόνο στην περίπτωση της καθυστέρησης των spout, που είναι ανεξάρτητη της φύσης του αλγορίθμου. Για την ακρίβεια υπάρχει μια μικρή μείωση του χρόνου εκτέλεσης για μεγαλύτερο μέγεθος δεσμίδων, η οποία είναι της τάξης των min. Αυτό συμβαίνει γιατί οι ακμές αποθηκεύονται στη βάση μια προς μια και όχι σαν δεσμίδα, για μεγαλύτερη ακρίβεια των αποτελεσμάτων. Αν οι ακμές αποθηκευόταν σαν δεσμίδες, θα είχαμε ακρίβεια σε αυτό το επίπεδο, αφού θα έπρεπε να αποθηκευτεί ολόκληρη η δεσμίδα, για να ανιχνευθεί η ύπαρξη μιας ακμή από την δεύτερη ροή κατά τον έλεγχο της ομοιότητας.

Ουσιαστικά, για κάθε ακμή εισόδου γίνονται 2 αιτήσεις στη βάση: μια για την αποθήκευση της και μία για τον έλεγχο ύπαρξης της ακμής στην δεύτερη ροή. Η ενημέρωση των μετρητών γίνεται στο τέλος της δεσμίδας συνολικά, με αποτέλεσμα να βλέπουμε κάποια μικρή βελτίωση στο χρόνο εκτέλεσης για μεγαλύτερο μέγεθος δεσμίδας. Η επικοινωνία για τη βάση σε επίπεδο ακμής, αποτελεί και το σημείο συμφόρησης της τοπολογίας, καθώς δεν μπορεί να αξιοποιηθεί σημαντικά το πακετάρισμα των τουπλών σε δεσμίδα, σε αυτό το στάδιο.

Όσον αφορά την ικανότητα επεξεργασίας, για μέγεθος δεσμίδας 2000 και βαθμό παραλληλίας 10, εκτελούμε την τοπολογία και λαμβάνουμε μετρήσεις σε συγκεκριμένα χρονικά διαστήματα:

Χρόνος Μέτρησης	Αριθμός Επεξεργασμένων Τουπλών
3 m	101200
6 m	223513
15 m	566600
30 m	1155780

Πίνακας 6.5.2. Αποτελέσματα εκτέλεσης σε αριθμό επεξεργασμένων τουπλών ανά μονάδα χρόνου.



Εικόνα 6.5.3. Αριθμός τουπλών υπό επεξεργασία σε συνάρτηση με το χρόνο μέτρησης

Βλέπουμε στο διάγραμμα, ότι ο ρυθμός επεξεργασίας παραμένει σταθερός ως προς το χρόνο, καθώς με την χρήση της δομής Hash του Redis που χρησιμοποιήσαμε, η αποθήκευση των ακμών στη βάση δεν επιβαρύνει την εκτέλεση του προγράμματος. Εφόσον οι ακμές δεν αποθηκεύονται σε ένα αντικείμενο ανά ροή, αλλά σε ανεξάρτητα αντικείμενα που φέρουν τον αριθμό της ροής στον οποίο ανήκουν, η αναζήτηση μιας ακμής γίνεται σε χρόνο O(1) και είναι ανεξάρτητη του μεγέθους του γράφου.

7 Επίλογος

7.1 Σύνοψη και συμπεράσματα

Με βάση τόσο την θεωρητική ανάλυση που προηγήθηκε, όσο και τα αποτελέσματα που λάβαμε από τις τοπολογίες που υλοποιήθηκαν καταλήξαμε σε κάποια συμπεράσματα σε σχέση με τη λειτουργία του Storm. Αρχικά, είναι ένα εργαλείο που εστιάζει στην υλοποίηση αλγορίθμων πραγματικού χρόνου, που σημαίνει ότι δεν έχει τόσο καλή εφαρμογή σε κλασικούς αλγόριθμους δεσμίδας με επαναλήψεις πάνω στα δεδομένα, όπως το Hadoop. Ένα τέτοιο παράδειγμα είναι ο αλγόριθμος PLDA, που έχει αναπτυχθεί από τους Wang et al [26], ο οποίος αποτελεί μια επέκταση του κλασικού LDA, για παράλληλη εκτέλεση των επαναλήψεων. Αυτός ο αλγόριθμος παρότι χαρακτηρίζεται από μεγάλο βαθμό παραλληλίας, δεν είναι κατάλληλος για υλοποίηση με το Storm, γιατί απαιτεί συγκεκριμένο αριθμό επαναλήψεων πάνω στο σύνολο των δεδομένων, που δεν ορίζεται σε μια ατέρμονη τοπολογία. Οι αλγόριθμοι που απαιτούν ένα μοναδικό πέρασμα των δεδομένων εισόδου είναι πιο κοντά στη φύση του εργαλείου. Συνεπώς, το παραπάνω πρέπει να λαμβάνεται υπόψιν κατά τη σχεδίαση των τοπολογιών.

Όσον αφορά την επεξεργασία κειμένου για εξαγωγή συμπερασμάτων σε πραγματικό χρόνο, το Storm είναι ένα στοχευμένο εργαλείο που χρησιμοποιείται από το Twitter, κυρίως για αυτό το λόγο. Συνεπώς, οι λειτουργικότητες που προσφέρει είναι σχεδιασμένες προς

αυτή την κατεύθυνση και η ανάπτυξη ανάλογων συστημάτων είναι απλή και κατανοητή. Λαμβάνοντας υπόψιν την πολυπλοκότητα ανάπτυξης αντίστοιχων συστημάτων, μπορούμε να πούμε ότι δικαίως εξελίσσεται σε ένα πολύ δημοφιλές εργαλείο. Ειδικότερα, με την χρήση του πλαισίου Trident, ο προγραμματισμός ενός συστήματος γίνεται σε πολύ υψηλό επίπεδο, καθώς προσφέρονται όλες οι βασικές συναρτήσεις που μπορούν να εφαρμοστούν σε μια ροή δεδομένων, ενώ οι συνδέσεις των bolt βελτιστοποιούνται αυτόματα.

Όσον αφορά τη συστοιχία, η επικοινωνία μεταξύ των διαφορετικών κόμβων – εργατών αυτοματοποιείται από τις διάφορες οντότητες. Σημαντικότερα, γίνεται αποτελεσματικός χειρισμός της αποτυχίας των κόμβων κατά την εκτέλεση της τοπολογίας, έτσι ώστε να εξασφαλιστεί η επεξεργασία κάθε τούπλας. Όταν ένα μηχάνημα αποτύχει, ή επανέλθει ξανά στη συστοιχία, η ανάθεση των υποεργασιών επαναλαμβάνεται με διάφανο για τους υπόλοιπους εργάτες τρόπο.

Τέλος, ανάλογα με τη φύση του αλγορίθμου, είναι δυνατή η χρήση της παραλληλίας. Οι αλγόριθμοι που υλοποιήθηκαν στη συγκεκριμένη εργασία δεν χαρακτηρίζονταν από μεγάλο βαθμό παραλληλίας, καθώς η ενημέρωση της κατάστασης έπρεπε να γίνεται καθολικά στο τελευταίο bolt με τις συναρτήσεις *.persistentAggregate* και *.global*. Αυτός ο περιορισμός εισάγεται λόγω της φύσης του αλγορίθμου και αποτελεί το σημείο συμφόρησης των τοπολογιών. Παρόλα αυτά, για τοπολογίες άλλου είδους, όπως η τοπολογία WordCount, στην οποία η ενημέρωση της κατάστασης γίνεται για κάθε λέξη ξεχωριστά, το επίπεδο παραλληλίας που μπορεί να επιτευχθεί και ως αποτέλεσμα η αντίστοιχη μείωση του χρόνου, είναι πολύ μεγαλύτερη.

7.2 Μελλοντικές επεκτάσεις

Σε έναν τόσο ευρύ χώρο, όπως αυτός της ανάλυσης διαδικτυακού περιεχομένου για εξαγωγή συμπερασμάτων, οι μελλοντικές κατευθύνσεις είναι ανεξάντλητες. Όσον αφορά το αλγοριθμικό κομμάτι θα μπορούσαν να δοκιμαστούν περισσότεροι αλγόριθμοι, έτσι ώστε να εξεταστεί η αποδοτικότητα τους με τη χρήση του Storm. Ένας σημαντικός στόχος θα ήταν η παραπάνω αύξηση της παραλληλίας, που θα μπορούσε να επιτευχθεί με την χρήση κατάλληλων αλγορίθμων.

Τεχνικά, θα μπορούσαν να χρησιμοποιηθούν συστήματα, όπως το Kafka και το RabbitMQ, για την δημιουργία συστημάτων ουρών που λαμβάνουν δεδομένα και υλοποιούν

μηχανισμό διατήρησης και ειδοποιήσεων. Με αυτό τον τρόπο, παρέχουν με αποτελεσματικό και ασφαλή τρόπο την είσοδο των δεδομένων στα αντίστοιχα Spout της τοπολογίας Storm.

Τέλος, όσον αφορά το κομμάτι της μηχανικής μάθησης και της εκπαίδευσης των αλγορίθμων, μια μελλοντική προοπτική είναι η μελέτη του εργαλείου SAMOA. Το SAMOA είναι ένα εργαλείο που έχει αναπτυχθεί, ώστε να χρησιμοποιείται πάνω από το Storm και αυτοματοποιεί τις συνηθέστερες πρακτικές των αλγορίθμων μηχανικής μάθησης. Βασικές οντότητες του SAMOA είναι ο Εκπαιδευτής, που αναλαμβάνει τη διαδικασία της εκπαίδευσης με βάση τα αρχικά δεδομένα και ο Αξιολογητής, που αναλαμβάνει την ταξινόμηση με βάση την εκπαίδευση που έχει προηγηθεί. Αυτά τα δύο στάδια είναι τα κυριότερα ενός αλγορίθμου μηχανικής μάθησης. Σύμφωνα με την περιγραφή των Bakshi et al [3], αποτελεί ένα ισχυρό εργαλείο για την δημιουργία παράλληλων συστημάτων μηχανικής μάθησης, που επεξεργάζονται πολύ μεγάλο όγκο δεδομένων.

8 *Βιβλιογραφία*

[1] Aisopos Fotis, Papadakis George, Tserpes Konstantinos, Varvarigou Theodora. (2012). "Content vs. Context for Sentiment Analysis: a Comparative Analysis over Microblogs".

[2] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, Dmitriy Ryaboy. (2014). "Storm @ Twitter".

[3] Bakshi Rohit Prasad, Sonali Agarwa. (2014). "Handling Big Data Stream Analytics using SAMOA Framework -A Practical Experience".

[4] Blei David M., Ng Andrew, Jordan Michael. (2003). "Latent Dirichlet Allocation"

[5] Blei David M. Lafferty Hohn D. (2009). "Topic Models".

[6] Blei David M., Bach Francis, Hoffman Matthew D., (2010). "Online Learning for Latent Dirichlet Allocation".

[7] Blei David M. (2012). "Probabilistic topic models".

[8] Blei David M. (2012). "Introduction to Probabilistic Topic Models".

[9] Bo Pang, Lillian Lee. (2008). "Opinion mining and sentiment analysis".

[10] Gimpel Kevin. (2006). "Modelling Topics".

[11] Goetz P. Taylor, O'Neill Brian. (2014). "Storm Blueprints: Patterns for Distributed Real-time Computation".

[12] Howard Taylor M., Samuel Karlin. (1998) "An Introduction To Stochastic Modeling".

[13] Igor Brigadir, Derek Greene, Padraig Cunningham, Gavin Sheridan. (2013). “Real Time Event Monitoring with Trident”.

[14] Ishana Raina, Sourabh Gujar, Parth Shah, Aishwarya Desai, Balaji Bodkhe. (2014). "Twitter Sentiment Analysis using Apache Storm".

[15] Krestel Ralf, Frankhauser Peter, Nejdl Wolfgang. (2009). "Latent Dirichlet Allocation for Tag Recommendation".

[16] Liangjie Hong, Brian D. Davison. (2010). "Empirical Study of Topic Modeling in Twitter".

[17] Marz Nathan. (2012). “Storm: Distributed and fault-tolerant realtime computation”

[18] Michael J. Paul, Mark Dredze. (2011). "You Are What You Tweet: Analyzing Twitter for Public Health"

[19] Pak Alexander, Paroubek Patrick. (2010). "Twitter as a Corpus for Sentiment Analysis and Opinion Mining".

[20] Paksula Matti. (2010). "Persisting Objects in Redis Key-Value Database"

[21] Saif, Hassan; He, Yulan and Alani, Harith (2012). “Semantic sentiment analysis of twitter. In: The 11th International Semantic Web Conference (ISWC 2012), 11-15 November 2012, Boston, MA, USA”.

[22] Shin, K.G.; Ramanathan, P. (1994). "Real-time computing: a new discipline of computer science and engineering".

[23] Stonebraker Michael, Çetintemel Uğur, Zdonik Stan. (2005). "The 8 Requirements of Real-Time Stream Processing".

[24] Steyvers Mark, Griffiths Tom. (2007). "Probabilistic Topic Models".

[25] Yogesh Tewari, Rajesh Kawad. (2013). "Real-Time Topic Modeling of Microblogs"

[26] Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, Edward Y. Chang. (2009). "PLDA: Parallel Latent Dirichlet Allocation for Large-scale Applications".

9 *Παράρτημα:*

9.1 Εγκατάσταση, παραμετροποίηση και εκκίνηση μιας συστοιχίας Storm

Για να εγκαταστήσουμε μια συστοιχία Storm σε περιβάλλον Linux εκτελούμε μια σειρά από βήματα, σε όλους τους υπολογιστές που απαρτίζουν τη συστοιχία:

1. Εγκατάσταση Java JDK
2. Εγκατάσταση και παραμετροποίηση Zookeeper (αρχείο zoo.cfg)

```
tickTime=2000
dataDir=/usr/local/zookeeper-3.4.5/myid
clientPort=2181
initLimit=5
syncLimit=2
server.1=IP1:2888:3888
server.2=IP2:2888:3888
.....
server.N=IPN:2888:3888
```

3. Εγκατάσταση jzmq και zeroMQ
4. Εγκατάσταση Python
5. Εγκατάσταση DaemonTools
6. Εγκατάσταση και παραμετροποίηση Storm-Apache (αρχείο storm.yaml)

```
storm.zookeeper.servers:
    - IP1
    - IP2 ...
nimbus.host: IP
supervisor.slots.ports:
     - 6700
     - 6701
     - 6702
     - 6703
drpc.servers:
 - IP1
 - IP2 ...
```

Η συστοιχία Zookeeper που δημιουργείται, είναι υπεύθυνη για την ανταλλαγή μηνυμάτων μεταξύ των κόμβων της συστοιχίας. Τα μηνύματα, που στέλνονται μέσω του πρωτοκόλλου TCP, αφορούν τον συγχρονισμό και συντονισμό των κόμβων και όχι δεδομένα της εκτέλεσης του προγράμματος.

Το DaemonTools χρησιμοποιείται για εκτέλεση του Storm υπό επίβλεψη και είναι απαραίτητο γιατί οι κόμβοι μπορεί ανά πάσα στιγμή να αντιμετωπίσουν σφάλμα και να σκοτωθούν. Το DaemonTools επιβλέπει την εκτέλεση και επανεκκινεί τους κόμβους σε τέτοιες περιπτώσεις.

Για να ξεκινήσουμε την συστοιχία εκτελούμε την εντολή *bin/storm nimbus* στον Nimbus και *bin/storm supervisor* στους εργάτες. Επίσης, εκτελούμε την εντολή *bin/storm ui* στον Nimbus για να ξεκινήσουμε το γραφικό περιβάλλον του Storm. Για να επιβεβαιώσουμε ότι όλα πήγαν καλά, ανοίγουμε τη σελίδα http://{IP Nimbus}:8080 σε κάποιον περιηγητή. Εκεί θα δούμε το γραφικό περιβάλλον, όπου θα υπάρχουν πληροφορίες για τους συνδεδεμένους κόμβους της συστοιχίας.