Επεξεργασία Πολύπλοκων Γεγονότων στο Κατανεμημένο Σύστημα Storm

Γεώργιος Διδυμιώτης – Βεντούρης

Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών (ΗΜΜΥ)

Δεκέμβριος 2015, Χανιά

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γεώργιος Διδυμιώτης-Βεντούρης

με θέμα

Επεξεργασία Πολύπλοκων Γεγονότων στο Κατανεμημένο Σύστημα Storm

Complex Event Processing in Storm Distributed System

Εξεταστική Επιτροπή:

Αντώνιος Δεληγιαννάκης, Αναπληρωτής Καθηγητής (Επιβλέπων)
Μίνως Γαροφαλάκης, Καθηγητής
Πολυχρόνης Κουτσάκης, Αναπληρωτής Καθηγητής

Η διπλωματική εργασία μου είναι αφιερωμένη...

Στους γονείς μου, στην αδερφή μου και στην υπόλοιπη οικογένεια

Στους φίλους μου

Σε όλους μου τους συντρόφους

Περίληψη

Η Επεξεργασία Πολύπλοκων Γεγονότων (Complex Event Processing) είναι μια μέθοδος εντοπισμού, ανάλυσης και επεξεργασίας ροών πληροφορίας και πιο συγκεκριμένα γεγονότων που συμβαίνουν (events). Διαφέρει από το Επεξεργασία Γεγονότων (Event Processing) διότι συνδυάζει πληροφορία από πολλαπλές πηγές, για να ανταποκριθεί σε πιο περίπλοκες συνθήκες. Στόχος του είναι να εντοπίσει καταστάσεις που εμπίπτουν σε συνθήκες βασισμένες σε μια σειρά από γεγονότα που έχουν συμβεί μέσα σε ένα δυναμικό χρονικό πλαίσιο. Αυτές οι συνθήκες μπορεί να είναι είτε αλληλουχία (SEQUENCE) γεγονότων είτε λογικές πράξεις μεταξύ γεγονότων (AND, OR) είτε και συνδυασμός αυτών. Με αυτής τη μέθοδο καθίσταται εφικτός ο εντοπισμός σημαντικών καταστάσεων και η αντιμετώπιση αυτών το τάχιστο δυνατό.

Για να επιτύχουμε την κλιμάκωση ενός CEP συστήματος σε μεγάλο όγκο δεδομένων, είναι επιθυμητή η δυνατότητα παραλληλοποίησης του σε μεγάλο αριθμό από υπολογιστές οι οποίοι ανήκουν σε μία συστάδα (cluster). Για αυτό το λόγο, σε αυτή την εργασία η μέθοδος Επεξεργασίας Πολύπλοκων Γεγονότων (CEP) υλοποιείται χρησιμοποιώντας το κατανεμημένο υπολογιστικό σύστημα Storm που λειτουργεί σε πραγματικό χρόνο. Κατασκευάσαμε μια τοπολογία που χρησιμοποιεί το σύστημα και υλοποιεί τη μέθοδο ΕΠΓ με πλήρη επεκτασιμότητα. Στη τοπολογία μας γίνεται συνεχή επεξεργασία γεγονότων σε πραγματικό χρόνο και ανίχνευση μοτίβων που δίνει, ανανεώνει και διαγράφει ο χρήστης.

Abstract

Complex Event Processing is a method of tracking, analyzing and processing streams of data and specifically occurring events. CEP differs from Event Processing because it combines information from different sources, in order to correspond to complicated conditions. Its goal is to trace on a system states that fulfill specific conditions based on a series of events that occurred in a dynamic time window. These conditions can be either an event sequence, a series of logical operations between events (AND, OR), or a combination of both. This method enables the detection of important states and accelerates their resolution.

In order to achieve the extend of a CEP system to big data, it should be parallelized between a great number of computers belonging to a cluster. For this reason, in this thesis the Complex Event Processing method is implemented on the distributed real time computation system of “Storm”. We developed a fully scalable topology that uses Storm and implements the CEP method. Our topology continuously processes events in real time and traces in real time patterns that the user can provide, refresh and/or remove.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Δεληγιαννάκη για τη συνεργασία που είχαμε όλο το διάστημα της εκπόνησης αυτής της διπλωματικής. Επίσης για την συνεχή εποπτεία και βοήθεια που μου παρείχε καθώς και ότι ήταν πάντοτε διαθέσιμος για τις απορίες μου και προβλήματα που ήθελα να του εκφράσω. Ευχαριστώ ακόμα τους συμφοιτητές μου Μανικάκη και Παυλάκη που με βοήθησαν σε κρίσιμες στιγμές της διπλωματικής. Τέλος ευχαριστώ τα υπόλοιπα μέλη της εξεταστικής επιτροπής, κ. Γαροφαλάκη και κ. Κουτσάκη για το χρόνο που ξόδεψαν για να διαβάσουν και να αξιολογήσουν τη διπλωματική μου εργασία.

Περιεχόμενα

1. Εισαγωγή

1.1 Επεξεργασία Πολύπλοκων Γεγονότων

Για τις ανάγκες επικοινωνίας υπολογιστή με υπολογιστή δημιουργούνται σε πραγματικό χρόνο, μεγάλου όγκου δεδομένα που πλέον έχουν ξεπεράσει σε όγκο τα δεδομένα που ο ίδιος ο άνθρωπος παράγει. Τα δεδομένα αυτά αποτελούνται από μεγάλο πλήθος πληροφορίας που αφορά ερωτήσεις, απαντήσεις μεταξύ των υπολογιστών, την μεταξύ τους ενημέρωση για σημαντικές ή κρίσιμες καταστάσεις που βρίσκονται, και πολλά άλλα απαραίτητα για τη σωστή λειτουργία τους. Η αποκωδικοποίηση τους και η διαλογή των πραγματικά χρήσιμων πληροφοριών από μέρους μας σε πραγματικό χρόνο θα μπορούσε να μας παρέχει γνώση χρήσιμη για τον εντοπισμό και την πρόληψη καταστάσεων που μας ενδιαφέρουν.

Αυτού του είδους τα δεδομένα επειδή ακριβώς είναι για εσωτερική επικοινωνία υπολογιστών είναι επαρκώς δομημένα στη μορφή και συνήθως αποτελούν τη σειρά διάφορων γεγονότων που τελέστηκαν. Έτσι η επεξεργασία τους με σκοπό τον εντοπισμό των καταστάσεων ενδιαφέροντος που προαναφέραμε γίνεται αρκετά πιο εύκολη. Αρκεί να ορίσουμε σωστά αυτές τις καταστάσεις με τρόπο όπου δεν θα μπορούν να ξεφύγουν και εν τέλη ο χρήστης να μην λάβει γνώση για αυτές. Επίσης είναι αναγκαίο να μπορούμε να επεξεργαστούμε το όγκο των γεγονότων που συμβαίνουν σε πραγματικό χρόνο ώστε να μπορέσουμε να έχουμε όχι απλά εντοπισμό αλλά και πρόληψη. Η Επεξεργασία Πολύπλοκων Γεγονότων αποτελεί ένα εξελισσόμενο πεδίο με σημαντικές εφαρμογές σε συστήματα πραγματικού χρόνου. Σκοπός των συστημάτων ΕΠΓ είναι ο εντοπισμός ορισμένων μοτίβων, σε μια ροή από πρωταρχικά γεγονότα. Πιο συνοπτικά μπορούμε να πούμε ότι η ΕΠΓ έχει την ικανότητα να αντλεί γνώση υψηλού επιπέδου από απλά γεγονότα.

Για να αντιληφθούμε τη λειτουργία του ΕΠΓ χρησιμοποιείται ένα χαρακτηριστικό παράδειγμα: [11] Μεταξύ χιλιάδων γεγονότων που λαμβάνει το σύστημα μας θα μπορούσε να λάβει και το χτύπημα της καμπάνας, την εμφάνιση ενός άνδρα με κουστούμι, το λευκό μακρύ φόρεμα μιας γυναίκας και ρύζι να πετιέται στον αέρα. Από αυτά τα πρωταρχικά γεγονότα εφόσον έχει οριστεί από τον χρήστη το σύστημα ΕΠΓ μπορεί να συμπεράνει ότι συμβαίνει κάποιος γάμος. Ο γάμος αποτελεί και το πολύπλοκο γεγονός στο συγκεκριμένο παράδειγμα. Η ΕΠΓ χρησιμοποιεί πλήθος τεχνικών για να επιτύχει το σκοπό της. Μερικές από αυτές είναι: ο εντοπισμός γεγονότος-μοτίβου, η εννοιολογική αφαίρεση ενός γεγονότος, φιλτράρισμα των γεγονότων, μετατροπή και συγκέντρωση γεγονότων, η μοντελοποίηση της ιεραρχίας των γεγονότων, εντοπισμός των σχέσεων μεταξύ των γεγονότων.

Εμπορικές εφαρμογές των συστημάτων ΕΠΓ υπάρχουν σε πληθώρα από βιομηχανίες και συμπεριλαμβάνουν αλγορίθμους εμπορίου χρηματιστηριακών μετοχών, εντοπισμού ύποπτων κινήσεων πιστωτικών καρτών, ελέγχου ασφαλείας, ελέγχου της δραστηριότητας στο εσωτερικό μια επιχείρησης. Τέτοιες εφαρμογές απαντούν στις ανάγκες πολλών και διαφορετικών τομέων όπως τα λογιστικά, η διαχείριση αποθηκών, συστήματα εφοδιασμού, χρηματιστηρίου κ.α.

1.2 Αντικείμενο διπλωματικής

Στα πλαίσια αυτής της διπλωματικής εργασίας υλοποιήθηκε μια εφαρμογή Επεξεργασίας Πολύπλοκων Γεγονότων με σκοπό την αποδοτική επεξεργασία των δεδομένων και τον άμεσο εντοπισμό ενδιαφέροντων συνθηκών τη στιγμή που συμβαίνουν. Για την υλοποίηση αυτού του συστήματος χρησιμοποιήσαμε το κατανεμημένο σύστημα Storm και πάνω σε αυτό αναπτύξαμε την εφαρμογή μας.

Το "Storm" ή πιο σωστά "Apache Storm" είναι ένα κατανεμημένο υπολογιστικό σύστημα γραμμένο κυρίως σε Clojure και Java. Δημιουργήθηκε το 2011 από τον Nathan Marz και την Backtype και είναι ανοικτού κώδικα από την στιγμή που αποκτήθηκε από το Twitter. Σήμερα το Storm ανήκει στην εταιρία Apache. Το Storm είναι εύχρηστο και μπορεί να χρησιμοποιηθεί με κάθε γλώσσα προγραμματισμού. Πολλές εταιρίες το χρησιμοποιούν ήδη για την επεξεργασία του μεγάλου όγκου δεδομένων που έχουν. Χαρακτηριστικά παραδείγματα είναι η Yahoo, Twitter, Groupon, Spotify, Flipboard, Yelp, Alibaba κ.α. [10]

Τα ιδιαίτερα χαρακτηριστικά του Storm μας είναι εξαιρετικά χρήσιμα και για αυτό επιλέχτηκε. Συγκεκριμένα ο συνεχής υπολογισμός, οι αναλύσεις πραγματικού χρόνου και ο κατανεμητικός χαρακτήρας του ήταν βασικά στοιχεία που συνιστούν το συγκεκριμένο εργαλείο κατάλληλο για την ανάπτυξης μιας εφαρμογής CEP.

Πιο αναλυτικά βασικές υποθέσεις χρήσης του συστήματος Storm είναι:

Επεξεργασία ροών
Το Storm χρησιμοποιείται για να επεξεργαστούμε ροές δεδομένων και να ανανεώσουμε μια ποικιλία από βάσεις δεδομένων σε πραγματικό χρόνο. Αυτή η επεξεργασία συμβαίνει σε πραγματικό χρόνο και η επεξεργαστική ταχύτητα πρέπει να είναι όμοια της ταχύτητας εισόδου δεδομένων.

Συνεχής υπολογισμός
Το Storm μπορεί να κάνει συνεχής υπολογισμούς στις ροές δεδομένων και να μεταδίδει σε πραγματικό χρόνο στους πελάτες(clients). Αυτό ίσως απαιτεί την επεξεργασία κάθε μηνύματος που έρχεται. Παράδειγμα του συνεχή υπολογισμού είναι η συνεχής ροή των τάσεων του Twitter στους περιηγητές

Αναλύσεις πραγματικού χρόνου
Το Storm μπορεί να αναλύει και να ανταποκρίνεται σε δεδομένα που έρχονται από διαφορετικές πηγές δεδομένων σε πραγματικό χρόνο.

Κατανεμημένο RPC
Το Storm έχει τη δυνατότητα να παραλληλίσει ένα δύσκολο αίτημα έτσι ώστε να μπορεί να το υπολογίσει σε πραγματικό χρόνο.

Βασικό κριτήριο στη υλοποίηση της διπλωματικής μας, της εφαρμογής Επεξεργασίας Πολύπλοκων Δεδομένων για μεγάλο όγκο δεδομένων, είναι το πως αυτή η εφαρμογή θα είναι εύκολα και άμεσα κλιμακώσιμη, πως θα εγγυόμαστε αξιοπιστία στο χρήστη ενώ ταυτόχρονα επεξεργαζόμαστε σε πραγματικό χρόνο τα μεγάλου όγκου δεδομένα. Επρεπε λοιπόν το εργαλείο που θα επιλέγαμε να έχει κάποια στοιχεία που θα χρησιμοποιούσαμε για να πετύχουμε τους στόχους μας. Το Storm εμπεριέχει όλα αυτά τα κριτήρια που θέσαμε πριν και έχει τις ιδιότητες που το κάνουν χρήσιμο και στη δικιά μας περίπτωση.

Πιο συγκεκριμένα οι βασικές του ιδιότητες είναι:

Εύκολο να το προγραμματίσεις
Η σε πραγματικό χρόνο επεξεργασία δεδομένων είναι πολύ κουραστική και επώδυνη δουλειά. Η πολυπλοκότητα αυτής της δουλειάς μειώνεται δραματικά με το Storm.

Υποστήριξη πολλαπλών γλωσσών προγραμματισμού
Αν και είναι ευκολότερο να υλοποιήσεις εργασίες σε Java το Storm υποστηρίζει και κάθε άλλη γλώσσα προγραμματισμού αρκεί να χρησιμοποιείται σαν ενδιάμεσος η απαραίτητη βιβλιοθήκη.

Ανεκτικό σε σφάλματα
Το Storm φροντίζει για τυχόν σφάλματα που προκύπτουν κατά την διάρκεια λειτουργίας του και αναθέτει ξανά εργασίες αν είναι απαραίτητο.

Κατακόρυφα κλιμακώσιμο
Το μόνο που χρειάζεται να κάνεις για να διευρύνεις το σύστημα σου είναι να προσθέσεις μηχανήματα. Το Storm μόνο του θα αναδιατάξει και θα επαναδιατάξει τις εργασίες στα νέα μηχανήματα.

Αξιόπιστο
Όλα τα μηνύματα είναι εγγυημένο ότι θα επεξεργαστούν τουλάχιστον μία φορά και ποτέ δεν θα χαθεί κάποιο μήνυμα.

Γρήγορο
Βασικός στόχος από την αρχή της υλοποίησης του Storm ήταν η ταχύτητα και εν τέλη καταφέρνει να πετύχει γρήγορη επεξεργασία δεδομένων σε πραγματικό χρόνο.

1.3 Οργάνωση κειμένου

Η διπλωματική εργασία χωρίζεται σε 8 κεφάλαια. Το κεφάλαιο 1 είναι μια εισαγωγή σε αυτή τη διπλωματική εξηγώντας κάποια βασικά στοιχεία της Επεξεργασία Πολύπλοκων Γεγονότων και γιατί υλοποιήθηκε αυτή η εφαρμογή στο σύστημα Storm. Στο 2ο κεφάλαιο αναλύουμε το θεωρητικό υπόβαθρο και τις βασικές αρχές του Storm και κάποια βασικά χαρακτηριστικά του CEP. Αυτά τα δύο στοιχεία (Storm, CEP) που είναι και τα βασικά της διπλωματικής μας τα αναλύουμε διεξοδικά ως προς τα συγκεκριμένα χαρακτηριστικά που έχουν, τις παραμέτρους που πρέπει να ορίσουμε αλλά και εργαλεία που μας παρέχουν στα επόμενα δυο κεφάλαια αντίστοιχα. Στο κεφάλαιο 5 αναλύουμε την πορεία προς την εύρεση της σωστής τοπολογίας καθώς και τα δομικά κομμάτια αυτής που τελικά υλοποιήσαμε. Στο επόμενο κεφάλαιο μπαίνουμε στην καρδία της υλοποίησης. Δίνουμε για κάθε spout ή bolt την πορεία επεξεργασίας των δεδομένων, εξηγούμε όλες τις συναρτήσεις που κάνουμε χρήση αλλά και πως διαχέουμε τα δεδομένα κατά μήκος της τοπολογίας. Λίγο πριν το τέλος δείχνουμε τις μετρήσεις και τις δοκιμές που κάναμε και βγάζουμε κάποια σημαντικά συμπεράσματα για το ποια είναι τα όρια και οι δυνατότητες της εφαρμογής μας. Κλείνοντας κάνουμε μια αναφορά στις μελλοντικές επεκτάσεις και δυνατότητες που έχει η συνέχιση αυτής της διπλωματικής.

2. Storm

2.1 Αρχιτεκτονική του Storm

Ένα σύμπλεγμα Storm ακολουθεί το μοντέλο master-slave, όπου οι διαδικασίες master και slave συντονίζονται μέσω του *Zookeeper*. Ο κόμβος *Nimbus* είναι ο master σε ένα σύμπλεγμα Storm. Είναι υπεύθυνη για τη διανομή του κώδικα εφαρμογής σε διάφορους κόμβους των εργαζομένων, την ανάθεση καθηκόντων σε διαφορετικά μηχανήματα, παρακολούθηση για τυχόν αποτυχίες και την επαναλειτουργία τους, όπως και όταν απαιτείται. Υπάρχει ένας μόνο κόμβος Nimbus σε ένα σύμπλεγμα Storm. *Επιβλέπων* κόμβοι είναι οι κόμβοι worker σε ένα σύμπλεγμα Storm. Κάθε κόμβος Storm τρέχει ένα δαίμονα επόπτη που είναι υπεύθυνος για τη δημιουργία, την έναρξη και τη διακοπή της εκτέλεσης των διαδικασιών worker που έχουν ανατεθεί σε αυτόν τον κόμβο. Ένας δαίμονα επόπτης χειρίζεται συνήθως πολλαπλές διαδικασίες worker που τρέχουν σε αυτό το μηχάνημα. Ο Zookeeper που συντονίζει τον Nimbus και τους Επιβλέποντες αποθηκεύει τις καταστάσεις που σχετίζονται με το σύμπλεγμα καθώς και τις διάφορες εργασίες που έχουν ανατεθεί στο Storm. Ο Nimbus κόμβος και οι Επιβλέποντες κόμβοι επικοινωνούν χρησιμοποιώντας ως ενδιάμεσο τον Zookeeper. [2]

2.1 Αρχιτεκτονική συμπλέγματος Storm[3]

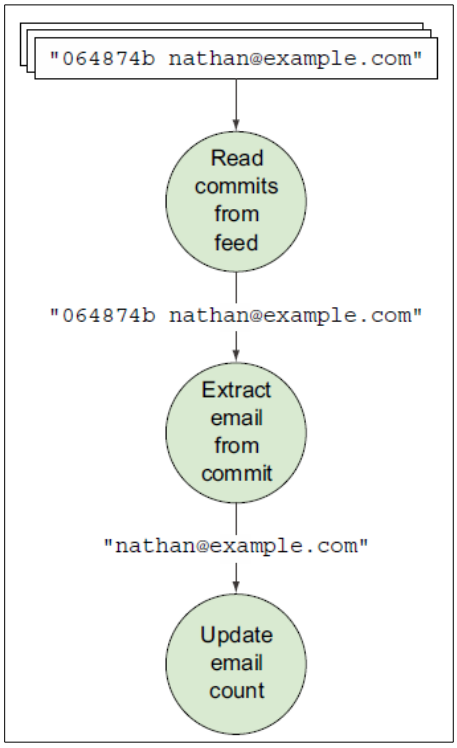
2.2 Μοντέλο Δεδομένων

Η βασική μονάδα δεδομένων που επεξεργάζεται από μια εφαρμογή Storm ονομάζεται πλειάδα(tuple). Κάθε πλειάδα αποτελείται από μια προκαθορισμένη λίστα πεδίων. Η τιμή του κάθε πεδίου μπορεί να είναι byte, char, integer, long, float, double, Boolean, ή πίνακας από bytes. Το Storm παρέχει επίσης μια διεπαφή για να καθορίσουμε τους δικούς μας τύπους δεδομένων, τα οποία μπορεί στη συνέχεια να είναι πεδία σε μια πλειάδα.

Μια πλειάδα έχει δυναμικούς τύπους δεδομένων, πιο συγκεκριμένα, το μόνο που χρειάζεται να καθορίσετε τα ονόματα των πεδίων σε μια πλειάδα και όχι τον τύπο των δεδομένων τους. Κάθε πεδίο μιας πλειάδας μπορεί να προσπελαστεί από το όνομα της είτε από τη θέση της στον κατάλογο της πλειάδας. [2]

2.3. Ορισμός Τοπολογίας

Βάση της ορολογίας του Storm, μια τοπολογία είναι μια αφηρημένη έννοια που ορίζει τον υπολογιστικό γράφο. Μπορούμε να δημιουργήσουμε μια τοπολογία Storm και να την αναπτύξουμε στο σύμπλεγμα Storm για να επεξεργαστούμε τα δεδομένα. Η τοπολογία μπορεί να αναπαρασταθεί με ένα κατευθυνόμενο άκυκλο γράφο όπου κάθε κόμβος κάνει κάποιου είδους επεξεργασία και τη προωθεί στο/ους επόμενο/ους κόμβους της ροής.

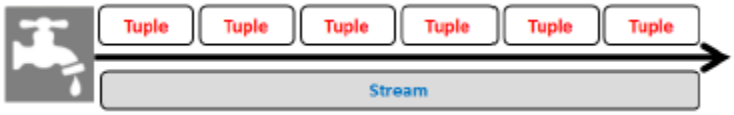


2.2 Παράδειγμα μια συμβατικής τοπολογίας όπου μπορούμε να παρατηρήσουμε τα βήματα επεξεργασίας των δεδομένων[1]

Συστατικά μιας τοπολογίας του Storm είναι:

2.3.1 Spout

Ένα spout είναι η πηγή των πλειάδων σε μια τοπολογία του Storm. Είναι υπεύθυνο για την ανάγνωση των δεδομένων από μια εξωτερική πηγή και την εκπομπή(emit) τους στις ροές. Ένα spout μπορεί να εκπέμπει σε πολλαπλά ρεύματα.



2.3 Παράδειγμα λειτουργίας ενός τυπικού Spout

Σημαντικές μέθοδοι ενός spout είναι:

nextTuple(): Αυτή η μέθοδος καλείται από το Storm για να πάρει την επόμενη πλειάδα από την πηγή εισόδου. Μέσα σε αυτή τη μέθοδο, υπάρχει η λογική της ανάγνωσης των δεδομένων από τις εξωτερικές πηγές και της εκπομπής τους στις ροές. *(Εικόνα 2.4)*

open(): Αυτή η μέθοδος καλείται μόνο μια φορά, όταν το spout αρχικοποιείται. Εάν είναι απαραίτητο να συνδεθεί με μια εξωτερική πηγή για τα δεδομένα εισόδου, καθορίζουμε τη σύνδεση με την εξωτερική πηγή στην μέθοδο open, και μετά μεταφέρουμε δεδομένα από αυτήν την εξωτερική πηγή διοχετεύοντας τα στη μέθοδο nextTuple για να εκπέμπουν παραπέρα. Στην *εικόνα 2.4* φαίνεται μια απλή υλοποίηση αυτής της μεθόδου όπου συνδέεται το Spout με ένα αρχείο .txt.

ack(): Το Storm καλεί αυτή τη μέθοδο όταν μια πλειάδα με ένα δεδομένο όρισμα επεξεργαστεί πλήρως από την τοπολογία. Εγγυάται με αυτό τον τρόπο το σύστημα ότι δεν θα επεξεργαστεί η ίδια πλειάδα ξανά στο μέλλον.

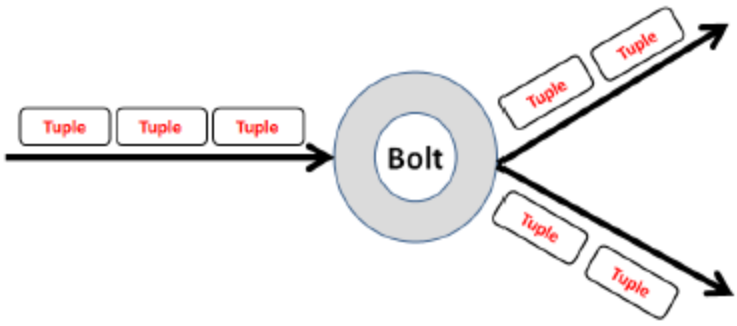
*fail():*Το Storm καλεί αυτή τη μέθοδο όταν διακρίνει ότι μια πλειάδα με ένα δεδομένο όρισμα δεν έχει επεξεργαστεί επιτυχώς ή έχει τελειώσει το χρονικό περιθώριο. Έτσι δίνεται η δυνατότητα από το σύστημα να επαναληφθεί η επεξεργασία της πλειάδας που πριν είχε αποτύχει.

```
public class CommitFeedListener extends BaseRichSpout {
  private SpoutOutputCollector outputCollector;
  private List<String> commits;

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("commit"));
  }

  @Override
  public void open(Map configMap,
                   TopologyContext context,
                   SpoutOutputCollector outputCollector) {
    this.outputCollector = outputCollector;

    try {
      commits = IOUtils.readLines(
         ClassLoader.getSystemResourceAsStream("changelog.txt"),
         Charset.defaultCharset().name()
      );
    } catch (IOException e) {
      throw new RuntimeException(e);
    }
  }

  @Override
  public void nextTuple() {
    for (String commit : commits) {
      outputCollector.emit(new Values(commit));
    }
  }
}
```

2.4 Παράδειγμα κώδικα ενός τυπικού Spout [1]

2.3.2 Bolt

Ένα bolt είναι η βασική δομή επεξεργασίας μιας τοπολογίας Storm και είναι υπεύθυνη για τη μετατροπή μιας ροής. Ιδανικά, κάθε bolt στην τοπολογία θα πρέπει να κάνει μια απλή μετατροπή των πλειάδων, και πολλά bolts μπορούν να συντονιστούν μεταξύ τους για να επιδείξουν ένα σύνθετο μετασχηματισμό. Ένα bolt μπορεί να εγγραφεί σε πολλαπλές ροές των άλλων συστατικών, είτε spout είτε bolt, στην τοπολογία και ομοίως μπορεί να εκπέμπει έξοδο σε πολλαπλές ροές.



2.5 Παράδειγμα λειτουργίας ενός τυπικού Bolt

Σημαντικές μέθοδοι ενός bolt είναι:

execute(): Αυτή η μέθοδος εκτελείται από το Storm για κάθε πλειάδα που έρχεται μέσα από τα ρεύματα εισόδου που το κάθε bolt έχει εγγραφεί. Στη μέθοδο αυτή, πραγματοποιείται ότι επεξεργασία χρειάζεται για κάθε πλειάδα και στη συνέχεια, παράγει την έξοδο είτε με τη μορφή της εκπομπής περισσότερων πλειάδων στις δηλωμένες ροές εξόδου είτε με κάποιο άλλο τρόπο, όπως η διατήρηση των αποτελεσμάτων σε μια βάση δεδομένων. *(Εικόνα 2.6)*

prepare(): Αυτή η μέθοδος καλείται ακριβώς πριν το bolt ξεκινήσει την επεξεργασία των πλειάδων. Με την μέθοδο αυτή το σύστημα διαμορφώνει κατάλληλα το bolt για να μπορεί να επεξεργαστεί στη συνέχει τα δεδομένα. (Εικόνα 2.6)

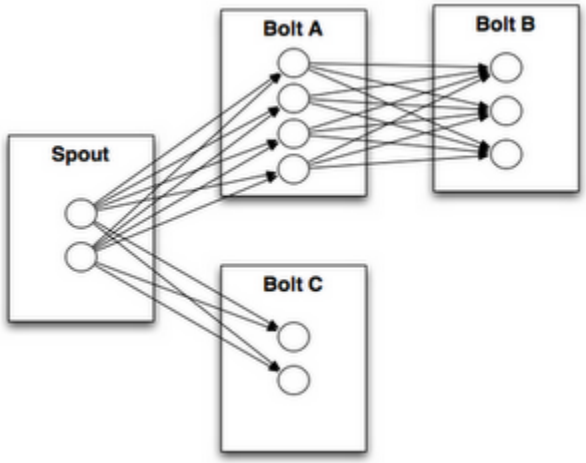
```
public class EmailCounter extends BaseBasicBolt {
  private Map<String, Integer> counts;

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    // This bolt does not emit anything and therefore does
    // not declare any output fields.
  }

  @Override
  public void prepare(Map config,
                      TopologyContext context) {
    counts = new HashMap<String, Integer>();
  }

  @Override
  public void execute(Tuple tuple,
                      BasicOutputCollector outputCollector) {
    String email = tuple.getStringByField("email");
    counts.put(email, countFor(email) + 1);
    printCounts();
  }

  private Integer countFor(String email) {
    Integer count = counts.get(email);
    return count == null ? 0 : count;
  }

  private void printCounts() {
    for (String email : counts.keySet()) {
      System.out.println(
        String.format("%s has count of %s", email, counts.get(email)));
    }
  }
}
```

2.6 Παράδειγμα κώδικα ενός τυπικού Bolt [1]

Κάθε spout ή bolt εκτελεί όσες εργασίες(tasks) θέλει ο χρήστης στο σύμπλεγμα. Κάθε εργασία αντιστοιχεί σε ένα νήμα εκτέλεσης, και οι ομαδοποιήσεις ροών καθορίζουν τον τρόπο που στέλνονται πλειάδες από ένα σύνολο εργασιών σε ένα άλλο. Μπορούμε να ορίσουμε τον παραλληλισμό για κάθε spout ή bolt με την χρήση ειδικών μεθόδων πολύ εύκολα.

2.3.3 Ροή(Stream)

Η ροές ίσως είναι η σημαντικότερη έννοια του Storm. Πρόκειται για μια απεριόριστη ακολουθία των πλειάδων που μπορεί να επεξεργαστούν παράλληλα. Κάθε ροή μπορεί να επεξεργάζεται από έναν ή πολλαπλούς τύπους bolt. Το Storm λοιπόν μπορεί επίσης να θεωρηθεί ως μια πλατφόρμα για να μετατρέπει ροές. Κάθε ροή έχει όνομα και αναγνωριστικό για να μπορούν τα bolts να λαμβάνουν είσοδο και να παράγουν έξοδο σε αυτές τις ροές. Στα διαγράμματα των τοπολογιών απεικονίζουμε τις ροές με βέλη.



2.7 Ροές δεδομένων σε ένα παράδειγμα τοπολογίας

2.3.4 Ομαδοποιήσεις Ροών

Μέρος του καθορισμού μιας τοπολογίας είναι ο προσδιορισμός για κάθε bolt ποία ρέματα θα πρέπει να λαμβάνουν ως είσοδο. Η ομαδοποίηση μιας ροής καθορίζει πώς αυτή η ροή θα πρέπει να κατανείμει τα δεδομένα μεταξύ των εργασιών(tasks) ενός bolt. Υπάρχουν οκτώ ομαδοποιήσεις ροών ενσωματωμένες στο Storm ενώ μπορούμε να υλοποιήσουμε την προσαρμοσμένη σε αυτά που θέλουμε να πετύχουμε, δικιά μας ομαδοποίηση ροής:

1. *Τυχαία(Shuffle) Ομαδοποίηση:* Οι πλειάδες τυχαία κατανέμονται μεταξύ των εργασιών ενός bolt με τέτοιο τρόπο, ώστε όλα τα bolt να δεχθούν ίσο αριθμό πλειάδων.

2. *Ομαδοποίηση Πεδίων(Fields):* Η ροή διαχωρίζεται βάση των πεδίων που έχουν προσδιοριστεί στην ομαδοποίηση. Αυτός ο τρόπος εγγυάται ότι ένα δεδομένο σύνολο τιμών των πεδίων πάντα θα στέλνεται στην ίδια εργασία του bolt.

3. *Ομαδοποίηση Μερικού Κλειδιού(Partial Key):* Η ροή διαχωρίζεται βάση των πεδίων που έχουν προσδιοριστεί στην ομαδοποίηση, όπως και στην

ομαδοποίηση πεδίων, αλλά ο φόρτος μοιράζεται στα bolt που έχουν είσοδο από αυτή τη ροή. Αυτός ο τρόπος παρέχει καλύτερη χρήση των πόρων.

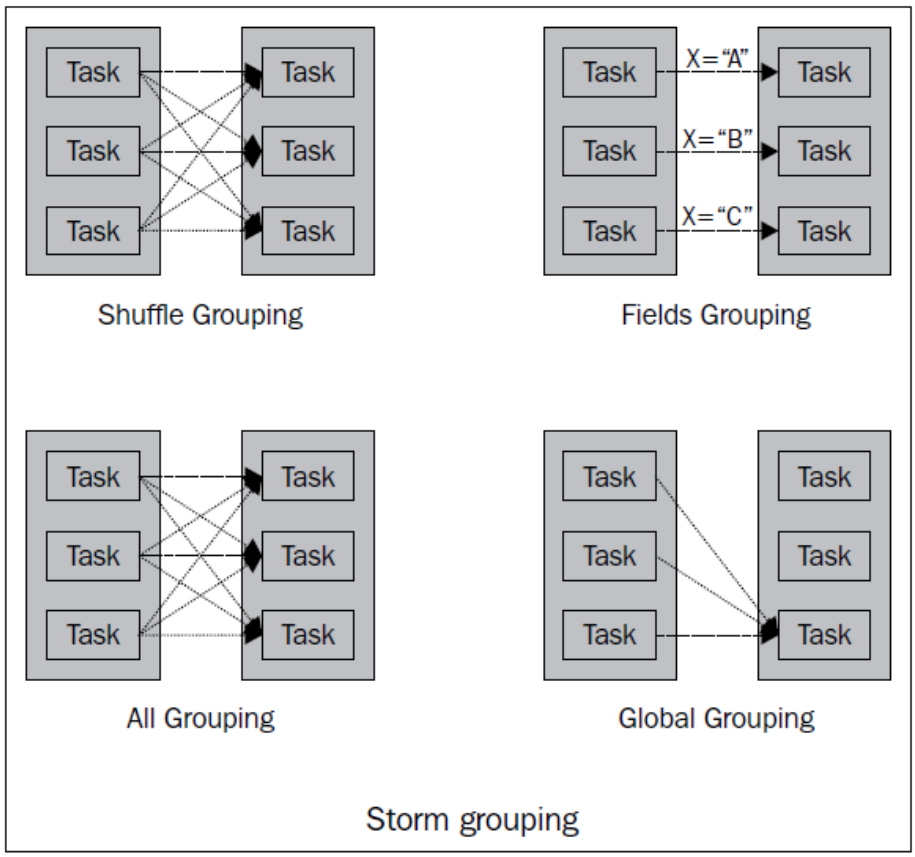
4. *Όλα(All) Ομαδοποίηση:* Η ροή επαναλαμβάνεται σε όλες τις εργασίες του bolt.

5. *Γενική(Global) Ομαδοποίηση:* Ολόκληρη η ροή πηγαίνει σε μια από τις εργασίες του bolt (συγκεκριμένα σε αυτή με το χαμηλότερο id).

6. *Καμιά(None) Ομαδοποίηση:* Η ομαδοποίηση αυτή δηλώνει ότι δεν μας ενδιαφέρει το πώς θα ομαδοποιηθεί η ροή. Επί του παρόντος, η καμιά ομαδοποίηση είναι ισοδύναμη με την τυχαία ομαδοποίηση.

7. *Στοχευμένη(Direct) Ομαδοποίηση*: Σε αυτή την ομαδοποίηση ο παραγωγός της πλειάδας αποφασίζει ποιο task του καταναλωτή θα λάβει αυτή τη πλειάδα.

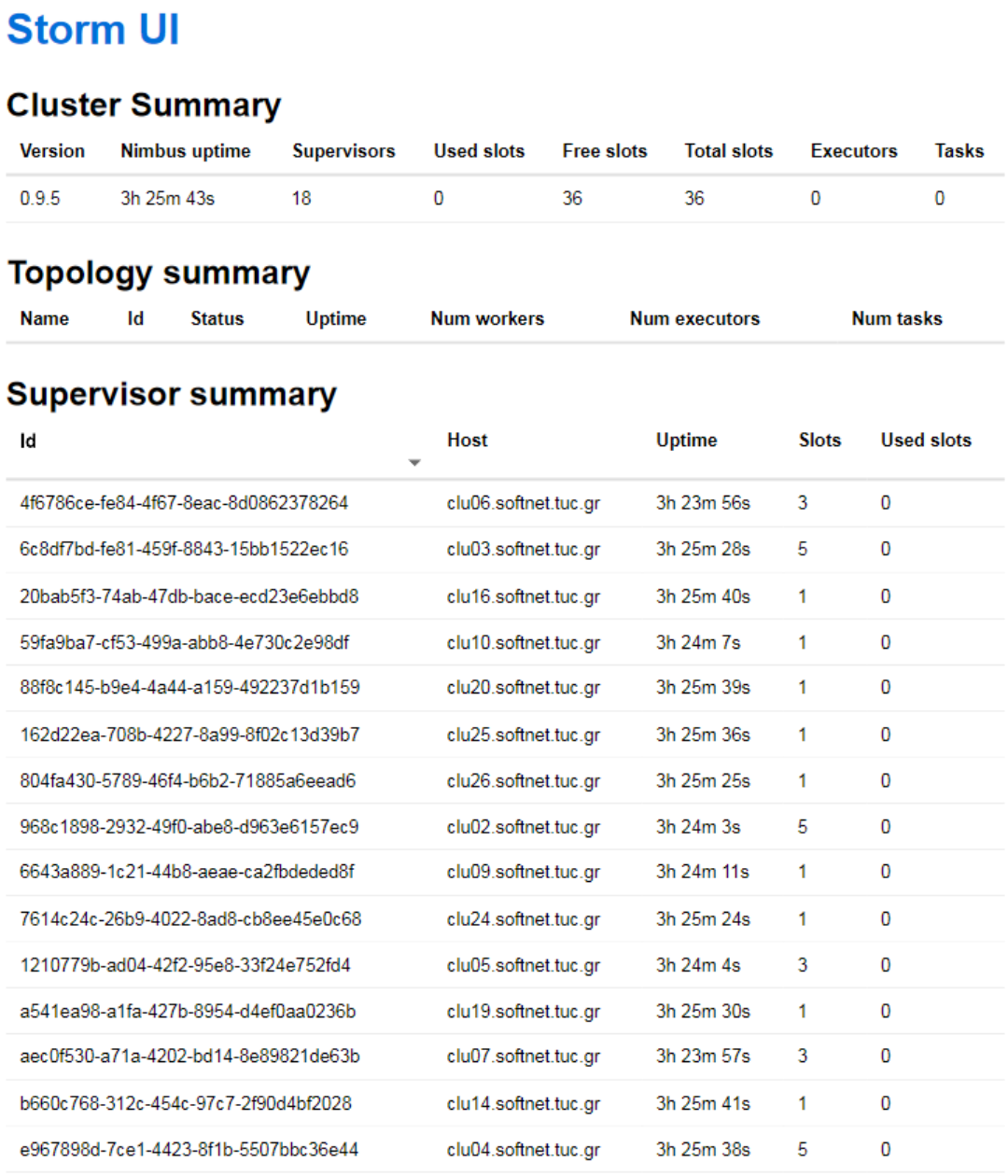
8. *Τοπική ή Τυχαία(Local or Shuffle) Ομαδοποιήση:* Αν ο παραγωγός και ο καταναλωτής της πλειάδας βρίσκονται στον ίδιο κόμβο worker τότε αυτή η ομαδοποίηση θα λειτουργήσει σαν τυχαία ομαδοποίηση μεταξύ των εργασιών που τρέχουν στον ίδιο worker κόμβο. Αλλιώς, αυτή η ομαδοποίηση λειτουργεί σαν μια κανονική τυχαία ομαδοποίηση.



2.8 Παράδειγμα μια συμβατικής τοπολογίας[2]

2.4 Διεπαφή Χρήστη Storm

Το Storm μέσω της διεπαφής χρήστη μας παρέχει τις απαραίτητες πληροφορίες για το σύμπλεγμα Storm. Με αυτό τον τρόπο μπορούμε να επιβλέπουμε την κατάσταση των διάφορων επιμέρους κομματιών που 'τρέχουν' στο σύμπλεγμα. Τα στατιστικά που συλλέγει η διεπαφή και με τις πληροφορίες που παρέχει ο διαχειριστής μπορεί να εντοπίσει λάθη, προβλήματα και περιορισμούς που υπάρχουν στο σύμπλεγμα.



Storm UI

Cluster Summary

Version	Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
0.9.5	3h 25m 43s	18	0	36	36	0	0

Topology summary

Name	Id	Status	Uptime	Num workers	Num executors	Num tasks

Supervisor summary

Id	Host	Uptime	Slots	Used slots
4f6786ce-fe84-4f67-8eac-8d0862378264	clu06.softnet.tuc.gr	3h 23m 56s	3	0
6c8df7bd-fe81-459f-8843-15bb1522ec16	clu03.softnet.tuc.gr	3h 25m 28s	5	0
20bab5f3-74ab-47db-bace-ecd23e6ebbd8	clu16.softnet.tuc.gr	3h 25m 40s	1	0
59fa9ba7-cf53-499a-abb8-4e730c2e98df	clu10.softnet.tuc.gr	3h 24m 7s	1	0
88f8c145-b9e4-4a44-a159-492237d1b159	clu20.softnet.tuc.gr	3h 25m 39s	1	0
162d22ea-708b-4227-8a99-8f02c13d39b7	clu25.softnet.tuc.gr	3h 25m 36s	1	0
804fa430-5789-46f4-b6b2-71885a6eead6	clu26.softnet.tuc.gr	3h 25m 25s	1	0
968c1898-2932-49f0-abe8-d963e6157ec9	clu02.softnet.tuc.gr	3h 24m 3s	5	0
6643a889-1c21-44b8-aeae-ca2fbdeded8f	clu09.softnet.tuc.gr	3h 24m 11s	1	0
7614c24c-26b9-4022-8ad8-cb8ee45e0c68	clu24.softnet.tuc.gr	3h 25m 24s	1	0
1210779b-ad04-42f2-95e8-33f24e752fd4	clu05.softnet.tuc.gr	3h 24m 4s	3	0
a541ea98-a1fa-427b-8954-d4ef0aa0236b	clu19.softnet.tuc.gr	3h 25m 30s	1	0
aec0f530-a71a-4202-bd14-8e89821de63b	clu07.softnet.tuc.gr	3h 23m 57s	3	0
b660c768-312c-454c-97c7-2f90d4bf2028	clu14.softnet.tuc.gr	3h 25m 41s	1	0
e967898d-7ce1-4423-8f1b-5507bbc36e44	clu04.softnet.tuc.gr	3h 25m 38s	5	0

2.9 Το Storm UI του cluster που έχει στηθεί στο Πολυτεχνείο Κρήτης

3. CEP

3.1. Δομικά Στοιχεία Εισόδου στο σύστημα CEP

Τα δύο δομικά στοιχεία που ορίζουν τους τύπους εισόδου και άρα την πληροφορία που λαμβάνει το σύστημα CEP είναι τα γεγονότα και τα ερωτήματα [4] :

3.1.1 Γεγονότα

Τα γεγονότα είναι το κυριότερο στοιχείο ενός συστήματος Επεξεργασίας Πολύπλοκων Γεγονότων. Υπάρχουν 2 βασικά είδη γεγονότων. Τα *πρωταρχικά γεγονότα* που είναι ήδη ορισμένα συμβάντα μοναδικού ενδιαφέροντος τα οποία δεν μπορούν να αναλυθούν σε απλούστερα/μικρότερα γεγονότα. Τα *σύνθετα γεγονότα* εντοπίζονται από το σύστημα μας , αποτελούνται από συνδυασμό πρωταρχικών γεγονότων και/ή από άλλα σύνθετα γεγονότα. Τα πρωταρχικά γεγονότα έρχονται από πολλές διαφορετικές πηγές σε αντίθεση με τα σύνθετα γεγονότα που δημιουργούνται εντός του συστήματος ΕΠΓ. Τα δεδομένα/γεγονότα (events) στο δικό μας σύστημα ορίζονται σαν πλειάδες με δύο συστατικά στοιχεία. Το πρώτο είναι τα ίδια χαρακτηριστικά που το ορίζουν και το δεύτερο η χρονική στιγμή που συνέβη το γεγονός. Τα χαρακτηριστικά μπορεί να είναι είτε πολλά είτε ένα μοναδικό και δίνουν το είδος, το νόημα και το χαρακτήρα του γεγονότος. Η χρονική στιγμή μπορεί να χαρακτηρίζεται είτε από χρονικό στιγμιότυπο που συνέβη το γεγονός είτε από το χρονικό στιγμιότυπο που σταμάτησε να συμβαίνει άμα έχει διάρκεια ή/ και την διάρκειά του.

3.1.2 Ερωτήματα

Τα ερωτήματα (queries) αποτελούν το δεύτερο τύπου εισόδου στο σύστημα μας και είναι τα μοτίβα που ουσιαστικά προσπαθούμε να ανιχνεύσουμε ανάμεσα στα γεγονότα των δεδομένων που φτάνουν στο σύστημα CEP. Τα ερωτήματα αυτά αποτελούνται από ένα ή παραπάνω τύπους γεγονότων (π.χ. A, B, C), τελεστέους. Επίσης περιέχουν ένα ή παραπάνω τύπους τελεστών και όλους τους συνδυασμούς αυτών. Οι κυριότεροι τύποι τελεστών που υλοποιεί ένα τυπικό σύστημα Επεξεργασίας Πολύπλοκων Γεγονότων είναι: SEQ (ακολουθία), AND (λογικό ΚΑΙ), Or (λογικό 'Η), NOT (λογικό ΟΧΙ), Kleene Closure (επανάληψη ενός γεγονότος μηδέν ή περισσότερες φορές). Στο ερώτημα είναι εξαιρετικά σημαντικό να οριστεί το ακριβές μοτίβο που αναζητείται από το σύστημα καθώς και το χρονικό περιθώριο (παράθυρο) μέσα στο οποίο πρέπει να εντοπιστεί στο σύνολο του το ερώτημα. Ο σωστός ορισμός των ερωτημάτων εγγυάται ότι θα υπάρχει

πάντοτε εντοπισμός των κρίσιμος και ενδιαφερόντων για το χρήστη καταστάσεων και δεν θα υπάρχουν περιπτώσει που θα ξεφεύγουν αυτοί από το σύστημα Εφόσον υπάρχει εντοπισμός του μοτίβου ανάμεσα στα γεγονότα το σύστημα επιστρέφει τη λίστα των πρωταρχικών γεγονότων που εντόπισε ή ένα εξαγώγιμο σύνθετο γεγονός.

Βασικός τρόπος πλήρους και δομημένης έκφρασης ερωτήματος, που χρησιμοποιείται σε πολλές εφαρμογές ΕΠΓ, είναι η γλώσσα SASE. Η γλώσσα αυτή στη πιο απλή μορφή της αποτελείται από τρία πεδία: το μοτίβο, τις επιμέρους συνθήκες και το χρονικό παράθυρο.

```
PATTERN SEQ(E a, E b, E c)
WHERE (a.ticker = MSFT) AND (b.ticker=GOOG)
AND (c.ticker = AAPL) AND (a.price < b.price) AND
(b.price<c.price)
WITHIN 4 hours
```

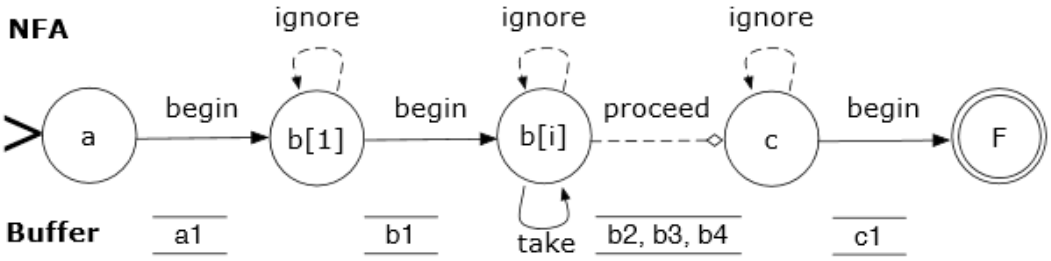
3.1 Παράδειγμα ενός ερωτήματος εκφρασμένου με τη γλώσσα SASE[6]

Στη δικιά μας εφαρμογή ΕΠΓ τα ερωτήματα είναι λιγότερο σύνθετα και για αυτό έχουμε επιλέξει ένα πιο απλό τρόπο. Παραμένει επαρκώς δομημένος παρόλαυτα για να μπορεί να επικοινωνεί σωστά με το spout το λαμβάνει και να μπορεί να αναλυθεί . Συγκεκριμένα ένα ερώτημα στη εφαρμογή μας έχει τη μορφή : '(((A AND B) AND (A OR C)) OR C), 2000'.

3.2 Εντοπισμός Δεδομένων

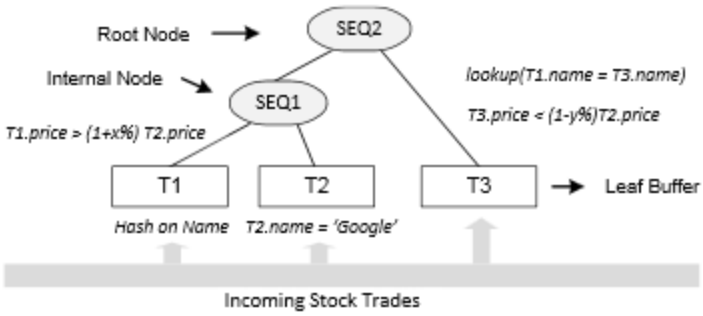
Καθώς εισέρχονται γεγονότα στο σύστημα CEP τα ερωτήματα που αναζητούμε εντοπίζονται σταδιακά. Πιο συγκεκριμένα στην περίπτωση που το συνολικό ερώτημα είναι σύνθετο τότε κατακερματίζεται σε υποερωτήματα μικρότερα τα οποία εντοπίζονται πρώτα και έπειτα όλα μαζί ικανοποιούν το συνολικό ερώτημα. Προϋπόθεση για να γίνει αυτό είναι σε κάθε υποερώτημα που το σύστημα εντοπίζει να γίνεται έλεγχος αν τυχών ικανοποιείται πλέον στο σύνολο του και το αρχικό ερώτημα. Για την αναπαράσταση αυτής της λογικής υπάρχουν δύο βασικοί τρόποι:

- Η αναπαράσταση με την χρήση ενός **μη-ντετερμινιστικού αυτόματου** στην οποία έχουμε μετάβαση κατάστασης όταν ένα γεγονός ικανοποιεί μερικό ή συνολικό εντοπισμό ερωτήματος.



3.2 Παράδειγμα αναπαράστασης ερωτήματος με μη-ντετερμινιστικό πεπερασμένο αυτόματο[12]

- Η αναπαράσταση με την χρήση **γράφου** κατά την οποία δεντρικές δομές απεικονίζουν ένα πολύπλοκο γεγονός. Σε αυτή την περίπτωση τα φύλλα του δένδρου είναι τα πρωταρχικά γεγονότα και οι ενδιάμεσοι κόμβοι οι τελεστές.



3.3 Παράδειγμα αναπαράστασης ερωτήματος με δέντρο[7]

Για τον εντοπισμό του ερωτήματος εδώ είναι καλό να αναφέρουμε μια επιλογή που κάναμε στα πλαίσια της διπλωματικής. Στην βιβλιογραφία υπάρχουν διάφοροι τρόποι εντοπισμού. Σε αυτή τη διπλωματική όταν αναζητούμε ένα ερώτημα επιλέγουμε τα γεγονότα που συνέβησαν πιο πρόσφατα χρονικά και ικανοποιούν το ερώτημα. Επίσης ένα γεγονός δεν μπορεί να χρησιμοποιηθεί για πολλαπλό εντοπισμό του ερωτήματος. υπάρχουν άλλες περιπτώσεις οι οποίες για τον εντοπισμό ενός ερωτήματος επιλέγουν γεγονότα που συνέβησαν πιο παλιά ή σε άλλες περιπτώσεις που τα γεγονότα έχουν διάρκεια και δεν συμβαίνουν στιγμιαία τότε μπορεί να επιλεγούν είτε τα γεγονότα που τελείωσαν πιο πρόσφατα είτε αυτά που ξεκίνησαν πιο πρόσφατα. Τέλος άλλες υλοποιήσεις αντίστοιχων συστημάτων, το ίδιο γεγονός μπορεί να χρησιμοποιηθεί για τον εντοπισμό του ίδιου ερωτήματος πάνω από μια φορά (πχ αν το ερώτημα είναι: A AND B και έχουμε με σειρά εισόδου τα γεγονότα: A, A, A, B τότε έχουμε 3 εντοπισμούς του ερωτήματος). Στο 5ο κεφάλαιο γίνεται εκτενής αναφορά και με παραδείγματα για το πως συγκεκριμένα γίνεται ο κατακερματισμός και ο εντοπισμός ενός ερωτήματος.

3.3 Μέτρηση απόδοσης

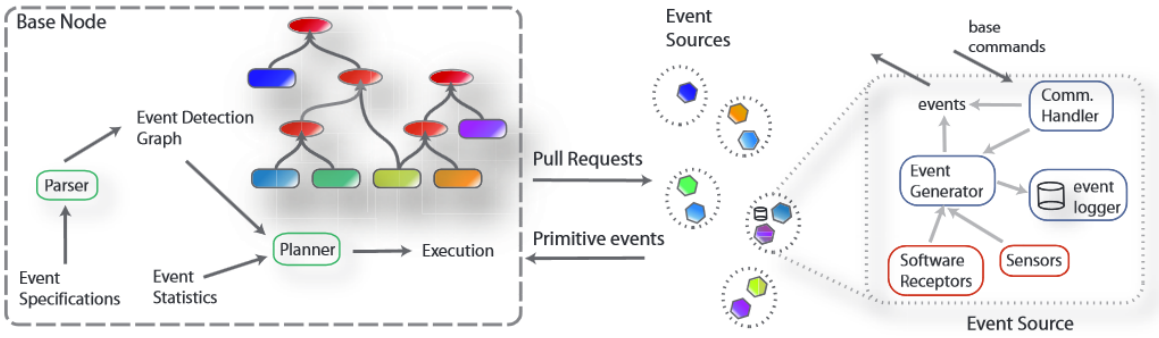
Τα συστήματα CEP κρίνονται ως προς την απόδοση τους βάση του πόσο γρήγορα μπορούν να επεξεργαστούν τα δεδομένα που υποδέχονται. Έτσι κύρια ένδειξη απόδοσης του συστήματος αποτελεί η ικανότητα διαβίβασης δεδομένων(throughput) δηλαδή το πόσα γεγονότα μπορούν έχουν επεξεργαστεί κάθε μονάδα χρόνου. Άλλες βασικές μετρήσεις επίδοσης είναι το κόστος της επεξεργαστικής ισχύς που χρειάζεται το σύστημα, το κόστος μετάδοσης (σε συστήματα που έχουμε πολλές ροές δεδομένων), η καθυστέρηση εντοπισμού του ερωτήματος (δηλαδή ο χρόνος που μεσολάβησε από την στιγμή που όντως συνέβη μέχρι που το σύστημα το εντόπισε), καθώς και η διαχείριση της μνήμης όταν υπάρχει ανάγκη επεξεργασίας πολλών ερωτημάτων με μεγάλα χρονικά παράθυρα.

3.4 Συναφείς εργασίες

Στο πεδίο της Επεξεργασίας Πολύπλοκων δεδομένων τα τελευταία χρόνια έχουν γίνει πολλές έρευνες και εργασίες. Πολλές από αυτές είναι ιδιαίτερα αξιόλογες και με μεγάλη συνεισφορά στον τομέα. Παρουσιάζουμε κάποιες από αυτές σε μια προσπάθεια εντοπισμού καινοτομιών που εισήγαγαν αυτές αλλά και για να μπορέσουμε να δούμε τι πέτυχαν παρεμφερείς δουλειές που έχουν γίνει στο παρελθόν.

Η δημοσίευση [5] παρουσιάζει μια νέα προσέγγιση βασισμένη στην εκπόνηση σχεδίου για επικοινωνιακά αποδοτικό Εντοπισμό Πολύπλοκων Γεγονότων κατά μήκος κατανεμημένων πηγών. Οι **κατανεμημένες πηγές** παράγουν απλά γεγονότα, τα οποία συνεχώς στέλνονται στον **κεντρικό κόμβο συντονιστή** όπου ελέγχονται αν ικανοποιούν κάποιο πολύπλοκο γεγονός. Η εργασία αυτή ασχολείται με την ορισμένη από το χρήστη/σύστημα **καθυστέρηση στον εντοπισμό πολύπλοκων γεγονότων και το έλεγχο του επικοινωνιακού κόστος μεταξύ των πηγών και του κόμβου συντονιστή** .

Με αυτά σαν κριτήρια και επειδή η εξέταση κάθε γεγονότος όταν συμβαίνει δεν είναι ούτε αποδοτική, αφού απαιτεί την επικοινωνία όλων των γεγονότων με το κόμβο συντονιστή, ούτε αναγκαία αφού μόνο ένα μικρό κομμάτι των γεγονότων θα γίνει τελικά σύνθετο γεγονός επιχειρείται μια προσπάθεια να ισορροπήσει το επικοινωνιακό κόστος με την καθυστέρηση εντοπισμού. Χρησιμοποιούν **FSM** για τον εντοπισμό των γεγονότων και γράφους για την μοντελοποίηση των ερωτημάτων. Ο γράφος εμπεριέχει όλα τα ερωτήματα και έχει σαν φύλλα-κόμβους τα πρωταρχικά γεγονότα και σαν ενδιάμεσους κόμβους τους τελεστές ή τα σύνθετα γεγονότα.



3.4 Το λειτουργικό πλαίσιο με βάση το οποίο γίνεται ο Εντοπισμός Πολύπλοκων Γεγονότων[5]

Η δημοσίευση αυτή προτείνει λοιπόν τη δημιουργία σχεδίων πολλών βημάτων, με βάση τους χρονικούς περιορισμούς μεταξύ των γεγονότων που ενδιαφέρουν ένα ερώτημα, των στατιστικών συχνότητας των γεγονότων και της πολλαπλής συμμετοχής των γεγονότων σε διαφορετικά ερωτήματα. Τα σχέδια μετέπειτα κατανέμονται στις πηγές και μέσα από μηνύματα push/pull ο κόμβος συντονιστής εντοπίζει τα πολύπλοκα γεγονότα.

Ανάλογα τον αριθμό των σταδίων της FSM, μπορούν να εντοπίσουν κάθε γεγονός που ανήκει σε ένα ερώτημα. Ο εντοπισμός του πρωταρχικού γεγονότος που αντιστοιχεί σε μια

κατάσταση ενεργοποιεί την επόμενη κατάσταση της FSM και άρα τον έλεγχο νέων γεγονότων. Οι σειρά των καταστάσεων ορίζονται με σκεπτικό ότι η επεξεργασία των 'συχνών' γεγονότων στην περίπτωση που έχει ήδη συμβεί το πιο 'σπάνιο' μειώνει το κόστος επικοινωνίας σε αντάλλαγμα με την αύξηση της καθυστέρησης εντοπισμού του γεγονότος. Άρα προηγούνται τα σπάνια και έπειτα τα περισσότερο συχνά γεγονότα.

Μια ακόμα σημαντική εργασία [6] επικεντρώνεται στο πως θα λύσει το ζήτημα του να μην εξετάζουμε ολόκληρο το πλήθος των γεγονότων κατά την άφιξη τους στο σύστημα διότι όπως προείπαμε είναι μη αποδοτικό. Η συγκεκριμένη δημοσίευση αναφέρεται μόνο στην περίπτωση που αναζητούμε ακολουθία (sequence) και όχι άλλες λογικές πράξεις γεγονότων. Είναι επίσης φτιαγμένη και δοκιμασμένη σε ένα **κεντρικό υπολογιστή** και όχι σε κατανεμημένο σύστημα ούτε κάποια συστάδα (cluster).

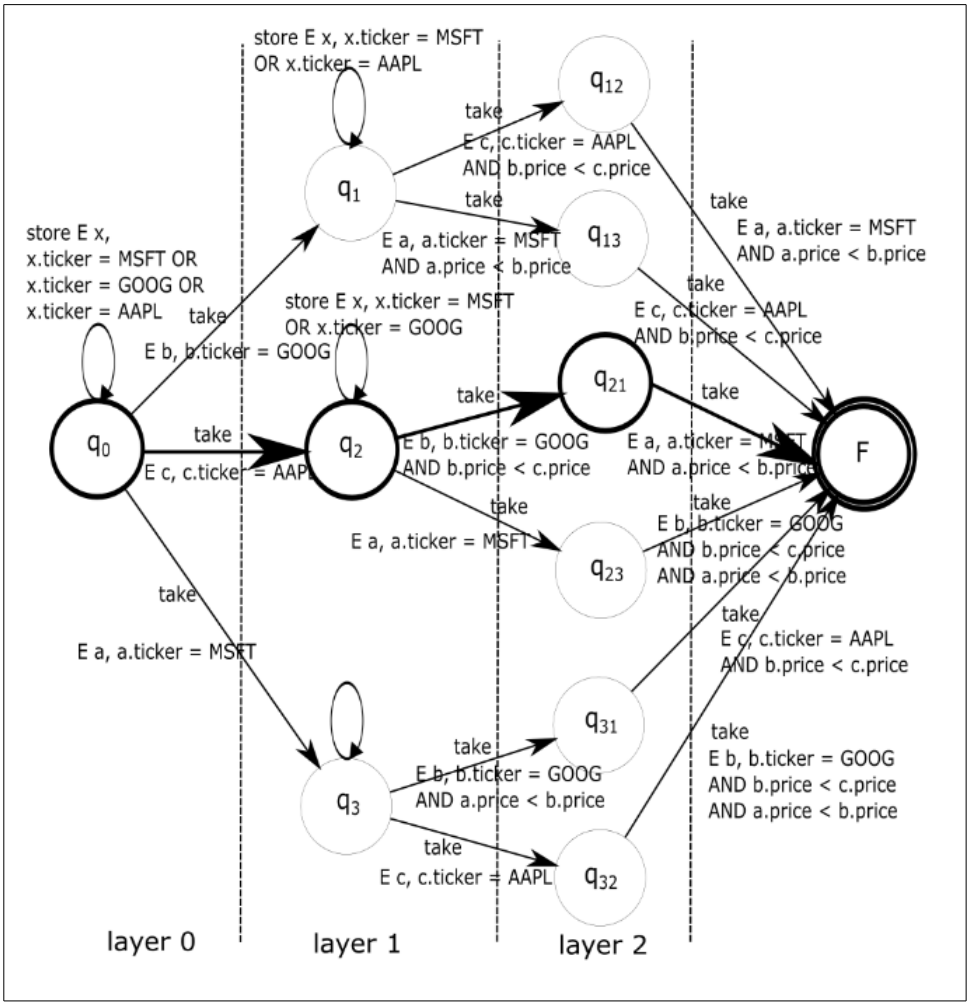
Υλοποιούν λοιπόν για να ξεπεράσουν το πρόβλημα μη-αποδοτικότητας του "ανυπόμονου αλγορίθμου" όπως τον αποκαλούν ένα δικό τους αλγόριθμο πού επεξεργάζεται τα γεγονότα με φθίνουσα σειρά σπανιότητας. Ο προτεινόμενος μηχανισμός ξεκινάει τον εντοπισμό από το πιο σπάνιο γεγονός και όχι από το πρώτο στη σειρά κ.ο.κ. Αυτό έχει σαν αποτέλεσμα να μειώνει το μέγεθος μνήμης που χρειαζόμαστε αλλά και το να μειώνει τον χρόνο επεξεργασίας. **Πιο συγκεκριμένα ο μηχανισμός αυτός είτε επεξεργάζεται ένα γεγονός κατά την άφιξη του είτε το αποθηκεύει σε ένα buffer, που αναφέρεται σαν "input buffer", για να επεξεργαστεί αργότερα αν χρειάζεται.** Φτιάχνουν μετέπειτα 2 μεθόδους εντοπισμού.

Ο πρώτος είναι το **"μη ντετερμινιστικό αυτόματο σε σειρά"** *(εικόνα 3.2)*. Περιέχει ν+1 βήματα με κάθε ν βήμα να είναι και ο εντοπισμός ενός πρωταρχικού γεγονότος και το τελευταίο το βήμα αποδοχής των προηγούμενων. Τα βήματα είναι με φθίνουσα σειρά σπανιότητας. Σε αυτή τη μέθοδο βέβαια υπάρχουν κάποιες προβληματικές. Πρώτον δεν είναι πάντοτε εύκολο σε πραγματικές συνθήκες να ξέρουμε από πριν τη σπανιότητα ενός γεγονότος και δεύτερον και να την ξέρουμε δεν μας εγγυάται κανείς ότι με τη πάροδο του χρόνου δεν θα αλλάξει και τελικά η σειρά στα βήματα θα καταλήξει να είναι λανθασμένη.

Για να ξεπεράσουμε το πρόβλημα αυτό και να μην εξαρτόμαστε από μια και μοναδική σειρά σπανιότητας δημιούργησαν τη δεύτερη μέθοδο. Αυτή ονομάζεται **"δέντρο μη ντετερμινιστικό αυτόματο"** όπου αποφασίζεται η σειρά βάση της κάθε στιγμής σπανιότητας ενός γεγονότος. Χρησιμοποιούν λοιπόν τον "input buffer" για να μετράνε τα γεγονότα που έφτασαν από τη ροή δεδομένων σε ένα ορισμένο χρονικό περιθώριο. Για κάθε νέο γεγονός λοιπόν αυξάνεται ο μετρητής αυτού του γεγονότος και κάθε φορά που αυτό το γεγονός αφαιρείται (συνεισφέρει σε εντοπισμό πολύπλοκου γεγονότος) μειώνεται. Έτσι αφού εισαχθεί τουλάχιστον ένα στοιχείο για κάθε γεγονός, ελέγχουν ποιο είναι το πλέον σπάνιο, ξεκινούν από αυτό και μετέπειτα μεταβαίνουν κατάσταση κοκ.

Οι δοκιμές τους έδειξαν ότι και οι δυο μέθοδοι αποτελούν βελτίωση του απλοϊκού και μη αποδοτικού "ανυπόμονου" αλγορίθμου και σε απόδοση και σε κατανάλωση μνήμης. Συγκριτικά οι δυο μέθοδοι που προτείνουν για περιπτώσεις που κατά την διάρκεια της δοκιμής οι σπανιότητα των γεγονότων αλλάζει αποδείχτηκε ότι το "δέντρο μη

ντετερμινιστικό αυτόματο" που έχει τη δυνατότητα να προσαρμόζεται είναι σταθερά καλύτερο από τη μέθοδο "μη ντετερμινιστικό αυτόματο σε σειρά"



3.5 Η μέθοδος δένδρο μη-ντετερμινιστικό αυτόματο για αποδοτικότερη διαχείριση μνήμης[6]

Στο [7] υλοποιούν μια ακόμα εφαρμογή εντοπισμού σύνθετων γεγονότων. Σημαντικές συνεισφορές της δουλείας αυτής είναι η **προσαρμοστικότητα του συστήματος** κατά τη διάρκεια λειτουργίας του και η **μείωση του κόστους σε μνήμη**.

Για τη μοντελοποίηση του εντοπισμού των σύνθετων γεγονότων χρησιμοποιήθηκαν δεντρικές δομές. Η **δενδρική δομή** φαίνεται χρήσιμη εδώ γιατί ανάλογα προς τα πού αναπτύσσεται το δένδρο (δλδ από τα δεξιά, από τα αριστερά κ.α.) αλλάζει και η απόδοση του συστήματος για να μπορεί να προσαρμοστεί σε νέες συνθήκες *(εικόνα 3.3)*.

Για κάθε κόμβο χρησιμοποιούνται buffers για να αποθηκεύσουμε είτε τα πρωταρχικά γεγονότα (στα φύλλα της δενδρικής δομής) είτε τα σύνθετα γεγονότα (στους ενδιάμεσους κόμβους). Έτσι όπως εισέρχονται τα πρωταρχικά γεγονότα και αποθηκεύονται στους buffers, όταν γίνει ένας μερικός εντοπισμός τότε ελέγχονται τα αποθηκευμένα γεγονότα στις ανώτερα επίπεδα του δένδρου ώστε να απομακρυνθούν γεγονότα που δεν

συμβαδίζουν με το ήδη εντοπισμένο. Buffers δημιουργούνται όχι απαραίτητα σε όλα τα φύλλα των κόμβων. Για παράδειγμα άμα είναι AND ο τελεστής τότε δημιουργούνται σε όλα τα φύλλα ενώ για το SEQ αποθηκεύεται το πρώτο γεγονός που αφορά την ακολουθία.

Στη συγκεκριμένη εργασία έχει γίνει και ανασύνταξη ερωτημάτων με σκοπό το μικρότερο υπολογιστικό κόστος και άρα τη βελτίωση της απόδοσης. Βάση λοιπόν αλγεβρικών αλλαγών στα ερωτήματα αλλάζουν τελεστές με μεγάλο κόστος όπως το AND ή το SEQ με το OR κοκ. Έτσι η έκφραση SEQ(A, AND(!B, !C), D) μετασχηματίζεται σε SEQ(A, ! OR(B, C), D).

Πέρα από αυτή τη βελτίωση η εργασία έχει επιτύχει όπως προαναφέραμε την ικανότητα να προσαρμόζεται. Συγκεκριμένα κατά την διάρκεια που λειτουργεί το σύστημα και παρόλο που μπορεί να έχει επιλεγεί το κατάλληλο πλάνο στην αρχή λειτουργίας του, επειδή οι πιθανότητες να αλλάξουν οι τιμές και οι πιθανότητες με τις οποίες φθάνουν τα γεγονότα το αρχικό πλάνο δεν είναι πλέον το κατάλληλο και πρέπει να αντικατασταθεί. Για αυτό κρατιούνται στατιστικά για κάθε τελεστή και γεγονός με αποτέλεσμα όταν αυτά ξεπεράσουν ένα ορισμένο όριο που μπορούν να διαφοροποιούνται από τις αρχικές τους τιμές να γίνεται αλλαγή του πλάνου εφόσον αυτό εκτιμάται από τον αλγόριθμο ότι θα φέρει καλύτερα αποτελέσματα. Η αλλαγή γίνεται εκείνη τη στιγμή χωρίς να χάνονται δεδομένα. Τα ενδιάμεσα αποτελέσματα ανακτούνται όταν εγκατασταθεί το νέο πλάνο από τους buffers των φύλλων του δέντρου. Με αυτή τη τεχνική επιτυγχάνεται σημαντική βελτίωση στην απόδοση του συστήματος.

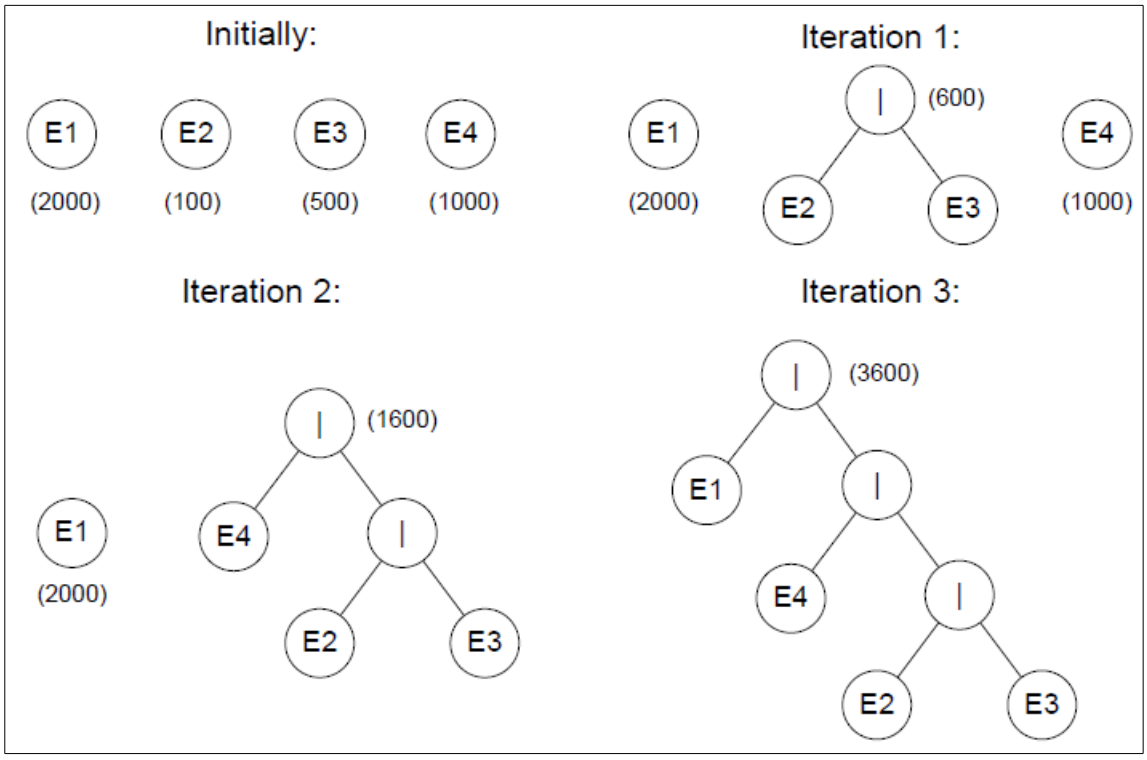
Στο [8] επιχειρείται η κατασκευή ενός κατανεμημένου **ΕΠΔ συστήματος σε συστάδα** (cluster). Για τη μοντελοποίηση του εντοπισμού των πολύπλοκων γεγονότων στην εργασία αυτή χρησιμοποίησαν **FSM**. Κάθε μετάβαση σε νέα κατάσταση σημαίνει και τον εντοπισμό μερικού ή ολόκληρου του πολύπλοκου γεγονότος.

Σημαντική, ερευνητικά, παρέμβαση αυτής της εργασίας είναι η ανασύνταξη του ερωτήματος με πιο αποδοτικό τρόπο. Το ερώτημα "ξαναγράφεται", με προσοχή ώστε το τελικό αποτέλεσμα να είναι το ίδιο και με σκοπό να επιτευχθούν καλύτερες αποδόσεις από το σύστημα. Συγκεκριμένα μοτίβα με τελεστές ΚΑΙ, ΑΚΟΛΟΥΘΙΑ, ΟΧΙ μετασχηματίζονται σε νέα αντίστοιχα με τα αρχικά αλλά με μικρότερο επεξεργαστικό κόστος. Γνωρίζοντας τη συχνότητα εμφάνισης ενός γεγονότος κάθε στιγμή βρίσκεται, από τον αλγόριθμο που τρέχει κεντρικά σε ένα κόμβο, η βέλτιστη διαδρομή εντοπισμού.

Για την επεξεργασία των δεδομένων η [8] προτείνει την **κεντρική παρακολούθηση των ροών δεδομένων και την κατανεμημένη(παράλληλοποιήμενη) επεξεργασία των πολύπλοκων γεγονότων**. Η ιδέα της κατανεμημένης επεξεργασίας του ερωτήματος κατά μήκος πολλών κόμβων βελτιώνει δραστικά την επίδοση του συστήματος στο ζήτημα της διεκπεραιωτικής ικανότητας του(throughput). Έτσι πολλές πηγές στέλνουν γεγονότα στο σύστημα και αυτά πάνε μέσω ενός κεντρικού κόμβου στους πολλούς επεξεργαστικούς κόμβους. Ο κεντρικός κόμβος δεν κάνει κάποια επεξεργασία στα εισερχόμενα γεγονότα αλλά είναι υπεύθυνος για την βελτιστοποίηση των ερωτημάτων που μιλήσαμε πριν. Ο κεντρικός κόμβος είναι επίσης υπεύθυνος για το πλάνο με το οποίο οι τελεστές θα πάνε

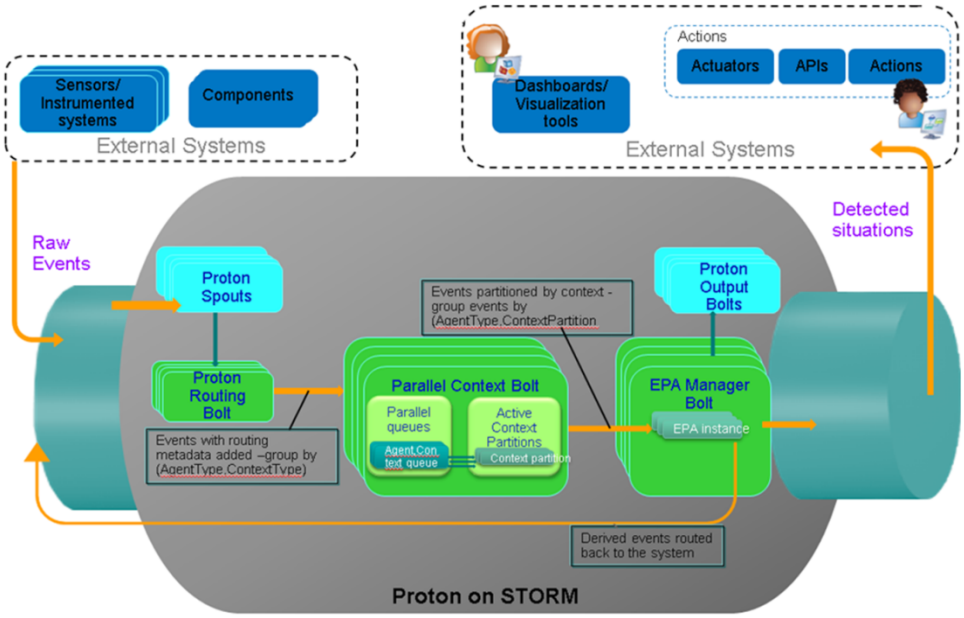
στους κατάλληλους κόμβος. Έτσι ο κεντρικός κόμβος ψάχνει ποιος από πλευράς κόστους είναι ο πιο αποδοτικός κόμβος για να εισαχθεί ένας τελεστής.

Η κατανεμημένη επεξεργασία των γεγονότων ελαττώνει επίσης το πρόβλημα της διαχείρισης της μνήμης και της διεκπεραιωτικής ικανότητας του συστήματος όπως είπαμε και πριν. Εκμεταλλευόμενοι λοιπόν την ανασύνταξη των ερωτημάτων τελεστές αναθέτονται σε επεξεργαστικές μονάδες ενώ ταυτόχρονα αυτοί μπορούν να στείλουν μεταξύ του ίδιου κόμβου ή και άλλων για να επιτύχουν τον πλήρη εντοπισμό ενός πολύπλοκου γεγονότος. Αρνητικό αυτής της λογικής παραλληλοποιήσης είναι ότι δεν επιδέχεται αναδιαμόρφωση και είναι επιρρεπείς σε ανισορροπίες του φόρτου εργασίας κατά μήκος των επεξεργαστικών κόμβων.



3.6 Παράδειγμα του αλγορίθμου ανασύνταξης ερωτήματος βάση κόστους που υλοποιείται στο [8]

Τέλος οφείλουμε να αναφερθούμε στην σημαντική και πλήρως ολοκληρωμένη δουλεία που έχει κάνει η IBM με την εφαρμογή της “Proton(Proactive Technology Online) [9] on Storm”. Αποτελεί μια έρευνα ανοιχτού κώδικα. Όταν λαμβάνει ένα γεγονός τότε αρχικά ερευνάται σε ποιο ερώτημα μπορεί να συμμετέχει. Έπειτα το γεγονός πάει στις ουρές παραλληλοποίησης που φροντίζουν για το αν μπορεί και πως θα γίνει με σωστή σειρά η παράλληλη επεξεργασία του γεγονότος που μπορεί να απασχολεί παραπάνω από ένα ερώτημα. Αφού εισαχθεί στις ουρές αποφασίζεται αν αυτό το γεγονός εκκινεί ή τερματίζει αναζητήσεις αλλά και με ποια άλλα γεγονότα θα ομαδοποιηθεί για να επεξεργαστεί μαζί. Τέλος το γεγονός πηγαίνει στον επεξεργαστικό κόμβο όπου γίνεται η επεξεργασία του και ερευνάται αν είναι σχετικό με το ερώτημα και στέλνεται το παραγόμενο γεγονός.



3.7 Η αρχιτεκτονική του Proton on STORM [9]

4. Τοπολογία Επεξεργασίας Πολύπλοκων Γεγονότων

4.1 Πορεία προς τη σωστή τοπολογία

Μέχρι να καταλήξουμε στην τελική τοπολογία την οποία και υλοποιήσαμε δοκιμάσαμε και εξετάσαμε διάφορες άλλες λύσεις. Οι ελλείψεις τους, οι αδυναμίες τους και τα προβλήματα που προέκυπταν από αυτές μας οδήγησαν σε αυτή που θα περιγράψουμε παρακάτω. Προς το παρών ας δούμε τη πορεία μέχρι εκεί.

Αρχικά θεωρήσαμε ως κομβικό και δομικό στοιχείο της εφαρμογής μας τα ερωτήματα (queries). Με αυτή την παραδοχή και με δεδομένη την μη εξοικείωση μας με το storm σαν εργαλείο και τα ιδιαίτερα χαρακτηριστικά του, ξεκινήσαμε αφελώς να λέμε ότι χρειάζεται κάθε task (του δομικού ως προς τον εντοπισμό bolt) να αναλαμβάνει τον εντοπισμό ενός ολόκληρου ερωτήματος. Κάτι τέτοιο όμως θα είχε μεγάλο υπολογιστικό κόστος, θα είχαμε σπατάλη πολλών πόρων γιατί θα έπρεπε να εξετάζουν ολόκληρο το ερώτημα ανεξάρτητα την πολυπλοκότητα του, αλλά και ουσιαστικά καμία διευκόλυνση αφού κάθε event θα έπρεπε να σταλθεί σε πλήθος από ερωτήματα άρα και σε πλήθος από task.

Προσπαθώντας να αλλάξουμε το κέντρο της σκέψης μας όσων αναφορά το ερώτημα και την μετατόπιση αυτού στα γεγονότα (events) σκεφτήκαμε μια υλοποίηση που κάθε task του κεντρικού bolt επεξεργασίας θα αναλαμβάνει κάποιους συγκεκριμένους τύπους γεγονότων. Εκεί λοιπόν για κάθε είσοδο/γεγονός που απασχολεί ένα ερώτημα στέλνεται ποιο γεγονός έφτασε σαν είσοδο και ποιο ερώτημα απασχολεί σε ένα επόμενο Bolt το οποίο γνωρίζει όλα τα ερωτήματα και στη ουσία αυτό θα εξετάζει αν υπάρχει εντοπισμός του ερωτήματος ή όχι. Και αυτό όμως ήταν σπατάλη πόρων καθώς στην ουσία μεταβιβάζεται ολόκληρος ο φόρτος σε επόμενο bolt το οποίο έχει πάλι το πρόβλημα του μεγάλου υπολογιστικού κόστους.

Αντιληφθήκαμε λοιπόν την ανάγκη να "σπάσει" το σύνθετο και πολύπλοκο ερώτημα σε μικρότερης δυσκολίας υποερωτήματα. Για την ακρίβεια κατακερματίζουμε το ερώτημα σε απλές λογικές πράξεις με ένα τελεστή κάθε φορά. Αυτό θα ήταν αποκλειστική ευθύνη ενός Bolt που παρενέβαινε μεταξύ του spout που παίρνει τα ερωτήματα και του βασικού bolt εντοπισμού.

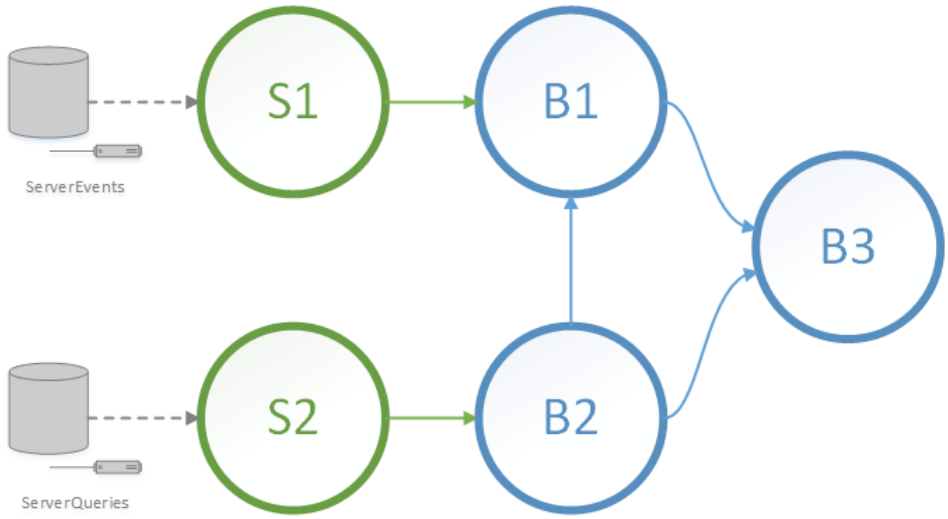
Σκεφτήκαμε έπειτα μια τοπολογία που το βασικό Bolt επεξεργασίας θα έχει τόσα task όσο το μέγιστο μέγεθος πολυπλοκότητας. Έτσι όταν εντοπιζόταν ένα υποερώτημα (μια λογική πράξη με ένα τελεστέο) τότε θα ενημέρωνε αυτό το task το επόμενο και εκείνο με την σειρά του είτε θα ενημέρωνε για τον συνολικό εντοπισμό είτε θα συνέχιζε την δουλειά του. Αυτό όμως έβαζε απόλυτα όρια στην πολυπλοκότητα των ερωτημάτων και στην ουσία χώριζε σε στάδια-βήματα με ένα γραμμικό τρόπο τον εντοπισμό ενός ερωτήματος ενώ όπως έχουμε αναφέρει και πιο πριν σαν απεικόνιση έχουν περισσότερο την μορφή δενδρικής δομής/γράφου ή αυτόματου.

Τέλος για να ταιριάζει με τη μορφή αυτόματου σκεφτήκαμε την αντιστοίχηση κάθε task με κάποια συγκεκριμένα γεγονότα ενώ ταυτόχρονα κάθε task θα ξέρει ποιο υποερωτήματα απασχολεί αυτό το γεγονός, ποιο event πρέπει να εντοπιστεί στην συνέχεια (και άρα ποιο task θα συνεχίσει τη αναζήτηση του ερωτήματος) καθώς και ποια event προηγήθηκαν. Κάτι τέτοιο θα απαιτούσε όμως μεγάλο όγκο δεδομένων γιατί κρατάμε αρχείο με το τι γεγονός ήρθε πριν. Επίσης πρόκειται για στρεβλή αντίληψη του αυτόματου αφού για μια τιμή εισόδου μπορεί να έχουμε ποικίλες πιθανές συνέχειες επίλυσης του αυτομάτου. Επίσης δεν είναι γραμμικός ο εντοπισμός του ερωτήματος σαν να πρόκειται για ένα μεγάλο Sequence γεγονότων αλλά αντίθετα εμπεριέχονται στα ερωτήματα και λογικές πράξεις όπως AND, OR που δεν εξυπηρετεί καθόλου αυτή η λογική.

4.2. Τελική τοπολογία

4.2.1 Γενική περιγραφή

Η τοπολογία που δημιουργήσαμε για την επεξεργασία πολύπλοκων γεγονότων περιλαμβάνει δύο spouts που διαβάζουν τις τιμές εισόδου και τις στέλνουν για επεξεργασία στα bolts που ακολουθούν. Το πρώτο spout με την ονομασία ‘S1’ διαχειρίζεται τις εισόδους για τα νέα γεγονότα που φθάνουν στο σύστημα . Το άλλο spout με όνομα ‘S2’ λαμβάνει τα ερωτήματα που υποβάλλονται για εντοπισμό. Το ‘S2’ στέλνει το ερώτημα στο bolt ‘B2’ και το ‘S1’ τα γεγονότα στο bolt ‘B1’ αντίστοιχα. Στο B2 “σπάμε” το ερώτημα σε υποερωτήματα και ενημερώνουμε το B3 για αυτά καθώς και το B1 για το ποια γεγονότα ενδιαφέρουν ποια ερωτήματα. Και τα δύο αυτά bolt στέλνουν τα επεξεργασμένα δεδομένα τους στο bolt ‘B3’ που είναι υπεύθυνο για τον εντοπισμό των ερωτημάτων ή υποσυστημάτων αυτών.



5.1 Αφαιρετικό σχέδιο της τελικής μας τοπολογίας

4.2.2 Τα Spout

Η τοπολογία μας αποτελείται από δυο είδη spout που είναι υπεύθυνα για την υποδοχή των ροών εισόδου. Τα spouts διαβάζουν τις ροές εισόδου που μπορεί να έρχονται από διαφορετικές πηγές (αρχείο, sockets κ.α.). Στη δικιά μας τοπολογία τα δεδομένα μας έρχονται μέσω socket. Πιο συγκεκριμένα:

S1

Το 'S1' είναι το spout που δέχεται τα νέα γεγονότα. Τα δεδομένα φτάνουν στο 'S1' με τη μορφή ενός αλφαριθμητικού στοιχείου που ορίζει το είδος του γεγονότος. Κάθε ένα αλφαριθμητικό στοιχείο είναι ένα γεγονός. Για κάθε γεγονός που φτάνει στο σύστημα το συνδέουμε με την χρονική στιγμή που το λάβαμε και δημιουργούμε για καθένα μια πλειάδα (tuple) με δύο τιμές. Πρώτη τιμή είναι το αλφαριθμητικό χαρακτηριστικό στοιχείο που ορίζει το γεγονός και δεύτερη η χρονική στιγμή (timestamp) που συνέβη το γεγονός. Η πλειάδα μετέπειτα αποστέλλεται στο bolt 'B1'.

S2

Το δεύτερο spout 'S2' είναι υπεύθυνο για την διαχείριση της εισόδου των ερωτημάτων που ψάχνει να εντοπίσει το σύστημα μας. Τα δεδομένα που έρχονται στο 'S2' αποτελούνται από ένα μεγάλο αλφαριθμητικό στοιχείο που μέσα σε αυτό έχει οριστεί από τον χρήστη: αν το ερώτημα είναι για εισαγωγή είτε για διαγραφή από το σύστημα, τη μοναδική ταυτότητα αναγνώρισης του (ID), την πηγή που μας έστειλε αυτά τα δεδομένα, το μοτίβο των γεγονότων που αναζητούμε (το κυριότερο χαρακτηριστικό του ερωτήματος) και το χρονικό περιθώριο μέσα στο οποίο θα αναζητούμε το συγκεκριμένο μοτίβο. Αυτά τα πέντε χαρακτηριστικά του ερωτήματος τα διαχωρίζουμε και τα περνάμε σαν τιμές μιας πλειάδας. Κάθε ερώτημα/πλειάδα στέλνεται μετά στο bolt 'B2'.

4.2.3 Τα Bolt

Για την επεξεργασία των δεδομένων μας από την τοπολογία χρειαστήκαμε να δημιουργήσουμε 3 διαφορετικά bolts καθένα από τα οποία αναλαμβάνει να κάνει ένα διακριτό και αυτοτελές κομμάτι της επεξεργασίας των δεδομένων που στέλνονται από τα spouts.

B1

Το 'B1' παίρνει ως είσοδο τις πλειάδες που του στέλνει το 'S1' δηλαδή τα γεγονότα. Την ίδια στιγμή παίρνει και είσοδο από το bolt 'B2' (βλ. αναλυτικότερα παρακάτω) που του στέλνει ποιο πρωταρχικό γεγονός συνδέεται με ποια ερωτήματα ή υποερωτήματα εφόσον μιλάμε για πολύπλοκο ερώτημα που είναι και το πλέον σύνηθες. Με αυτό τον τρόπο το 'B1' γνωρίζει για κάθε πρωταρχικό γεγονός που λαμβάνει από το 'S1' σε ποιο ερώτημα αντιστοιχεί. Έτσι κάθε νέο γεγονός που δέχεται από το 'S1' μαζί με το χρονικό του αποτύπωμα το στέλνει σε όλα τα ερωτήματα που ψάχνουν για αυτό το γεγονός. Ομαδοποιεί λοιπόν σε μια πλειάδα το γεγονός, τη χρονική στιγμή που αυτό συνέβη και το ερώτημα/υποερώτημα που αντιστοιχεί και το στέλνει στο 'B3'.

B2

Το 'B2' λαμβάνει για είσοδο τις πλειάδες που του στέλνει το 'S2'. Παίρνει δηλαδή τα ερωτήματα και σκοπός του είναι να τα κατακερματίσει σε μικρότερα υποερωτήματα με σκοπό να μειωθεί η πολυπλοκότητα αυτών και έπειτα να ενημερώσει τα άλλα δυο bolt. Παίρνει λοιπόν το μοτίβο που είναι προς αναζήτηση και με έναν εσωτερικό αναλυτή (parser) σπάει ένα πολύπλοκο ερώτημα σε απλά υποερωτήματα. Σε καθένα από αυτά δίνει μια μοναδική ταυτότητα αναγνώρισης. Δημιουργεί επί της ουσίας μια δενδρική δομή που αναπαριστά το ερώτημα. Σε αυτή την αναπαράσταση κάποια υποερωτήματα αφορούν πρωταρχικά γεγονότα, κάποια άλλα αφορούν σύνθετα γεγονότα (δηλαδή το αποτέλεσμα προηγούμενου σε σειρά προτεραιότητας ερώτημα) και κάποια και τα δυο είδη. Για κάθε υποερώτημα που δημιουργεί το 'B2' ενημερώνει εφόσον αφορά πρωταρχικά γεγονότα το 'B1' δίνοντας του την αντιστοιχία του γεγονότος με το υποερώτημα (πιο συγκεκριμένα με τη μοναδική ταυτότητα αναγνώρισης αυτού). Στέλνει έτσι μια πλειάδα με το γεγονός/τελεστέος, τη πράξη που αφορά, και το αν πρόκειται για εισαγωγή ή διαγραφή. Ταυτόχρονα ενημερώνει και το 'B3' για το υποερώτημα που δημιούργησε. Το 'B2' έχει επίσης την ευθύνη να ελέγχει αν υπάρχουν επαναλήψεις όμοιων υποερωτημάτων ανάμεσα στο πλήθος ερωτημάτων που λαμβάνει. Εφόσον υπάρχει τότε απλά ενημερώνει τη βάση δεδομένων του, ότι το συγκεκριμένο υποερώτημα είναι κομμάτι δυο ή και παραπάνω ερωτημάτων, χωρίς να κάνει στην ουσία εγγραφή νέου υποερωτήματος και εν συνεχεία ενημερώνει και το 'B3'. Η πλειάδα που στέλνεται στο B3 περιέχει τις τιμές: το μοναδικό κλειδί της πράξης (ID), τον τύπο της πράξης, τους τελεστέους /γεγονότα που αφορά, τα ερωτήματα που αποτελεί κομμάτι συνδεδεμένα με τις πράξεις-γονέα και το χρονικό παράθυρο, το αν πρόκειται για εισαγωγή ή διαγραφή.

B3

Το ‘B3’ αποτελεί και το τελευταίο στάδιο επεξεργασίας το δεδομένων που εισέρχονται στο σύστημα μας. Δέχεται τις πλειάδες των ‘B1’ και του ‘B3’ σαν είσοδο. Έτσι γνωρίζει το σύνολο των υποερωτημάτων που του στέλνει το ‘B2’ και τυχών ενημερώσεις αυτών ενώ ταυτόχρονα λαμβάνει κάθε νέο γεγονός από το ‘B1’ και την αντιστοίχηση του με κάποια από τα υποερωτήματα. Για κάθε νέο γεγονός που λαμβάνει λοιπόν το συγκεκριμένο bolt ελέγχει αν ικανοποιείται κάποιο υποερώτημα και κατά συνεπεία αν τυχών αυτό σηματοδοτεί και την ολοκλήρωση του εντοπισμού όλου του ερωτήματος. Αν δεν ικανοποιείται ολόκληρο το ερώτημα τότε δημιουργείται από τον εντοπισμό του υποερωτήματος ένα νέο γεγονός, σύνθετο όπως το αποκαλούμε, το οποίο ανοίγει νέο κύκλο ελέγχου για το αν ικανοποιεί εκείνο κάποιο άλλο υποερώτημα είτε του ίδιου συνολικού ερωτήματος είτε κάποιου άλλου (σε περίπτωση που όπως αναφέραμε υπάρχει ομοιότητα υποερωτημάτων) κ.ο.κ. η επεξεργασία του εκάστοτε γεγονότος ολοκληρώνεται είτε με την ικανοποίηση κάποιου υποερωτήματος είτε εφόσον δεν ικανοποιείται με την εισαγωγή του νέου γεγονότος στις λίστες με τα χρονικά στιγμιότυπα των γεγονότων που αφορούν αυτό το υποερώτημα.

5. Υλοποίηση Τοπολογίας

5.1 Συναρτήσεις

5.1.1 Το S1

Για τα δεδομένα που λαμβάνει το spout δημιουργήσαμε ένα server που μέσω socket που ανοίγουμε μεταξύ των δυο μεταφέρουμε τα δεδομένα. Τα δεδομένα είναι της μορφής:

A187
A72
A29
.
.
.

Αυτά τα δεδομένα παράγονται από μια τυχαία γεννήτρια στο server. Εμείς όταν τρέχουμε την εφαρμογή μπορούμε να επιλέξουμε το πλήθος των διαφορετικών event μπορεί να έχουμε (π.χ. 100 -> A1-A100) καθώς και το πλήθος αυτών.

Έπειτα από το spout προστίθεται σε κάθε event το timestamp που το λάβαμε και μαζί σαν μια πλειάδα στέλνονται στο bolt “B1” .

6.1 Σχέδιο λειτουργίας του Spout 'S1'

5.1.2 Το S2

Αντίστοιχα και εδώ υλοποιήσαμε ένα άλλο server που πάλι μέσω socket στέλνει δεδομένα στο S2. Τα δεδομένα είναι της μορφής:

Add, 1, dataq1, ((((A AND B) AND (A OR C)) OR B) SEQ G), 1000
Remove, 1, dataq1, ((((A AND B) AND (A OR C)) OR B) SEQ G), 1000
Add, 2, dataq2, (((A AND B) AND (A OR C)) OR C), 2000
.
.
.

Στον server υπάρχουν αρκετά πρότυπα ερωτήματα (queries) από το οποία επιλέγει τυχαία κάποιο κάθε φορά. Τυχαία επιλέγεται και τα event τα οποία θα αφορά αυτό το ερώτημα. Και εδώ επιλέγουμε το πλήθος των διαφορετικών event μπορεί να υπάρχουν μέσα στα ερωτήματα (queries) καθώς και το πλήθος των ερωτημάτων που θέλουμε να αναζητήσουμε από το σύστημα. Έτσι στέλνει τα δεδομένα με την μορφή που είπαμε παραπάνω στο 'B2'.

Μετά όπως έχουμε ήδη περιγράψει σπάμε στα πέντε δομικά του κομμάτια το string. Εξάγουμε δηλαδή από κάθε ένα δεδομένο το αν πρόκειται για εισαγωγή ή διαγραφή κάποιου ερωτήματος (query), την πηγή από όπου προήλθε αυτό, το ίδιο το ερώτημα (query) και το χρονικό παράθυρο μέσα στο οποίο πρέπει να εντοπιστεί (σε ms). Αυτά τα πέντε χαρακτηριστικά κάθε ερωτήματος τα στέλνουμε μαζί σαν τιμές μιας πλειάδας στο B2.

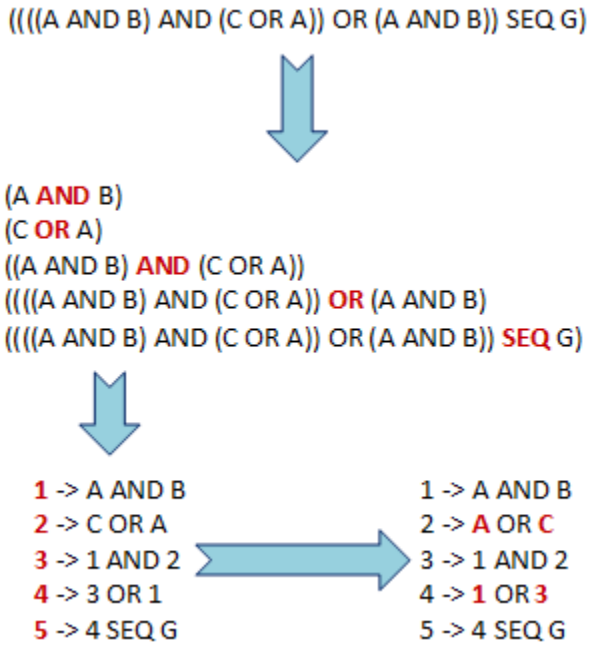
6.2 Σχέδιο λειτουργίας του Spout 'S2'

5.1.3 Το B2

Ξεκινάμε να αναλύουμε το B2 πρώτο από τα Bolt, βήμα βήμα τα στάδια επεξεργασίας, για να μπορέσουμε να παρακολουθήσουμε καλύτερα την πορεία των δεδομένων.

Στο B2 λαμβάνουμε δεδομένα από πολλές ροές. Στο execute() ανάλογα την πηγή των δεδομένων και τις τιμές αυτών θα ακολουθηθεί και ανάλογη πορεία επεξεργασίας.

Τα δεδομένα που έρχονται με την μορφή που περιγράψαμε από το S2 στέλνονται στη συνάρτηση SubdivideQuery() η οποία σκοπό έχει να κατακερματίσει το συνολικό ερώτημα. Αυτή η συνάρτηση λοιπόν σπάει σε απλές λογικές πράξεις τα σύνθετα ερωτήματα. Στις απλές πράξεις μετέπειτα ταξινομεί αλφαβητικά τους τελεστέους (εφόσον η λογική πράξη δεν είναι η SEQ (ακολουθία) που η σειρά αυτών έχει ιδιαίτερο νόημα) και για κάθε μια ελέγχει αν τυχών επαναλαμβάνεται στο ίδιο ερώτημα ήδη. Έπειτα δίνεται στη πράξη ένα μοναδικό κλειδί (ID) και αν αυτή, με τη σειρά της, είναι τελεστέος στο συνολικό ερώτημα (ή σε κάποια άλλη πράξη) τότε ο τελεστέος αντικαθίσταται με το μοναδικό κλειδί, της πράξης, για να μπορέσει να δομηθεί και η δενδρική δομή που προαναφέραμε.



6.3 Το στάδιο κατακερματισμού του μοτίβου ενός ερωτήματος

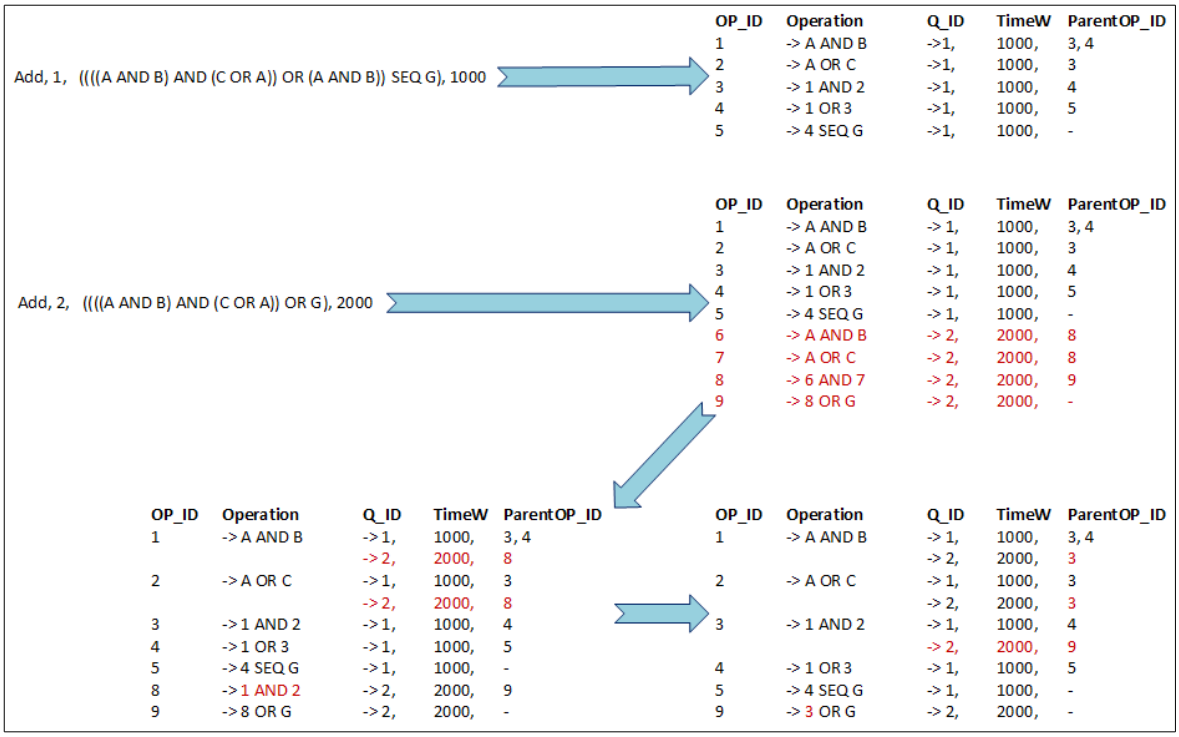
Έτσι λοιπόν κάθε πράξη/operation, που εξάγεται από ένα ερώτημα, την αποθηκεύουμε σε μια λίστα. Κάθε πράξη έχει τα εξής χαρακτηριστικά: το μοναδικό κλειδί (ID) της, το τύπο της πράξης (AND ή OR ή SEQ), τους τελεστέους, το μοναδικό κλειδί του συνολικού ερωτήματος που ανήκει, το χρονικό παράθυρο μέσα στο οποίο πρέπει να εντοπιστεί, ποιες πράξεις είναι γονείς αυτής της πράξης (δηλαδή εφόσον εντοπιστεί η πράξη πρέπει να ενημερώσει κάποια άλλη για να συνεχιστεί ο συνολικός εντοπισμός ενώ αν δεν έχει πράξη-γονέα τότε εντοπίστηκε συνολικά το ερώτημα) και εάν αυτό το ερώτημα έχει σταλθεί για εισαγωγή ή για διαγραφή.

Κάθε μια πράξη λοιπόν, από αυτές που συγκροτούν κάθε ερώτημα, πρέπει να ελεγχθεί αν τυχών έχει ξαναεισαχθεί στο σύστημα μας. Έχοντας ήδη ελέγξει αν υπάρχει στο ίδιο ερώτημα πάνω από μια φορά στη SubdivideQuery(), η μεθοδολογία που ακολουθούμε είναι ότι πρώτα από όλα ελέγχουμε αν υπάρχει στο ίδιο το task που κατακερμάτισε το ερώτημα και έπειτα στα υπόλοιπα tasks του bolt B2. Αν δεν υπάρχει ούτε εκεί τότε σημαίνει ότι δεν υπάρχει ήδη στο σύστημα μας και ότι πρέπει να το εισάγουμε εμείς.

Στέλνουμε έτσι κάθε μια πράξη στην συνάρτηση CheckInOperation() η οποία ελέγχει αν στο υπάρχον task υπάρχει όμοια πράξη.

- Εάν υπάρχει τότε κατά βάση κρατάμε τα χαρακτηριστικά της αρχικής πράξης. Δηλαδή κρατάμε το μοναδικό κλειδί, τον τύπο της πράξης, τους τελεστέους. Αυτό που αλλάζουμε είναι ότι ανανεώνουμε, αφαιρώντας ή προσθέτοντας (αν έχουμε εισαγωγή ή διαγραφή ερωτήματος) στα ήδη υπάρχοντα, το χαρακτηριστικό τύπου λίστας που μας ενημερώνει για τα συνολικά ερωτήματα που η συγκεκριμένη πράξη αποτελεί κομμάτι και μαζί με τα ερωτήματα και τα χρονικά παράθυρα που έχουν αυτά αλλά και τις νέες πράξεις-γονείς που πρέπει να ειδοποιήσει εφόσον εντοπιστεί. Αποθηκεύουμε λοιπόν την πράξη σε μια λίστα με όλες τις τελικώς διαμορφωμένες πράξεις αυτού του task και έπειτα:
    - Στέλνουμε τα ανανεωμένα στοιχεία της πράξης στο B3 μέσω του Stream2.
    - Στέλνουμε, μέσω του stream3, στα υπόλοιπα task του B2 για να αλλάξουν σε τυχόν πράξεις που έχουν γονέα τη συγκεκριμένη πράξη το ανανεωμένο μοναδικό κλειδί (ID) της.
    - Καλούμε τη RefreshPOP() και την RefreshLeaves().
- Εάν δεν το υπάρχει στο συγκεκριμένο task τότε στέλνουμε, μέσω του stream3, στα υπόλοιπα tasks του B2 για να εξετάσουμε αν υπάρχει εκεί όμοια πράξη.

Μετά συνεχίζουμε με τις επόμενες πράξεις του ερωτήματος που κατακερματίσαμε.



6.4 Παράδειγμα του ελέγχου και ανανέωση των πράξεων. Στη προκειμένη περίπτωση είναι απλοποιημένη η διαδικασία γιατί έχουμε 1 task για όλο το B2 αλλά φαίνονται τα βήματα ελέγχου και αλλαγών που γίνονται.

Εφόσον λοιπόν δεν βρεθεί όμοια πράξη στο αρχικό task, τα υπόλοιπα tasks του B2 λαμβάνουν μέσω του stream3 την πράξη για να αναζητήσουν εάν αυτά έχουν μια όμοια ήδη καταχωρημένη. Όταν λοιπόν λαμβάνουμε από αυτή τη ροή τα δεδομένα εισόδου στέλνονται από την execute() στη συνάρτηση CheckOUTOperation().

Εκεί κάθε ένα task εξετάζει αν η πράξη που του στάλθηκε έχει κάποια όμοια ήδη καταχωρημένη.

- Εφόσον υπάρχει απλά ανανεώνει την ήδη υπάρχουσα πράξη με το ίδιο τρόπο που ανανεώνει τη πράξη και η συνάρτηση CheckInOperation(). Έπειτα:
  - Στέλνουμε τα τελικά στοιχεία της πράξης στο B3 μέσω του stream2.
  - Καλούμε τη RefreshPOP().
  - Στέλνουμε ξανά, μέσω του stream3, στα υπόλοιπα task του B2 για να αλλάξουν σε τυχόν πράξεις που έχουν γονέα τη συγκεκριμένη πράξη το ανανεωμένο μοναδικό κλειδί (ID) της.
  - Στέλνουμε ενημέρωση στο αρχικό task, μέσω του stream4, ότι η συγκεκριμένη πράξη εντοπίστηκε στο συγκεκριμένο task.

- Εάν δεν το εντοπίσαμε τότε απλά ενημερώνουμε το αρχικό task, μέσω του stream4, ότι η συγκεκριμένη πράξη δεν έχει όμοια στο συγκεκριμένο task.

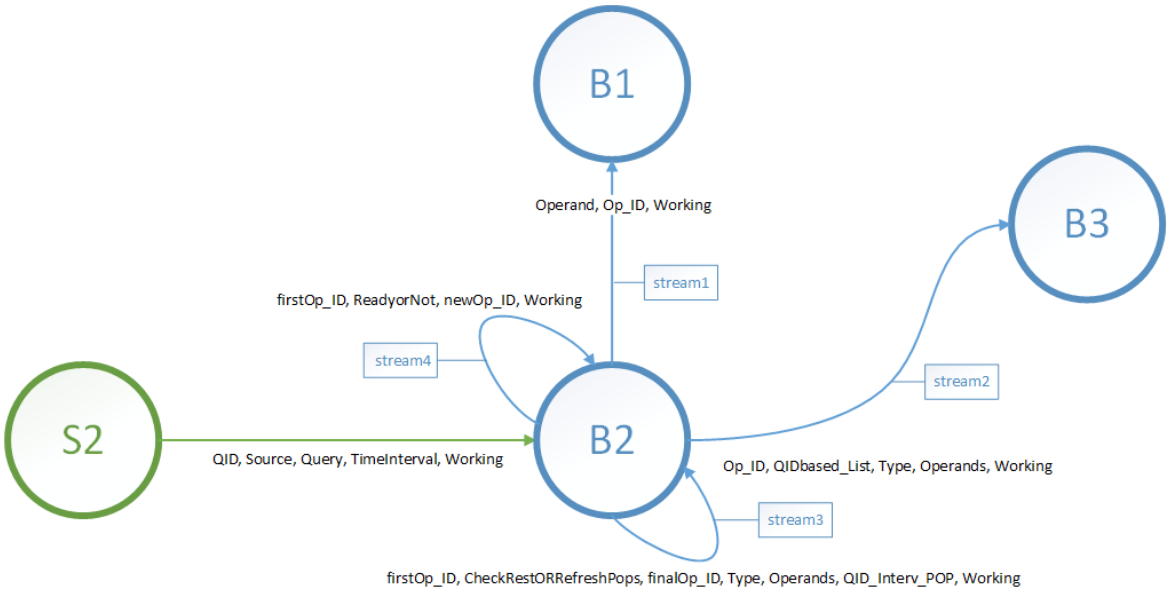
Αποστέλλεται λοιπόν μέσω της stream4, που πρόκειται για συντομευμένη ροή, στο αρχικό task η πληροφορία για το αν εντοπίστηκε από κάποιο άλλο task όμοια πράξη ή όχι. Τα δεδομένα από αυτή τη ροή στέλνονται στη συνάρτηση Checked Operation(). Εκεί:

- Αν για μια πράξη έχει γίνει εντοπισμός της σε άλλο task τότε καλεί τη RefreshLeaves() και τερματίζει με τη καταχώρηση αυτής της πράξης.
- Αν δεν υπάρχει όμως σε άλλο task γίνεται καταχώρηση της πράξης και τέλος στέλνεται στο B3 και στο B1 μέσω του stream2 και του stream1 αντίστοιχα για να εισαχθεί και εκεί η νέα πράξη.

Τέλος πρέπει να εξηγήσουμε τη χρήση των συναρτήσεων RefreshPoP() και RefreshLeaves() που καλούνται βοηθητικά σε πολλές περιπτώσεις.

Η RefreshPoP() καλείται όταν καταχωρείται μια πράξη που προϋπάρχει όμοια της σε αυτό ή άλλο task. Συνεπακόλουθο αυτού είναι να κρατηθεί το αρχικό μοναδικό κλειδί (ID) και άρα να απαιτείται ανανέωση αυτού αν η πράξη αυτή είναι γονέας ήδη καταχωρημένων πράξεων ενός task. Αυτές τις ανανεώσεις τις αναλαμβάνει αποκλειστικά η RefreshPoP(). Η συνάρτηση αυτή πριν τελειώσει τη λειτουργία της στέλνει τις ανανεωμένες πράξεις στο B3.

Η RefreshLeaves() καλείται και αυτή όταν καταχωρείται μια πράξη που προϋπάρχει όμοια της σε αυτό ή άλλο task. Εκείνη όμως ανανεώνει το μοναδικό κλειδί (ID) των τελεστέων μιας σύνθετης πράξης που είναι προς καταχώρηση.



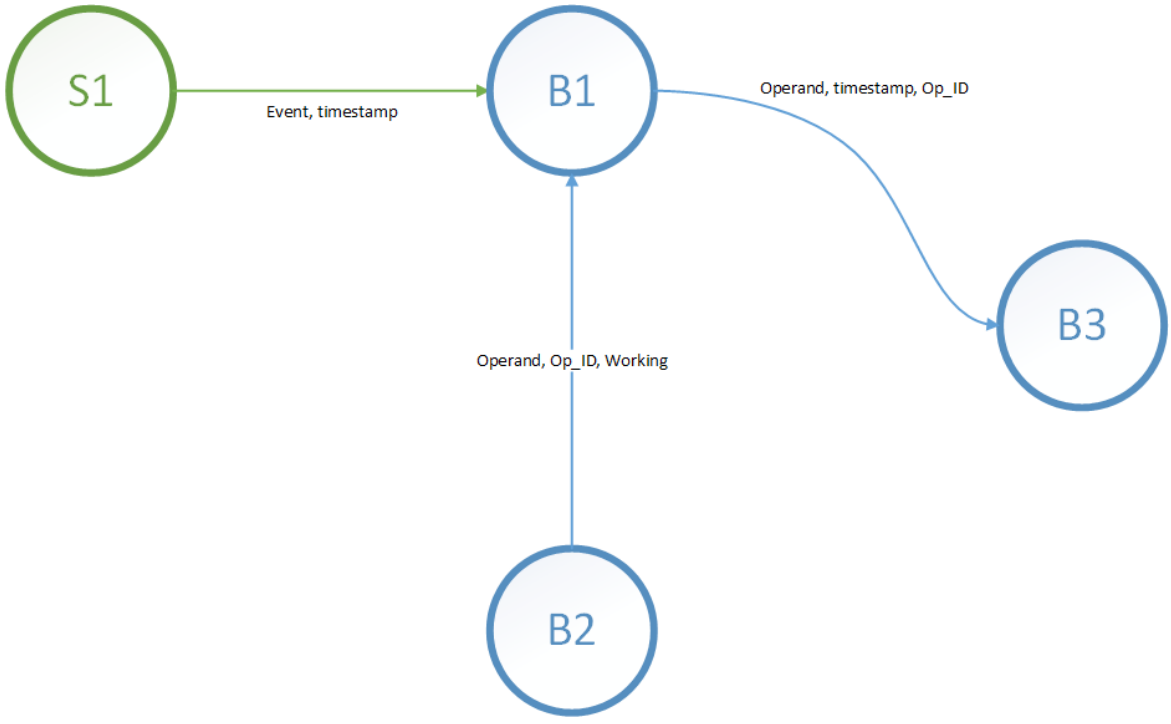
6.5 Σχέδιο λειτουργίας του Bolt 'B2'

5.1.4 Το B1

Το συγκεκριμένο Bolt όπως συνοπτικά αναφέραμε παραπάνω δέχεται πλειάδες από δύο πηγές. Δέχεται και από το S1 και από το B2. Έτσι ελέγχοντας την πηγή των δεδομένων κάθε φορά, στη συνάρτηση execute(), αντίστοιχες ενέργειες εκτελούνται.

Αν τα δεδομένα είναι από το B2 τότε λαμβάνουμε σε κάθε πλειάδα αυτής της ροής το γεγονός, το μοναδικό κλειδί της πράξης που αυτό το γεγονός είναι τελεστέος και το αν πρόκειται για εισαγωγή ή διαγραφή αυτής της πράξης. Έτσι για κάθε διαφορετικό τύπο γεγονότος που λαμβάνουμε από το B2 κάνουμε και μια καταχώρηση σε μια δομή HashMap που έχουμε φτιάξει. Σε αυτή τη δομή το κλειδί είναι ο τύπος του γεγονότος και τιμή κάθε κλειδιού είναι μια λίστα με τα μοναδικά κλειδιά των πράξεων στις οποίες τα γεγονότα αυτά είναι τελεστέοι. Καταχωρούμε λοιπόν στο HashMap το γεγονός που μας έστειλε το B2 και σαν τιμή το μοναδικό κλειδί της πράξης. Εφόσον έχει γίνει στο παρελθόν καταχώρηση αυτού του κλειδιού τότε απλά εισάγουμε στη λίστα ένα ακόμα μοναδικό κλειδί για το οποίο το συγκεκριμένο γεγονός αποτελεί τελεστέος.

Αν τα δεδομένα είναι από το S1 τότε λαμβάνουμε σε κάθε πλειάδα αυτής της ροής το γεγονός μαζί με τη χρονική στιγμή τελεστής του. Κάθε γεγονός που έρχεται από το S1 το χρησιμοποιούμε σαν κλειδί για να ανακαλέσουμε από το HashMap τη λίστα με τα μοναδικά κλειδιά (ID) των πράξεων που είναι τελεστέος. Έτσι στέλνουμε, στο B3, για κάθε στοιχείο της λίστας (πράξη) μια πλειάδα με τιμές: το τύπο του γεγονότος, το χρονικό στιγμιότυπο που τελέστηκε και το μοναδικό κλειδί της πράξης που είναι τελεστέος.



6.6 Σχέδιο λειτουργίας του Bolt 'B1'

5.1.5 Το B3

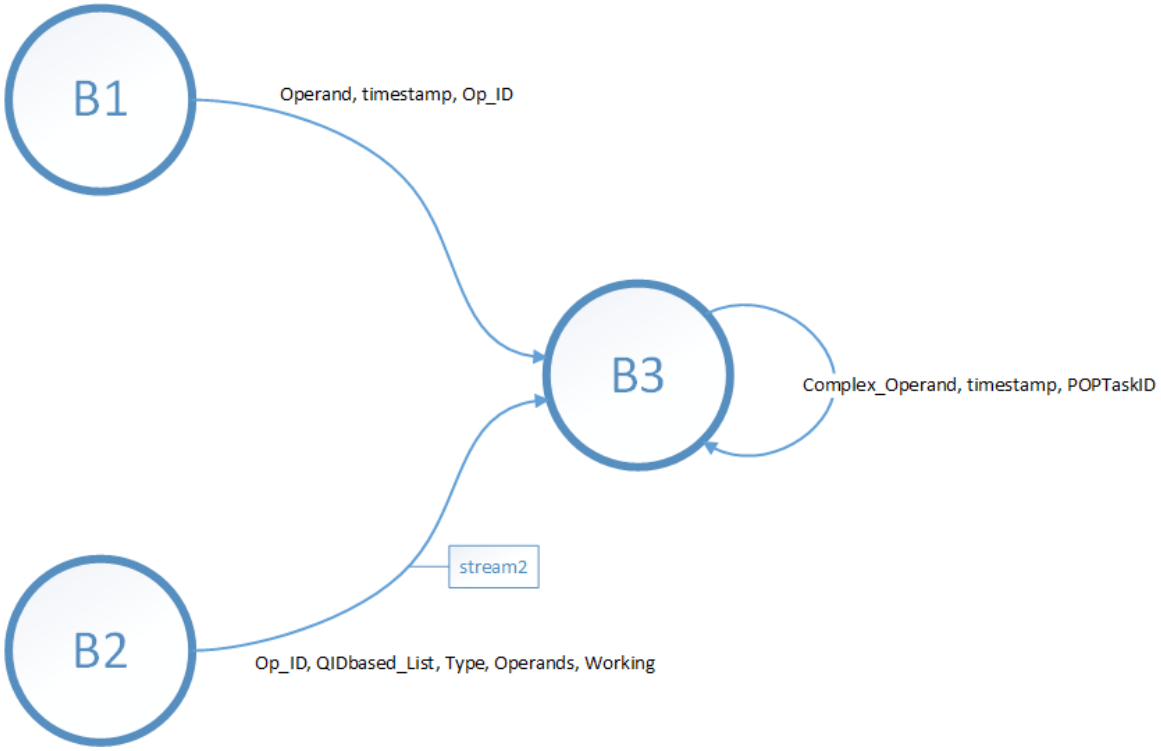
Το B3 αποτελεί το τελευταίο στάδιο επεξεργασίας των αρχικών δεδομένων. Το στάδιο όπου συνδυάζουμε κάθε νέο γεγονός με τη βάση δεδομένων των πράξεις που αναζητούμε για να ελέγξουμε αν υπάρχει εντοπισμός κομματιού ή ολόκληρου του ερωτήματος. Είσοδο λαμβάνει το B3 από το B2 από τη ροή stream2 και από το B1. Ανάλογα με την είσοδο που έχουμε ακολουθείται και η ανάλογη επεξεργασία.

Εάν η είσοδος είναι από το B2 τότε έχουμε να κάνουμε με κάποια εισαγωγή νέας πράξης, ενημέρωση ή διαγραφή κάποιας κάποιας υπάρχουσας. Καλούμε τότε τη refreshOperations() και ελέγχεται, βάσει του μοναδικού κλειδιού, εάν η πράξη που έλαβε το B3 έχει εισαχθεί στο παρελθόν και αν πρόκειται για ανανέωση αυτής (πρόσθεση ή αφαίρεση ερωτημάτων, πράξεων-γονέα, χρονικών παραθύρων) ή άμα δεν υπάρχει εισαγωγή της πράξης στο υπάρχον task του B3.

Εάν πρόκειται για είσοδο από το B1 τότε καλείται η newEvents() και βάση της τιμής της πλειάδας που μας δείχνει για ποια πράξη (βάσει του μοναδικού κλειδιού ID) αναφέρεται αυτό το γεγονός εξετάζουμε πρώτα αν υπάρχει η συγκεκριμένη πράξη και έπειτα εντοπίζουμε το τύπου του τελεστέου που ταιριάζει το συγκεκριμένο γεγονός. Εφόσον περάσουμε επιτυχώς αυτού τους ελέγχους, εισάγουμε το υπάρχον χρονικό στιγμιότυπο σε μια λίστα με όλα τα γεγονότα που έχουν εισαχθεί στο σύστημα, αφορούν αυτό τον τελεστέο και δεν έχουν χρησιμοποιηθεί για τον εντοπισμό αυτής της πράξης στο παρελθόν. Μετά από κάθε εισαγωγή νέου γεγονότος καλείται από την newEvents() η συνάρτηση findpattern().

Η συνάρτηση αυτή είναι υπεύθυνη για τον τυχόν εντοπισμό των προαπαιτούμενων μιας πράξης, δηλαδή αν έχουν εισαχθεί όλοι απαιτούμενοι τελεστέοι-γεγονότα μέσα σε ένα ορισμένο χρονικό παράθυρο. Έτσι για κάθε πράξη που ενεργοποιείται από την είσοδο του B1 ελέγχουμε αν το νέο γεγονός σε συνδυασμό με τα προηγούμενα ή και μόνο του καλύπτει τα κριτήρια που αναφέραμε. Ανάλογα με το αν η πράξη είναι τύπου ΚΑΙ, Ή, ΑΚΟΛΟΥΘΊΑ ελέγχουμε τα χρονικά στιγμιότυπα που έχουμε εισάγει για κάθε τελεστέο και αν είναι μέσα στα χρονικά περιθώρια. Αν τα κριτήρια καλύπτονται τότε τα γεγονότα που συμβάλουν στον εντοπισμό αφαιρούνται από τις λίστες για να μην ξαναχρησιμοποιηθούν (πιο συγκεκριμένα τα χρονικά στιγμιότυπα αυτών των γεγονότων). Από τον εντοπισμό προκύπτει ένα εξαγόμενο γεγονός το οποίο δείχνει επί της ουσίας τον εντοπισμό της συγκεκριμένης πράξης. Αυτό το γεγονός στέλνεται ξανά στο B3 και πιο συγκεκριμένα στις πράξεις-γονέα της πράξης που τα κριτήρια ικανοποιήθηκαν. Η πλειάδα που στέλνει έχει τις ίδιες ακριβώς τιμές με αυτές της πλειάδας που στέλνει το B1 στο B3. Το εξαγόμενο γεγονός κατόπιν λαμβάνεται από το B3 και αντιμετωπίζεται σαν κάθε άλλο γεγονός.

Με το εξαγόμενο γεγονός έπρεπε να υπερβούμε ένα πρόβλημα που προέκυπτε. Το πρόβλημα ήταν ποια από τα χρονικά στιγμιότυπα των πολλαπλών τελεστέων μιας πράξης θα χρησιμοποιούσαμε για χρονικό στιγμιότυπο του εξαγόμενου γεγονότος. Για να επιλύσουμε αυτό το πρόβλημα πήραμε τις πιθανές περιπτώσεις των τελεστέων των οποίων τα χρονικά στιγμιότυπα θέλουμε να χρησιμοποιήσουμε. Έτσι καταλήξαμε ότι οι δύο πιθανές περιπτώσεις είναι να θέλουμε είτε το χρονικά παλαιότερο είτε το νεότερο στιγμιότυπο. Ανάλογα με τις περιπτώσεις και τη διάρθρωση της πράξης-γονέα λοιπόν μπορεί να χρησιμοποιήσουμε είτε το ένα είτε το άλλο. Παρόλαυτα αυτά δεν μπορούμε να το ξέρουμε όταν εξάγουμε το γεγονός και το στέλνουμε, αλλά μόνο εκ των υστέρων. Έτσι επιλέξαμε να στέλνουμε στη πράξη γονέα σαν χρονικό στιγμιότυπο, του εξαγόμενου γεγονότος, και το παλιότερο και το νεότερο από αυτά των τελεστέων της πράξης που εντοπίστηκε. Έτσι ενώσαμε σε ένα κοινό αλφαριθμητικό και τα δυο στιγμιότυπα. Με αυτό τον τρόπο μπορούμε έπειτα στην πράξη-γονέα να κρίνουμε ποιο από αυτά τα δύο θέλουμε να χρησιμοποιήσουμε.



6.7 Σχέδιο λειτουργίας του Bolt 'B3'

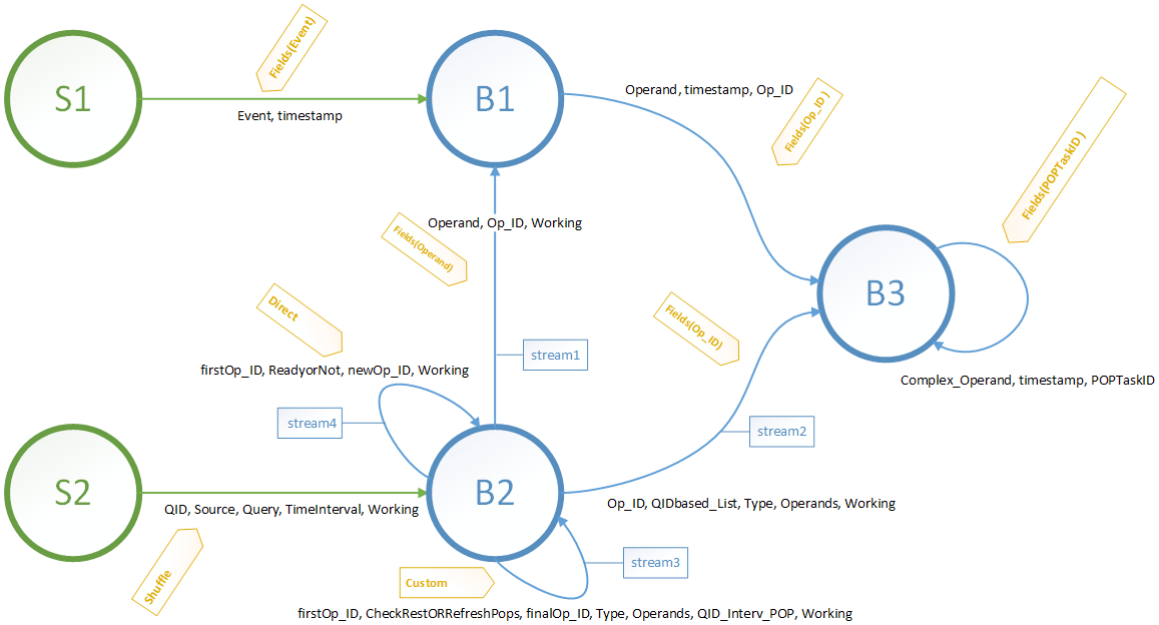
5.1.6 Η μέθοδος Main

Στη κλάση Main υλοποιήσαμε τη main μέθοδο που ενώνει τα bolts με τα spouts και δείχνει με ποιο τρόπο στέλνονται δεδομένα από το ένα στο άλλο καθώς και πως ομαδοποιούνται αυτά.

Επιλέξαμε λοιπόν για το B1 που λαμβάνει δεδομένα από το S1 και το B2 από τη ροή stream1 αυτά από κοινού να υλοποιούν την *Ομαδοποίηση Πεδίων,* με πεδίο τη τιμή του τύπου των δεδομένων (π.χ. A24). Με αυτό τον τρόπο είμαστε σίγουροι ότι τα νέα δεδομένα/γεγονότα θα πάνε στο task του B1 που έχουν καταχωρηθεί στο Hashmap κλειδιά με τα ίδιο τύπου γεγονότα.

Για το B2 που λαμβάνει από το S2 όπως και από τον εαυτό του μέσα από δύο ροές τη stream3 και τη stream4 επιλέξαμε διαφορετικές ομαδοποιήσεις. Για τα δεδομένα από το S2 επιλέξαμε τη τυχαία ομαδοποίηση για να διασφαλίσουμε ότι ο φόρτος θα μοιραστεί σε όλα τα task του B2. Για τα δεδομένα από τη ροή stream3 που στέλνει το B2 στον εαυτό του υλοποιήσαμε μια προσωπική ομαδοποίηση. Σε αυτή προσέχουμε ποιο task είναι ο αποστολέας των δεδομένων και ορίζουμε ως παραλήπτες τα υπόλοιπα tasks. Καταφέρνουμε έτσι όταν ένα task B2 στέλνει μια πράξη, για να δει αν έχει εισαχθεί παλαιότερα σε άλλο task, να σταλεί σε όλα πέρα του εαυτού και άρα γλιτώνουμε την αποστολή επιπλέον πλειάδων που δεν υπάρχει αξία να σταλθούν. Τέλος για τα δεδομένα που στέλνει το B2 στον εαυτό του από την ροή stream4 επιλέγουμε τη στοχευμένη ομαδοποίηση και αυτό το κάνουμε γιατί όταν κάποιο task παίρνει είσοδο από το stream3 για να ερευνήσει αν έχει ήδη αποθηκευμένο κάποια πράξη τότε βάση του κώδικα πρέπει να απαντήσει για το αποτέλεσμα της έρευνας στο αρχικό task/αποστολέα. Έτσι μπορεί με μεθόδους του Storm να εντοπιστεί ο αποστολέας και με στοχευμένη ομαδοποίηση να στείλει στοχευμένα την απάντηση στο task που την αναμένει.

Το B3 τέλος λαμβάνει δεδομένα από τρεις πηγές. Από το B1 από το B2 μέσω της ροής stream2 και από το ίδιο το B3. Πρέπει έτσι να διασφαλίσουμε ότι στο ίδιο task του B3 που έχουν αποθηκευτεί κάποιες από τις πράξεις που έχουν σταλεί από το B2, στο ίδιο task πρέπει να σταλούν από το B1 ή το B3 και τα γεγονότα που είναι τελεστέοι σε αυτές τις πράξεις. Έτσι χρησιμοποιούμε ξανά την *Ομαδοποίηση Πεδίων* με πεδίο το μοναδικό κλειδί (ID) της πράξης. Έτσι στέλνονται στο ίδιο task η πράξη από το B2 και οι τελεστέοι που ανήκουν σε αυτή από τα B1 και B3.



6.8 Πλήρης σχεδίαση της τοπολογίας μας

6. Μετρήσεις και Συμπεράσματα

Για να δοκιμάσουμε τη τοπολογία μας κάναμε πρώτα εξαντλητική αποσφαλμάτωση και τοπικά σε ένα υπολογιστή και έπειτα στον cluster. Αφού σιγουρευτήκαμε ότι τρέχει σωστά σε όλα τα στάδια επεξεργασίας αυξήσαμε το μέγεθος των δεδομένων που εισάγονται στη τοπολογία μας για να δούμε τη συμπεριφορά της, να πάρουμε μετρήσεις και να βγάλουμε χρήσιμα συμπεράσματα.

Πριν κάνουμε τις δοκιμές όμως στον cluster ορίσαμε τις βασικές παραμέτρους του συστήματος τις οποίες θα μεταβάλλαμε. Αρχικά θεωρήσαμε ότι οι workers που θέλουμε να έχει το σύστημά μας μαζί με τον ορισμό της παραλληλίας κάθε spout και bolt ορίζουν και τους πόρους που παρέχονται στη τοπολογία κάθε φορά άρα και την υπολογιστική της δυνατότητα. Έπειτα βλέποντας ότι ανάλογα τους πόρους έχουμε και διαφορετικά αποτελέσματα θέσαμε άλλες 2 παραμέτρους. Αυτές είναι ο αριθμός των ξεχωριστών γεγονότων αλλά και ο αριθμός των ερωτημάτων που αναζητά το σύστημα. Τέλος ξεκινώντας να κάνουμε τις δοκιμές είδαμε ότι η ταχύτητα της ροής των δεδομένων στο σύστημα και πιο συγκεκριμένα στα spouts είναι καθοριστική παράμετρος. Για να μπορεί έτσι το σύστημα να αναζητεί επαρκές ή μεγάλο πλήθος ερωτημάτων, έπρεπε να μπει ένας περιορισμός στη ταχύτητα ροής των δεδομένων. Ειδάλλως το σύστημα μας αδυνατούσε να διαχειριστεί το φόρτο εργασίας και κατέρρεε.

Μετά από 90 περίπου μετρήσεις που πήραμε προκύπτουν αρκετά συμπεράσματα για τις δυνατότητες και τις προοπτικές του συστήματος που δημιουργήσαμε. Καταδεικνύουν επίσης επαρκώς τα όρια λειτουργίας του ανάλογα τη ρύθμιση των παραμέτρων. Παραθέτουμε κάποιες μετρήσεις παρακάτω, ως αποδείξεις, συνοδευόμενες από τα συμπεράσματα που εξάγουμε από αυτές.

6.1 Σύγκριση μετρήσεων με διαφορετικό αριθμό workers

Αρχικά μεταβάλουμε τη μεταβλητή των workers και βλέπουμε πώς διαμορφώνονται τα αποτελέσματα σε κάθε αλλαγή που γίνεται.

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	12	1-1-2-2-6	150	500	Working
event/1ms query/1ms	12	1-1-2-2-6	150	750	Cluttered
event/1ms query/1ms	6	1-1-2-2-6	150	500	Working
event/1ms query/1ms	6	1-1-2-2-6	150	750	Cluttered

7.1.1 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	12	1-1-2-2-6	200	800	Working
event/1ms query/1ms	12	1-1-2-2-6	200	1.000	Cluttered
event/1ms query/1ms	6	1-1-2-2-6	200	800	Working
event/1ms query/1ms	6	1-1-2-2-6	200	1.000	Cluttered

7.1.2 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	12	1-1-2-3-5	100	500	Working
event/2ms query/2ms	12	1-1-2-3-5	100	750	Cluttered
event/2ms query/2ms	6	1-1-2-3-5	100	500	Working
event/2ms query/2ms	6	1-1-2-3-5	100	750	Cluttered

7.1.3 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	12	1-1-2-2-6	200	2.000	Working
event/2ms query/2ms	12	1-1-2-2-6	200	2.500	Cluttered
event/2ms query/2ms	6	1-1-2-2-6	200	2.000	Working
event/2ms query/2ms	6	1-1-2-2-6	200	2.500	Cluttered

7.1.4 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	6	1-1-2-2-9	200	2.700	Working
event/2ms query/2ms	6	1-1-2-2-9	200	3.000	Cluttered
event/2ms query/2ms	15	1-1-2-2-9	200	3.000	Working

7.1.5 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	6	1-1-2-2-9	200	2.700	Working
event/2ms query/2ms	2	1-1-2-2-9	200	2.700	Cluttered

7.1.6 Μετρήσεις

Είναι λοιπόν εμφανές ότι για ένα μεγάλο πλήθος μετρήσεων στις οποίες δεν αναζητούμε πολύ μεγάλο αριθμό ερωτημάτων τότε δεν παρατηρείται κάποια διαφορά στα αποτελέσματα. Αντίθετα όταν έχουμε πλήθος ερωτημάτων και σχετικά μεγάλη αναντιστοιχία μεταξύ του συνολικού παραλληλισμού και των workers τότε βλέπουμε διαφορά. Πιο συγκεκριμένα στα τελευταία δύο παραδείγματα φαίνεται όταν τα ερωτήματα που αναζητά το σύστημα είναι της τάξης των 2.700-3.000 και ταυτόχρονα ενώ ο συνολικός παραλληλισμός είναι 15 το να αλλάζουμε τους workers από 6 σε 15 και σε 2 αλλάζουν και τα όρια λειτουργίας του συστήματος προς το καλύτερο και προς το χειρότερο αντίστοιχα.

6.2 Σύγκριση μετρήσεων με διαφορετική παραλληλία

Στη συνέχεια ασχοληθήκαμε με το να εξετάσουμε κατά πόσο επηρεάζει η παραλληλία που θα ορίζαμε στα bolt την απόδοση του συστήματος. Γνωρίζοντας από πριν το μέγεθος της επεξεργασίας που σε καθένα έχει ανατεθεί και βλέποντας στις δοκιμές μέσα από τη διεπαφή του Storm το αν χρειάζονται επιπλέον πόρους δώσαμε βάση και μεγάλο παραλληλισμό στο B3. Συγκρίνουμε λοιπόν παρακάτω 3 συνδυασμούς παραλληλίας (1-1-3-3-3, 1-1-2-2-6, 1-1-2-2-9) σε πολλές και διαφορετικές μετρήσεις.

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	6	1-1-3-3-3	100	750	Cluttered
event/2ms query/2ms	6	1-1-2-2-6	100	800	Working
event/2ms query/2ms	6	1-1-2-2-6	100	1.000	Cluttered
event/2ms query/2ms	6	1-1-2-2-9	100	1.000	Working

7.2.1 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	6	1-1-3-3-3	200	1.800	Cluttered
event/2ms query/2ms	6	1-1-2-2-6	200	2.000	Working
event/2ms query/2ms	6	1-1-2-2-6	200	2.500	Cluttered
event/2ms query/2ms	6	1-1-2-2-9	200	2.700	Working

7.2.2 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-3-3-3	100	400	Cluttered
event/1ms query/1ms	6	1-1-2-2-6	100	400	Working
event/1ms query/1ms	6	1-1-2-2-6	100	500	Cluttered
event/1ms query/1ms	6	1-1-2-2-9	100	500	Working

7.2.3 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-3-3-3	200	700	Cluttered
event/1ms query/1ms	6	1-1-2-2-6	200	800	Working
event/1ms query/1ms	6	1-1-2-2-6	200	1.000	Cluttered
event/1ms query/1ms	6	1-1-2-2-9	200	1200	Working

7.2.4 Μετρήσεις

Γίνεται παραπάνω από εμφανές ότι δίνοντας επεξεργαστική ισχύ (δλδ παραπάνω παραλληλία) έχουμε σε όλες τις περιπτώσεις αύξηση των ορίων του συστήματος και βελτίωση της απόδοσης. Επίσης να παρατηρήσουμε ότι άμα θέλαμε να ορίσουμε ποιο bolt έχει έπειτα από το B3 ανάγκη για μεγαλύτερη παραλληλία αυτό είναι το B2 σε περιπτώσεις που θέλουμε να ερευνήσουμε πολύ μεγάλο πλήθος ερωτημάτων 3.000+.

6.3 Σύγκριση μετρήσεων με διαφορετικό αριθμό ξεχωριστών γεγονότων

Έπειτα όπως πιθανώς να έχει γίνει ήδη αντιληπτό από τα προηγούμενα παραδείγματα που δείξαμε σημαντικό ρόλο έχει το πόσα είναι τα ξεχωριστά είδη γεγονότων που εισέρχονται στο σύστημα. Αποτελεί σημαντική παράμετρος, με αυτή την διάταξη της τοπολογίας και την υπάρχουσα ροή δεδομένων, γιατί όταν έρχεται ένα νέο γεγονός στο B1 τότε βλέπουμε με ποιες πράξεις συσχετίζεται και στέλνει στο B3 μια πλειάδα με αυτό το γεγονός για κάθε μια πράξη. Σε αυτό το σημείο να αναφέρουμε ότι κάθε είδος γεγονότος έχει τις ίδιες πιθανότητες με κάθε άλλο. Όταν λοιπόν έχουμε 500 πράξεις που αφορούν πρωταρχικά γεγονότα και 100 διαφορετικά γεγονότα τότε αναλογούν 5 πράξεις ανά τύπο γεγονότος και άρα στέλνονται στο B3 5 πλειάδες ανά νέο γεγονός που φτάνει στο B1. Στο ίδιο παράδειγμα κάνοντας 200 τα διαφορετικά γεγονότα τότε αντιστρόφως ανάλογα μειώνονται και πλειάδες που στέλνονται στο B3 από το B1 με αποτέλεσμα μειώνεται ο φόρτος στο B3.

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-2-2-6	**100**	400	**Working**
event/1ms query/1ms	6	1-1-2-2-6	**100**	500	**Cluttered**
event/1ms query/1ms	6	1-1-2-2-6	**150**	500	**Working**
event/1ms query/1ms	6	1-1-2-2-6	**150**	750	**Cluttered**
event/1ms query/1ms	6	1-1-2-2-6	**200**	800	**Working**
event/1ms query/1ms	6	1-1-2-2-6	**200**	1.000	**Cluttered**

7.3.1 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	6	1-1-2-2-6	100	800	Working
event/2ms query/2ms	6	1-1-2-2-6	100	1.000	Cluttered
event/2ms query/2ms	6	1-1-2-2-6	200	2.000	Working
event/2ms query/2ms	6	1-1-2-2-6	200	2.500	Cluttered

7.3.2 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-2-3-5	100	300	Working
event/1ms query/1ms	6	1-1-2-3-5	100	500	Cluttered
event/1ms query/1ms	6	1-1-2-3-5	200	750	Working
event/1ms query/1ms	6	1-1-2-3-5	200	1.000	Cluttered

7.3.3 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/2ms query/2ms	6	1-1-2-3-5	100	500	Working
event/2ms query/2ms	6	1-1-2-3-5	100	750	Cluttered
event/2ms query/2ms	6	1-1-2-3-5	200	2.000	Working
event/2ms query/2ms	6	1-1-2-3-5	200	2.500	Cluttered

7.3.4 Μετρήσεις

Αυτή η σχέση πλήθους ερωτημάτων που αναζητούμε με τα ξεχωριστά είδη γεγονότων μπορούμε να την παρατηρήσουμε και στις παραπάνω μετρήσεις. Σημαντικό είναι ότι αυτό το συμπέρασμα επαληθεύεται για όλα τα μεγέθη των υπόλοιπων μεταβλητών που περιγράψαμε καθώς και των συνδυασμών τους. Όπου λοιπόν αυξήσαμε το πλήθος των ξεχωριστών γεγονότων αυξήθηκε και η δυνατότητα αναζήτησης περισσότερων ερωτημάτων και το αντίστροφο.

6.4 Σύγκριση μετρήσεων με διαφορετική καθυστέρηση ροής δεδομένων

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-2-2-6	100	400	Working
event/1ms query/1ms	6	1-1-2-2-6	100	600	Cluttered
event/2ms query/2ms	6	1-1-2-2-6	100	800	Working
event/2ms query/2ms	6	1-1-2-2-6	100	1.000	Cluttered

7.4.1 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-2-2-6	200	800	Working
event/1ms query/1ms	6	1-1-2-2-6	200	1.000	Cluttered
event/2ms query/2ms	6	1-1-2-2-6	200	2.000	Working
event/2ms query/2ms	6	1-1-2-2-6	200	2.500	Cluttered

7.4.2 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	12	1-1-2-3-5	100	300	Working
event/1ms query/1ms	12	1-1-2-3-5	100	400	Cluttered
event/2ms query/2ms	12	1-1-2-3-5	100	500	Working
event/2ms query/2ms	12	1-1-2-3-5	100	750	Cluttered

7.4.3 Μετρήσεις

query delay event delay (tuple/ms)	# of Workers	Parallelism S1-S2-B1-B2-B3	Distinct Events	# of Queries	Result
event/1ms query/1ms	6	1-1-2-3-5	200	750	Working
event/1ms query/1ms	6	1-1-2-3-5	200	1.000	Cluttered
event/2ms query/2ms	6	1-1-2-3-5	200	2.000	Working
event/2ms query/2ms	6	1-1-2-3-5	200	2.500	Cluttered

7.4.4 Μετρήσεις

Τέλος εξετάζουμε μετρήσεις που συγκρίνουμε την απόδοση του συστήματος σε σχέση με την αλλαγή της ταχύτητας της ροής δεδομένων. Όπως και στα προηγούμενα παραδείγματα βλέπουμε παραπάνω ότι για όλες τις δοκιμές που κάναμε η αύξηση της ροής δεδομένων στο σύστημα και κυρίως των γεγονότων σήμαινε και μείωση της απόδοσης και αντίστοιχα μείωση της ταχύτητας ροής των δεδομένων σήμαινε αύξηση της απόδοσης.

Σε όλες τις μετρήσεις που πήραμε ο χρόνος επεξεργασίας των δεδομένων ήταν ίσος με το χρόνο μετάδοσης των δεδομένων. Δηλαδή δεν ξεπερνούσε το 1ms.

7. Μελλοντική έρευνα

Στα πλαίσια αυτής της διπλωματικής έγινε μια πρώτη προσπάθεια κατασκευής του αλγορίθμου που θα υλοποιεί η τοπολογία μας και πετυχαίνει όλους τους αρχικούς στόχους που είχαμε. Δηλαδή να είναι λειτουργική, επεκτάσιμη και να μπορεί να διαχειριστεί ένα επαρκή αριθμό ερωτημάτων. Παρολαυτα σίγουρα υπάρχουν αρκετές επεκτάσεις και αλλαγές που μπορούν να γίνουν για να βελτιωθούν τα όρια λειτουργίας του συστήματος είτε και η ταχύτητα επεξεργασίας αυτών.

Βλέποντας ότι σημαντικό ρόλο στην απόδοση του συστήματος παίζει ο αριθμός των ερωτημάτων θα ήταν χρήσιμος ένας εξαρχής σωστός κατακερματισμός του ερωτήματος. Στις λογικές εκφράσεις υπάρχουν αλγόριθμοι μετασχηματισμού τους ώστε να έχουν το ίδιο αποτέλεσμα άλλα με διαφορετική σειρά πράξεων. Τέτοιοι μετασχηματισμοί πιθανώς οδηγούν στο αντίστοιχο αποτέλεσμα αλλά με λιγότερες υπολογιστικές ανάγκες. Αυτό παρόλο που θα πρόσθετε επιπλέον φόρτο στο B2 θα ήταν βοηθητικό για το B3 που επί της ουσίας εντοπίσαμε ως bottleneck της εφαρμογής μας.

Επίσης σημαντική προσθήκη θα ήταν η δυνατότητα αναπροσαρμογής του φόρτου εργασίας κατά τη διάρκεια που εκείνο επεξεργάζεται δεδομένα. Πιο συγκεκριμένα όταν θέλουμε να μεταφέρουμε την εφαρμογή αυτή σε συνθήκες πραγματικές όπου τα δεδομένα δεν είναι ισοπίθανα μεταξύ τους και που έχουμε στιγμές που υπάρχει μεγάλη κίνηση δεδομένων και στιγμές που αυτά έρχονται πιο αραιά τότε θα δημιουργείται και πρόβλημα στην εφαρμογή. Για την επίλυση του χρειάζεται ένας αλγόριθμος που ελέγχει τη ροή νέων event και αναπροσαρμόζει την κατανομή αυτών κυρίαρχα στο B2 και κατ' επέκταση στο B3.

Βιβλιογραφική αναφορά

[1] S. Allen, M. Jankowski, P. Pathirana. Storm Applied: Strategies for real-time event processing. Greenwich, CT: Manning Publications Co.

[2] A. Jain, A. Nalya. Learning Storm. Packt Publishing.

[3] J.Leiviusky, G. Eisbruch, D. Simnassi. Getting Started with Storm. O'Reilly Media, Inc.

[4] I. Flouris, N. Giatrakos, M Garofalakis, A. Deligiannakis Issues in Complex Event Processing Systems. Trustcom/BigDataSE/ISPA, 2015 IEEE (Volume:2), Pages 241-246, Aug. 2015.

[5] M. Akdere, U. Cetintemel, N. Tatbul, Plan-based complex event detection across distributed sources, in: Proceedings of VLDB, 2008.

[6] I. Kolchinsky, T. Sharfman, A. Schuster. Lazy Evaluation Methods for Detecting Complex Events. The 9th ACM International Conference on Distributed Event-Based Systems (DEBS). July 2015

[7] Y. Mei, S. Madden, Zstream: A cost-based query processor for adaptively detecting composite events, in: Proceedings of SIGMOD, 2009.

[8] N. P. Schultz-Moller, M. Migliavacca, P. Pietzuch, Distributed complex event processing with query rewriting, in: Proceedings of DEBS, 2009.

[9] IBM Proactive Technology Online on STORM

[10] Apache Storm

[11] https://en.wikipedia.org/wiki/Complex_event_processing

[12] H. Zhang, Y. Diao, and N. Immerman. Optimizing expensive queries in complex event processing. In Proceedings of SIGMOD, pages 217–228, June 2014