ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

"ΠΡΟΗΓΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Complex event processing on streaming sensor data using Esper and Storm

Βασίλειος Η Γιαννούτσος
Σοφία Χ Νομικού

Επιβλέπων: **Ευστάθιος Χατζηευθυμιάδης,** Καθηγητής

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΙΟΣ 2018

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Complex event processing on streaming sensor data using Esper and Storm

Βασίλειος Η. Γιαννούτσος
Σοφία Χ. Νομικού

Α.Μ.: M1456
Α.Μ.: M1351

ΕΠΙΒΛΕΠΩΝ: **Ευστάθιος Χατζηευθυμιάδης,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Κωνσταντίνος Κολομβάτσος,** ΕΔΙΠ

ΔΕΚΕΜΒΡΙΟΣ 2018

ΠΕΡΙΛΗΨΗ

Η συμβολή της ανάλυσης δεδομένων στην απόκτηση νέας γνώσης και στη διαδικασία λήψης αποφάσεων είναι ιδιαίτερα σημαντική. Ο μεγάλος πλέον όγκος των δεδομένων καθιστά αναγκαία τη διαδικασία της επεξεργασίας τους έτσι ώστε να παραχθεί χρήσιμη γνώση.

Η εξόρυξη δεδομένων (data mining) είναι μία διαδικασία επιθεώρησης, καθαρισμού, μετατροπής και μοντελοποίησης δεδομένων με στόχο την ανακάλυψη χρήσιμων πληροφοριών, την υποβολή συμπερασμάτων και τη λήψη αποφάσεων. Η ανάλυση δεδομένων έχει πολλαπλές πτυχές και προσεγγίσεις, που περιλαμβάνουν ποικίλες τεχνικές με ποικίλα ονόματα, σε διαφορετικούς τομείς των επιχειρήσεων, της επιστήμης και της κοινωνικής επιστήμης. Η εξόρυξη δεδομένων είναι μια συγκεκριμένη τεχνική ανάλυσης δεδομένων που επικεντρώνεται στη μοντελοποίηση και την ανακάλυψη γνώσεων για λόγους πρόβλεψης και όχι για καθαρά περιγραφικούς σκοπούς.

Στην παρούσα διπλωματική εργασία εξετάζεται η διαδικασία συλλογής και επεξεργασίας δεδομένων από αισθητήρες (sensors). Για τη διαδικασία αυτή, μιας και ο όγκος των προς επεξεργασία δεδομένων είναι μεγάλος, θα χρησιμοποιηθούν τεχνολογίες και πλαίσια λογισμικού (frameworks) σχεδιασμένα να επεξεργάζονται μεγάλα δεδομένα (big data) καθώς τα παραδοσιακά λογισμικά εφαρμογών επεξεργασίας δεδομένων είναι ανεπαρκή για την αντιμετώπισή τους. Οι μεγάλες προκλήσεις στον τομέα των δεδομένων περιλαμβάνουν τη συλλογή δεδομένων, την αποθήκευση δεδομένων, την ανάλυση δεδομένων, την αναζήτηση, την κοινή χρήση, την μεταφορά, την οπτικοποίηση, την ερώτηση, την ενημέρωση, την προστασία προσωπικών δεδομένων και την προέλευση δεδομένων.

Με τη χρήση του λογισμικού Apache Storm πραγματοποιήθηκε αξιόπιστη επεξεργασία των ροών δεδομένων σε πραγματικό χρόνο. Επιπροσθέτως, εκτός από το προαναφερθέν πακέτο, χρησιμοποιήθηκε και το λογισμικό Esper με το οποίο αναλύθηκαν σειρές από συμβάντα με σκοπό την παραγωγή χρήσιμων συμπερασμάτων.

Τα δεδομένα που χρησιμοποιήθηκαν στην εργασία αυτή είναι δεδομένα που αποκτήθηκαν με χρήση αισθητήρων. Η επεξεργασία αυτού του είδους δεδομένων μπορεί να πραγματοποιηθεί με δύο τρόπους: επεξεργασία παρτίδας (batch processing) και επεξεργασία ροής (stream processing).

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ανάλυση Μεγάλων Δεδομένων από Αισθητήρες

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ανάλυση δεδομένων, μεγάλα δεδομένα, εξόρυξη δεδομένων, Apache, Storm, δεδομένα αισθητήρων, Kafka, Kibana, Elasticsearch

ABSTRACT

The contribution of data analysis to the acquisition of new knowledge and the decision-making process is particularly important. The huge volume of data makes it necessary to process them in order to generate useful knowledge.

Data mining is a process of inspecting, cleaning, transforming, and modeling data in order to discover useful information, draw conclusions and make decisions. Data analysis has multiple aspects and approaches, including varied techniques with varying names, in different business, science and social sciences. Data mining is a specific data analysis technique that focuses on modeling and discovery of knowledge for predictive purposes rather than for purely descriptive purposes.

In this diploma thesis we examine the process of collecting and processing data from sensors. For this process, as the volume of data to be processed is large, technologies and frameworks designed to process large data will be used.

Big data is data sets that are so bulky and complex that traditional data processing software is inadequate to deal with. Major data challenges include data collection, data storage, data analysis, search, sharing, transportation, visualization, questioning, updating, privacy and data origination.

Apache Storm software has been reliably used in order to process the data in real time. Additionally, in addition to the aforementioned package, Esper software was also used to analyze series of events to produce useful conclusions.

The data used in this work is data acquired using sensors. Processing this kind of data can be processed in two ways: batch processing and stream processing.

SUBJECT AREA: Analysis of Big Data from Sensors

KEYWORDS: Data Analysis, Big Data, Data Mining, Apache, Storm, sensor data, Kafka, Kibana, Elasticsearch

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα, καθ. Ευστάθιο Χατζηευθυμιάδη για τη συνεργασία και την πολύτιμη συμβολή του στην ολοκλήρωση της.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα διπλωματική εργασία διενεργήθηκε στα πλαίσια του Προγράμματος Μεταπτυχιακών Σπουδών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών από τους μεταπτυχιακούς φοιτητές Γιαννούτσο Βασίλειο και Νομικού Σοφία υπό την επίβλεψη του κ καθηγητή Χατζηευθυμιάδη Ευστάθιο.

Η εργασία αυτή εκπονήθηκε στην Αθήνα Αττικής στο χρονικό διάστημα από 1/1/2018 έως ότου 30/10/2018 και ο κύριος σκοπός της ήταν η εντριβή και η εκμάθηση στην πράξη τεχνολογιών ανάλυσης δεδομένων υψηλής κλίμακας. Επιπροσθέτως, μελετήθηκαν τα ευρέως χρησιμοποιούμενα πλαίσια λογισμικού στον τομέα της ανάλυσης των μεγάλων δεδομένων με σκοπό την πιθανή εύρεση και πρόταση βελτιώσεων στις διάφορες πτυχές τους.

Το θέμα αυτό βρίσκεται υπό έντονη συζήτηση τα τελευταία χρόνια μιας και τα προς ανάλυση δεδομένα είναι πολλά, οπότε αυτό λειτούργησε σαν επιπλέον κίνητρο για την ενασχόληση μας.

Επιθυμία υπήρξε όχι μόνο για την προσομοίωση της χρήσης των επιλεγμένων τεχνολογιών για την λύση ενός προβλήματος αλλά και για την εκμάθηση εις βάθος των τεχνολογιών αυτών με σκοπό να εντοπιστούν τα πλεονεκτήματα καθώς και τα μειονεκτήματα που τις συνοδεύουν.

Η εργασία για λειτουργικούς και πρακτικούς λόγους θα μπορούσε να διαχωριστεί σε τρία επιμέρους κεφάλαια.

Το πρώτο κεφάλαιο αφορά την ανάλυση των δεδομένων εισόδου του προβλήματος. Θα πρέπει να γίνει εκτίμηση του όγκου δεδομένων, ανάλυση της μορφής τους και εφαρμογή τεχνικών για τη μετάφρασή τους σε μορφή κατανοητή και επεξεργάσιμη από το τμήμα της λογικής του συστήματος.

Το δεύτερο κεφάλαιο αφορά την επιλογή, εγκατάσταση και παραμετροποίηση των τεχνολογιών στο σύστημα με σκοπό την επεξεργασία των δεδομένων εισόδου.

Το τρίτο κεφάλαιο αφορά τη δημιουργία της λογικής και την ενσωμάτωσή της στο σύστημα έτσι ώστε να πραγματοποιηθεί η επεξεργασία και να παραχθούν τα δεδομένα εξόδου.

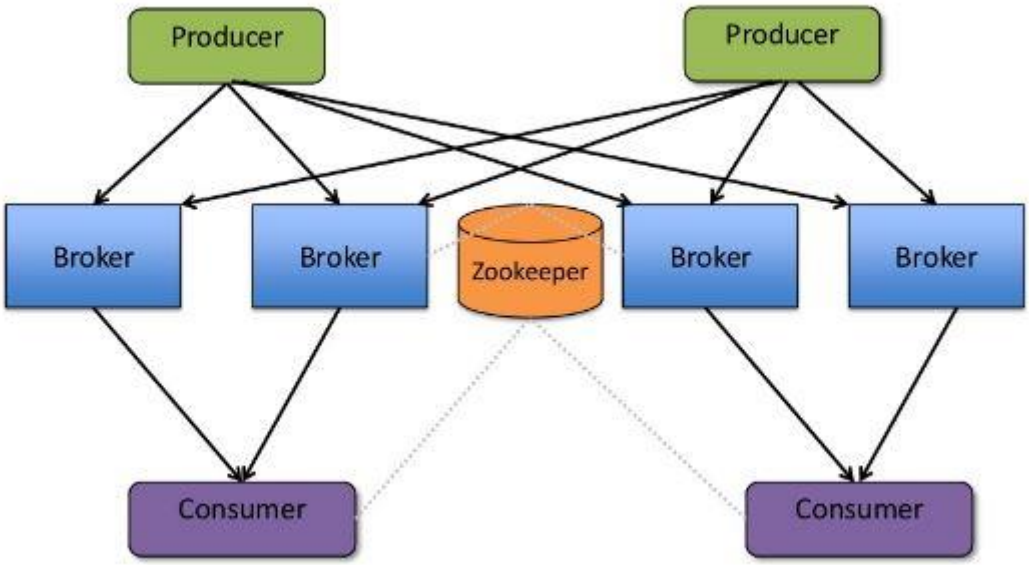
Στο τέταρτο και τελευταίο κεφάλαιο πραγματοποιείται η ανάλυση των αποτελεσμάτων και η αξιολόγηση της λογικής που εφαρμόστηκε.

1. ΤΟ ΣΥΣΤΗΜΑ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ

1.1 Apache Kafka

Το Kafka είναι ένα σύστημα της Apache το οποίο είναι ανεπτυγμένο στις γλώσσες προγραμματισμού Scala και Java. Πρόκειται για ένα κατανεμημένο σύστημα του οποίου ο ρόλος είναι η δημοσιοποίηση και η εγγραφή σε ροές πληροφορίας. Γενικότερα τέτοια συστήματα ανήκουν στην κατηγορία “publish-subscribe” και αποτελούν μία ουρά μηνυμάτων.

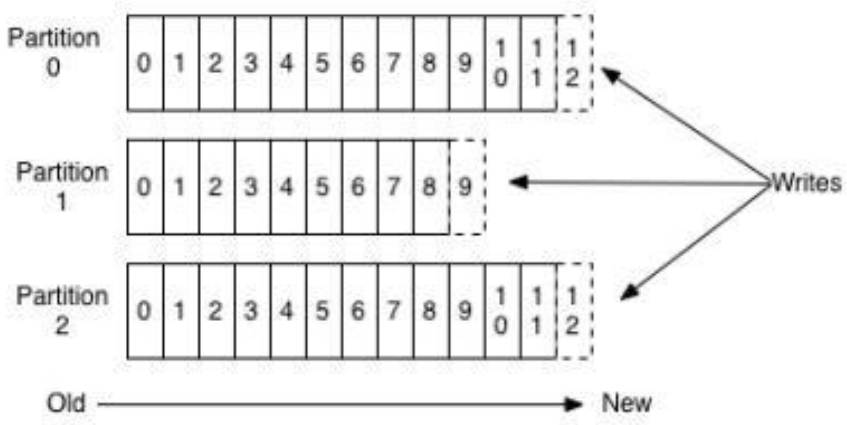
Τα κύρια χαρακτηριστικά ενός τέτοιου συστήματος είναι η μεγάλη ταχύτητα δεδομένου του μεγάλου όγκου δεδομένων που μεταφέρονται, η υψηλή δυνατότητα κλιμάκωσης καθώς και η δυνατότητα αποθήκευσης των μηνυμάτων με τρόπο ικανό να αποτρέπονται απώλειες δεδομένων σε περίπτωση αποτυχίας ή βλάβης του συστήματος.
Επίσης αυτό το σύστημα επιτρέπει την άμεση επεξεργασία των δεδομένων σε πραγματικό χρόνο. Οι παραπάνω λόγοι καθιστούν το Kafka την ιδανική λύση τόσο για τη δημιουργία pipelines πραγματικού χρόνου που μεταφέρουν δεδομένα μεταξύ συστημάτων ή υπηρεσιών με αξιόπιστο τρόπο όσο και για τη σχεδίαση εφαρμογών που επεξεργάζονται ή αντιδρούν με συγκεκριμένο τρόπο σε ροές δεδομένων.



Εικόνα 1: Η αρχιτεκτονική ενός συστήματος Apache Kafka

Το βασικό δομικό στοιχείο του συστήματος Kafka είναι το topic (άρθρο). Όλα τα μηνύματα/αρχεία που εισέρχονται στο οικοσύστημα οργανώνονται σε διάφορα topics. Για να αποσταλεί και να διαβαστεί οποιαδήποτε πληροφορία πρέπει να ανατρέξει κάποιος στο αντίστοιχο topic. Ουσιαστικά το topic μπορεί να μεταφραστεί σαν ένας πίνακας σε μία βάση δεδομένων. Ένα topic έχει συγκεκριμένο όνομα το οποίο πρέπει να είναι μοναδικό σε ένα σύμπλεγμα Kafka. Είναι το μέρος στο οποίο τα δεδομένα με την μορφή μηνυμάτων δημοσιεύονται από έναν producer (παραγωγό). Το topic χωρίζεται σε partitions (τμήματα), όπου κάθε ένα από αυτά τα τμήματα συντηρείται σε έναν broker (μεσίτη). Επίσης, το topic αντιγράφεται και αναπαράγεται αυτόματα

σύμφωνα με το αρχείο ρυθμίσεών μας για λόγους ανάκτησης δεδομένων από αντίγραφα ασφαλείας σε περίπτωση απώλειας δεδομένων από το κύριο σύστημα.



Εικόνα 2: Παράδειγμα ανάλυσης ενός άρθρου σε τμήματα

Σύμφωνα με την Apache, το σύμπλεγμα Kafka διατηρεί ένα δομημένο αρχείο καταγραφής για κάθε καταχώρηση για κάθε ένα από τα topics. Κάθε ένα διατμηματισμένο κομμάτι του topic διατηρείται σε αλληλουχία συνεχόμενων προσαρτημένων εγγραφών. Τα τμήματα αυτά είναι διαμοιρασμένα στους διάφορους διακομιστές του συμπλέγματος Kafka. Οι Kafka brokers είναι ανιθαγενείς (stateless) με αποτέλεσμα να μην καταγράφουν την κατανάλωση, αφήνοντας ένα μήνυμα για διαγραφή σε μία πολιτική ρυθμιζόμενης διατήρησης.

Ένα ακόμη βασικό στοιχείο του συστήματος Kafka είναι το record (εγγραφή). Τα δεδομένα στο συγκεκριμένο σύστημα απεικονίζονται ως records και όχι ως ροές και με αυτή τη μορφή εισέρχονται στα διάφορα topics. Κάθε μία εγγραφή αποτελείται από ένα κλειδί, την πληροφορία και μία χρονοσφραγίδα.

Το σύστημα Kafka διαθέτει 4 κύρια API τα οποία δίνουν τη δυνατότητα ανάπτυξης χρήσιμων εφαρμογών.

Producer API (Παραγωγός)

Μέσω αυτού δημιουργούνται εφαρμογές που δημοσιεύουν ροές από records σε ένα ή περισσότερα topics.

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.0.0</version>
</dependency>

```

Εικόνα 3: Τα maven dependencies για την ενσωμάτωση του Producer API

Consumer API (Καταναλωτής)

Δίνει την δυνατότητά σε εφαρμογές να εκδηλώσουν ενδιαφέρον σε ορισμένα topic και να επεξεργαστούν records που εισέρχονται σε αυτά.

Streams API

Αφορά την δημιουργία εφαρμογών που δρουν ως stream processors, καταναλώνοντας τα records κάποιου topic για να δημιουργήσουν καινούριες ροές records που αυτές με την σειρά τους θα αποθηκευτούν σε άλλα topics.

Connect API

Επιτρέπει την δημιουργία επαναχρησιμοποιήσιμων καταναλωτών ή παραγωγών για την σύνδεση των topics της Kafka με άλλες εφαρμογές ή υπολογιστικά συστήματα.

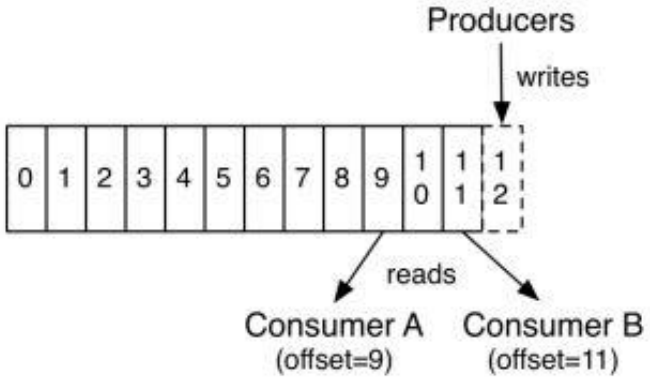
AdminClient API

Υποστηρίζει τη διαχείριση και επιθεώρηση των θεμάτων, μεσιτών, acls και όλων των υπόλοιπων αντικειμένων του Kafka συστήματος.

Legacy API

Αποτελεί ένα API περιορισμένης κληρονομιάς (legacy) τόσο για τον παραγωγό όσο και για τον καταναλωτή.

Οι γλώσσες προγραμματισμού που χρησιμοποιούνται κατά κόρον για ανάπτυξη εφαρμογών στο σύστημα Kafka είναι η Java και η Scala. Πραγματοποιείται όμως μεγάλη προσπάθεια τον τελευταίο καιρό για την ενσωμάτωση του Kafka σε όσο δυνατόν περισσότερα συστήματα με αποτέλεσμα πλέον να υποστηρίζονται κι άλλες γλώσσες προγραμματισμού όπως η Python, η C και τη C++.

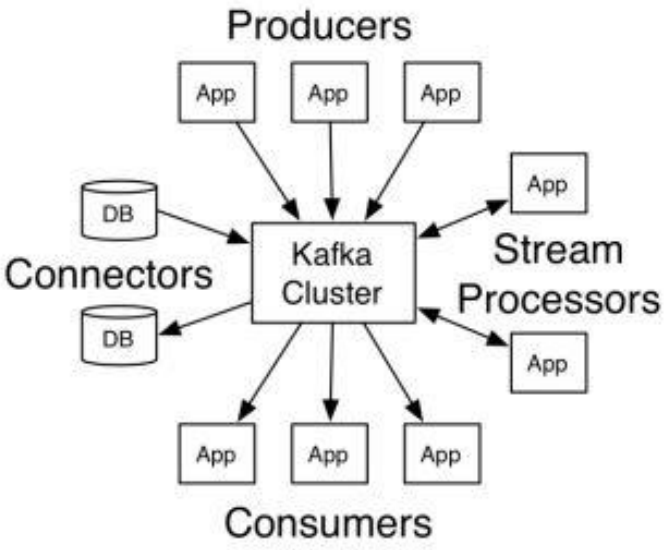


Εικόνα 4: Το μοντέλο παραγωγών/καταναλωτών στο Apache Kafka

Όπως ήδη ειπώθηκε το Kafka, όντας ένα κατανεμημένο σύστημα, τρέχει σε αρχιτεκτονική συμπλέγματος όπου κάθε κόμβος του συμπλέγματος ονομάζεται Kafka broker. Σε αυτό το σημείο πρέπει να αναλυθεί η ανατομία ενός topic του Kafka. Τα topics είναι διατμηματισμένα σε έναν αριθμό από τμήματα (partitions). Αυτό συμβαίνει για να υπάρχει παραλληλισμός. Επειδή, επιτρέπεται η ύπαρξη πολλών καταναλωτών σε πάνω από ένα topic, ο διαχωρισμός των δεδομένων και ο διαμοιρασμός τους σε διαφορετικούς brokers επιτρέπει σε διαφορετικούς consumers να διαβάζουν από το ίδιο topic παράλληλα.

Επιπροσθέτως, παρέχεται και η δυνατότητα πολλοί καταναλωτές να διαβάζουν διαφορετικά partitions παράλληλα με αποτέλεσμα να δημιουργείται πολύ υψηλή διακίνηση και γρήγορος ρυθμός επεξεργασίας δεδομένων.

Τα partitions του κάθε topic βρίσκονται ταυτόχρονα σε διαφορετικά σημεία σε μορφή αντιγράφων για να υπάρχει ανοχή σε σφάλματα. Οπότε τελικά, η ουσιαστική αποστολή των brokers είναι να διατηρούν έναν αριθμό από partitions και καθ' ένα από αυτά θα έχει είτε την ιδιότητα του πρωτότυπου είτε αυτή του αντιγράφου. Όλες οι αναγνώσεις και οι εγγραφές πρέπει να περάσουν από τον leader και αυτός θα συγχρονίσει όλα τα αντίγραφα του κατάλληλα. Αν για κάποιο λόγο υπάρχει αποτυχία από την πλευρά του leader, δηλαδή σταματήσει να λειτουργεί, ένα αντίγραφο θα πάρει την θέση του και δεν θα γίνει διακοπή των εργασιών που εκτελούνται.



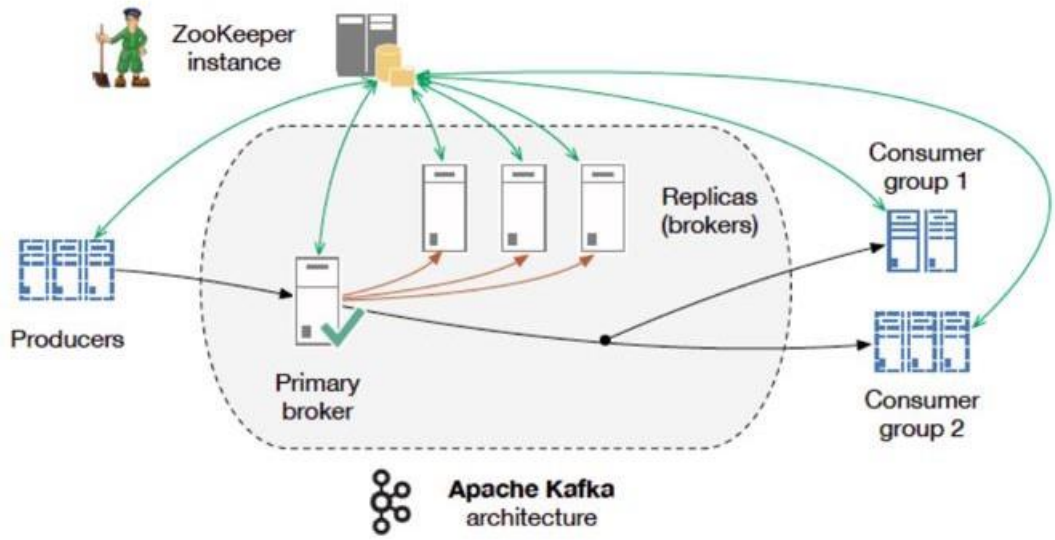
Εικόνα 5: Το σχεδιάγραμμα ενός συμπλέγματος Kafka

1.2 Apache ZooKeeper

Το Apache ZooKeeper είναι μία κατανεμημένη υπηρεσία συντονισμού για κατανεμημένες εφαρμογές. Ο κύριος ρόλος του είναι να συντονίζει τους κόμβους κάποιου συμπλέγματος και να διατηρεί τα δεδομένα που μοιράζονται μεταξύ αυτών χρησιμοποιώντας διάφορες τεχνικές συγχρονισμού.

Ορισμένες από τις υπηρεσίες που προσφέρει ο ZooKeeper είναι η ονοματοδοσία των κόμβων με τρόπο παρόμοιο όπως ένας DNS, ο συγχρονισμός σε πρόσβαση κοινών πόρων στο cluster, η αποθήκευση και διαχείριση της διαμόρφωσης ενός κατανεμημένου συστήματος και η εκλογή αρχηγού για την εκτέλεση αντιφατικών ενεργειών. Ο ZooKeeper όπως προαναφέρθηκε, αποτελεί κι αυτός ένα κατανεμημένο σύστημα από μόνο του. Ακολουθεί μοντέλο πελάτη-εξυπηρετητή κατά το οποίο οι πελάτες είναι οι κόμβοι (για παράδειγμα τα μηχανήματα) που χρησιμοποιούν την υπηρεσία και εξυπηρετητές οι κόμβοι που την προσφέρουν. Οι ZooKeeper διακομιστές ονομάζονται ensemble και μεταξύ αυτών υπάρχει ένας leader (ηγέτης) ο οποίος εκλέγεται μόλις ξεκινήσει η υπηρεσία.

Οι πελάτες μπορούν να συνδεθούν σε κάποιον ZooKeeper διακομιστή και ο κάθε εξυπηρετητής μπορεί να διαχειριστεί μεγάλο αριθμό πελατών. Αν κάποιος server σταματήσει να λειτουργεί οι clients που του αντιστοιχούν ανατίθενται σε κάποιον άλλο που ανήκει στο ensemble.

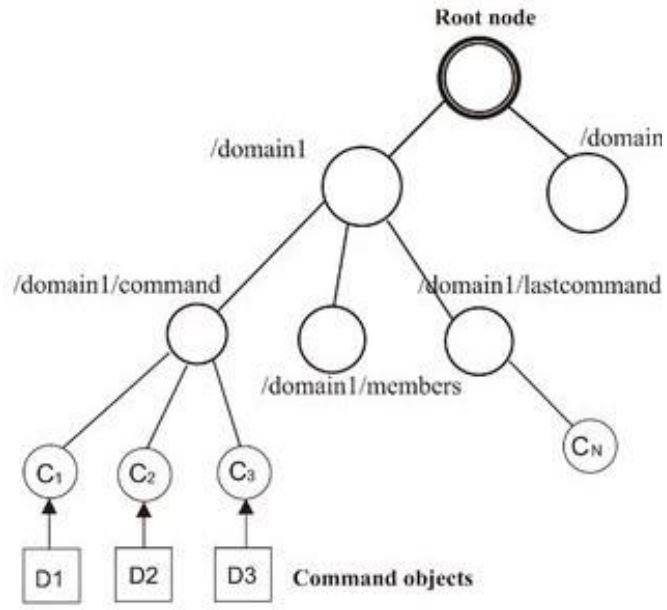


Εικόνα 6: Η αρχιτεκτονική ενός στιγμιότυπου ZooKeeper

Το συνολικό σύστημα του ZooKeeper μοιάζει με ένα σύστημα αρχείων ιεραρχικού μοντέλου και αποτελείται από τα λεγόμενα znodes (ZooKeeper data nodes). Αυτά μοιάζουν με αρχεία συστήματος UNIX με τη διαφορά ότι μπορούν να έχουν και κόμβους παιδιά. Η ιεραρχία αυτών είναι αποθηκευμένη στη μνήμη κάθε εξυπηρετητή ώστε να ικανοποιούνται γρήγορα τα αιτήματα ανάγνωσης των πελατών. Το μέγεθος των znodes δεν πρέπει να ξεπερνάει το 1 MB οπότε αποθηκεύονται μόνο οι απαραίτητες πληροφορίες που θα προσφέρουν αξιοπιστία, διαθεσιμότητα και συντονισμό στην κατανεμημένη εφαρμογή.

Η replicated βάση του ZooKeeper αποτελείται ουσιαστικά από ένα δέντρο από znodes τα οποία αποτελούν οντότητες που χονδρικά αναπαριστούν τους κόμβους του

συστήματος. Θα μπορούσε κάποιος να τα παρομοιάσει με ευρετήρια. Κάθε znode έχει τη δυνατότητα να εμπλουτιστεί με έναν πίνακα ψηφιολέξεων (byte array) οποίος αποθηκεύει δεδομένα.

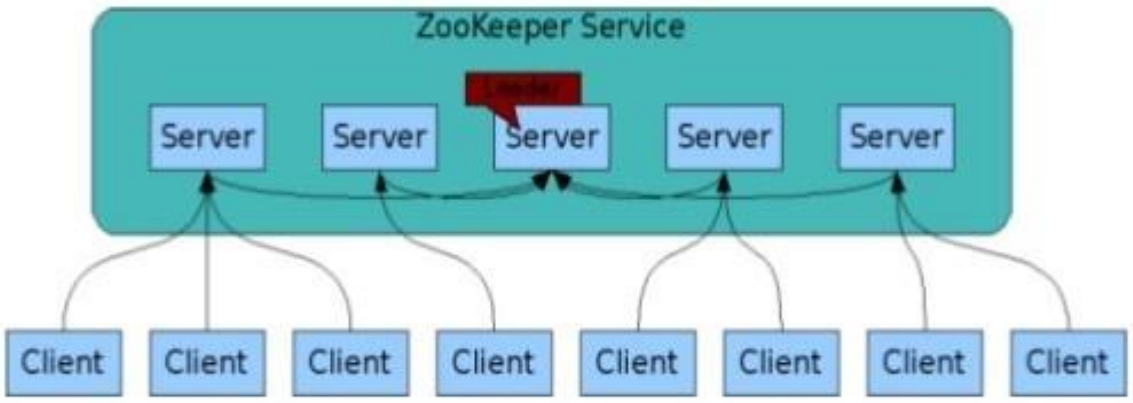


Εικόνα 7: Δεντρική αναπαράσταση znodes ενός παραδείγματος υλοποίησης ZooKeeper

Ο τύπος ενός znode μπορεί να είναι εφήμερος. Αυτό σημαίνει ότι καταστρέφεται μόλις αποσυνδεθεί ο πελάτης που το δημιούργησε. Αυτό είναι κυρίως χρήσιμο για να γνωρίζουμε πότε ένας πελάτης αποτυγχάνει, το οποίο μπορεί να είναι σχετικό όταν ο ίδιος ο πελάτης έχει ευθύνες που πρέπει να πάρει ένας νέος πελάτης. Λαμβάνοντας το παράδειγμα της κλειδαριάς, μόλις ο πελάτης ο οποίος έχει την κλειδαριά αποσυνδεθεί, οι άλλοι πελάτες μπορούν να ελέγξουν αν έχουν δικαίωμα στην κλειδαριά.

Επιπροσθέτως, το όνομα ενός znode μπορεί να είναι ακολουθιακό, κάτι το οποίο σημαίνει πως το όνομα το οποίο παρέχει ο πελάτης όταν δημιουργεί το znode είναι απλώς ένα πρόθεμα. Το όνομα στην πλήρη του μορφή επίσης δίνεται από έναν ακολουθιακό αριθμό επιλεγμένο από ένα σύνολο. Αυτό είναι ιδιαιτέρως χρήσιμο, για παράδειγμα, σε περιπτώσεις συγχρονισμού. Αν πολλαπλοί πελάτες επιθυμούν να κλειδώσουν και να κατοχυρώσουν έναν πόρο, μπορούν ταυτόχρονα να δημιουργήσουν ένα ακολουθιακό znode σε μία τοποθεσία. Οποιοσδήποτε έχει τον μικρότερο σε τιμή αριθμό, έχει τη δικαιοδοσία να κατοχυρώσει τον πόρο.

Ένα ακόμη χαρακτηριστικό του ZooKeeper είναι το ότι μπορούν να τεθούν σκανδάλες βασισμένες σε συμβάντα. Πιο συγκεκριμένα αυτά ονομάζονται «watches» και στην ουσία μπορούν να τεθούν και να ενεργοποιηθούν όταν πραγματοποιηθεί κάποιο συμβάν σε ένα znode. Αυτό το συμβάν μπορεί να είναι μία τροποποίηση σε κάποιο znode, είτε η αφαίρεση κάποιου ακόμη και η δημιουργία znodes κόμβων παιδιών κάτω από αυτόν που παρακολουθούμε.



Εικόνα 8: Η υπηρεσία ZooKeeper στην κατανεμημένη της μορφή

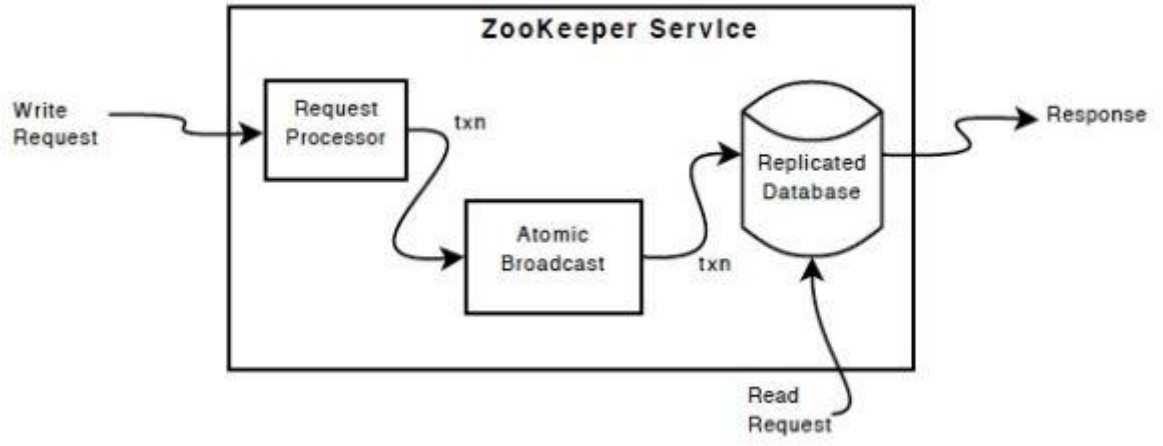
Το ZooKeeper ουσιαστικά αποτελεί μία υπηρεσία συγχρονισμού η οποία βρίσκεται σε αναπαραγωγή διατηρώντας αντίγραφα με τελική συνέπεια.

Πρόκειται για ένα εύρωστο σύστημα μιας και τα δεδομένα κατανέμονται μεταξύ πολλαπλών κόμβων (αυτό ονομάζεται ensemble) και ένας πελάτης συνδέεται σε οποιοδήποτε από αυτά (δηλαδή σε οποιονδήποτε από τους διακομιστές). Σε περίπτωση αποτυχίας ενός κόμβου ένας πελάτης θα μεταναστεύσει σε κάποιον άλλο. Όσο λειτουργεί μια αυστηρή πλειοψηφία κόμβων, το σύνολο των κόμβων του συστήματος ZooKeeper είναι ζωντανό. Συγκεκριμένα, ένας κύριος κόμβος επιλέγεται δυναμικά με συναίνεση εντός του συνόλου. Εάν ο κύριος κόμβος αποτύχει, ο ρόλος του κύριου μεταναστεύει σε άλλο κόμβο.

Ο master κόμβος έχει την εξουσία για να διευθύνει τις διαδικασίες που γράφουν δεδομένα. Κατ' αυτόν τον τρόπο οι διαδικασίες που γράφουν είναι εγγυημένες να παραμένουν στην τάξη, δηλαδή γραμμικά. Κάθε φορά που κάποιος πελάτης γράφει στο σύνολο, η πλειοψηφία των κόμβων διατηρεί τις πληροφορίες. Αυτοί οι κόμβοι συμπεριλαμβάνουν τους κόμβους του εξυπηρετητή για τον πελάτη και τον κόμβο master.

Αυτό σημαίνει ότι κάθε διαδικασία γραψίματος καθιστά τον εξυπηρετητή ενημερωμένο με τον κόμβο master. Επίσης σημαίνει πως δεν γίνεται να πραγματοποιούνται ταυτόχρονα διαδικασίες γραψίματος που να συμπίπτουν.

Η εγγύηση των γραμμικών εγγραφών είναι ο λόγος για τον οποίο ο ZooKeeper δεν έχει καλές επιδόσεις για φόρτο εργασίας που το μεγαλύτερο ποσοστό του αποτελείται από διαδικασίες γραψίματος. Ειδικότερα, δεν πρέπει να χρησιμοποιείται για την ανταλλαγή μεγάλων δεδομένων, όπως στις περιπτώσεις των δεδομένων των μέσων ενημέρωσης. Όσο η επικοινωνία περιλαμβάνει διαμοιρασμένα δεδομένα, το ZooKeeper μπορεί να βοηθήσει. Όμως, όταν τα δεδομένα πρέπει να γραφτούν ταυτόχρονα, το ZooKeeper αποτυγχάνει πραγματικά, επειδή επιβάλλει μια αυστηρή διάταξη των λειτουργιών, ακόμη και αν δεν είναι απολύτως απαραίτητη από την οπτική γωνία των κόμβων που γράφουν. Η ιδανική χρήση του είναι για τον συντονισμό, όπου ανταλλάσσονται μηνύματα μεταξύ των πελατών.



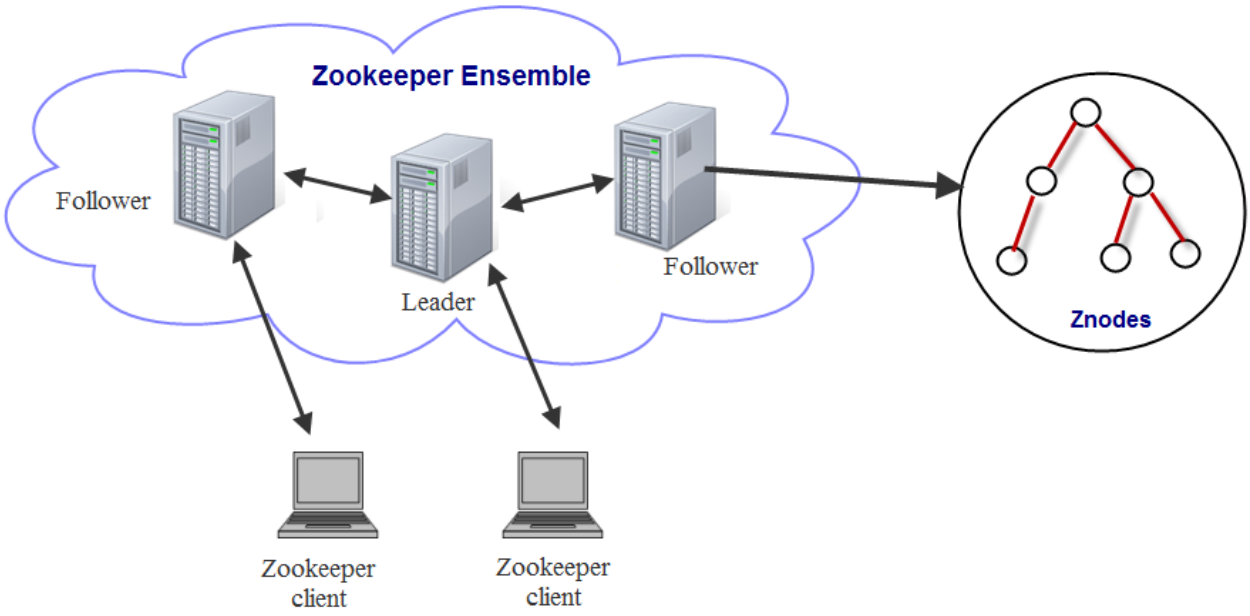
Εικόνα 9: Η διαδικασία ανάγνωσης και εγγραφής

Σχετικά με τη διαδικασία της ανάγνωσης των δεδομένων εδώ είναι το σημείο που το ZooKeeper υπερέχει. Οι αναγνώσεις πραγματοποιούνται ταυτόχρονα μιας και διακομίζονται από τον συγκεκριμένο διακομιστή στον οποίο έχει συνδεθεί ο πελάτης. Ταυτοχρόνως, αυτός είναι και ο λόγος για την τελική συνέπεια. Η όψη των δεδομένων για έναν πελάτη μπορεί να μην είναι ενημερωμένη μιας και ο master ενημερώνει τον αντίστοιχο διακομιστή με μία φραγμένη αλλά αόριστη καθυστέρηση.

Τα δεδομένα που είναι αποθηκευμένα στους znodes διαβάζονται και ενημερώνονται από τους clients. Στις αναγνώσεις εμπλέκεται μόνο ο server στον οποίο στάλθηκε το αίτημα. Έτσι οι αναγνώσεις γίνονται πολύ γρήγορα και μπορούν να πραγματοποιηθούν και παράλληλα.

Από την άλλη τα αιτήματα για εγγραφή περνούν όλα από τον leader. Όταν αποσταλεί ένα τέτοιο αίτημα ο server που το λαμβάνει το αποστέλλει στον εκλεγμένο αρχηγό και αυτός με την σειρά του το αναπαράγει σε όλους τους υπόλοιπους κόμβους. Αν υπάρχει απαρτία (quorum), δηλαδή απαντήσει θετικά η αυστηρή πλειοψηφία, τότε το αίτημα πραγματοποιείται και ενημερώνεται αντίστοιχα ο χρήστης. Σε περίπτωση που για κάποιο λόγο δεν υπάρχει απαρτία η υπηρεσία του ZooKeeper δεν είναι λειτουργική.

Η έννοια της απαρτίας ή, όπως ονομάζεται στο οικοσύστημα του ZooKeeper, quorum είναι πολύ σημαντική. Επειδή χρειαζόμαστε την αυστηρή πλειοψηφία των κόμβων να είναι λειτουργική επιλέγουμε μονούς αριθμούς για το πόσοι θα είναι οι servers. Αν, για παράδειγμα, είχαμε δύο τότε θα έπρεπε και οι δύο να λειτουργούν καθώς ο ένας από τους δύο δεν αποτελεί πλειοψηφία. Όμως, αν έχουμε τρείς χρειαζόμαστε μόνο δύο από αυτούς. Βέβαια, αυτό που παρατηρείται είναι ότι αν είχαμε τέσσερις τότε το σύστημα μας δεν επωφελείται καθόλου καθώς όπως και στην περίπτωση των τριών αν πέσουν δύο από αυτούς τότε δεν λειτουργεί η υπηρεσία. Γι’ αυτό το λόγο επιλέγονται οι μονοί αριθμοί όπως 3, 5, 7.



Εικόνα 10: Το ZooKeeper Ensemble και η οργάνωσή του

Σε γενικές γραμμές ένα κανονικό παράδειγμα χρήσης του ZooKeeper είναι προβλήματα υπολογισμού με κατανεμημένη μνήμη όπου ένα τμήμα των δεδομένων βρίσκεται μοιρασμένο σε τμήματα μεταξύ των κόμβων πελατών και πρέπει να προσπελαστεί και να ενημερωθεί με πολύ προσεκτικό τρόπο ενώ θα διασφαλίζεται ο συγχρονισμός.

Το ZooKeeper προσφέρει τη βιβλιοθήκη για την κατασκευή των πρωτότυπων συγχρονισμού, παράλληλα παρέχοντας τη δυνατότητα εκτέλεσης ενός κατανεμημένου εξυπηρετητή. Με αυτόν τον τρόπο αποφεύγεται το πρόβλημα ενός σημείου αποτυχίας που υπάρχει όταν χρησιμοποιείται ένα κεντρικό αποθετήριο μηνυμάτων.

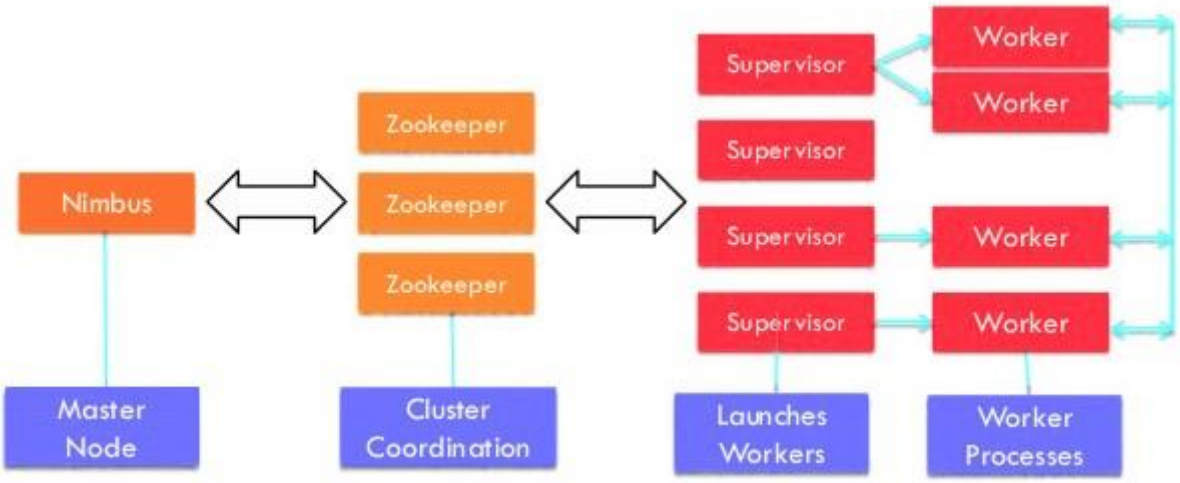
Το ZooKeeper είναι ένα ελαφρύ από την άποψη των χαρακτηριστικών του, που σημαίνει ότι δεν υπάρχουν ήδη μηχανισμοί, όπως εκλογές ηγέτη, κλειδαριές, εμπόδια κλπ. Όμως αν η φύση του προβλήματος που προσπαθούμε να επιλύσουμε το απαιτεί, μπορούν να γραφτούν πάνω από τα πρωτόγονα του ZooKeeper. Εάν η γλώσσα προγραμματισμού C και το JAVA API είναι αρκετά πολύπλοκα για τους σκοπούς του προβλήματος, θα πρέπει κανείς να βασιστεί σε βιβλιοθήκες που είναι χτισμένες στο ZooKeeper, όπως τα κλουβιά και ειδικά οι επιμελητές.

1.3 Apache Storm

Το Apache Storm είναι ένα ελεύθερο και ανοικτού κώδικα λογισμικό, ικανό να επεξεργάζεται μεγάλο όγκο δεδομένων γρήγορα και αποτελεσματικά Έχει σχεδιαστεί να εφαρμόζεται σε ροές δεδομένων και να ενσωματώνεται σε ποικίλες γλώσσες προγραμματισμού

Το Storm έχει πολυάριθμες χρήσεις όπως η ανάλυση δεδομένων σε πραγματικό χρόνο, μηχανική μάθηση σε πραγματικό χρόνο, συνεχείς υπολογισμοί, κτλ. Θεωρείται γρήγορο, αποδοτικό, ανεκτικό σε σφάλματα και εγγυάται πως η επεξεργασία των δεδομένων θα πραγματοποιηθεί με επιτυχία. Επίσης, είναι κλιμακώσιμο (scalable) και η εγκατάστασή του εύκολη και προφανής (straightforward).

Το Storm ενσωματώνεται σε τεχνολογίες επερωτήσεων και βάσεων δεδομένων γενικότερα που χρησιμοποιούνται ήδη ευρέως.



Εικόνα 11: Η αρχιτεκτονική του συστήματος Apache Storm

Επιφανειακά, ένα σύμπλεγμα (cluster) Storm μοιάζει αρκετά με ένα σύμπλεγμα Hadoop. Η διαφορά είναι πως αντί για τις διεργασίες MapReduce (MapReduce jobs) υπάρχουν οι τοπολογίες (topologies). Η βασική διαφορά μεταξύ τους είναι ότι ένα MapReduce job τρέχει έως ότου ολοκληρωθεί ενώ μία τοπολογία θα τρέχει για πάντα εκτός κι αν τη διακόψει ο χρήστης.

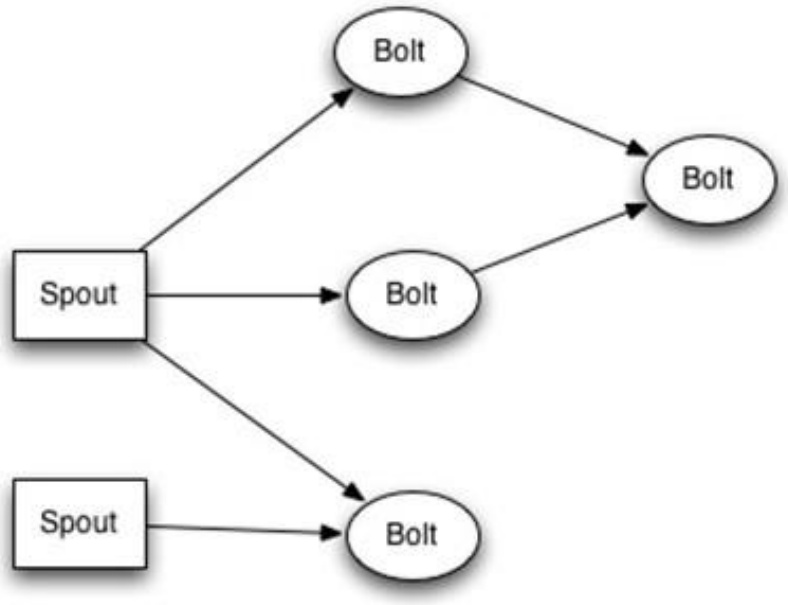
Ένα Storm cluster αποτελείται από δύο ειδών κόμβους: τον κόμβο αφέντη (master node) και τους κόμβους εργάτες (worker nodes). Η βασική ευθύνη του master node είναι το να τρέχει μία διαδικασία δαίμονα (daemon) που ονομάζεται “Nimbus”. Αυτή η διαδικασία είναι υπεύθυνη να διαμοιράζει τον κώδικα στο cluster, να αναθέτει εργασίες στα ξεχωριστά μηχανήματα και να επιβλέπει για αποτυχίες εκτέλεσης.

Κάθε ένας από τους worker nodes είναι υπεύθυνος να εκτελεί ένα daemon που ονομάζεται “Supervisor”. Αυτή η διαδικασία δέχεται μηνύματα σχετικά με εργασίες που είναι ανατεθειμένες στο συγκεκριμένο μηχάνημα και αναλαμβάνει να ξεκινά και να διακόπτει διεργασίες ανάλογα με τις ανάγκες βασιζόμενη σε αυτά που το Nimbus έχει αναθέσει. Κάθε worker job εκτελεί ένα υποσύνολο από τη τοπολογία όπου κάθε τοπολογία σε εκτέλεση αποτελείται από πολλές worker processes οι οποίες βρίσκονται διασκορπισμένες στα διάφορα μηχανήματα του cluster.

Το συντονισμό μεταξύ του Nimbus και των Supervisors τον αναλαμβάνει ένα cluster που ονομάζεται “ZooKeeper” και το οποίο περιεγράφηκε αναλυτικά προηγουμένως. Ολόκληρη η κατάσταση του συμπλέγματος βρίσκεται αποθηκευμένη είτε στο Zookeeper είτε στον τοπικό δίσκο. Αυτό σημαίνει πως μπορούμε να διακόψουμε τη λειτουργία των κόμβων “Nimbus” και “Supervisor” και αυτοί χωρίς κανένα πρόβλημα θα ξεκινήσουν πάλι τη λειτουργία τους. Αυτό είναι ένα από τα σημαντικότερα θετικά του Apache Storm μιας και το καθιστά εξαιρετικά σταθερό ως σύστημα.

Η βασική εννοιολογική αφαίρεση (abstraction) του Apache Storm είναι το ρεύμα (stream). Ένα stream είναι μία χωρίς όρια σειρά από πλειάδες (tuples). Το Apache Storm είναι υπεύθυνο να παρέχει τις βάσεις και τα θεμελιακά στοιχεία για την μεταμόρφωση (transformation) ενός ρεύματος σε ένα νέο ρεύμα με κατανεμημένο (distributed) και αξιόπιστο (reliable) τρόπο.

Τα βασικά στοιχεία που παρέχει το Apache Storm για την επίτευξη του μετασχηματισμού είναι τα “spouts” και τα “bolts”. Και τα δύο αυτά βασικά στοιχεία παρέχουν διεπαφές (interfaces) για εύκολη ενσωμάτωση στον κώδικα έτσι ώστε να εξυπηρετηθούν οι εξειδικευμένες από τη λογική του προβλήματος προδιαγραφές (application-specific logic).



Εικόνα 12: Ένα δίκτυο από spouts και bolts

Ένα spout είναι μία πηγή των streams. Είναι το σημείο εισόδου ή η πηγή ροών στην τοπολογία. Είναι υπεύθυνο για να έρχεται σε επαφή με την πραγματική πηγή δεδομένων, να λαμβάνει συνεχώς δεδομένα, να μετατρέπει αυτά τα δεδομένα σε πραγματικό ρεύμα πλειάδων και τελικά να τα στέλνει στα bolts που πρόκειται να υποστούν επεξεργασία.

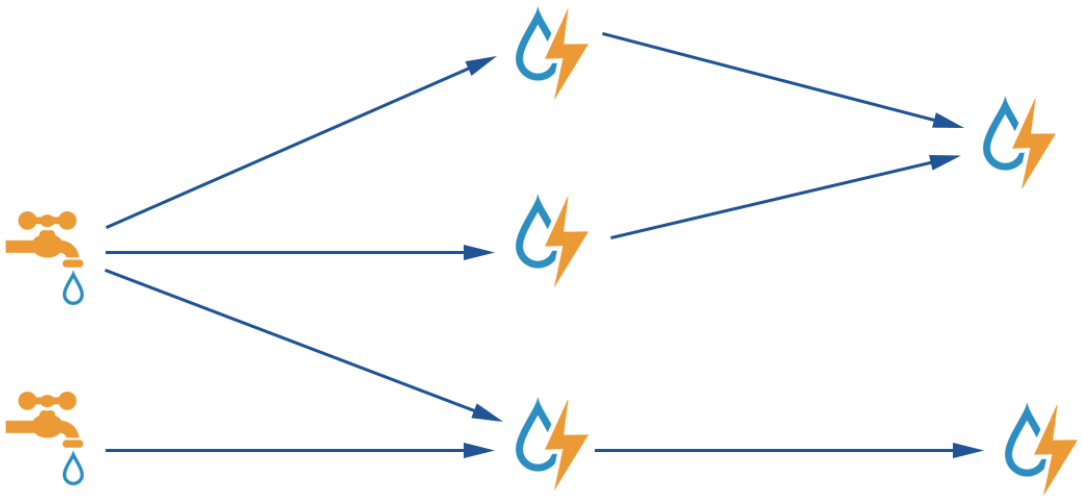
Ένα bolt καταναλώνει ένα αριθμό από spouts, τα επεξεργάζεται και σε ορισμένες περιπτώσεις εκπέμπει καινούρια streams. Είναι δυνατόν να πραγματοποιηθούν περίπλοκοι μετασχηματισμοί και υπολογισμοί σε αυτή τη διαδικασία οι οποίοι θα απαιτήσουν και πολυάριθμα βήματα, που σημαίνει πολυάριθμα bolts. Τα bolts μπορούν να πραγματοποιήσουν διάφορες διαδικασίες, από την εκτέλεση συναρτήσεων, το

φιλτράρισμα πλειάδων, τις συσσωματώσεις (aggregations) ρεύματος, συνενώσεις πηγών δεδομένων αλλά και επικοινωνία με βάσεις.

Τα bolts διατηρούν τη λογική που απαιτείται για την επεξεργασία. Αυτά είναι υπεύθυνα για την εκπομπή των ρευμάτων για επεξεργασία από άλλα bolts και την αποθήκευση ή την αποστολή των δεδομένων για αποθήκευση. Αυτά είναι ικανά να εκτελούν λειτουργίες, να φιλτράρουν πλειάδες, να συγκεντρώνουν και να συνδέουν ροές, να συνδέονται με βάση δεδομένων κ.λπ.

Δίκτυα από spouts και bolts συνεργάζονται για να δημιουργήσουν μία τοπολογία. Η τοπολογία αποτελεί ένα abstraction κορυφαίου επιπέδου που υποβάλλεται στα Storm clusters προς εκτέλεση. Ουσιαστικά η τοπολογία είναι ένας γράφος (graph) από μετασχηματισμούς ρεύματος όπου ο κάθε κόμβος είναι είτε ένα spout είτε ένα bolt. Πιθανές ακμές στο γράφο υποδεικνύουν ποια bolts συνεισφέρουν σε ποια streams. Όταν ένα spout ή ένα bolt στέλνουν μία πλειάδα σε ένα ρεύμα, τότε αποστέλλει την πλειάδα σε κάθε bolt που είναι συνδεδεμένο με αυτό το ρεύμα

Σύνδεσμοι μεταξύ κόμβων της τοπολογίας υποδεικνύουν το πόσες πλειάδες πρέπει να δια-μοιραστούν. Ανάλογα με τις ακμές, θα πραγματοποιούνται και αντίστοιχες αποστολές πλειάδων.



Εικόνα 13: Spouts και bolts συνθέτουν το σύστημα Storm

Η τοπολογία έχει σχεδιαστεί για να εκτελείται για πάντα εκτός κι αν διακοπεί από το διαχειριστή. Το Apache Storm είναι σχεδιασμένο να επαναναθέτει διεργασίες που έχουν αποτύχει. Επίσης, εγγυάται πως θα υπάρχει μηδαμινή απώλεια δεδομένων ακόμη κι αν τα μηχανήματα πάψουν να λειτουργούν ή αν τα μηνύματα χαθούν.

Οι ειδικοί στη βιομηχανία λογισμικού αναφέρουν το Storm ως το Hadoop για επεξεργασία σε δεδομένα πραγματικού χρόνου. Ενώ η επεξεργασία σε πραγματικό χρόνο ήταν ένα θέμα που είχε πάρει μεγάλες διαστάσεις μεταξύ των επαγγελματιών της επιχειρηματικής λογικής και των αναλυτών δεδομένων, το Apache Storm εξελίχθηκε με όλες εκείνες τις δυνατότητες που απαιτούνται για να στερεώσουν τις παραδοσιακές διαδικασίες.

Στη συνέχεια παρουσιάζονται παρακάτω τα χαρακτηριστικά που το καθιστούν κατάλληλο για επεξεργασία σε πραγματικό χρόνο.

Αρχικά, το Storm UI REST API αποτελεί ένα REST API που παρέχεται από το Storm UI

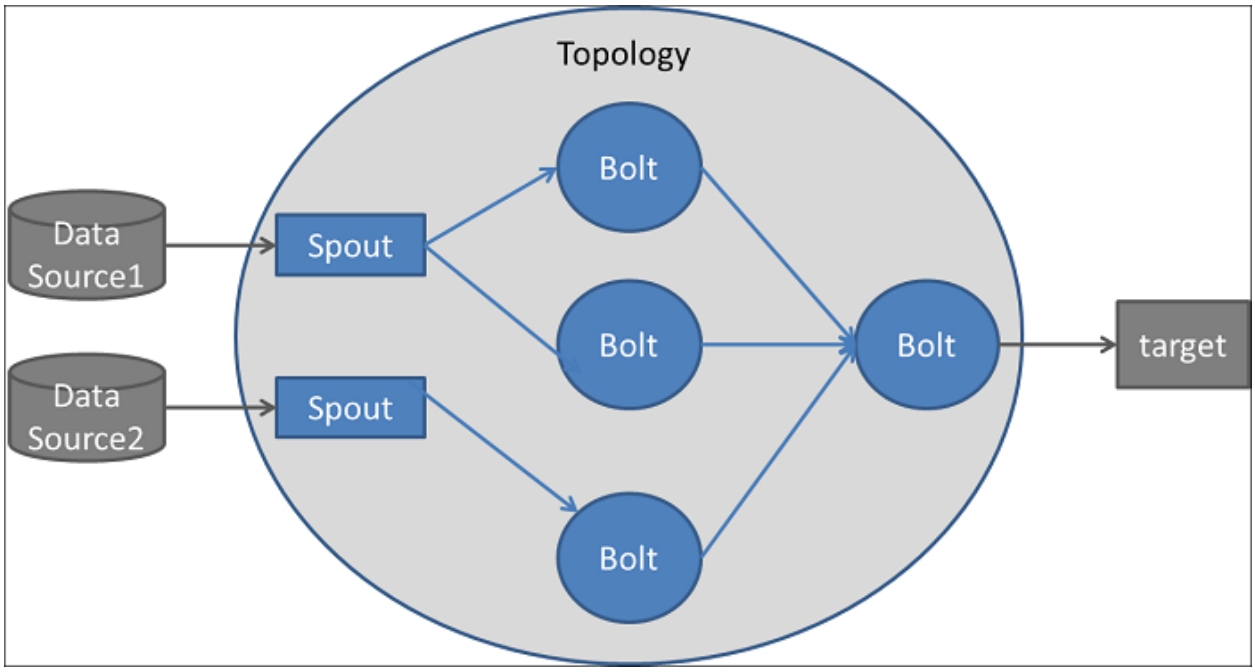
daemon και επιτρέπει στους προγραμματιστές να αλληλοεπιδρούν με ένα σύμπλεγμα Storm. Αυτό περιλαμβάνει την ανάκτηση δεδομένων μετρήσεων και πληροφοριών διαμόρφωσης, καθώς και λειτουργίες διαχείρισης όπως εκκίνηση ή διακοπή τοπολογιών. Είναι ικανό να επεξεργάζεται 1 εκατομμύριο μηνύματα των 100 bytes σε έναν και μοναδικό κόμβο. Μπορεί να δουλεύει σε λογική γρήγορης και άμεσης επανεκκίνησης σε περίπτωση αποτυχίας.

Επίσης, κάθε κόμβος επεξεργάζεται τουλάχιστον μία φορά ακόμη κι αν συμβεί κάποιο σφάλμα.

Το Storm είναι εξαιρετικά κλιμακωτό με την ικανότητα να συνεχίζει τους υπολογισμούς παράλληλα με την ίδια ταχύτητα υπό αυξημένο φορτίο. Τούτου λεχθέντος, είναι σημαντικό να σημειωθεί ότι έχει δημιουργήσει ένα σημείο αναφοράς για την επεξεργασία 1 εκατομμυρίου μηνυμάτων με μέγεθος 100 bytes σε έναν μοναδικό κόμβο που καθιστά την πλατφόρμα επεκτάσιμη και ταχεία. Εξοπλισμένο με τα χαρακτηριστικά της ταχύτητας και της κλιμάκωσης, αυτή η τεχνολογία υπερπηδά σε δυνατότητες άλλες υπάρχουσες όταν έρχεται η επεξεργασία όγκων δεδομένων με μια άνευ προηγουμένου ταχύτητα.

Όχι μόνο αυτό, αλλά το Storm βασίζεται στην προσέγγιση "αποτυχία γρήγορης επανεκκίνησης", η οποία της επιτρέπει να ξεκινήσει εκ νέου τη διαδικασία μόλις αποτύχει ένας κόμβος χωρίς να διαταράξει ολόκληρη τη λειτουργία, καθιστώντας την μηχανή του ανθεκτική σε σφάλματα. Εξασφαλίζει ότι κάθε πλειάδα θα υποβληθεί σε επεξεργασία "τουλάχιστον μία φορά ή ακριβώς μία φορά" ακόμη και αν κάποιος από τους κόμβους αποτύχει ή χάσει ένα μήνυμα.

Η τυποποιημένη διαμόρφωση του Storm το καθιστά κατάλληλο για άμεση παραγωγή. Μόλις αναπτυχθεί το σύμπλεγμα Storm, μπορεί να λειτουργήσει εύκολα. Επιπλέον, είναι μια στιβαρή και φιλική προς το χρήστη τεχνολογία που το καθιστά κατάλληλο τόσο για μικρές όσο και για μεγάλες επιχειρήσεις.



Εικόνα 14: Ανάλυση ενός συμπλέγματος Apache Storm

Το Storm, ακόμη κι αν παρουσιάζεται ως το σύστημα που είναι τέλειο, είναι ανοικτό σε

μελλοντικές αναβαθμίσεις. Πιο συγκεκριμένα μερικές από τις πιθανές αναβαθμίσεις αυτές που μπορούν να καταστήσουν το Apache Storm πλουσιότερο σε σχέση με χαρακτηριστικά που θα οδηγήσουν τη ζήτηση και την αγορά του παρουσιάζονται παρακάτω.

Αρχικά, η ενσωμάτωση με το YARN. Τα πλεονεκτήματα του YARN έχουν ήδη αναθεωρήσει τον Hadoop με την εφαρμογή νέων υπηρεσιών στην υποδομή του. Ομοίως, το YARN θα ενσωματωθεί με το Storm, καθιστώντας το, πιο βιώσιμο για τους προγραμματιστές, καθώς θα ήταν σε θέση να επικεντρωθεί στις εφαρμογές τους ανεξάρτητα από την υλοποιούμενη υποδομή.

Έπειτα να ενσωματωθούν περισσότερες διατάξεις ασφαλείας. Το Apache Storm θεωρήθηκε αρχικά ως επιρρεπές σε απειλές και λιγότερο ασφαλές. Αλλά πρόκειται να καλύψει αυτό το κενό τα επόμενα χρόνια με τα ακόλουθα χαρακτηριστικά:

- Έλεγχος ταυτότητας Kerberos με αυτόματη ενεργοποίηση πιστοποίησης και ανανέωση
- Χρονοδρομολόγηση πολλαπλών ενοίκων
- Ασφαλής ενσωμάτωση με άλλα συστήματα Hadoop (όπως ZooKeeper, HDFS, HBase κλπ.)
- Απομόνωση χρηστών διότι οι τοπολογίες Storm λειτουργούν μέσω του χρήστη που τις υπέβαλε
- Αύξηση της επεκτασιμότητας: Το Storm παρέχει ήδη στους χρήστες της δυνατότητα επέκτασης με λιγότερους από 20 κόμβους για να επεξεργαστεί ένα εκατομμύριο διεργασίες ανά δευτερόλεπτο. Αλλά αυτή η ταχύτητα θα αυξηθεί τα επόμενα χρόνια καθώς πρόκειται να κλιμακωθεί μέχρι σε και αρκετές χιλιάδες κόμβους για επεξεργασία σε πραγματικό χρόνο.
- Τέλος, θα ήταν καλό να ενσωματωθούν περισσότερες γλώσσες προγραμματισμού. Το Apache Storm θεωρείται ότι είναι υποστηρικτικό για πολυάριθμες γλώσσες προγραμματισμού οι οποίες θα αυξηθούν στο μέλλον, για να διευκολύνουν την αύξηση της παραγωγικότητας των χρηστών τους.

1.4 Esper

Το Esper είναι ένα ανοιχτού κώδικα λογισμικό βασισμένο στη γλώσσα προγραμματισμού JAVA. Είναι σχεδιασμένο για τη διαδικασία της πολύπλοκης επεξεργασίας συμβάντων (complex event processing) και της επεξεργασίας συμβάντων ρεύματος. Κατά τη διάρκεια αυτών των διαδικασιών αναλύονται σειρές συμβάντων με σκοπό την ανακάλυψη χρήσιμων για εμάς συμπερασμάτων.

Το Esper επεκτείνει το SQL-62 πρότυπο (standard) για τη μηχανή και το πλαίσιο λογισμικού του παρέχοντας τη λειτουργία επεξεργασίας συνόλων (aggregate function), το ταίριασμα των προτύπων (pattern matching), την παραθυροποίηση συμβάντων (event windowing) και τη συνένωση (joining).

Αυτό το λογισμικό δημιουργήθηκε το 2006 από την εταιρεία EsperTech Inc. Προσφέρει μία γλώσσα εξειδικευμένη για τον τομέα της επεξεργασίας συμβάντων και για αυτό το λόγο αποτελεί μία Γλώσσα Επεξεργασίας Συμβάντων (EPL). Είναι μία γλώσσα δηλωτικού προγραμματισμού (declarative programming language) η οποία αναλύει δεδομένα συμβάντων βασισμένα στη μεταβλητή του χρόνου ικανή να ανιχνεύει καταστάσεις καθώς προκαλούνται.

Τα σημαντικότερα θετικά σημεία του λογισμικού Esper είναι το ότι παρέχει ένα μεγάλο βαθμό κλιμάκωσης, είναι αρκετά αποδοτικό σε θέματα διαχείρισης μνήμης, υποστηρίζει το πρότυπο SQL, έχει ελάχιστες καθυστερήσεις και τέλος το ότι είναι ένα λογισμικό ικανό να εφαρμόζεται σε ροές δεδομένων πραγματικού χρόνου.

Τα παραπάνω το καθιστούν ένα ολοκληρωμένο λογισμικό το οποίο μπορεί να εφαρμοστεί σχετικά εύκολα, ευθέως και με αποδοτικό τρόπο για την επίλυση πολυάριθμων προβλημάτων. Είναι μία μηχανή επεξεργασίας και ανάλυσης μεγάλων δεδομένων ικανή να εφαρμοστεί και να επεξεργαστεί δεδομένα οποιουδήποτε όγκου και οποιασδήποτε ποικιλομορφίας καθώς και ιστορικά δεδομένα.

Το Esper δεν είναι περιορισμένο στο να εκτελείται σε ένα και μοναδικό μηχάνημα. Έχει σχεδιαστεί να αναπτύσσεται μέσα σε κατανεμημένα πλαίσια ανάλυσης ροών δεδομένων. Αν και μπορεί να χρησιμοποιηθεί και σε απλά προβλήματα, μεγάλη σημασία έχει πως είναι ικανό να εκτελεστεί σε οποιαδήποτε αρχιτεκτονική και σε οποιοδήποτε κιβώτιο (container) καθώς δεν έχει εξαρτήσεις (dependencies) σε εξωτερικές διαδικασίες και υπηρεσίες και δεν προαπαιτεί την ύπαρξη κάποιου συγκεκριμένου μοντέλου νημάτων (threading model) ή κάποιου μοντέλου του πως προκαταβάλλεται ο χρόνος και δεν προαπαιτεί κάποιο εξωτερικό χώρο αποθήκευσης. Επιπροσθέτως, το Esper συνεργάζεται καλά με διαχείριση χρόνου είτε συμβάντος-χρόνου είτε βασισμένης σε υδατογράφημα.

Η αρχιτεκτονική του είναι οριζόντια με γραμμική και οριζόντια δυνατότητα κλιμάκωσης, Η κλιμάκωση επίσης είναι ελαστική και υποστηρίζεται κατανομή του φόρτου επεξεργασίας, η λειτουργία της εξισορρόπησης καθώς και της αντίστροφης διαδικασίας (balancing, re-balancing). Εκτός από αυτές τις λειτουργίες, υπάρχει ανοχή σε σφάλματα έτσι ώστε να μην διακόπτεται η εκτέλεση αν εντοπιστεί κάποιο σφάλμα. Υπάρχει η δυνατότητα της δυναμικής ανακάλυψης των κόμβων μέσω κόμβων-σπόρων (seed-nodes) και της αναπαραγωγής έτσι ώστε να εξασφαλίζονται η εγκυρότητα και η ασφάλεια των δεδομένων επεξεργασίας. Τέλος, υπάρχει η δυνατότητα υποστήριξης πολλαπλών κέντρων δεδομένων (datacenters).

Ορισμένα τυπικά παραδείγματα χρήσης του λογισμικού Esper είναι τα παρακάτω:

- Διαχείριση επιχειρηματικών διαδικασιών και αυτοματοποίηση των διαδικασιών (παρακολούθηση διαδικασιών επεξεργασίας, εξαιρέσεις αναφοράς, επιχειρησιακή νοημοσύνη)
- Τομέας των οικονομικών (αλγοριθμικό εμπόριο, ανίχνευση απάτης, διαχείριση ρίσκου)
- Παρακολούθηση διαδικασιών δικτύου και και εφαρμογών (ανίχνευση εισβολών)
- Εφαρμογές δικτύου αισθητήρων (διάβασμα RFID, προγραμματισμός και έλεγχος γραμμών κατασκευής, εναέρια κίνηση)

Τα βασικά στοιχεία του Esper περιγράφονται παρακάτω.

Ένα από τα βασικά χαρακτηριστικά του Esper είναι το *Byte Code Generation*. Αυτό το χαρακτηριστικό αποτελεί μία τεχνική η οποία αναμειγνύει τις τελευταίες και πιο σύγχρονες τεχνολογικά τεχνικές από τους σύγχρονους μεταγλωττιστές και από τις βάσεις MPP. Μέσω αυτής της λειτουργίας, επιτυγχάνεται αισθητά καλύτερη ταχύτητα επεξεργασίας καθώς εξολοθρεύονται πολλαπλές εικονικές κλήσεις, διανομές ρόλων και διακλαδώσεις

Επιπροσθέτως, επιτρέπεται με αυτόν τον τρόπο στο εικονικό μηχάνημα να βελτιστοποιήσει τον παραγόμενο κώδικα και στο υλικό να εκτελεί τις εντολές με μεγαλύτερη ταχύτητα. Το Esper ενσωματώνει την καλύτερη αρχιτεκτονική για μηχανική εκτέλεση κατά την επεξεργασία δεδομένων εκτελώντας τη λειτουργία αυτή. Δυστυχώς, όχι όλα τα φορτία εργασίας μπορούν να επωφεληθούν από τη λειτουργία αυτή κατά τον ίδιο βαθμό.

Ένα δεύτερο βασικό χαρακτηριστικό του Esper είναι τα *Data Windows*. Μέσω αυτών, πραγματοποιείται η διαχείριση ή λήξη των συμβάντων με αποδοτικό τρόπο. Οδηγούν την μηχανή σχετικά με το χρόνο για τον οποίο διατηρούνται σχετικά συμβάντα ή τις προϋποθέσεις κάτω από τις οποίες ορισμένα συμβάντα θα απορριφθούν. Λειτουργούν στο επίπεδο των ατομικών επερωτημάτων, ροών δεδομένων και υποερωτημάτων. Για παράδειγμα, ένα data window μπορεί να χρησιμοποιηθεί έτσι ώστε να διατηρούνται τα συμβάντα για ένα συγκεκριμένο και ορισμένο χρονικό διάστημα. Σε αυτήν την περίπτωση, η μηχανή αφήνει να φύγουν (λήγει) διάφορα συμβάντα που έχουν ξεπεράσει τον προκαθορισμένο από το χρήστη χρόνο. Έχοντας μία ικανοποιητική ποικιλία από μεταβλητά και συνδυάσιμα data windows μας επιτρέπει να διευθύνουμε περισσότερες απαιτήσεις ανάλυσης και να διευθύνουμε τις κοινές/συνηθισμένες απαιτήσεις σε σύντομο χρονικό διάστημα.

Ένα τρίτο χαρακτηριστικό είναι τα *Named Windows*. Αυτά είναι ευρέως ορατά data windows τα οποία επιτρέπουν το διαμοιρασμό των συμβάντων μεταξύ των επερωτημάτων με αποδοτικό τρόπο. Με αυτόν τον τρόπο απαλείφεται η ανάγκη για τη διατήρηση παρόμοιων συμβάντων σε πολλαπλά σημεία. Επιτρέπεται ο ορισμός της οδηγίας για λήξη των συμβάντων μία φορά με πολλαπλές εφαρμογές.

Ένα τέταρτο βασικό στοιχείο είναι τα *Tables*. Αυτά είναι δομές δεδομένων οι οποίες είναι ικανές να αποθηκεύουν καταστάσεις συσσωμάτωσης κατά μήκος των δεδομένων και των συμβάντων και επιτρέπουν την ενημέρωση στο σημείο. Καθιστούν εφικτές τις συσσωματώσεις πολλαπλών εντολών στην ίδια κατάσταση. Η συγκατοίκηση και η ενημέρωση στο σημείο των δεδομένων μπορούν να βοηθήσουν αισθητά στην αύξηση της απόδοσης αλλά και στη μείωση της χρήσης της συνολικής μνήμης.

Η βασική λειτουργία του Esper είναι η ανάλυση σειρών από γεγονότα τα οποία μπορούν να αποτελούν είτε ιστορικά γεγονότα είτε γεγονότα ροής χρόνου. Υποστηρίζονται διάφορες λειτουργίες σχετικές με την ανάλυση των γεγονότων. Αρχικά,

υποστηρίζεται το *match-recognize* το οποίο ουσιαστικά είναι ένα μοντέλο επερωτημάτων σχεδιασμένο για ταίριασμα προτύπων βασισμένο σε κανονικές εκφράσεις. Οι κανονικές εκφράσεις είναι σημαντικό να υποστηρίζονται καθώς θεωρείται σχετικά εύκολο να γίνουν κατανοητές. Αρκετά πρότυπα μπορούν να αναπαρασταθούν με ακρίβεια και συνέπεια χρησιμοποιώντας την παραπάνω λειτουργία.

Μία δεύτερη λειτουργία που υποστηρίζεται είναι τα *patterns (πρότυπα)*. Αυτά αποτελούν μία γλώσσα η οποία παρέχει συσχέτιση λογικού και προσωρινού χρόνου. Ο κύκλος χρόνου των προτύπων είναι δυνατόν να ελεγχθεί με τη χρήση χρονομετρητή και διαφόρων συντελεστών. Τα patterns προσφέρουν έναν εκφραστικό τρόπο να καθοριστούν πιο πολύπλοκες σχέσεις χρόνου και συσχετίσεων. Τέλος, τα πρότυπα συχνά συνδυάζονται με άλλα ρεύματα δεδομένων ή χρησιμοποιούνται ως σκανδάλες που εκκινούν διάφορες άλλες λειτουργίες. Πρότυπα που χρησιμοποιούν αυτήν την λειτουργία είναι τα πρότυπα επαναλαμβανόμενου χρόνου.

Διάφορες λοιπές λειτουργίες που υποστηρίζονται είναι η *ομαδοποίηση (grouping)*, *συσσωμάτωση (aggregation)*, *συλλογή (rollup)*, *κύβος (cube)*, *ταξινόμηση (sorting)*, *φιλτράρισμα (filtering)*, *μεταμόρφωση (transforming)*, *συγχώνευση (merging)*, *διαχωρισμός (splitting)* και *διπλασιασμός (duplicating)* είτε των σειρών των συμβάντων είτε των ροών.

Εκτός από τα προαναφερθέντα, ορισμένες από τις σημαντικές λειτουργίες είναι οι παρακάτω:

- Περιορισμός και σταθεροποίηση του ρυθμού εξόδου
- Η κατανάλωση συμβάντων βρίσκεται υπό έλεγχο
- Μέθοδοι απαρίθμησης
- Μέθοδοι ημέρας/χρόνου
- Άλγεβρα διαστημάτων του Allan
- Δηλωμένες εκφράσεις
- Ενσωμάτωση τμημάτων κώδικα (scripts)
- Συνένωση εξωτερικών δεδομένων
- Μεταβλητές και σταθερές
- Προσεγγιστικοί αλγόριθμοι

Το Esper έχει πολλαπλές χρήσεις και πολυάριθμα οφέλη καθιστώντας το μία γλώσσα προγραμματισμού αναγκαία και χρήσιμη για την επίλυση συγκεκριμένων προβλημάτων. Η ικανότητά του να συνδυάζει την επεξεργασία συμβάντων ροών δεδομένων (filtering, joins, aggregation) και την πολύπλοκη επεξεργασία συμβάντων σε μία γλώσσα προγραμματισμού καθιστούν το Esper καινοτόμα και χρήσιμη τεχνολογία.

Η γλώσσα προγραμματισμού που χρησιμοποιείται ως βάση συμφωνεί με τα πρότυπα SQL με αποτέλεσμα η εκμάθησή της να είναι γρήγορη και εύκολη και ταυτόχρονα υψηλά προσανατολισμένη προς την υποστήριξη των νέων τεχνολογιών. Επίσης αποτελεί μία αντικειμενοστραφή γλώσσα με αποτέλεσμα να είναι εύκολα επεκτάσιμη.

Τα προβλήματα που λύνει με μέγιστη απόδοση είναι τα πραγματικού χρόνου οδηγούμενα από συμβάντα. Ορισμένα τυπικά πεδία εφαρμογής είναι η διαχείριση επιχειρηματικών διαδικασιών, ο αυτοματισμός, τα οικονομικά, τα δίκτυα, η παρακολούθηση εφαρμογών και οι εφαρμογές αισθητήρων. Η βασική διαφορά με τα συστήματα που εξαρτώνται σε κλασικές SQL βάσεις είναι το γεγονός πως δεν εκτελούνται επερωτήματα σε μια αποθήκη συμβάντων που πληρούν ορισμένα φίλτρα

αναζήτησης αλλά η αναζήτηση αυτή πραγματοποιείται σε ενέργειες προσαρμοσμένες με triggers σε πραγματικό χρόνο, όπως συμβαίνουν τα γεγονότα, μειώνοντας αισθητά τη χρονική καθυστέρηση.

Η λειτουργία του Esper μπορεί να γίνει εύκολα κατανοητή αν αναλυθεί ένα παράδειγμα χρήσης.

Παρακάτω αναλύεται ένα παράδειγμα χρήσης όπου το Esper χρησιμοποιείται για να ανιχνεύονται συμβάντα σχετικά με την παρακολούθηση, προειδοποίηση πιθανού σφάλματος και αναγγελία σημαντικού προβλήματος σχετικά με τη θερμοκρασία ενός συστήματος.

Τα παραδείγματα που θα εξεταστούν είναι τα παρακάτω:

- Πληροφόρηση σχετικά με τη μέση θερμοκρασία κάθε 10 δευτερόλεπτα για ενημερωτικούς σκοπούς
- Προειδοποίηση για την περίπτωση όπου 2 συνεχόμενες τιμές της ημερομηνίας υπερβαίνουν ένα συγκεκριμένο και προκαθορισμένο όριο
- Ειδοποίηση κινδύνου στην περίπτωση όπου αναγνωριστούν 4 συνεχόμενα συμβάντα έχοντας την πρώτη τιμή κάτω από το προκαθορισμένο όριο και κάθε επόμενη μεγαλύτερη από την προηγούμενή της και την τελευταία 1,5 φορές μεγαλύτερη από την πρώτη. Έτσι ορίζεται στο παράδειγμα ένα ξαφνικό και με αυξανόμενη κλιμάκωση συμβάν.

Τα απαραίτητα βήματα για την μοντελοποίηση του προβλήματος που πρέπει να πραγματοποιηθούν είναι τα παρακάτω:

- Χρησιμοποιώντας το Esper γίνεται ο ορισμός 3 επερωτημάτων που θα περιγράψουν τα προαναφερθέντα πρότυπα
- Ένας listener συνάπτεται σε κάθε ένα από τα επερωτήματα. Κάθε φορά που θα ικανοποιούνται οι συνθήκες για το κάθε ένα από αυτά θα ενεργοποιείται η αντίστοιχη σκανδάλη
- Δημιουργείται μία υπηρεσία Esper η οποία καταγράφει τα επερωτήματα αυτά καθώς και τους listeners
- Το επόμενο βήμα είναι να τροφοδοτηθεί το σύστημα με δεδομένα θερμοκρασίας και να επιτραπεί στο Esper να ενημερώσει τους listeners όταν ικανοποιούνται οι συνθήκες ενεργοποίησης των σκανδαλών.

Τα επερωτήματα παρουσιάζονται παρακάτω:

Επερώτημα #1

```
select avg(value) as avg_val
from TemperatureEvent.win:time_batch(10 sec)
```

Επερώτημα #2

```
select * from TemperatureEvent "
 match_recognize (
   measures A as temp1, B as temp2
   pattern (A B)
   define
     A as A.temperature > 400,
     B as B.temperature > 400)
```

Επερώτημα #3

```
select * from TemperatureEvent
match_recognize (
measures A as temp1, B as temp2, C as temp3, D as temp4
pattern (A B C D)
define
A as A.temperature > 100,
B as (A.temperature < B.value),
C as (B.temperature < C.value),
D as (C.temperature < D.value) and D.value >
(A.value * 1.5))
```

1.5 ElasticSearch

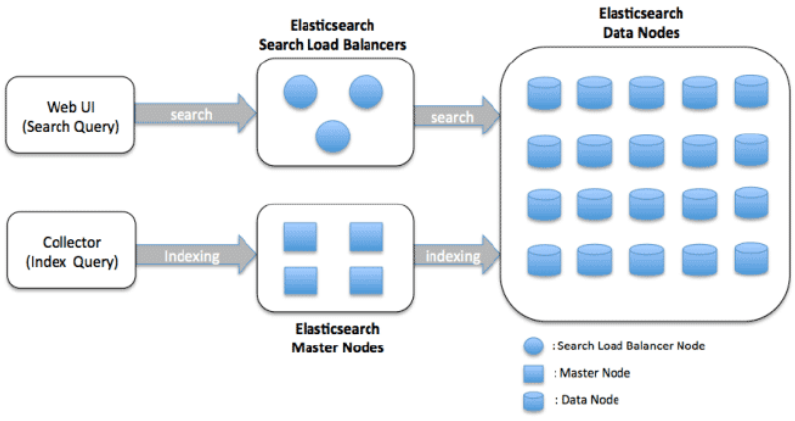
Το Elasticsearch είναι μια μηχανή αναζήτησης ανοικτού λογισμικού, ευρέως διανεμητή, εύκολα κλιμακούμενη και επιχειρηματική. Προσβάσιμο μέσω ενός εκτεταμένου και περίπλοκου API, το Elasticsearch μπορεί να τροφοδοτήσει εξαιρετικά γρήγορες αναζητήσεις που υποστηρίζουν τις εφαρμογές για την ανεύρεση δεδομένων.

Ένα σύμπλεγμα είναι μια ομάδα εξυπηρετητών Elasticsearch. Αυτοί οι διακομιστές ονομάζονται κόμβοι. Ανάλογα με τις προτιμήσεις χρήσης και τις δυνατότητες κλιμάκωσης, ένα σύμπλεγμα μπορεί να έχει οποιοδήποτε αριθμό κόμβων. Κάθε κόμβος αναγνωρίζεται με ένα μοναδικό όνομα. Όλα τα δεδομένα κατανέμονται μεταξύ αυτών των κόμβων οι οποίοι με τη σειρά τους ομαδοποιούνται σε διαφορετικές ομάδες. Ένα σύμπλεγμα καθιστά δυνατή τη δημιουργία ευρετηρίων αλλά και την αναζήτηση στα αποθηκευμένα δεδομένα.

Ένας κόμβος είναι ένας διακομιστής. Δηλαδή, είναι μια ενιαία μονάδα του συμπλέγματος. Αν είναι ένα σύμπλεγμα μονάδων κόμβων, όλα τα δεδομένα θα αποθηκεύονται σε αυτόν τον μοναδικό κόμβο. Αλλιώς τα δεδομένα θα διανεμηθούν μεταξύ των διαφόρων κόμβων που αποτελούν το τμήμα αυτού του συμπλέγματος. Οι κόμβοι συμμετέχουν στις δυνατότητες αναζήτησης και ευρετηρίασης ενός συμπλέγματος. Ανάλογα με τον τύπο του ερωτήματος που εκτελέστηκε, πραγματοποιείται συνεργασία και έπειτα επιστρέφεται η αντίστοιχη απάντηση.

Ένα ευρετήριο είναι μια συλλογή ή ομαδοποίηση εγγράφων. Το ευρετήριο έχει μια ιδιότητα που ονομάζεται τύπος. Σε όρους σχεσιακής βάσης δεδομένων, ένα ευρετήριο είναι κάτι σαν βάση δεδομένων και ο τύπος είναι κάτι σαν έναν πίνακα. Αυτή η σύγκριση μπορεί να μην είναι πάντα αληθής επειδή εξαρτάται σε μεγάλο βαθμό από το πώς έχει σχεδιαστεί το σύμπλεγμα, αλλά στις περισσότερες περιπτώσεις, θα ισχύει.

Οποιοσδήποτε αριθμός ευρετηρίων μπορεί να οριστεί. Ακριβώς όπως ο κόμβος και το σύμπλεγμα, ένα ευρετήριο αναγνωρίζεται από ένα μοναδικό όνομα και αυτό το όνομα πρέπει να είναι σχετικά μικρό.

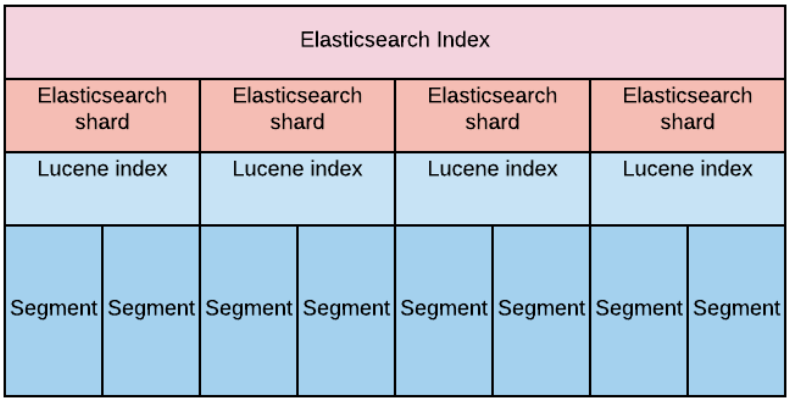


Εικόνα 15: Η αρχιτεκτονική του συστήματος Elasticsearch

Ο τύπος είναι μια κατηγορία ή μια κλάση παρόμοιων εγγράφων. Όπως εξηγείται στην προηγούμενη παράγραφο, πλησιάζει σε έναν πίνακα σε όρους σχεσιακής βάσης δεδομένων. Αποτελείται από ένα μοναδικό όνομα και χαρτογράφηση. Κάθε φορά που εκτελείται μία επερώτηση σε ένα ευρετήριο, το Elasticsearch διαβάζει τον τύπο από ένα

αρχείο μεταδιδόμενων και εφαρμόζει ένα φίλτρο σε αυτό το πεδίο. Η Lucene εσωτερικά δεν έχει ιδέα για το τι είναι ο τύπος. Ένα ευρετήριο μπορεί να έχει οποιοδήποτε αριθμό τύπων και οι τύποι μπορούν να έχουν τη δική τους χαρτογράφηση.

Η χαρτογράφηση είναι κάπως παρόμοια με το σχήμα μιας σχεσιακής βάσης δεδομένων. Δεν είναι υποχρεωτικό να οριστεί ρητά η χαρτογράφηση. Αν δεν παρέχεται χαρτογράφηση, η Elasticsearch θα την προσθέσει δυναμικά με βάση τα δεδομένα της, όταν προστεθεί το έγγραφο. Η χαρτογράφηση γενικά περιγράφει το πεδίο και τον τύπο δεδομένων του σε ένα έγγραφο. Περιλαμβάνει επίσης πληροφορίες σχετικά με τον τρόπο κατάδειξης και αποθήκευσης πεδίων από τη Lucene.



Εικόνα 16: Ανάλυση ενός δείκτη Elasticsearch

Ένα έγγραφο είναι η μικρότερη και πιο βασική μονάδα πληροφοριών που μπορεί να ευρετηριαστεί. Αποτελείται από ζεύγη κλειδιών-τιμών όπου οι τιμές μπορούν να είναι τύπου συμβολοσειράς, αριθμός, ημερομηνία, αντικείμενο κ.λπ. Ένα ευρετήριο μπορεί να έχει οποιοδήποτε αριθμό εγγράφων αποθηκευμένων μέσα σε αυτό. Σε αντικειμενοστραφείς όρους, ένα έγγραφο είναι κάτι σαν ένα αντικείμενο. Έχει τη μορφή JSON. Σε όρους σχεσιακής βάσης δεδομένων, ένα έγγραφο μπορεί να θεωρηθεί ως μία γραμμή ενός πίνακα.

Ένα ευρετήριο μπορεί να χωριστεί σε πολλαπλά ανεξάρτητα δευτερεύοντα ευρετήρια. Αυτά τα δευτερεύοντα ευρετήρια είναι απόλυτα λειτουργικά και ονομάζονται θραύσματα. Είναι χρήσιμα όταν ένα ευρετήριο χρειάζεται να έχει περισσότερα δεδομένα από την ικανότητα υλικού ενός κόμβου διακομιστή που υποστηρίζει όπως για παράδειγμα, δεδομένα 800 gb σε δίσκο χωρητικότητας 500 gb.

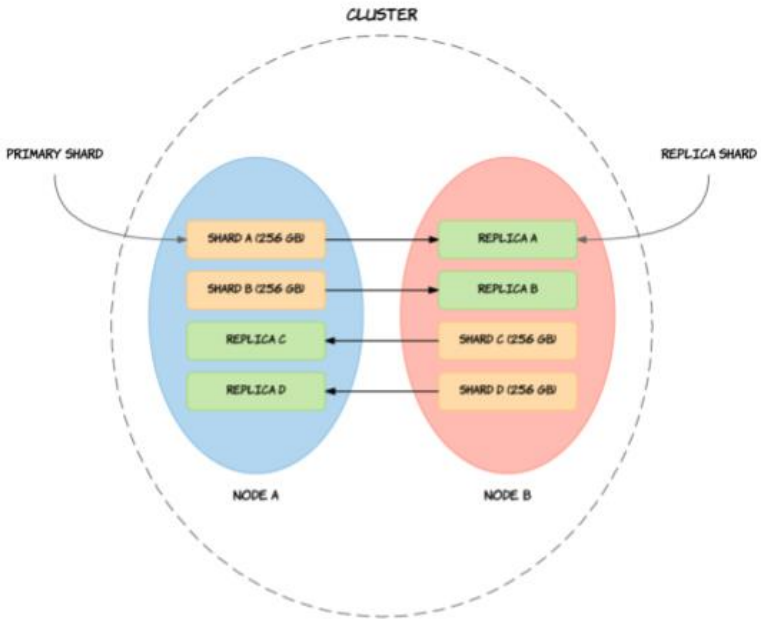
Η σκίαση επιτρέπει την οριζόντια κλιμάκωση όσον αφορά τις διαστάσεις του όγκου και του χώρου. Βελτιώνει την απόδοση ενός συμπλέγματος με την εκτέλεση παράλληλων λειτουργιών και τη διανομή φορτίων σε διαφορετικά θραύσματα. Από προεπιλογή το Elasticsearch προσθέτει 5 πρωτογενή κομμάτια για το ευρετήριο. Αυτό μπορεί να ρυθμιστεί χειροκίνητα ώστε να ανταποκρίνεται στις εκάστοτε απαιτήσεις.

Ένα αντίγραφο είναι ένα αντίγραφο ενός ξεχωριστού τεμαχίου. Το Elasticsearch δημιουργεί ένα αντίγραφο για κάθε θραύσμα. Το αντίγραφο και το αρχικό shard ποτέ δεν διαμένουν στον ίδιο κόμβο.

Το αντίγραφο έρχεται στο προσκήνιο όταν οι κόμβοι σε ένα σύμπλεγμα αποτυγχάνουν. Τα θραύσματα σε έναν κόμβο αποτυγχάνουν ή σημειώνεται μια αιχμή του συνολικού

φόρτου διαβάσματος. Το αντίγραφο υπόσχεται την υψηλή διαθεσιμότητα των δεδομένων σε τέτοιες καταστάσεις. Όταν ενεργοποιείται μία επερώτηση εγγραφής, το αρχικό θραύσμα πρώτα ενημερώνεται και στη συνέχεια τα αντίγραφα ενημερώνονται με κάποια υπερκείμενη καθυστέρηση. Αλλά τα ερωτήματα ανάγνωσης μπορούν να τρέξουν παράλληλα σε διαφορετικά αντίγραφα. Αυτό το χαρακτηριστικό βελτιώνει την απόδοση των λειτουργιών ανάγνωσης συνολικά.

Από προεπιλογή, ένα μόνο αντίγραφο κάθε πρωτογενούς τεμαχίου δημιουργείται, αλλά ένα θραύσμα μπορεί να έχει περισσότερα από ένα αντίγραφα σε ορισμένες ειδικές περιπτώσεις.



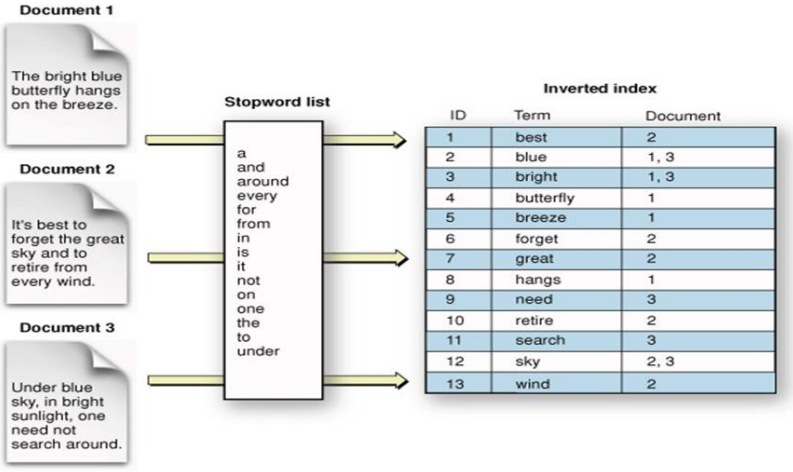
Εικόνα 17: Ένα παράδειγμα συμπλέγματος του Elasticsearch το οποίο αποτελείται από 2 κόμβους και περιέχει 4 θραύσματα μαζί με τα αντίγραφά τους

Ένα άλλο δομικό στοιχείο στο Elasticsearch είναι το ανεστραμμένο ευρετήριο που ουσιαστικά αποτελεί έναν μοναδικό τύπο δομής δεδομένων που χρησιμοποιεί η Lucene για να κάνει εύκολη την αναζήτηση τεράστιου συνόλου δεδομένων. Το ανεστραμμένο ευρετήριο είναι ένα σύνολο λέξεων, φράσεων ή μαρκών που σχετίζονται με διαφορετικά έγγραφα για να επιτρέπεται η αναζήτηση πλήρους κειμένου. Με απλά λόγια, ένα ανεστραμμένο ευρετήριο είναι κάτι σαν μια σελίδα προσάρτησης στο τέλος ενός βιβλίου μιας και θα αποτελεί μια δομή η οποία θα έχει αντιστοιχίες λέξεων σε έγγραφα.

Επιπροσθέτως, κάθε θραύσμα αποτελείται από πολλαπλά τμήματα. Αυτά τα τμήματα δεν είναι τίποτα διαφορετικό από ανεστραμμένους δείκτες, οι οποίοι θα αναζητούν παράλληλα, θα δέχονται τα αποτελέσματα και θα τα συνδυάζουν στην τελική παραγωγή για το συγκεκριμένο τεμάχιο.

Όπως και όταν τα έγγραφα είναι ευρετηριασμένα, το Elasticsearch τα γράφει σε νέα τμήματα, ανανεώνει τα δεδομένα αναζήτησης και ενημερώνει τα αρχεία καταγραφής συναλλαγών. Αυτό συμβαίνει πολύ συχνά για να καταστούν τα δεδομένα σε νέο τμήμα ορατά από όλα τα ερωτήματα. Το Elasticsearch δεν προορίζεται για ενημερώσεις και

διαγραφή, οπότε αν τα δεδομένα πρέπει να διαγραφούν ή να ενημερωθούν, στην πραγματικότητα απλώς καθιστά το παλιό έγγραφο ως διαγραμμένο και ευρετηριάζει ένα νέο έγγραφο που περιλαμβάνει τις τροποποιήσεις. Η διαδικασία συγχώνευσης εκθέτει επίσης αυτά τα παλιά διαγραμμένα έγγραφα. Το Elasticsearch συγχωνεύει συνεχώς παρόμοια τμήματα σε κοινά μεγάλα τμήματα στο παρασκήνιο. Η αναζήτηση σε πολλά μικρά τμήματα δεν είναι πολύ βέλτιστη. Αφού γραφεί το μεγαλύτερο τμήμα, τα μικρότερα τμήματα αποσύρονται και τα αρχεία καταγραφής ενημερώνονται και πάλι για να αντικατοπτρίζουν τις νέες αλλαγές.



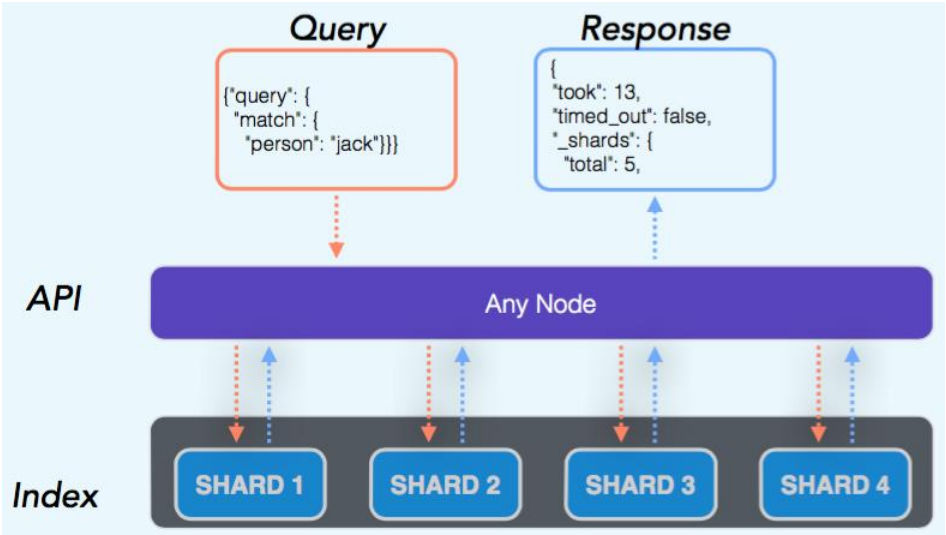
Εικόνα 18: Παράδειγμα ανεστραμμένου ευρετηρίου

Είναι σχετικά εύκολο να ενσωματωθεί το Elasticsearch σε ένα πρόβλημα. Χαρακτηρίζεται από λογική στις προκαθορισμένες μεταβλητές και κρύβει πολύπλοκη αναζήτηση και μηχανικές διανομής από αρχάριους χρήστες. Λειτουργεί αρκετά καλά, ακριβώς όπως παρέχεται χωρίς τροποποιήσεις. Με μια σύντομη καμπύλη μάθησης για την κατανόηση των βασικών αρχών, μπορεί κάποιος να γίνει παραγωγικός σχετικά πολύ γρήγορα.

Τα συμβατικά SQL συστήματα διαχείρισης βάσεων δεδομένων δεν είναι πραγματικά σχεδιασμένα για αναζητήσεις πλήρους κειμένου και σίγουρα δεν λειτουργούν καλά ενάντια στα χαλαρά δομημένα ακατέργαστα δεδομένα που βρίσκονται εκτός της βάσης δεδομένων. Στο ίδιο υλικό, ερωτήματα που θα χρειάζονταν περισσότερα από 10 δευτερόλεπτα με τη χρήση SQL θα επιστρέψουν τα αποτελέσματα σε λιγότερο από 10 χιλιοστά του δευτερολέπτου με τη χρήση του Elasticsearch.

Ένας χρήστης εκφράζει ένα ερώτημα Elasticsearch σε μια απλή γλώσσα που ονομάζεται Query DSL. Ένα ερώτημα εξετάζει μία ή περισσότερες τιμές-στόχους και βαθμολογεί κάθε ένα από τα στοιχεία στα αποτελέσματα ανάλογα με το πόσο κοντά ταιριάζουν με την εστίαση του ερωτήματος.

Οι χειριστές επερωτήσεων καθιστούν δυνατή τη βελτιστοποίηση απλών αλλά και πολύπλοκων ερωτημάτων που συχνά επιστρέφουν αποτελέσματα από μεγάλα σύνολα δεδομένων σε λίγα χιλιοστά του δευτερολέπτου. Ο σχεδιασμός του Elasticsearch είναι πολύ πιο απλός και πολύ πιο καθαρός από μια βάση δεδομένων που περιορίζεται από σχήματα, πίνακες, πεδία, γραμμές και στήλες.



Εικόνα 19: Παράδειγμα εκτέλεσης ενός επερωτήματος

Κατά τη διάρκεια μιας διαδικασίας δημιουργίας ευρετηρίου, το Elasticsearch μετατρέπει τα πηγαία ανεπεξέργαστα δεδομένα όπως αρχεία καταγραφής ή αρχεία μηνυμάτων σε εσωτερικά έγγραφα και τα αποθηκεύει σε μια βασική δομή δεδομένων παρόμοια με ένα αντικείμενο JSON. Κάθε έγγραφο είναι ένα απλό σύνολο συσχετισμένων κλειδιών και τιμών. Τα κλειδιά είναι συμβολοσειρές και οι τιμές είναι ένας από τους πολυάριθμους τύπους δεδομένων όπως για παράδειγμα συμβολοσειρές, αριθμοί, ημερομηνίες ή λίστες.

Η προσθήκη εγγράφων στο Elasticsearch είναι εύκολη και είναι εύκολο να αυτοματοποιηθεί. Εκτελώντας ένα απλό POST HTTP μεταδίδεται το έγγραφό ως ένα απλό αντικείμενο JSON.

Οι αναζητήσεις πραγματοποιούνται επίσης με JSON. Το ερώτημα αποστέλλεται με τη χρήση ενός HTTP GET με σώμα JSON. Το RESTful API διευκολύνει την ανάκτηση, την υποβολή και την επαλήθευση δεδομένων απευθείας από μια γραμμή εντολών.

Ακόμα και σε έργα τα οποία αναπτύσσονται με έναν πελάτη που χρησιμοποιεί τις γλώσσες προγραμματισμού Python ή Ruby, πολλοί προγραμματιστές χρησιμοποιούν το εργαλείο cURL για την αποσφαλμάτωση και την ανάπτυξη με το Elasticsearch.

Είναι σημαντικό το βασικό χαρακτηριστικό, ότι το Elasticsearch δεν είναι μια σχεσιακή βάση δεδομένων με αποτέλεσμα οι βασικές αρχές ΣΔΒΔ να μην ισχύουν. Η πιο σημαντική έννοια που πρέπει να αγνοηθεί από κάποιον που έχει μάθει να σκέφτεται με συμβατικές βάσεις δεδομένων είναι η εξομάλυνση. Το εγγενές Elasticsearch δεν επιτρέπει συνδέσεις ή υποερωτήσεις, οπότε η εξομοίωση των δεδομένων του προβλήματος κρίνεται απαραίτητη.

Το Elasticsearch συνήθως αποθηκεύει ένα έγγραφο μία φορά για κάθε αποθήκη στην οποία βρίσκεται. Αν και αυτό είναι αντίθετο από την άποψη ενός συμβατικού ΣΔΒΔ, είναι βέλτιστο για το Elasticsearch. Οι αναζητήσεις πλήρους κειμένου θα είναι εξαιρετικά γρήγορες επειδή τα έγγραφα αποθηκεύονται σε κοντινή απόσταση από τα αντίστοιχα μεταδεδομένα του ευρετηρίου. Αυτός ο σχεδιασμός μειώνει σημαντικά τον αριθμό των δεδομένων που διαβάζει και το Elasticsearch περιορίζει τον ρυθμό αύξησης του δείκτη διατηρώντας τον συμπιεσμένο.

Το Elasticsearch μπορεί να κλιμακώσει μέχρι χιλιάδες διακομιστές και να φιλοξενήσει petabytes δεδομένων. Η τεράστια χωρητικότητά του προκύπτει άμεσα από την

πολύπλοκη, κατανεμημένη αρχιτεκτονική του. Και παράλληλα με την τόσο γρήγορη επεξεργασία τεράστιων σε όγκο δεδομένων ο χρήστης του Elasticsearch μπορεί να είναι ευτυχής αγνοώντας σχεδόν όλη την αυτοματοποίηση και την πολυπλοκότητα που υποστηρίζει αυτό ο κατανεμημένος σχεδιασμός.

Στο Elasticsearch, οι παρακάτω ευαίσθητες και συχνά εντατικές λειτουργίες συμβαίνουν αυτόματα και ανεπαίσθητα:

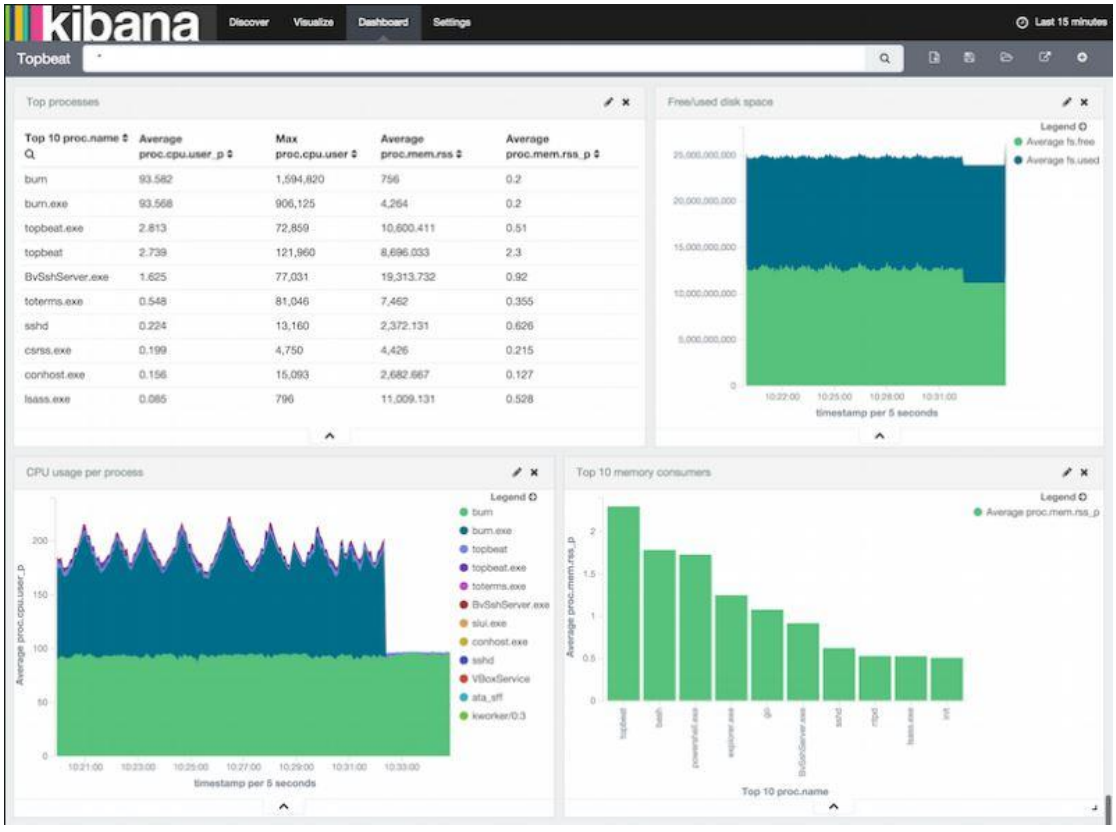
- Διαχωρισμός των εγγράφων σε μια διάταξη διακεκριμένων τεμαχίων (δοχείων)
- Σε ένα σύμπλεγμα πολλαπλών κόμβων, κατανομή εγγράφων σε θραύσματα ικανή να βρίσκεται σε όλους τους κόμβους
- Εξισορρόπηση θραυσμάτων σε όλους τους κόμβους σε ένα σύμπλεγμα για την ομοιόμορφη διαχείριση του φορτίου ευρετηρίου και αναζήτησης
- Αναπαραγωγή αλλά και αντιγραφή κάθε τεμαχίου για την παροχή πλεονασμού δεδομένων και ανακατεύθυνσης
- Αιτήματα δρομολόγησης από οποιονδήποτε κόμβο του συμπλέγματος σε συγκεκριμένους κόμβους που περιέχουν τα συγκεκριμένα δεδομένα που χρειάζονται
- Αδιάλειπτη προσθήκη και ενσωμάτωση νέων κόμβων καθώς δημιουργείται η ανάγκη για αύξηση του μεγέθους του συμπλέγματος
- Ανακατανομή των κομματιών για αυτόματη ανάκτηση από την απώλεια ενός κόμβου

1.6 Kibana

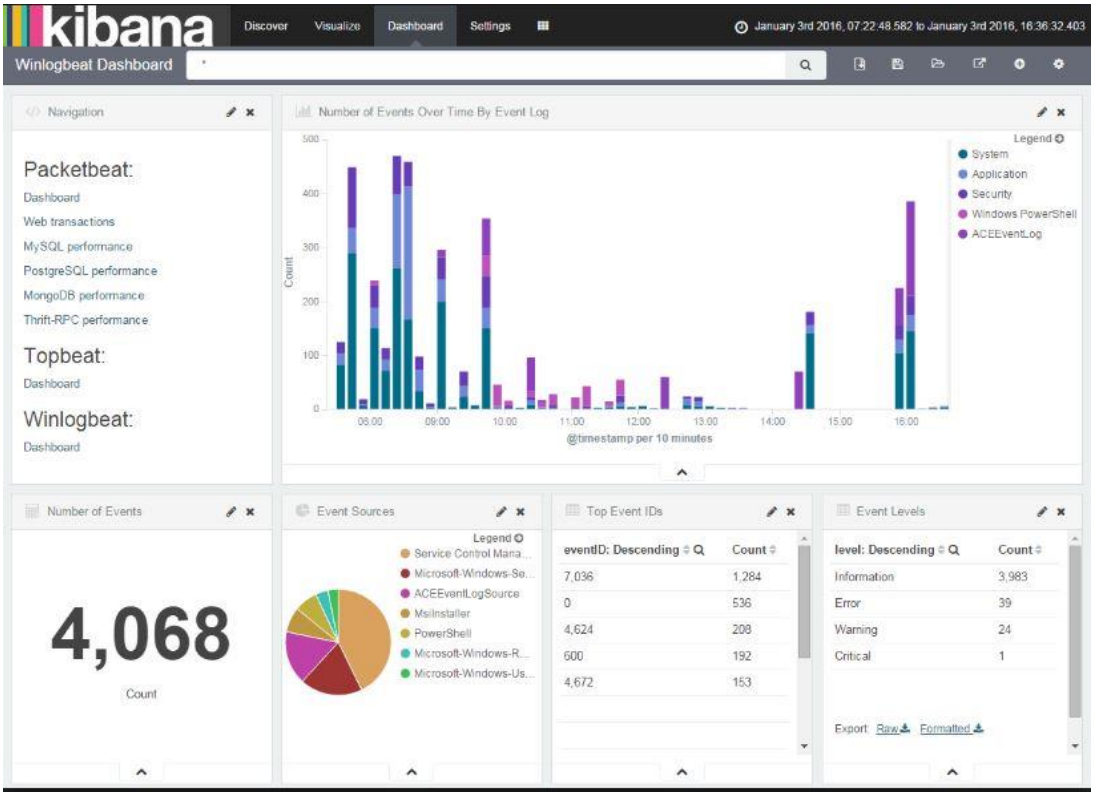
Το Kibana είναι μια πλατφόρμα οπτικοποίησης και εξερεύνησης δεδομένων από την Elasticsearch, η οποία, όπως αναφέρθηκε αναλυτικά παραπάνω, εξειδικεύεται σε μεγάλους όγκους δεδομένων ροής και σε πραγματικό χρόνο.

Το Kibana έχει σχεδιαστεί για να συνεργάζεται με την Elasticsearch για να κάνει πιο γρήγορα και εύκολα κατανοητά τις τεράστιες και σύνθετες ροές δεδομένων μέσω γραφικής παράστασης. Τα αναλυτικά στοιχεία της Elasticsearch παρέχουν μαθηματικούς μετασχηματισμούς στα δεδομένα καθώς και βελτιωμένη συνάθροιση. Το λογισμικό δημιουργεί αναφορές PDF είτε κατόπιν αιτήματος είτε με βάση το χρονοδιάγραμμα και παρέχει έναν ευέλικτο και ολοκληρωτικά δυναμικό πίνακα ελέγχου. Οι αναφορές που δημιουργούνται μπορούν να αντιπροσωπεύουν τα δεδομένα σε διαγράμματα γραμμών, διαγράμματα στηλών, διαγράμματα διασποράς, διαγράμματα πίτας ή γραφικά διαγραμματα με προσαρμοσμένα χρώματα και επισημασμένα αποτελέσματα αναζήτησης. Το Kibana περιλαμβάνει επίσης εργαλεία κοινής χρήσης για οπτικοποιημένα δεδομένα.

Παρακάτω παρατίθενται σε μορφή εικόνας ορισμένα παραδείγματα οθόνης του ταμπλό του Kibana:



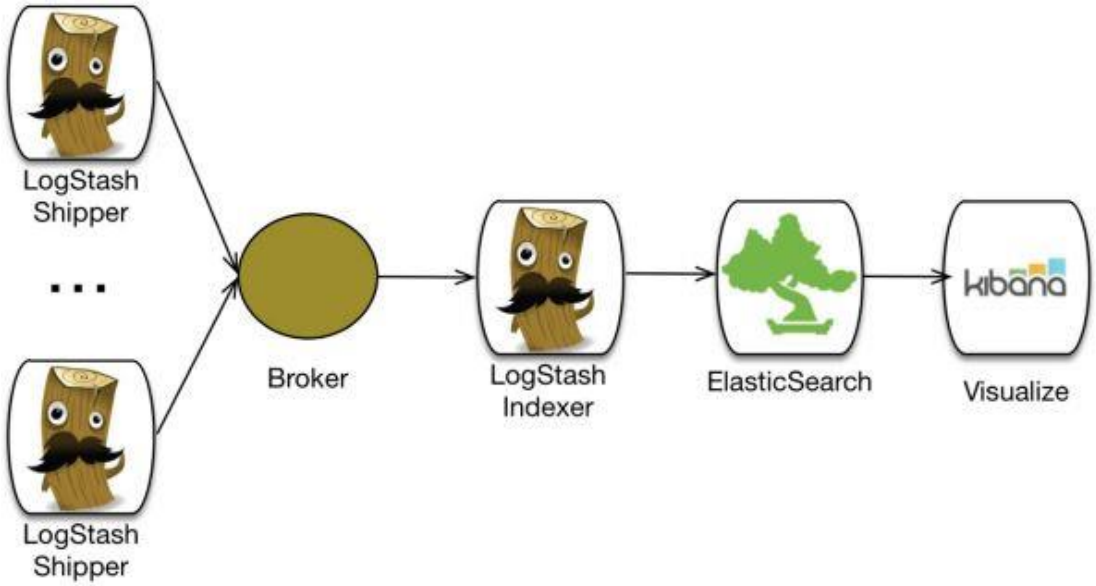
Εικόνα 20: Δείγμα οθόνης του Kibana



Εικόνα 21: Δείγμα οθόνης του Kibana

Το μεγάλο προσόν του Kibana είναι η δυνατότητα παρουσίασης μεγάλου όγκου πληροφορίας με τρόπο φιλικό προς το χρήστη. Μπορούν να κατασκευαστούν πολυάριθμα διαγράμματα μέσω των οποίων η ανάλυση των δεδομένων γίνεται εύκολα, γρήγορα και αποδοτικά.

Το προϊόν είναι ένα στοιχείο του Elastic Stack, μιας σουίτας εργαλείων απεικόνισης και ανάλυσης δεδομένων που περιλαμβάνει επίσης το Elasticsearch, το Logstash και το Beats. Η ασφάλεια παρέχεται από την αρχιτεκτονική client-server της Kibana με έλεγχο ταυτότητας. Το Shield, ένα εργαλείο ασφαλείας για την Elasticsearch και μπορεί να προστεθεί για την προστασία δεδομένων.



Εικόνα 22: Διάγραμμα συνεργασίας Kibana με το Elasticsearch και το LogStash

Εικόνα 23: Διάγραμμα συνεργασίας Kibana με το Elasticsearch και το LogStash

Στις παραπάνω εικόνες μπορεί κάποιος να καταλάβει τον τρόπο με τον οποίο το Kibana συνεργάζεται με τα υπόλοιπα στοιχεία του ELK Stack, δηλαδή το Elasticsearch και το Logstash. Όπως προαναφέρθηκε, αποτελεί το τελευταίο επίπεδο της αλληλουχίας και είναι αυτό με την άμεση αλληλεπίδραση με τον αναλυτή των δεδομένων.

1.7 Δεδομένα από αισθητήρες

Τα δεδομένα που χρησιμοποιήθηκαν ως είσοδος στο σύστημα είναι δεδομένα που συλλέγονται από αισθητήρες.

Οι αισθητήρες μπορούν να χρησιμοποιηθούν για τον εντοπισμό σχεδόν οποιουδήποτε φυσικού στοιχείου. Ακολουθούν μερικά παραδείγματα αισθητήρων, για να δοθεί μια ιδέα για τον αριθμό και την ποικιλία των εφαρμογών τους:

- Ένα επιταχυνσιόμετρο ανιχνεύει αλλαγές στη βαρυτική επιτάχυνση σε μια συσκευή στην οποία είναι εγκατεστημένη, όπως ένα έξυπνο κινητό τηλέφωνο ή ένας ελεγκτής παιχνιδιών, για τον προσδιορισμό της επιτάχυνσης, της κλίσης και των κραδασμών.
- Ένας φωτοαισθητήρας ανιχνεύει την παρουσία ορατού φωτός, υπέρυθρης μετάδοσης (IR) ή / και υπεριώδους (UV) ενέργειας.
- Το Lidar, μια μέθοδος ανίχνευσης με λέιζερ, εύρεση εύρους και χαρτογράφηση, συνήθως χρησιμοποιεί ένα παλμικό λέιζερ χαμηλής κατανάλωσης ενέργειας, που λειτουργεί σε συνδυασμό με μια φωτογραφική μηχανή.
- Μια συσκευή συζευγμένης φόρτισης (CCD) αποθηκεύει και εμφανίζει τα δεδομένα για μια εικόνα με τέτοιο τρόπο ώστε κάθε εικονοστοιχείο (pixel) να μετατρέπεται σε ηλεκτρικό φορτίο, η ένταση του οποίου σχετίζεται με ένα χρώμα στο φάσμα χρώματος.
- Οι έξυπνοι αισθητήρες δικτύου μπορούν να παρέχουν δεδομένα σε πραγματικό χρόνο σχετικά με τις συνθήκες δικτύου, ανίχνευση διακοπών, βλάβες και φόρτιση και ενεργοποίηση συναγερμών.
- Τα ασύρματα δίκτυα αισθητήρων συνδυάζουν εξειδικευμένους μετατροπείς με υποδομή επικοινωνιών για παρακολούθηση και καταγραφή συνθηκών σε διάφορες τοποθεσίες.

Οι συνήθεις παρακολουθούμενες παράμετροι περιλαμβάνουν τη θέση σημείων στο χώρο, τη θερμοκρασία, την υγρασία, την πίεση, την κατεύθυνση και ταχύτητα του ανέμου, την ένταση φωτισμού, την ένταση των κραδασμών, την ένταση του ήχου, την ηλεκτρική τάση, τις χημικές συγκεντρώσεις, τα επίπεδα ρύπων και τις ζωτικές λειτουργίες του σώματος.

Τα δεδομένα των αισθητήρων αποτελούν αναπόσπαστο στοιχείο της αυξανόμενης πραγματικότητας του περιβάλλοντος Internet of Things (IoT). Στο σενάριο IoT, σχεδόν οποιαδήποτε οντότητα μπορεί να φανταστεί κάποιος μπορεί να εξοπλιστεί με ένα μοναδικό αναγνωριστικό (UID) και την ικανότητα μεταφοράς δεδομένων μέσω ενός δικτύου. Μεγάλο μέρος των δεδομένων που μεταδίδονται είναι δεδομένα αισθητήρων.

Ο τεράστιος όγκος δεδομένων που παράγονται και μεταδίδονται από συσκευές ανίχνευσης μπορεί να παρέχει πολλές πληροφορίες, αλλά θεωρείται συχνά η επόμενη μεγάλη πρόκληση δεδομένων για τις επιχειρήσεις. Για να αντιμετωπιστεί αυτή η πρόκληση, οι αναλύσεις δεδομένων αισθητήρων είναι ένα αυξανόμενο πεδίο έρευνας.

Εικόνα 24: Το Internet of Things

2. Η ΜΕΘΟΔΟΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ

Οι τεχνολογίες που έχουν περιγραφεί σχολαστικά παραπάνω έχουν χρησιμοποιηθεί για το προς επίλυση πρόβλημα. Έχουν επιλεχθεί αυτές μεταξύ άλλων ανταγωνιστικών για ποικίλους λόγους.

Αρχικά το Apache Kafka, θεωρείται ένα από τα πιο ιδανικά εργαλεία όσον αφορά την επεξεργασία και τη διαχείριση δεδομένων ροών. Αυτό είναι πλέον επιθυμητό μιας και στο συγκεκριμένο πρόβλημα τα δεδομένα αισθητήρων αποτελούν μία πηγή που δύναται να στέλνει πολλές μετρήσεις ανά δευτερόλεπτο.

Στη συνέχεια, το Apache Storm παρέχει τη δυνατότητα να επεξεργάζεται δεδομένα με τη λογική exact once to tuple σε περίπτωση προβλημάτων που ενδέχεται να ξαναστέλνουν δεδομένα. Αυτό σημαίνει πως μπορεί να διαχειριστεί περιπτώσεις όπου μια πηγή στέλνει δεδομένα πολλαπλές φορές αγνοώντας τα διπλότυπα που λαμβάνονται πολυάριθμες φορές λόγω έλλειψης συγχρονισμού.

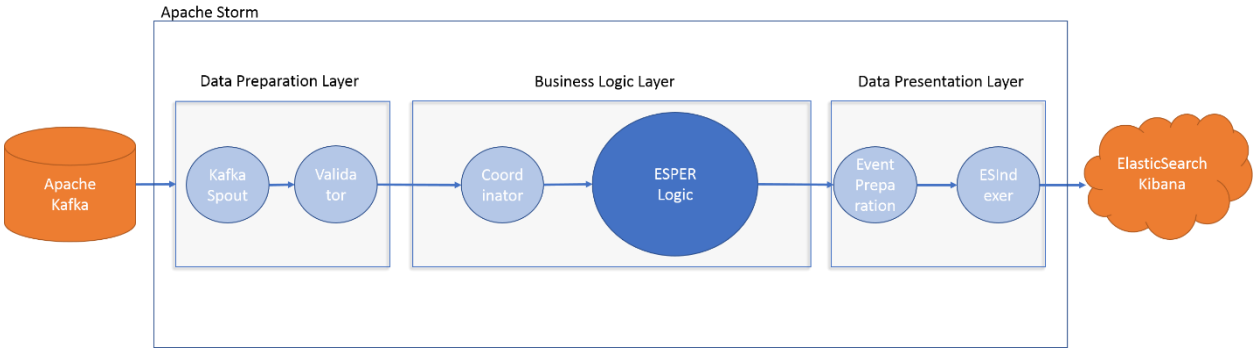
Επιπροσθέτως το Storm παρέχει εύκολο και ευθύ τρόπο για τη δημιουργία αγωγών ροών δεδομένων με bolts τα οποία είναι ρυθμισμένα από τους χρήστες με βάση το συγκεκριμένο πρόβλημα. Εκτός αυτών, αυτά τα σημαντικά οφέλη παρέχονται ταυτόχρονα με εξαιρετικές επιδόσεις.

Σχετικά με το Esper, αποτελεί μια λύση που συνεργάζεται άψογα με τις προαναφερθέντες τεχνολογίες. Αυτό, σε συνδυασμό με το ότι παρέχει τη δυνατότητα για εφαρμογή πολυσύνθετων επερωτημάτων στα δεδομένα, το καθιστά αναγκαίο για το συγκεκριμένο σύστημα.

Το ElasticSearch στην παρούσα φάση ουσιαστικά χρησιμοποιήθηκε με άμεσο στόχο την απεικόνιση των δεδομένων με τρόπο φιλικό προς το χρήστη. Η ανάλυση των δεδομένων εισόδου γίνεται εύκολη μιας και αντί ο χρήστης να διαβάζει κωδικοποιημένα δεδομένα, θα μπορεί να βλέπει την πληροφορία συσσωρευμένη και επεξηγημένη σε διαγράμματα και γραφήματα.

Τα δεδομένα που συλλέγονται από τους αισθητήρες μετά την επεξεργασία τους σε μία καθορισμένη μορφή, φορτώνονται στο Apache Kafka μέρος του έργου, έπειτα προετοιμάζονται με τη χρήση των spouts και bolts του Apache Storm τμήματος, έπειτα εφαρμόζονται τα επερωτήματα και στο τέλος τα αποτελέσματα αφού περάσουν από το Elasticsearch κομμάτι θα παρουσιαστούν με φιλικό προς το χρήστη τρόπο στο Kibana τμήμα του έργου, το οποίο ουσιαστικά αποτελεί τον τελευταίο κρίκο της αλυσίδας.

Όλη αυτή η διαδικασία περιγράφεται με το σχήμα παρακάτω:



Εικόνα 25: Η διαδικασία επίλυσης του προβλήματος

2.1 Τα Δεδομένα

Τα δεδομένα του προβλήματος αφορούν αισθητήρες οι οποίοι είναι τοποθετημένοι σε έξι διαφορετικά και διακριτά δωμάτια.

Στο συγκεκριμένο πρόβλημα, κάθε ένα από τα αναγνωριστικά αντιστοιχεί σε ένα διαφορετικό άτομο που κινείται και ενδεχομένως αλλάζει θέση μεταξύ των δωματίων.

Συνολικά υπάρχουν έξι διαφορετικές κεραίες/αισθητήρες και πολυάριθμα μοναδικά αναγνωριστικά ατόμων.

Οι έξι αισθητήρες που χρησιμοποιήθηκαν αποτελούν ένα μικρό υποσύνολο από αυτούς που υπάρχουν συνολικά αλλά και από αυτούς που το συγκεκριμένο σύστημα μπορεί να επεξεργαστεί αποτελεσματικά και αποδοτικά.

Πιο συγκεκριμένα, για τους έξι αισθητήρες υπάρχουν 200 μοναδικά αναγνωριστικά ατόμων. Το πρώτο βήμα είναι η δημιουργία των δεδομένων με τυχαίο τρόπο έτσι ώστε να έχουμε μία σειρά από θέσεις στο χρόνο και το χώρο για κάθε ένα από τα άτομα.

Για τις ανάγκες της παρούσας διπλωματικής εργασίας έχει δημιουργηθεί ένα πρόγραμμα με χρήση της γλώσσας προγραμματισμού Python, το οποίο δημιουργεί στιγμιότυπα των ατόμων με τυχαίο τρόπο. Με αυτόν τον τρόπο παράγονται πανομοιότυπα δεδομένα με τα αντίστοιχα πραγματικά που θα είχαμε αν ως πηγή στο Kafka υπήρχε η ροή από τους αισθητήρες απευθείας.

Τα δεδομένα που χρησιμοποιήθηκαν στην επίλυση του συγκεκριμένου προβλήματος έχουν κωδικοποιηθεί με τη μορφή <*user_id*, *sensor_id*, *timestamp*>. Αυτά συλλέγονται από μια ροή μηνυμάτων όπου στην περίπτωση αυτή είναι ένα *Kafka stream* και διαβάζονται με τη βοήθεια του Apache Storm.

Ένα πολύ μικρό δείγμα των δεδομένων αυτών παρουσιάζονται παρακάτω:

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x0c-0x4-0x97-0x0c, 5, 10:42:05.038>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x0f-0xcf-0x98-0x86, 5, 10:42:05.089>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x54-0x22-0x90-0x6c, 2, 10:42:05.861>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x66-0x3d-0x85-0x20, 5, 10:42:05.996>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x82-0x5c-0xec-0x6a, 2, 10:42:06.381>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0xe8-0x3-0x13-0xce, 5, 10:42:06.386>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x8d-0xeb-0x8a-0x32, 5, 10:42:06.667>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x71-0xb0-0xc5-0x35, 4, 10:42:06.822>

<0x34-0x0-0x35-0xe0-0x17-0x2-0x7-0x0-0x0-0x0-0x0-0x0-0x72-0x50-0x5f-0x11-0x4d, 4, 10:42:07.067>

Στο παραπάνω δείγμα των δεδομένων έχουν επισημανθεί, με γκρι χρώμα τα μοναδικά αναγνωριστικά των ατόμων που ανιχνεύονται από τους διάφορους αισθητήρες, με πράσινο χρώμα τα μοναδικά αναγνωριστικά των αισθητήρων/κεραιών και με κίτρινο χρώμα οι χρονοσφραγίδες των μετρήσεων.

2.2 Τα δομικά στοιχεία της Storm τοπολογίας

Από τη στιγμή που τα δεδομένα είναι έτοιμα, το επόμενο βήμα είναι να ξεκινήσει να τρέχει στο σύστημα η υπηρεσία Zookeeper και έπειτα ο εξυπηρετητής Kafka. Για να επιτευχθεί η επιθυμητή διαδικασία έχουν bolts που αναφέρονται παρακάτω:

Αρχικά, το bolt το οποίο είναι υπεύθυνο να μεταφράζει ένα event από το Kafka spout σε ένα event object ονομάζεται *TxtTupleValidatorBolt* και παρουσιάζεται εδώ.

```
- id: "txt_parser"
className: "com.cep.sensor.analytics.topology.bolt.TxtTupleValidatorBolt"
parallelism: 1
```

Το επόμενο είναι αυτό του οποίο ο κύριος ρόλος είναι να καταγράφει τα μηνύματα διάγνωσης και αποσφαλμάτωσης στο τέλος, έτσι ώστε να μπορεί να διασφαλίζεται η ομαλή λειτουργία του συστήματος και ονομάζεται *LoggingBolt*.

```
- id: "e_log_txt"
className: "com.cep.sensor.analytics.topology.bolt.LoggingBolt"
parallelism: 1
constructorArgs:
- "TXT_ERROR"
```

Το τρίτο bolt που έχει οριστεί για τη τοπολογία ονομάζεται *JsonTupleValidatorBolt* και ο ρόλος του είναι παρόμοιος με αυτόν του *TxtTupleValidatorBolt* που προαναφέρθηκε με την μόνη διαφορά ότι αυτό επεξεργάζεται πληροφορία που περιέχεται σε json αρχείο.

Στη συνέχεια υπάρχει το *CoordinatorBolt* το οποίο είναι υπεύθυνο για να δέχεται σαν είσοδο ένα συμβάν και έπειτα να το διαμοιράζει στα διάφορα άλλα Esper bolts.

```
- id: "event_coordinator"
className: "com.cep.sensor.analytics.topology.bolt.CoordinatorBolt"
parallelism: 1
```

Για τη διαδικασία μετατροπής του αποτελέσματος που προκύπτει από κάθε ένα Esper bolt σε hash είναι ευθύνη του *PrepareEsBolt* και παρουσιάζεται εδώ.

```
- id: "prepare_es"
className: "com.cep.sensor.analytics.topology.bolt.PrepareESBolt"
parallelism: 1
```

Το επόμενο της διαδικασίας ονομάζεται *EsIndexBolt* και ο ρόλος του στη τοπολογία είναι να ευρετηριάζει το hash το οποίο έχει επιστραφεί προηγουμένως από το *PrepareEsBolt*.

```
- id: "es_bolt"
className: "com.cep.sensor.analytics.topology.bolt.external.EsIndexBolt"
parallelism: 1
constructorArgs:
 - "http://localhost:9200"
```

Από τα ήδη δημιουργημένα του Esper χρησιμοποιήθηκε το *ExtendedEsperBolt* για τη δημιουργία των επερωτημάτων-μετρικών. Στη συνέχεια παρουσιάζονται εκτενέστερα αυτά τα bolts.

Το spout του Apache Storm τμήματος αναμένει να διαβάσει τα διάφορα events στην μορφή που έχουν περιγραφεί προηγουμένως από το Apache Kafka τμήμα του συστήματος.

Μόλις δημιουργηθεί κάποιο event και διαβαστεί από το Storm, το event αυτό προωθείται στο πρώτο bolt, δηλαδή στο TxtTupleValidatorBolt. Σε αυτό το σημείο γίνεται επαλήθευση σε event και από εκεί αποστέλλεται στο επόμενο bolt της διαδικασίας, το CoordinatorBolt.

Στη συνέχεια, αυτό το bolt το διαμοιράζει στα bolts που είναι υπεύθυνα για να πραγματοποιούν τους υπολογισμούς για τις μετρικές που έχουν οριστεί και παρουσιάστηκαν αναλυτικά προηγουμένως.

2.2.1 Η απεικόνιση των μετρικών στη τοπολογία

Προκειμένου να είναι δυνατός ο υπολογισμός της κίνησης καθώς και της επισκεψιμότητας των αισθητήρων, δηλαδή των δωματίων έχει δημιουργηθεί το *esperBoltRoomTraffic*. Πιο αναλυτικά, το επερώτημα που χρησιμοποιείται για να επιστραφούν τα συγκεκριμένα αποτελέσματα είναι το παρακάτω:

```
INERT INTO RoomTraffic
SELECT A.antenna as antenna,A.antenna as id,min(A.startDate) as minTime,min(A.startDate)
as timeid,count(distinct A.uid) as counter FROM Event.win:time_batch(5 sec) A GROUP BY
A.antenna
```

Στη συνέχεια, υπάρχει αυτό το οποίο υπολογίζει τον μέσο χρόνο παραμονής σε κάποιο από τα δωμάτια. Ονομάζεται *esperBoltAvgWaiting* και το επερώτημα για τον υπολογισμό αυτό παρουσιάζεται εδώ:

```
INSERT INTO AvgWaiting
SELECT A.antenna as id,A.antenna as antenna,cast(min((B.startDate.  toMillisec() -
A.startDate.toMillisec())/1000),long) as minTime,cast(max((B.startDate.toMillisec() -
A.startDate.toMillisec())/1000),long) as maxTime,min(A.startDate) as timeid FROM
Event.win:time_batch(5 sec) A INNER JOIN Event.win:time_batch(6 sec) B ON A.uid=B.uid
WHERE A.antenna != B.antenna and B.startDate IS NOT NULL  and A.startDate IS NOT NULL
and (B.startDate.toMillisec() -A.startDate.toMillisec()) > 0 GROUP BY A.antenna
```

Για την τρίτη μετρική του συστήματος, δηλαδή για την μετρική που υπολογίζει το μέσο πλήθος ατόμων που διασχίζουν έναν αισθητήρα στην μονάδα του χρόνου έχει δημιουργηθεί το *esperBoltAvgPersons* του οποίου το αντίστοιχο επερώτημα παρουσιάζεται εδώ:

```
INSERT INTO AvgPersons
SELECT A.antenna as id,A.antenna as antenna,count(DISTINCT A.uid) as cnt,min(A.startDate)
as timeid FROM Event.win:time_batch(5 sec) A GROUP BY A.antenna
```

Το τελευταίο bolt είναι υπεύθυνο για την μετρική που υπολογίζει την μέση ταχύτητα των μοναδικών αναγνωριστικών. Αυτό για να πραγματοποιηθεί, αρκεί να γίνει παρακολούθηση της διέλευσης των μοναδικών αναγνωριστικών από δύο διαδοχικούς αισθητήρες των οποίων γνωρίζουμε εκ των προτέρων την απόσταση. Έτσι, δεδομένου πως η απόσταση και ο χρόνος είναι γνωστά, είναι δυνατός ο υπολογισμός της ταχύτητας. Το αντίστοιχο επερώτημα είναι το παρακάτω:

```
INSERT INTO PersonVelocity
SELECT DISTINCT A.uid as uid,A.uid as id,cast(avg((B.startDate.  toMillisec() -
A.startDate.toMillisec())/1000),long) as avgTime,min(A.startDate) as timeid from
Event.win:time_batch(5 sec) A INNER JOIN Event.win:time_batch(6 sec) B ON A.uid=B.uid
WHERE A.antenna != B.antenna and B.startDate IS NOT NULL  and A.startDate IS NOT NULL
and (B.startDate.toMillisec() -A.startDate.toMillisec()) > 0
```

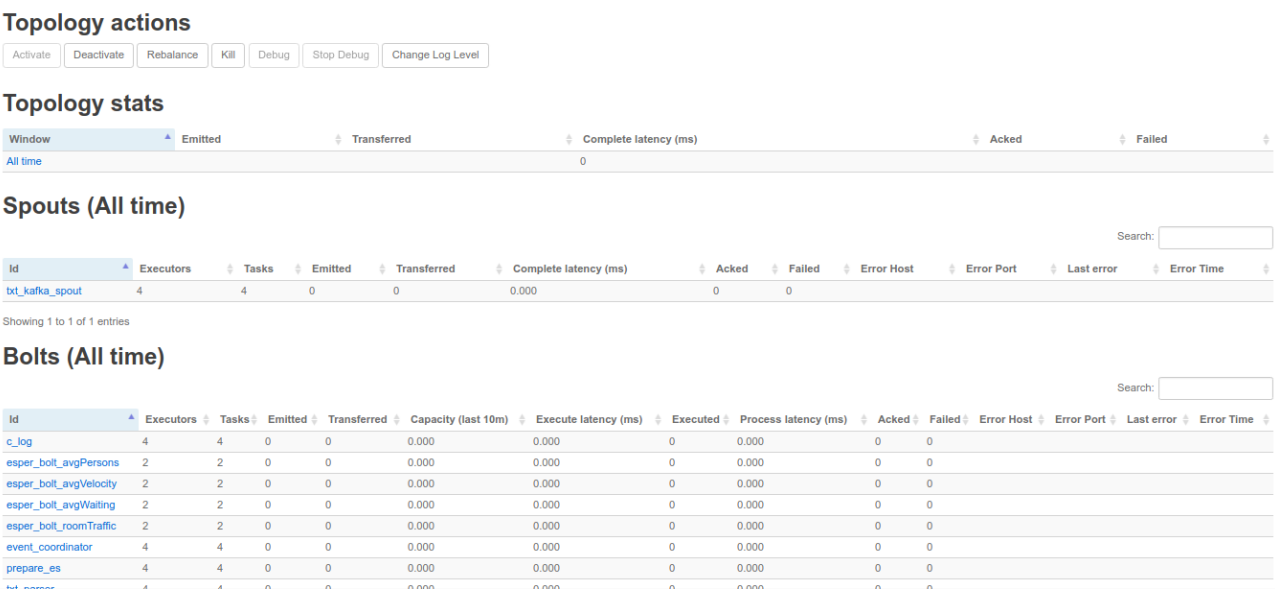
2.3 Το Storm configuration του συστήματος

Το βασικό κομμάτι όπου πραγματοποιείται ο ορισμός της τοπολογίας στο συγκεκριμένο σύστημα είναι το *Flux*. Αυτό ουσιαστικά αποτελεί ένα framework το οποίο δημιουργεί και αναπτύσσει υπολογισμούς του *Apache Storm* πιο εύκολα και με λιγότερες προστριβές. Όπως είναι γνωστό, συχνά προκαλούνται προβλήματα όταν οι βασικές ρυθμίσεις είναι κωδικοποιημένες με το χέρι σε διάφορα σημεία του κώδικα. Ιδανικά, οι προγραμματιστές και οι χρήστες του συστήματος γενικότερα δε θα πρέπει να μεταγλωττίζουν από την αρχή μία εφαρμογή με σκοπό να τροποποιήσουν ορισμένες ρυθμίσεις.

Για αυτό το λόγο υπάρχει το *Flux*, το οποίο με ένα σύνολο από βοηθητικά προγράμματα που καθιστούν τον καθορισμό και την ανάπτυξη των τοπολογιών του *Apache Storm* λιγότερο επώδυνη και εντατική διαδικασία. Ένα από τα σημεία που συχνά προκαλεί προβλήματα είναι το γεγονός πως η ενσωμάτωση ενός γραφήματος τοπολογίας είναι συνδεδεμένη με κώδικα στη γλώσσα προγραμματισμού JAVA, οπότε οποιεσδήποτε αλλαγές απαιτούν επαναμεταγλώττιση του αρχείου jar της τοπολογίας. Το Flux όμως, στοχεύει στο να λύσει αυτή τη δυσκολία επιτρέποντας να δημιουργηθεί ένα πακέτο με όλα τα συστατικά του Storm σε ένα μοναδικό αρχείο jar και να χρησιμοποιηθεί ένα εξωτερικό αρχείο κειμένου για να οριστεί η διάταξη και η ρύθμιση των τοπολογιών.

Επιπροσθέτως, το flux configuration είναι το σημείο όπου ορίζεται η τοπολογία που χρησιμοποιήθηκε για την επίλυση του συγκεκριμένου προβλήματος.

Σε αυτό το σημείο παρουσιάζονται με τη μορφή στιγμιότυπων οθόνης υπολογιστή οι βασικές ρυθμίσεις του *Storm*:



Εικόνα 26: Ρυθμίσεις για το Storm

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
c_log	4	4	0	0	0.000	0.000	0	0.000	0	0				
esper_bolt_avgPersons	2	2	0	0	0.000	0.000	0	0.000	0	0				
esper_bolt_avgVelocity	2	2	0	0	0.000	0.000	0	0.000	0	0				
esper_bolt_avgWaiting	2	2	0	0	0.000	0.000	0	0.000	0	0				
esper_bolt_roomTraffic	2	2	0	0	0.000	0.000	0	0.000	0	0				
event_coordinator	4	4	0	0	0.000	0.000	0	0.000	0	0				
prepare_es	4	4	0	0	0.000	0.000	0	0.000	0	0				
txt_parser	4	4	0	0	0.000	0.000	0	0.000	0	0				

Showing 1 to 8 of 8 entries

Topology Visualization

Show Visualization

Topology Configuration

Show 20 entries

Search:

Key	Value
backpressure.disruptor.high.watermark	0.9
backpressure.disruptor.low.watermark	0.4
client.blobstore.class	"org.apache.storm.blobstore.NimbusBlobStore"
dev.zookeeper.path	"/tmp/dev-storm-zookeeper"
drpc.authorizer.acl.filename	"drpc-auth-acl.yaml"
drpc.authorizer.acl.strict	false

Εικόνα 27: Ρυθμίσεις για το Storm

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
txt_kafka_spout	4	4	1180	1180	0.000	0	600				

Showing 1 to 1 of 1 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
c_log	4	4	0	0	0.000	0.444	180	0.500	140	0				
esper_bolt_avgPersons	2	2	60	60	0.002	0.220	1180	0.229	1220	0				
esper_bolt_avgVelocity	2	2	100	100	0.001	0.050	1200	0.050	1200	0				
esper_bolt_avgWaiting	2	2	40	40	0.002	0.167	1200	0.150	1220	0				
esper_bolt_roomTraffic	2	2	60	60	0.002	0.200	1200	0.067	1200	0				
event_coordinator	4	4	2400	5080	0.001	0.169	1180	0.000	0	0				
prepare_es	4	4	140	140	0.002	1.250	240	0.500	220	0				
txt_parser	4	4	1220	1220	0.007	1.133	1200	0.000	0	0				

Showing 1 to 8 of 8 entries

Topology Visualization

Show Visualization

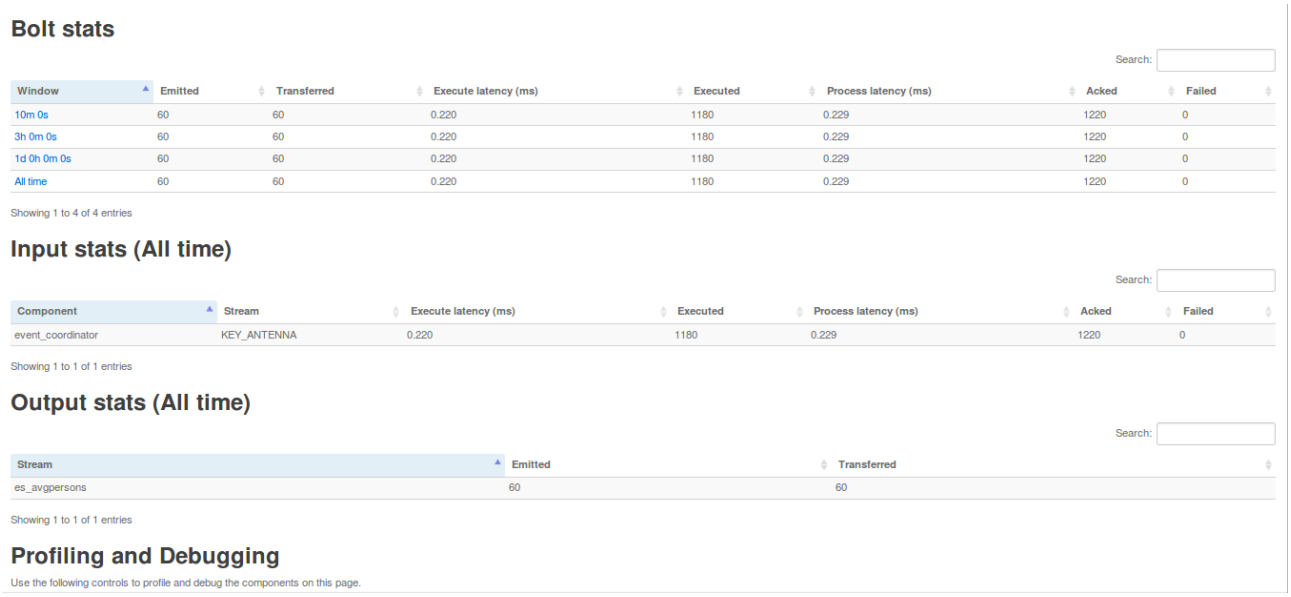
Topology Configuration

Show 20 entries

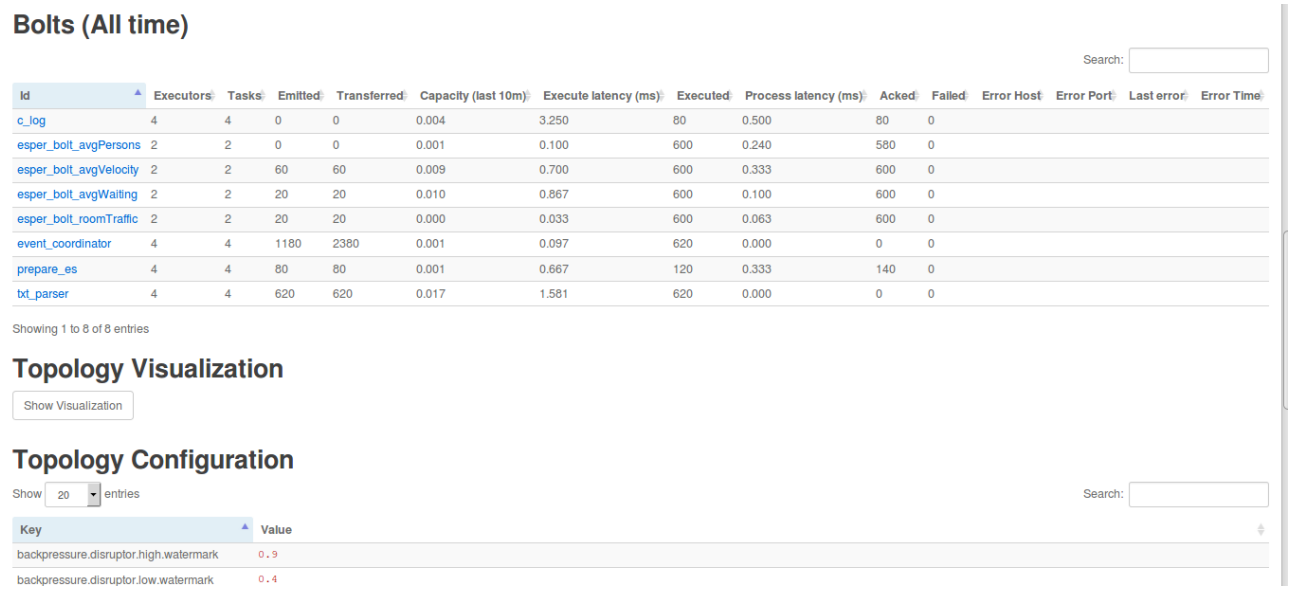
Search:

Key	Value

Εικόνα 28: Ρυθμίσεις για το Storm



Εικόνα 29: Ρυθμίσεις για το Storm



Εικόνα 30: Ρυθμίσεις για το Storm

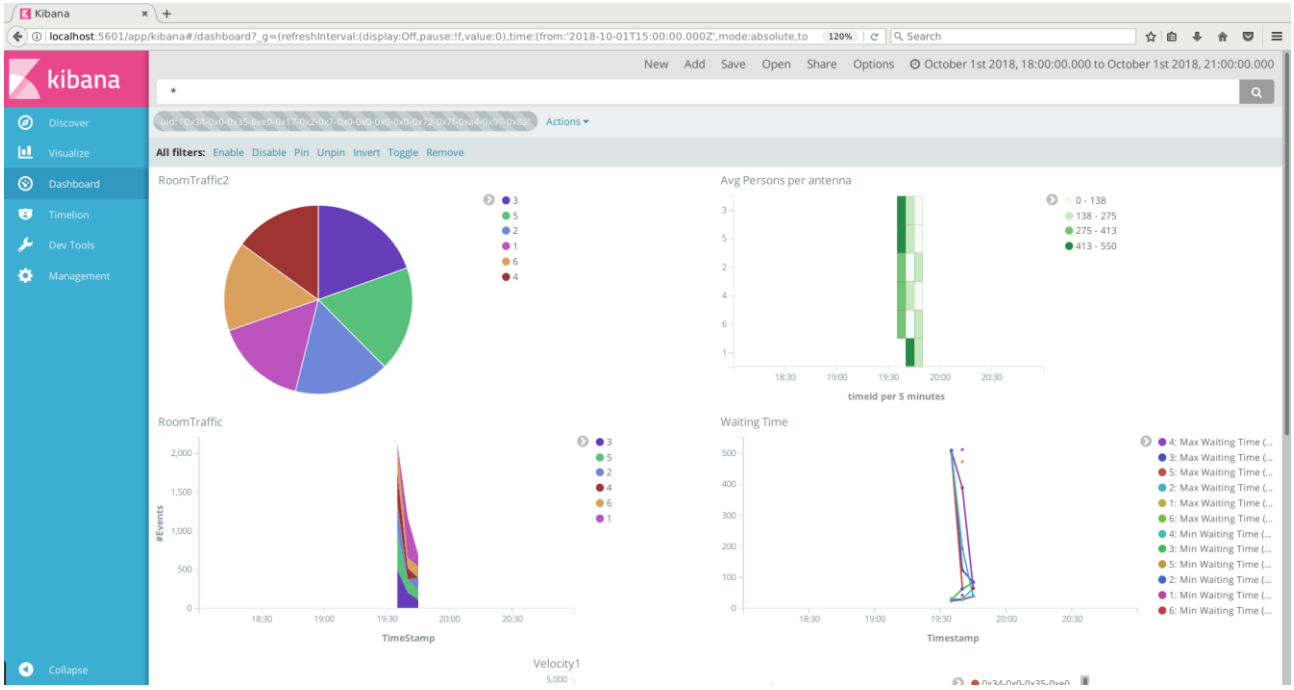
2.4 Η εξαγωγή των αποτελεσμάτων

Κάθε bolt-μετρική εκτελεί τους αντίστοιχους υπολογισμούς και αποστέλλει το αποτέλεσμα στο PrepareESBolt. Σε αυτό το σημείο γίνεται hash το αποτέλεσμα και το παραχθέν hash αποστέλλεται στο EsIndexBolt έτσι ώστε να ευρετηριαστεί στο τμήμα του ElasticSearch για μελλοντική παρουσίαση των δεδομένων αλλά και στο LoggingBolt με σκοπό να καταγραφούν οι σχετικές πληροφορίες εκτέλεσης και αποσφαλμάτωσης.

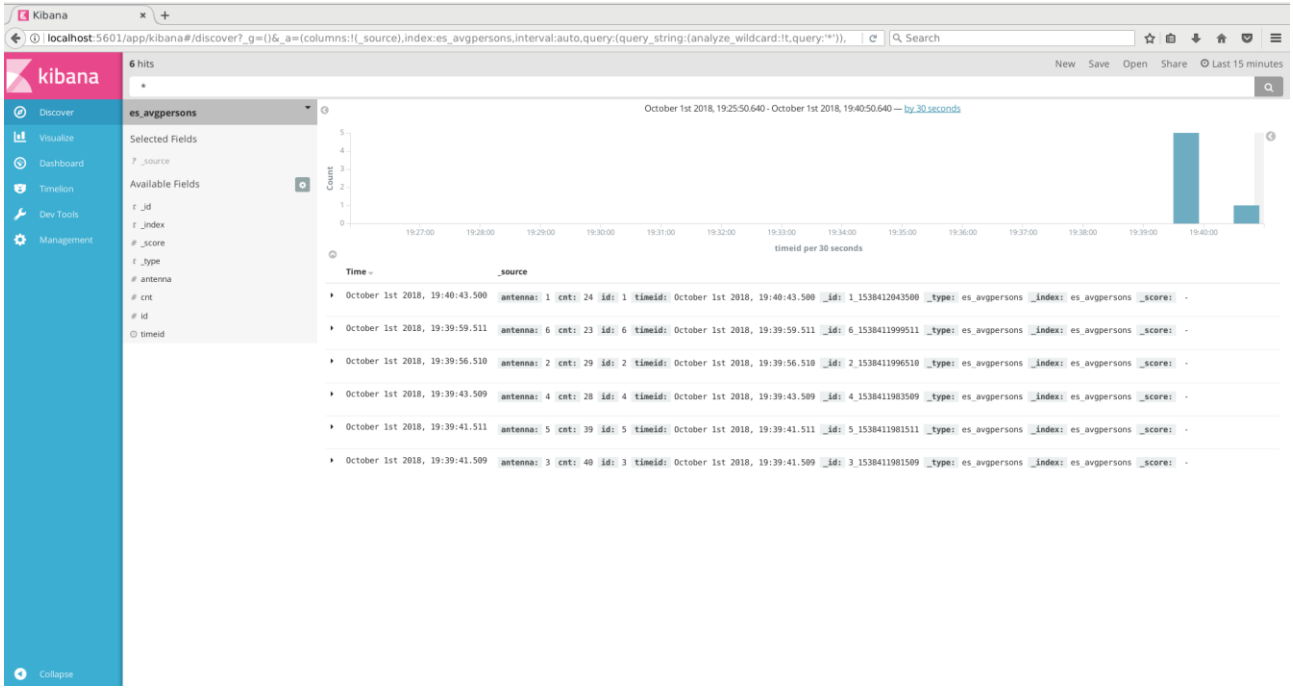
Έπειτα, από τη στιγμή που τα δεδομένα έχουν καταλήξει στο τμήμα του ElasticSearch, είναι πλέον έτοιμα να παρουσιαστούν. Στο τμήμα αυτό, υπάρχει αντιστοίχιση (mapping) στα αρχεία ρυθμίσεων (configuration files) έτσι ώστε να αντιστοιχηθούν τα αντίστοιχα πεδία και να δημιουργηθούν τα επιθυμητά στατιστικά από τις υπολογισμένες μετρικές.

Τέλος, τα αποτελέσματα από το τμήμα του ElasticSearch θα καταλήξουν στο τμήμα του Kibana το οποίο είναι αρμόδιο να τα παρουσιάζει με μορφή γραφημάτων (graphs), πινάκων και ταμπλό (dashboards). Αυτός ο τρόπος παρουσίασης είναι ο πλέον επιθυμητός μιας και η απεικόνιση μεγάλων δεδομένων γίνεται εύκολα και πολύ επεξηγηματικά στο χρήστη.

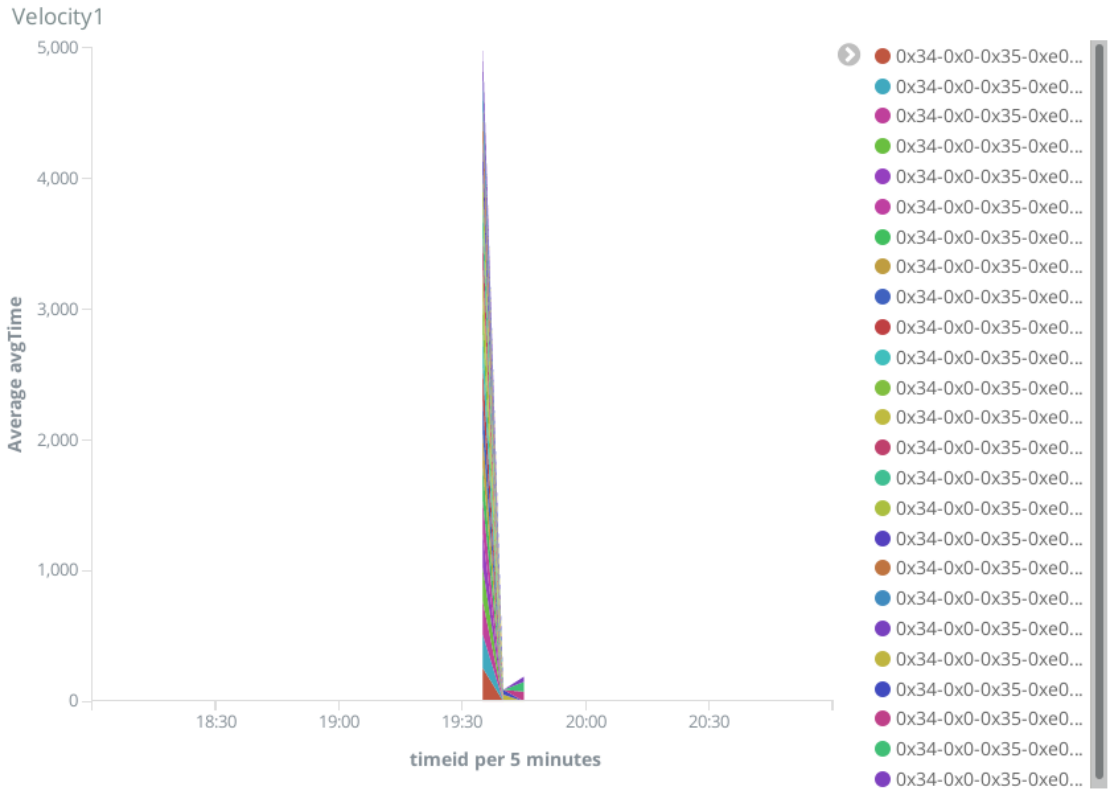
Ορισμένα παραδείγματα απεικόνισης των δεδομένων του προβλήματος με τη χρήση του Kibana είναι τα παρακάτω:



Εικόνα 31: Το dashboard του Kibana με όλες τα χρήσιμα διαγράμματα



Εικόνα 32: Στο διάγραμμα αυτό βλέπουμε τα δεδομένα που εισέρχονται στο σύστημα



Εικόνα 33: Διάγραμμα που απεικονίζει την μετρική της ταχύτητας των μοναδικών αναγνωριστικών.

3. Η λειτουργία των επαναλαμβανομένων συμβάντων

3.1 Ορισμός της επαναληψιμότητας

Η ανίχνευση της περιοδικότητας στα συμβάντα αποτελεί πλέον ένα ανερχόμενο ερευνητικό κλάδο της ανάλυσης δεδομένων και της μηχανικής μάθησης γενικότερα που συνεχώς αποκτά όλο και περισσότερο ενδιαφέρον.

Μία λειτουργία των σύγχρονων συστημάτων που ανιχνεύουν συμβάντα είναι το να εντοπίζουν την περιοδικότητα σε επαναλαμβανόμενα συμβάντα μιας και με τη χρήση αυτών των προτύπων είναι πιθανό να προβλεφθούν ορισμένα από τα μελλοντικά με ικανοποιητική ακρίβεια. Συστήματα πρόληψης βλαβών χρησιμοποιούν αυτήν την προσέγγιση με σκοπό να εντοπίζουν πιο αποδοτικά μελλοντικές βλάβες που τυχαίνει να επαναλαμβάνονται σε τακτά χρονικά διαστήματα. Συστήματα που αναλύουν τα δεδομένα από τα μέσα κοινωνικής δικτύωσης χρησιμοποιούν τέτοιες τεχνικές έτσι ώστε να αναγνωριστούν συμβάντα τα οποία έχουν επαναληφθεί στο παρελθόν.

Βέβαια, το μέγεθος των δεδομένων σήμερα καθώς και η πολυπλοκότητά τους, καθιστά τη διαδικασία της αναγνώρισης της επαναληψιμότητας συμβάντων μια απαιτητική και δύσκολη διαδικασία. Για τους παραπάνω λόγους αυτή η λειτουργία συνήθως εφαρμόζεται σε δεδομένα τα οποία έχουν ήδη υποστεί τεχνικές καθαρισμού, έτσι ώστε να γίνει η εφαρμογή σε όσο το δυνατόν πιο καθαρά από θόρυβο δεδομένα.

Στην παρούσα διπλωματική εργασία έχει μελετηθεί η έννοια της ανεύρεσης επαναλαμβανομένων συμβάντων με τη μορφή εντοπισμού περιπτώσεων όπου κάποιο μοναδικό αναγνωριστικό χρήστη επισκέπτεται κάποιον από τους αισθητήρες σε τακτά χρονικά διαστήματα.

Για τα δεδομένα του συγκεκριμένου προβλήματος δημιουργήθηκε ο μηχανισμός που εντοπίζει επαναλαμβανόμενες σειριακές μεταβάσεις μεταξύ των αισθητήρων. Δηλαδή, όταν ένα μοναδικό αναγνωριστική χρήστη μεταβεί από τον αισθητήρα Α στον αισθητήρα Β και έπειτα μετά από μία αλληλουχία μεταβάσεων επιστρέψει στον αισθητήρα Α και μεταβεί και πάλι στον αισθητήρα Β, τότε ανιχνεύεται ένα συμβάν επαναληψιμότητας.

Για το σκοπό αυτό έχουν δημιουργηθεί άλλα δύο bolts, τα οποία θα προστεθούν στα ήδη δημιουργημένα που αναφέρθηκαν εκτενώς προηγουμένως.

3.2 Μηχανισμός εντοπισμού μεταβάσεων

Το πρώτο από τα δύο *bolts* που ορίστηκαν είναι το *esper_Bolt_Recurring_1* του οποίου σκοπός είναι να βρίσκει τις μεταβάσεις ενός χρήστη στη μεταβλητή του χρόνου. Αυτό το *bolt* είναι υπεύθυνο χρησιμοποιώντας δύο παράλληλα *streams* να προσπαθεί να ταυτίσει τις χρονικές στιγμές όπου πραγματοποιήθηκε η κάθε μία μετάβαση με βάση το μοναδικό αναγνωριστικό χρήστη. Για να το πετύχει αυτό χρησιμοποιούνται δύο μικρά *buckets ελέγχου* έτσι ώστε να επιστραφεί ή πιο κοντινή χρονικά μετάβαση και όχι κάποια από τις μελλοντικές.

Το αντίστοιχο επερώτημα παρουσιάζεται εδώ:

```
INSERT INTO EventExt SELECT DISTINCT A.uid AS uid,A.startDate AS start_time, B.startDate
AS change_time,min(A.antenna,B.antenna) AS old_antenna,max(A.antenna,B.antenna) AS
new_antenna FROM Event.win:length(2) A INNER JOIN Event.win:length(2) B ON A.uid=B.uid
WHERE A.antenna != B.antenna AND B.startDate IS NOT NULL  AND A.startDate IS NOT NULL
AND (B.startDate.toMillisec() -A.startDate.toMillisec()) > 0 "
```

3.3 Συλλογή και ανάλυση δεδομένων μεταβάσεων

Στη συνέχεια της διαδικασίας το πρώτο *bolt* αφού εντοπίσει αυτές τις μεταβάσεις, τις προωθεί στο δεύτερο *bolt*, δηλαδή το *esper_Bolt_Recurring_2*, το οποίο είναι υπεύθυνο να συλλέγει τα στοιχεία αυτά, να τα ομαδοποιεί ανά κάποιο ορισμένο χρονικό διάστημα και τις μεταβάσεις τις οποίες έχουν σχετικά μεγάλη συχνότητα να τις προωθεί στο *prepare_es bolt* που προαναφέρθηκε έτσι ώστε να εμφανιστούν στο περιβάλλον χρήστη. Σε αυτό το σημείο μπορεί να γίνει η ανάλυση των δεδομένων με παρόμοιο τρόπο όπως και στις περιπτώσεις των υπολοίπων επερωτημάτων.

Το επερώτημα που δημιουργήθηκε παρουσιάζεται παρακάτω:

```
INSERT INTO RepeatEvent SELECT A.uid AS uid,A.uid AS id,A.new_antenna AS
new_antenna,A.old_antenna AS old_antenna, CAST(Math.round(COUNT(A.uid)/2),int) AS
cnt,MIN(A.start_time) AS start_time,MIN(A.start_time) AS timeid,MAX(A.change_time) AS
end_time FROM EventExt.win:time_batch(35 sec) A GROUP BY
A.uid,A.new_antenna,A.old_antenna HAVING COUNT(A.uid)/2 > 1"
```

4. ΜΕΛΛΟΝΤΙΚΕΣ ΒΕΛΤΙΩΣΕΙΣ

Σε αυτό το κεφάλαιο αναφέρονται ορισμένες ιδέες σχετικά με πιθανές μελλοντικές βελτιώσεις της παρούσας διπλωματικής εργασίας.

4.1 Σύστημα ροών δεδομένων

Αρχικά, μία πρώτη πιθανή βελτίωση θα ήταν το να δημιουργηθεί μία ροή δεδομένων στο *Apache Kafka*, η οποία θα διαβάζει κατευθείαν τα δεδομένα από τους αισθητήρες. Αυτό θα ήταν χρήσιμο γιατί με αυτόν τον τρόπο θα έχουμε πλήρως πραγματικού χρόνου επεξεργασία. Για να υλοποιηθεί αυτό θα μπορούσε να χρησιμοποιηθεί ο *Apache Flume Agent*. Πιο συγκεκριμένα, το *Apache Flume* είναι ένα διαθέσιμο σύστημα το οποίο είναι κατανεμημένο, αξιόπιστο και είναι ικανό για την αποτελεσματική συλλογή, συγκέντρωση και μεταφορά μεγάλων ποσοτήτων δεδομένων καταγραφής από πολλές διαφορετικές πηγές σε μια κεντρική αποθήκη δεδομένων.

Η χρήση του *Apache Flume* δεν περιορίζεται μόνο στη συσσωμάτωση δεδομένων καταγραφής. Δεδομένου ότι οι πηγές δεδομένων είναι προσαρμόσιμες, το *Flume* μπορεί να χρησιμοποιηθεί για να μεταφέρει τεράστιες ποσότητες δεδομένων συμβάντων, συμπεριλαμβανομένων δεδομένων που παράγονται από κοινωνικά μέσα, μηνύματα ηλεκτρονικού ταχυδρομείου, μετρήσεις από αισθητήρες και πιθανώς οποιασδήποτε πιθανής πηγής δεδομένων χωρίς περιορισμό στο φόρτο δικτύου. Ουσιαστικά, στην περίπτωση αυτή, το *Flume* θα διαβάζει τα ακατέργαστα δεδομένα και θα τα στέλνει στο *Apache Kafka* για να συνεχίσει η διαδικασία ακριβώς όπως έχει πραγματοποιηθεί ήδη.

4.2 Ενσωμάτωση στατικών δεδομένων

Μία δεύτερη βελτίωση που προτείνεται είναι να δημιουργηθεί ένας ενιαίος τρόπος για την αποθήκευση στατικής πληροφορίας, η οποία θα ρυθμίζει ορισμένα στοιχεία του συστήματος. Πιο συγκεκριμένα, θα μπορούσε να υπάρχει η πληροφορία για το πως ομαδοποιούνται πολυάριθμοι αισθητήρες σε ένα δωμάτιο ή ακόμη και να ομαδοποιούνται οι χρήστες. Αυτό είναι χρήσιμο αν γίνεται προσπάθεια για δημιουργία συμπερασμάτων σε ομάδες χρηστών αντί για μοναδικούς χρήστες ή σε ομάδες διαφορετικών αισθητήρων.

Αυτά τα στατικά δεδομένα, από τεχνική πλευρά, θα μπορούν να φορτώνονται στην αρχή της τοπολογίας και αφού διαβαστούν να αποθηκεύονται στην μνήμη του συστήματος για περαιτέρω χρήση.

4.3 Ενσωμάτωση κρυφής μνήμης

Μία ακόμη τροποποίηση θα ήταν να παρέχεται η δυνατότητα να αποθηκεύονται στην προσωρινή μνήμη τα αποτελέσματα των επερωτημάτων έτσι ώστε να μην εκτελούνται από την αρχή κάθε φορά ακόμη κι αν δεν έχουν τροποποιηθεί τα δεδομένα εισόδου. Με αυτόν τον τρόπο θα μειωθεί αισθητά η κατανάλωση των πόρων του συστήματος και θα αυξηθεί η ταχύτητα επιστροφής αποτελεσμάτων από τα διάφορα επερωτήματα.

Επιπροσθέτως, για να εξασφαλιστεί η εγκυρότητα και η επιστροφή αποτελεσμάτων με ενημερωμένα δεδομένα θα μπορούσε να οριστεί μεταβλητή που θα ορίζει το πότε η κρυφή μνήμη θεωρείται άκυρη και χρειάζεται πλέον ενημέρωση με σκοπό την εκτέλεση από την αρχή των επερωτημάτων. Αυτό μπορεί να οριστεί είτε σε κάποια εξωτερική βάση δεδομένων είτε κατευθείαν στην υποβολή της τοπολογίας.

Για να πραγματοποιηθεί αυτό θα πρέπει να δημιουργηθεί μία βάση δεδομένων όπου θα γίνει η καταχώρηση των αποτελεσμάτων αυτών. Αυτή η βάση δεδομένων αν τα δεδομένα είναι μικρά σε έκταση θα μπορούσε να είναι μια *MySQL*. Στην περίπτωση όμως που τα δεδομένα είναι πολλά, μονόδρομος είναι η χρήση κάποιας μη-σχεσιακής βάσης δεδομένων, όπως για παράδειγμα η *MongoDB* και η *HBase*. Στο πρόβλημα αυτό πιθανώς θα εξυπηρετούσε καλύτερα μία μη-σχεσιακή βάση δεδομένων λόγω της φύσης της. Δεν έχει πίνακες και σταθερή δομή καθιστώντας τη διαχείριση της πιο εύκολη. Επίσης παρέχει μεγαλύτερο επίπεδο ελαστικότητας σχετικά με νεότερα μοντέλα δεδομένων. Ακόμη, τέτοιες βάσεις δεδομένων είναι ανοικτού κώδικα και συνεπώς χαμηλού κόστους. Είναι καλύτερα κλιμακώσιμες όσον αφορά το μέγεθος των δεδομένων που στη συγκεκριμένη περίπτωση θα αυξάνεται ραγδαία. Τέλος, δεν υπάρχει η ανάγκη για δημιουργία ενός πολύπλοκου μοντέλου βάσης δεδομένων με όλες τις λεπτομέρειες εκ των προτέρων.

4.4 Ενσωμάτωση λειτουργιών μηχανικής μάθησης

Τέλος, μια τελευταία βελτίωση που προτείνεται αποτελεί η ενσωμάτωση τεχνικών μηχανικής μάθησης κατά την ανάλυση των αποτελεσμάτων έτσι ώστε να ανιχνευθούν συμβάντα και γενικότερα πληροφορία χρήσιμη για τον άνθρωπο με μη εποπτευόμενο τρόπο. Για παράδειγμα αν ανιχνεύονται ξεσπάσματα υψηλής κίνησης στις σκάλες πολυκατοικίας θα μπορούσε να ενσωματωθεί αλγόριθμος για βελτιστοποίηση της διαχείρισης του κόσμου από τον ανελκυστήρα. Ένα ακόμη παράδειγμα είναι η δημιουργία πιθανής πορείας ενός χρήστη από τους διάφορους αισθητήρες με βάση το ιστορικό του και διάφορες άλλες μεταβλητές έτσι ώστε να πραγματοποιηθούν εξατομικευμένες ενέργειες αποκλειστικά για αυτόν.

Μια ακόμη ιδέα αποτελεί την εγκατάσταση ενός *bolt* με σκοπούς μηχανικής μάθησης το οποίο θα είναι ικανό να αναγνωρίζει πιο σύνθετα πρότυπα μέσα στο χρόνο. Στην παρούσα εργασία έχει ενσωματωθεί η λειτουργία της επαναληψιμότητας αλλά αναγνωρίζει πιο απλά πρότυπα από αυτό που προτείνεται εδώ.

Ακόμη θα μπορούσε να εισαχθεί η ιδέα της συνάθροισης χρόνου και να οριστούν δείκτες απόδοσης σε περίπτωση που οι χρονικές στιγμές στις οποίες αναφέρονται τα γεγονότα αποτελούν εορτές και αργίες ή να γίνει διαχωρισμός μεταξύ εργάσιμων και μη ωρών. Με παρόμοιο τρόπο θα μπορούσε να ενσωματωθεί και μηχανισμός ικανός να προβλέπει τις αλλαγές της ώρας.

Οι πιθανές βελτιώσεις είναι πολυάριθμες καθώς ο συγκεκριμένος κλάδος βρίσκεται σε ραγδαία άνθιση τον τελευταίο καιρό. Ο αυτοματισμός είναι αναπόσπαστο τμήμα της ζωής του σύγχρονου ανθρώπου και μελέτες αποδεικνύουν πως θα συνεχίσει να είναι για αρκετές δεκαετίες ακόμη.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Αρχικά, από τη στιγμή που εγκαταστάθηκε ολόκληρο το σύστημα και ξεκίνησε η επεξεργασία των δεδομένων κρίθηκε αναγκαίο να πραγματοποιηθούν έλεγχοι απόδοσης έτσι ώστε να διαπιστωθεί στην πράξη κατά πόσο το σύστημα ανταποκρίνεται σωστά και σε λογικό χρόνο. Τα αποτελέσματα χαρακτηρίζονται ως άκρως θετικά μιας και ο χρόνος που απαιτείται για την επεξεργασία καθώς και την ανάλυση των δεδομένων τα οποία χρησιμοποιήθηκαν για τις δοκιμές απόδοσης είναι εξαιρετικά χαμηλός. Αυτό δικαίωσε τις προσδοκίες και ταυτόχρονα επιβεβαίωσε τις μελέτες σχετικά με τις ικανότητες του συγκεκριμένου συστήματος για επεξεργασία μεγάλων δεδομένων σε μικρό χρόνο.

Ένα ακόμη συμπέρασμα από την ανάπτυξη του συστήματος είναι το ότι η διαδικασία με τον τρόπο που έχει μοντελοποιηθεί είναι ολοκληρωτικά αυτοματοποιημένη. Αυτό έχει ως αποτέλεσμα ένας απλός χρήστης με απλές σχετικά αλλαγές και τροποποιήσεις στο αρχείο παραμετροποίησης να είναι σε θέση να προσαρμόσει ολόκληρο το σύστημα στις συνθήκες του προς επίλυση προβλήματός του. Αυτό είναι ιδιαίτερα χρήσιμο καθώς το σύστημα που έχει πραγματοποιηθεί είναι επαναχρησιμοποιούμενο και ανοικτό σε περαιτέρω βελτιώσεις από άλλους χρήστες.

Επιπροσθέτως, η ενασχόληση με τα προαναφερθέντα εργαλεία απέδειξε πως τα συγκεκριμένα συστήματα όχι μόνο λύνουν προβλήματα όπως και το αντικείμενο της εργασίας αυτής αλλά πως είναι ικανά να εξυπηρετήσουν παράλληλα διάφορους ακόμη σκοπούς. Αυτό το αποδεικνύει και το μέγεθος των δεδομένων που το σύστημα είναι ικανό να επεξεργαστεί καθώς έχει τις δυνατότητες να ανταπεξέλθει σε λογικούς χρόνους ακόμη κι αν τροφοδοτηθεί με πολυάριθμές πηγές δεδομένων αντί για μία όπου οι τάξεις μεγέθους τους θα είναι αρκετά μεγαλύτερες από την ήδη χρησιμοποιημένη.

Τέλος, η δυνατότητα εφαρμογής τεχνικών ανάλυσης δεδομένων σε μεγάλα δεδομένα θεωρείται αναγκαία για την εποχή μας κι αυτό καθιστά το προαναφερθέν σύστημα σημείο αναφοράς για την επίλυση προβλημάτων τέτοιου είδους.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Framework	Πλαίσιο λογισμικού
Big Data	Μεγάλα δεδομένα
Sensor	Αισθητήρας
Data Mining	Εξόρυξη δεδομένων
Batch processing	Επεξεργασία παρτίδας
Stream processing	Επεξεργασία ροής
Scalable	Κλιμακώσιμο
Straightforward	Προφανής
Topology	Τοπολογία
Master node	Κόμβος αφέντης
Worker node	Κόμβος εργάτης
Daemon Process	Διαδικασία δαίμονα
Tuple	Πλειάδα
Transformation	Μεταμόρφωση
Distributed	Κατανεμημένο
Reliable	Αξιόπιστο
Interface	Διεπαφή
Application-specific logic	Εξειδικευμένες από τη λογική του προβλήματος προδιαγραφές
Aggregation	Συσσωμάτωση
Graph	Γράφος
Complex event processing	Πολύπλοκη επεξεργασία συμβάντων
Standard	Πρότυπο
Aggregate function	Λειτουργία επεξεργασίας συνόλων
Pattern matching	Ταίριασμα προτύπων
Event windowing	Παραθυροποίηση συμβάντων
Joining	Συνένωση
Declarative programming language	Γλώσσα δηλωτικού προγραμματισμού
Container	Κιβώτιο
Dependencies	Εξαρτήσεις
Threading model	Μοντέλο σπειρώματος
Seed-node	Κόμβος-σπόρος
Datacenter	Κέντρα δεδομένων
Grouping	Ομαδοποίηση
Aggregation	Συσσωμάτωση
Rollup	Συλλογή
Cube	Κύβος
Sorting	Ταξινόμηση
Filtering	Φιλτράρισμα
Transforming	Μεταμόρφωση
Merging	Συγχώνευση
Splitting	Διαχωρισμός
Duplicating	Διπλασιασμός
Scripts	Τμήματα κώδικα
Topic	Άρθρο
Producer	Παραγωγός

Partition	Τμήμα
Broker	Μεσίτης
Legacy	Κληρονομιά
Stateless	Ανιθαγενής
Byte Array	Πίνακας Ψηφιολέξεων
Pixel	Εικονοστοιχείο
Mapping	Αντιστοίχιση
Configuration Files	Αρχεία Ρυθμίσεων
Graph	Γράφημα
Ταμπλό	Dashboard

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

ADONIS	Article Delivery Over Network Information Systems
ALISE	Association For Library Collections and Technical Services
TCP/IP	Transmission Control Protocol/ Internet Protocol
TEI	Text Encoding Initiative
UNISIST	Universal System for information in Science and technology
W3C	World Wide Web Consortium
EEXI	Ένωση Ελλήνων Χρηστών Internet
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
EPL	Event Processing Language
RFID	Radio-frequency Identification
MPP	Meta Preprocessor
DNS	Domain Name System
MB	Megabyte
ΣΔΒΔ	Σύστημα Διαχείρισης Βάσεων Δεδομένων
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
IR	Infrared
UV	Ultra Violet
CCD	Charge-coupled device
IoT	Internet of Things
UID	Unique Identifier

ΑΝΑΦΟΡΕΣ

[1] Esper correlate; http://www.espertech.com/esper/solution-patterns/#correlate-2. [Προσπελάστηκε 25/05/18]
[2] Complex Event Processing with Esper – Core Concepts; https://medium.com/@bruno.felix/complex-event-processing-with-esper-core-concepts-f97394b39c07. [Προσπελάστηκε 25/05/18]
[3] Esper – The Min and Max Functions; http://esper.espertech.com/release-5.1.0/esper-reference/html/functionreference.html#epl-single-row-function-ref-minmax. [Προσπελάστηκε 25/05/18]
[4] Esper – Data Windows; http://esper.espertech.com/release-7.1.0/esper-reference/html/epl-views.html#win-views. [Προσπελάστηκε 25/05/18]
[5] Esper – Repeat Operator; http://esper.espertech.com/release-5.4.0/esper-reference/html/event_patterns.html#pattern-repeat. [Προσπελάστηκε 25/05/18]
[6] Esper Bolt for Storm (CEP); https://github.com/miguelantonio/storm-esper-bolt. [Προσπελάστηκε 25/05/18]
[7] Mirror of Apache Storm; https://github.com/apache/storm. [Προσπελάστηκε 25/05/18]
[8] Kafka Spout; https://github.com/apache/storm/blob/master/external/storm-kafka-client/src/main/java/org/apache/storm/kafka/spout/KafkaSpout.java. [Προσπελάστηκε 25/05/18]
[9] Flux; http://storm.apache.org/releases/2.0.0-SNAPSHOT/flux.html. [Προσπελάστηκε 25/05/18]
[10] Apache Storm; http://storm.apache.org/index.html. [Προσπελάστηκε 25/05/18]
[11] Apache Storm Tutorial; http://storm.apache.org/releases/1.1.2/Tutorial.html. [Προσπελάστηκε 25/05/18]
[12] Esper (Software); https://en.wikipedia.org/wiki/Esper_(software). [Προσπελάστηκε 06/04/18]
[13] Espertech - Esper; http://www.espertech.com/esper/. [Προσπελάστηκε 11/06/18]
[14] Esper FAQ; http://www.espertech.com/esper/esper-faq/. [Προσπελάστηκε 11/06/18]
[15] Complex Event Processing Made Easy (using Esper): https://dzone.com/articles/complex-event-processing-made. [Προσπελάστηκε 12/06/18]
[16] What is a topic in Apache Kafka; https://www.quora.com/What-is-a-topic-in-Apache-Kafka. [Προσπελάστηκε 02/10/18]
[17] Υλοποίηση Αρχιτεκτονικής Ανάλυσης Ροών Δεδομένων σε πραγματικό χρόνο με υποστήριξη μεθόδων Αποθήκευσης Στοιχείων και Εξόρυξης Πληροφορίας; http://artemis.cslab.ece.ntua.gr:8080/jspui/bitstream/123456789/13541/1/DT2017-0199.pdf. [Προσπελάστηκε 02/10/18]
[18] Apache Kafka Documentation; http://kafka.apache.org/documentation/#api. [Προσπελάστηκε 02/10/18]
[19] Evaluation of a Data Messaging System Solution; https://www.theseus.fi/bitstream/handle/10024/130611/Nguyen_Huong.pdf?sequence=1. [Προσπελάστηκε 02/10/18]
[20] Explaining Apache ZooKeeper; https://stackoverflow.com/questions/3662995/explaining-apache-zookeeper. [Προσπελάστηκε 03/10/18]
[21] Explaining Apache ZooKeeper; https://stackoverflow.com. [Προσπελάστηκε 03/10/18]
[22] What is Elasticsearch and how can I use it?; https://qbox.io/blog/what-is-elasticsearch. [Προσπελάστηκε 03/10/18]
[23] Introduction to Elasticsearch; http://www.jaydeeptrivedi.in/blog/technology/elasticsearch/. [Προσπελάστηκε 03/10/18]
[24] What is Kibana; https://searchitoperations.techtarget.com/definition/Kibana. [Προσπελάστηκε 05/10/18]
[25] What is Apache Storm; https://intellipaat.com/blog/what-is-apache-storm/. [Προσπελάστηκε 09/10/18]
[26] How Logstash works; https://www.elastic.co/guide/en/logstash/master/pipeline.html. [Προσπελάστηκε 09/10/18]
[27] Differences between Kafka and Logstash; https://www.quora.com/What-are-some-of-the-major-differences-and-similarities-between-Logstash-and-Kafka#. [Προσπελάστηκε 09/10/18]
[28] ELK stack architecture; http://sysadminxpert.com/elk-stack-architecture-elasticsearch-logstash-and-kibana/. [Προσπελάστηκε 09/10/18]
[29] Logstash; https://aws.amazon.com/elasticsearch-service/logstash/. [Προσπελάστηκε 09/10/18]
[30] Sensor Data; https://internetofthingsagenda.techtarget.com/definition/sensor-data. [Προσπελάστηκε 09/10/18]
[31] Automatically Identifying Periodic Social Events from Twitter; http://www.aclweb.org/anthology/R15-1043. [Προσπελάστηκε 30/10/18]
[32] Apache Storm Flux; http://storm.apache.org/releases/2.0.0-SNAPSHOT/flux.html. [Προσπελάστηκε 30/10/18]

[33] Apache Storm: How to configure KafkaBolt with Flux; https://sourcevirtues.com/2016/05/03/apache-storm-how-to-configure-kafkabolt-with-flux/. [Προσπελάστηκε 30/10/18]
[34] Apache Flume: User Guide; https://flume.apache.org/FlumeUserGuide.html. [Προσπελάστηκε 30/10/18]
[35] The key differences between SQL and NoSQL DBs; http://www.monitis.com/blog/cc-in-review-the-key-differences-between-sql-and-nosql-dbs/. [Προσπελάστηκε 30/10/18]