Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Υλοποίηση Αρχιτεκτονικής Ανάλυσης Ροών Δεδομένων σε πραγματικό χρόνο με υποστήριξη μεθόδων Αποθήκευσης Στοιχείων και Εξόρυξης Πληροφορίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γιώργος Μ. Χατζηκυριάκος

**Επιβλέπων :** Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Υλοποίηση Αρχιτεκτονικής Ανάλυσης Ροών Δεδομένων σε πραγματικό χρόνο με υποστήριξη μεθόδων Αποθήκευσης Στοιχείων και Εξόρυξης Πληροφορίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γιώργος Μ. Χατζηκυριάκος

**Επιβλέπων :** Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Σεπτεμβρίου 2017.

| ............................ | ............................ | ............................ |
| --- | --- | --- |
| (Υπογραφή) | (Υπογραφή) | (Υπογραφή) |

Αθήνα, Σεπτέμβριος 2017

...................................

Χατζηκυριάκος Μ. Γιώργος

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

Copyright © Χατζηκυριάκος Γιώργος
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

# Περίληψη

Την σημερινή εποχή του Διαδικτύου και της πληροφορίας η παραγωγή δεδομένων είναι πιο μεγάλη από ποτέ και θα συνεχίζει να αυξάνεται με εκθετικούς ρυθμούς. Οι αλλαγές αυτές έχουν δημιουργήσει τον δημοφιλή όρο «Big Data», για την περιγραφή αυτών των μεγάλων ποσοτήτων πληροφορίας. Ένα πολύ σημαντικό χαρακτηριστικό αυτών των δεδομένων είναι η ταχύτητα με την οποία παράγονται καθώς και το γεγονός ότι υπάρχουν πια πολλές πηγές πληροφορίας που δεν υπήρχαν στο κοντινό παρελθόν όπως για παράδειγμα τα μέσα κοινωνικής δικτύωσης. Κάθε επιχείρηση, οργανισμός αλλά και κάθε ξεχωριστός άνθρωπος αποτελεί μία πηγή αναπαραγωγής δεδομένων. Η ανάλυση αυτών των πληροφοριών είναι κρίσιμη και απαιτεί ιδιαίτερη τεχνογνωσία, πράγμα που οδηγεί τους οργανισμούς να αναθέτουν συχνά την εργασία αυτή σε τρίτους (Outsourcing). Τα τελευταία χρόνια γίνεται λόγος για επεξεργασία σε πραγματικό χρόνο, δηλαδή την ώρα παραγωγής της πληροφορίας. Με αυτό τον τρόπο μπορούν να εξάγονται συμπεράσματα όσο το δυνατόν πιο γρήγορα και οι οργανισμοί μπορούν να προβλέψουν γεγονότα ή να έχουν ανταγωνιστικό πλεονέκτημα.

Γι’ αυτό στην παρούσα εργασία καλούμαστε να σχεδιάσουμε μία αρχιτεκτονική για την ανάλυση ροών δεδομένων σε πραγματικό χρόνο που όμως ταυτόχρονα θα προσφέρει και δυνατότητες αποθήκευσης στοιχείων καθώς και επεξεργασία σε παρτίδες. Οπότε, θα μελετήσουμε τεχνολογίες που αφορούν την επεξεργασία ροών πληροφορίας, όπως η Apache Kafka, καθώς και άλλες τεχνολογίες που ειδικεύονται στην επεξεργασία και αποθήκευση Big Data, όπως η Apache HBase και το Apache Spark. Στη πορεία θα φτιάξουμε και θα επεξηγήσουμε στοιχεία της αρχιτεκτονικής που θα συνδέει αυτά τα εργαλεία δημιουργώντας έτσι το τελικό σύστημα. Θα αναλύσουμε επίσης ορισμένα χαρακτηριστικά που αφορούν το σενάριο χρήσης που επιλέξαμε για την πειραματική αξιολόγηση της πλατφόρμας. Στο τέλος θα προσομοιώσουμε κάποιες ροές δεδομένων για να ελέγξουμε πώς δουλεύει το σύστημα και για να πάρουμε μετρήσεις που θα οδηγήσουν στην τελική αξιολόγηση.



# Λέξεις Κλειδιά

Big Data, HDFS, YARN, Data Warehousing, Outsourcing, Stream Processing, Apache Kafka, Spark Streaming, Batch Processing, Apache Spark, Μη Σχεσιακές Βάσεις Δεδομένων, Apache HBase, Apache Avro, Yelp Dataset Challenge, Apache ZooKeeper.

# Abstract

In the current era, of the Internet and information, the production rate of data is greater than ever and it will continue to increase exponentially. These changes have created the popular term “Big Data”, to describe these huge quantities of data. A significant characteristic of this data is how fast they are being produced and also the fact that there are many information sources that there were not around here in the recent past, like, for example, the social networks. Every company, organization and even every individual is an information source. The processing of this information is crucial and demands the necessary know-how, which leads these organizations to often assign this task to a third party (Outsourcing). In the last few years, a very hot topic is real time processing, that is at the time the data is created. That way, conclusions can be made as soon as possible and the organizations can foresee events or have competitive advantage.

That’s why in this particular thesis we will design an architecture for real time stream processing that will also offer data warehousing and batch processing capabilities. So, we will study technologies created for data stream processing, like Apache Kafka, and other technologies specializing in the storage and processing of Big Data, like Apache HBase and Apache Spark. Next, we will create and explain certain things about the architecture that will connect all these tools creating the final system. We will, also, analyze some specifications about the use case that we chose to evaluate our platform. And at the end we are going to simulate some data streams to see how the system works and get metrics that will lead to the final assessments.



# Keywords

Big Data, HDFS, YARN, Data Warehousing, Outsourcing, Stream Processing, Apache Kafka, Spark Streaming, Batch Processing, Apache Spark, NoSQL Databases, Apache HBase, Apache Avro, Yelp Dataset Challenge, Apache ZooKeeper.

# Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Distributed Knowledge and Media Systems (Κατανεμημένης γνώσης και Συστημάτων Πολυμέσων) του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη της Καθηγήτριας Θεοδώρας Βαρβαρίγου.

Αρχικά, θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια κ. Θεοδώρα Βαρβαρίγου που μου ανέθεσε αυτή την ενδιαφέρουσα εργασία και έτσι μου δόθηκε η δυνατότητα να έρθω σε επαφή με τεχνολογίες και εργαλεία που χρησιμοποιούνται ευρύτατα στον τομέα της πληροφορικής και της ανάπτυξης συστημάτων.

Ευχαριστίες θα ήθελα να απευθύνω στον υποψήφιο Διδάκτορα του Ε.Μ.Π., Βρεττό Μουλό και στους συνεργάτες του εργαστηρίου για την καθοδήγηση και την πολύτιμη βοήθεια που μου προσέφεραν στα προβλήματα και εμπόδια που συνάντησα κατά την πραγματοποίηση αυτής της εργασίας.

Τέλος, ένα μεγάλο ευχαριστώ στην οικογένεια μου και στα αγαπημένα μου πρόσωπα για όλη την υποστήριξη και εμπιστοσύνη που μου έδειξαν κατά την διάρκεια των σπουδών μου καθώς και στους συμφοιτητές φίλους μου για την συνεργασία που είχαμε κατά τη διάρκεια της φοίτησης μας.

Χατζηκυριάκος Γιώργος

Αθήνα, Σεπτέμβριος 2017

# Πίνακας Περιεχομένων

**Περίληψη** ........ **5**

**Abstract** ........ **6**

**Ευχαριστίες** ........ **7**

**Πίνακας Περιεχομένων** ........ **8**

**Πίνακας Γραφημάτων** ........ **10**

**1. Εισαγωγή** ........ **12**

**2. Ανάλυση Σημαντικών Εννοιών.** ........ **14**

2.1 Big Data ........ 14
2.2 Cloud Computing ........ 17
2.3 Βάσεις Δεδομένων ........ 20
2.3.1 Σχεσιακές (SQL) Βάσεις δεδομένων ........ 21
2.3.2 NoSQL Βάσεις δεδομένων ........ 23
2.3.3 Σύγκριση NoSQL – SQL βάσεων ........ 26
2.4 Εξωτερική Ανάθεση (Outsourcing) ........ 30
2.4.1 Χαρακτηριστικά Εξωτερικής Ανάθεσης ........ 30
2.4.2 Εξωτερική ανάθεση στην ανάλυση δεδομένων ........ 33

**3. Ανάλυση Προβλήματος** ........ **34**

3.1 Lambda Architecture ........ 34
3.2 Απαιτήσεις προβλήματος ........ 35
3.3 Σημαντικότητα και Περιπλοκότητα ........ 36
3.4 Περιγραφή του προβλήματος προς αντιμετώπιση ........ 37

**4. Τεχνική ανάλυση προτεινόμενης λύσης** ........ **41**

4.1 Ανάλυση components του συστήματος ........ 41
4.1.1 HDFS (Hadoop Distributed File System) ........ 41
4.1.2 YARN (Yet Another Resource Negotiator) ........ 43
4.1.3 Apache Avro ........ 45
4.1.4 Apache HBase ........ 46

4.1.5 Apache Kafka ........ 49
4.1.6 Apache ZooKeeper ........ 52
4.1.7 Apache Spark ........ 54
4.2 Αρχιτεκτονική ........ 58
4.3 Κύκλος ζωής της πληροφορίας στο σύστημα ........ 63
**5. Εργασίες προς εκτέλεση και Έναρξη συστήματος. ........ 66**
5.1 Εργασίες Batch και Stream Processing ........ 66
5.2 Εργαλεία Μετρήσεων ........ 67
5.3 Προετοιμασία και έναρξη συστήματος ........ 68
5.3.1 Προετοιμασία Apache Kafka και Apache Hbase ........ 68
5.3.2 Εισαγωγή αρχικών δεδομένων στο σύστημα ........ 69
5.3.3 Έναρξη batch processing χωρίς Streaming ........ 70
5.3.4 Έναρξη Stream Processing Χωρίς Batch ........ 70
5.3.5 Λειτουργία και των δύο τεχνικών ........ 71
**6. Μετρήσεις ........ 72**
**7. Αξιολόγηση και επίλογος ........ 87**
7.1 Αξιολόγηση Αρχιτεκτονικής ........ 87
7.2 Περαιτέρω Βελτιώσεις και Επίλογος ........ 89
**Βιβλιογραφία ........ 90**
**Παράρτημα ........ 92**
Α. Διαδικασία εγκατάστασης λογισμικού. ........ 92
Α.1 Εγκατάσταση Apache Hadoop v2.7.3 ........ 92
Α.2 Εγκατάσταση Apache ZooKeeper v3.4.9 ........ 105
Α3. Εγκατάσταση Apache HBase v1.2.4 ........ 106
Α.4 Εγκατάσταση Apache Kafka v0.10.2 – Πλατφόρμα Confluent v3.2.0 ........ 113
Α5. Εγκατάσταση Apache Spark v2.1.0 ........ 115
Β. Κώδικας εφαρμογών ........ 117
Β.1. Kafka Producers ........ 117
Β.2 Batch Applications ........ 126
Β.3 Streaming Application ........ 130

# Πίνακας Γραφημάτων

Εικόνα 1 Αλλαγή του Όγκου Δεδομένων τα επόμενα χρόνια ........ 15
Εικόνα 2 Ποικιλία των Big Data ........ 16
Εικόνα 3 Διαδρομές με ποδήλατο "Divvy" στο Chicago ........ 17
Εικόνα 4 Αναπαράσταση Cloud Computing ........ 18
Εικόνα 5 Παραδείγματα Μοντέλων Υπηρεσιών Νέφους ........ 20
Εικόνα 6 Παράδειγμα Σχεσιακής Βάσης Δεδομένων ........ 22
Εικόνα 7 Λογότυπα από 16 πολύ γνωστές NoSQL Βάσεις Δεδομένων ........ 24
Εικόνα 8 Παραγωγή Δεδομένων με την πάροδο των χρόνων ........ 28
Εικόνα 9 Κλιμακωσιμότητα NoSQL vs RDBMS ........ 28
Εικόνα 10 Θεώρημα CAP ........ 29
Εικόνα 11 Θετικά Στοιχεία Outsourcing ........ 31
Εικόνα 12 Παράδειγμα συστήματος που χρησιμοποιεί Lambda Architecture ........ 35
Εικόνα 13 Διαφορές Μεγεθών σε Batch και Stream Processing ........ 37
Εικόνα 14 Χρέος σε σχέση με το ΑΕΠ σε ορισμένες Ευρωπαϊκές χώρες μετά το 2008 ...... 38
Εικόνα 15 Αρχιτεκτονική HDFS ........ 42
Εικόνα 16 Replication στο HDFS ........ 43
Εικόνα 17 Αρχιτεκτονική Yarn ........ 44
Εικόνα 18 Παράδειγμα συνδεσιμότητας YARN ........ 44
Εικόνα 19 Σύγκριση Apache Avro με παρόμοια πρότυπα ........ 45
Εικόνα 20 Λογότυπο HBase ........ 46
Εικόνα 21 Πίνακας στην HBase ........ 48
Εικόνα 22 Αρχιτεκτονική HBase ........ 49
Εικόνα 23 Η Kafka ως pipeline δεδομένων σε άλλα συστήματα ........ 50
Εικόνα 24 Τα βασικά API της Kafka ........ 51
Εικόνα 25 Kafka Brokers ........ 52
Εικόνα 26 Αρχιτεκτονική ZooKeeper ........ 53
Εικόνα 27 ZooKeeper Data Node (znode) ........ 53
Εικόνα 28 Οικοσύστημα Spark ........ 55
Εικόνα 29 Ανατομία και πράξεις ενός RDD ........ 56
Εικόνα 30 Αρχιτεκτονική Spark ........ 58
Εικόνα 31 Αρχιτεκτονική Προτεινόμενης Λύσης ........ 59
Εικόνα 32 Κύκλος ζωής «πληροφορίας streaming» ........ 63
Εικόνα 33 Εγγραφές Review και Tip προς αποθήκευση στην HBase ........ 64

Εικόνα 34 Κύκλος ζωής «πληροφορίας batch» ........ 65
Εικόνα 35 Batch Processing Jobs Times........ 73
Εικόνα 36 Executors για Query1 σε διαφορετικά στάδια υπολογιστικού φόρτου........ 74
Εικόνα 37 Executors για Query1 σε διαφορετικά στάδια υπολογιστικού φόρτου........ 74
Εικόνα 38 Κομμάτι χρονοδιαγράμματος εκτέλεσης του query2 όταν γίνεται kai stream processing ταυτόχρονα........ 75
Εικόνα 39 Οι πόροι του Cluster σε διάφορα στάδια ........ 76
Εικόνα 40 Executors στις για τις εργασίες streaming ........ 77
Εικόνα 41 Γενικά στοιχεία Spark Streaming όταν έχουμε μόνο streaming job........ 79
Εικόνα 42 Γενικά στοιχεία Spark Streaming μετά το τέλος του query1........ 80
Εικόνα 43 Γενικά στοιχεία Spark Streaming μετά το τέλος του query2........ 81
Εικόνα 44 Μετρήσεις Kafka Broker σε διαφορετικά σενάρια........ 82
Εικόνα 45 Μετρήσεις των topics μόνο όταν έχουμε streaming jobs ........ 83
Εικόνα 46 Μετρήσεις των topics όταν έχουμε streaming job και ολοκληρώθηκε το query1 ........ 84
Εικόνα 47 Μετρήσεις των topics όταν έχουμε streaming job και ολοκληρώθηκε το query2 ........ 85
Εικόνα 48 Consumer Lag κατά τη διάρκεια λειτουργίας........ 86

Πίνακας 1 Τρόπος αποθήκευσης στοιχείων μίας επιχείρησης, ενός review και ενός tip ..... 64
Πίνακας 2 Δείγμα αποτελέσματος query1 ........ 75
Πίνακας 3 Δείγμα αποτελέσματος query2 ........ 76
Πίνακας 4 Πίνακες αποτελεσμάτων δουλείας streaming........ 78

# 1. Εισαγωγή

Στην παρούσα διπλωματική εργασία θα φτιαχτεί και θα δοκιμαστεί μια αρχιτεκτονική για ένα σύστημα ανάλυσης δεδομένων. Τη σημερινή εποχή, που θα μπορούσε να ονομαστεί και εποχή της πληροφορίας, η ανταλλαγή και η ανάλυση δεδομένων αποτελεί σημαντικό και αναπόσπαστο κομμάτι της καθημερινής ζωής. Αυτό γίνεται καλύτερα σαφές αν σκεφτούμε την εξάπλωση του διαδικτύου και το μεγάλο όγκο δεδομένων που κυκλοφορούν. Ένας πολύ δημοφιλής όρος που έχει επικρατήσει και αναφέρεται σε αυτή την καινούρια «τάξη πραγμάτων» είναι το Big Data, όπου η λέξη «Big» δίνει έμφαση στη ποσότητα.

Τα τελευταία χρόνια μεγάλοι οργανισμοί αλλά και μικρομεσαίες επιχειρήσεις τείνουν να ψηφιοποιούν τις διαδικασίες παραγωγής τους και για το λόγο αυτό η παραγωγή δεδομένων και πληροφοριών, από άποψη μεγέθους καθώς και ταχύτητας βρίσκεται σε πολύ μεγάλα επίπεδα. Η ανάλυση και εξαγωγή συμπερασμάτων από αυτά τα δεδομένα είναι κρίσιμη καθώς κρύβεται γνώση ικανή να προσφέρει βελτιώσεις στη παραγωγική διαδικασία. Μεγάλοι όγκοι δεδομένων παράγονται επίσης και σε ατομικό επίπεδο μέσω διαφόρων τεχνολογιών, όπως τα κοινωνικά δίκτυα. Ο CEO της Google είχε πει ότι «από την έναρξη του πολιτισμού μέχρι το 2003 είχαν δημιουργηθεί 5 exabytes δεδομένων, τώρα αυτό το ποσό παράγεται σε 2 μέρες»[16] πράγμα που δείχνει αυτή τη θηριώδη παραγωγή πληροφορίας.

Σέ ένα τέτοιο περιβάλλον φαίνεται ξεκάθαρα η ανάγκη ύπαρξης μιας αρχιτεκτονικής που θα ικανοποιεί τις ανάγκες των Big Data. Όμως και τα Big Data εξελίσσονται κι αυτά, δίνοντας δυνατότητα ανάπτυξης σε νέες τεχνικές και εργαλεία που θα προσφέρουν το επιθυμητό αποτέλεσμα.

Η ζήτηση για οργανισμούς και άτομα που ειδικεύονται στα Big Data είναι πολύ αυξημένη και οι ειδήμονες αυτοί είναι πολλές φορές δύσκολο να βρεθούν. Ταυτόχρονα οι εταιρίες έχουν ανάγκη για την ανάλυση των δεδομένων τους ώστε να μείνουν ανταγωνιστικές στην αγορά και έτσι οδηγούνται στην ανάθεση αυτής της εργασίας σε τρίτους οργανισμούς με αυτή την ειδίκευση. Η διαδικασία αυτή λέγεται outsourcing.

Επίσης, πέρα από την τεχνογνωσία, απαραίτητος είναι ο καταμερισμός και η εξοικονόμηση υπολογιστικών πόρων καθώς και η κατανομή του φόρτου εργασίας ώστε οι εργασίες πάνω στα Big Data να γίνονται αποδοτικά. Όλα αυτά προσφέρονται από την τεχνολογία του Υπολογιστικού Νέφους ή όπως είναι ευρέως γνωστό Cloud Computing. Η ανάλυση δεδομένων μεγάλης κλίμακας θα πρέπει επίσης να συνοδεύεται και από άλλες διαφορετικές τεχνολογίες οι οποίες θα πρέπει να εκμεταλλεύονται τον παραλληλισμό και

γενικά το Cloud Computing. Και γι’ αυτό, θα μιλήσουμε για εργαλεία που προσφέρουν ανάλυση της πληροφορίας με κατανεμημένο τρόπο. Στο θέμα της αποθήκευσης χρειαζόμαστε κάποια κατανεμημένη βάση δεδομένων αλλά επειδή τα Big Data δεν διαθέτουν αυστηρή δομή θα κοιτάξουμε περισσότερο Μη Σχεσιακές (NoSQL) βάσεις δεδομένων.

Πέρα απ’ όλα αυτά, νέες ανάγκες εμφανίζονται στον ορίζονται, ανάγκες που οι αναλυτές καλούνται να καλύψουν. Η εξέλιξη αυτών των αναγκών αλλά και των ίδιων των συστημάτων διαχείρισης οδηγεί σε καινούρια ήδη επεξεργασίας, αυτή των ροών πληροφορίας (data stream processing) σε πραγματικό χρόνο, δηλαδή τη στιγμή που τα δεδομένα εισέρχονται στο σύστημα μας. Αυτός είναι και ο απώτερος σκοπός αυτής της εργασίας, να δημιουργηθεί και να αξιολογηθεί ένα σύστημα που θα ειδικεύεται σε αυτού του είδους τις αναλύσεις αλλά θα προφέρει ταυτόχρονα και επεξεργασία μεγάλου όγκου αποθηκευμένης πληροφορίας για την αντιμετώπιση των αναγκών των Big Data.

# 2. Ανάλυση Σημαντικών Εννοιών.

Όπως αναφέραμε και προηγουμένως, το μέγεθος των δεδομένων που δημιουργούνται αυξάνεται με πολύ μεγάλους ρυθμούς. Για την αποτελεσματική διαχείριση και αποθήκευσή τους δεν αρκούν οι συμβατικές τεχνικές που χρησιμοποιούσαμε παλαιότερα, καθώς δεν ήταν σχεδιασμένες για την επεξεργασία τόσο μεγάλου όγκου δεδομένων.

Σε αυτό το κεφάλαιο αναλύουμε σύντομα κάποιες σημαντικές έννοιες και τι είδους αλλαγές και καινούριες τεχνολογίες φέρνει η εποχή αυτή στη διαχείριση των δεδομένων καθώς επίσης και τι εργαλεία μπορούμε να χρησιμοποιήσουμε για να επιλύσουμε τα προβλήματα που προκύπτουν.

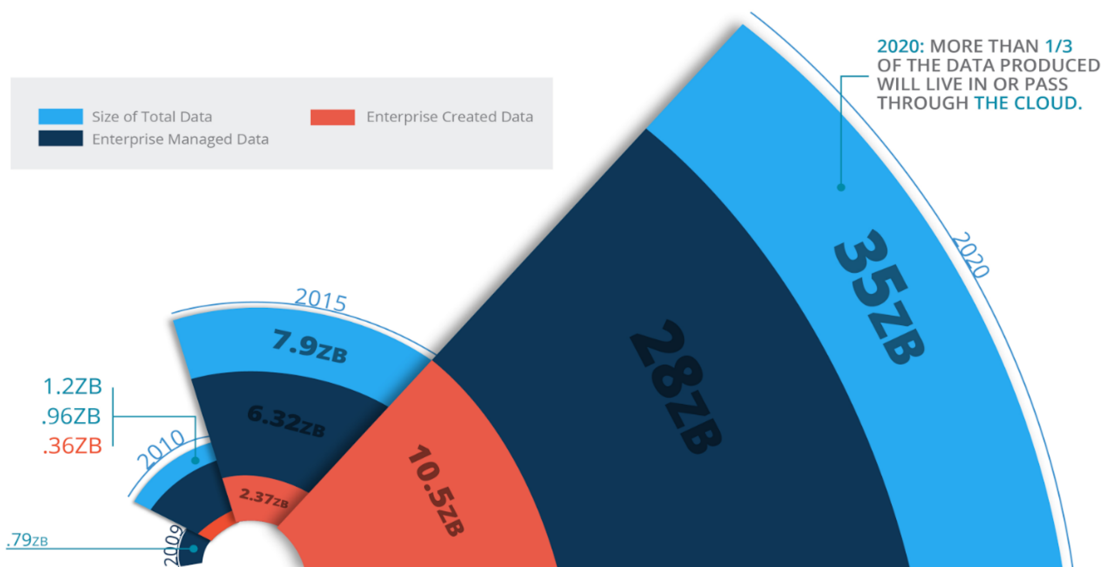
## 2.1 Big Data

Ένας ορισμός των Μεγάλων Δεδομένων που μπορεί να δοθεί είναι ότι «Τα Big Data αποτελούν σύνολα δεδομένων τόσο μεγάλα και περίπλοκα όπου οι παραδοσιακές μέθοδοι επεξεργασίας δεν μπορούν να ανταπεξέλθουν στις ανάγκες τους»[25].Το πρόβλημα ανάλυσης των Big data ξεκίνησε στις αρχές του 21ου αιώνα μαζί με την εξάπλωση της ψηφιακής τεχνολογίας. Από τους πρώτους κλάδους που ανέδειξαν το πρόβλημα αυτό ήταν οι μηχανές αναζήτησης και τα κοινωνικά δίκτυα. Για να καταλάβουμε καλύτερα την έκταση του προβλήματος αρκεί να σκεφτούμε ότι η Google επεξεργάζεται 5.5 δισεκατομμύρια αναζητήσεων κάθε μέρα[3]. Παρόλα αυτά καταφέρνει να παρουσιάζει τα αποτελέσματα αυτών σε εκατοστά του δευτερολέπτου.

Για τον ορισμό της έννοιας των Big data χρησιμοποιούμε τον χαρακτηρισμό των 7 V's, Volume, Velocity, Variety, Variability, Veracity, Visualization, Value[1]. Είναι ενδιαφέρον να αναφερθεί ότι στα χαρακτηριστικά αυτά, με την εξέλιξη των συστημάτων ανάλυσης και των αναγκών, προστίθενται και καινούρια V's . Για να κατανοήσουμε αν ένα πρόβλημα σχετίζεται με τα Big data, αρκεί να δούμε αν περιέχει τις παρακάτω παραμέτρους.

*Όγκος (Volume)*

Αναφέρεται στο πόσα δεδομένα έχουμε. Για τη μέτρηση του μεγέθους χρησιμοποιούσαμε Gibabytes (GB) αλλά τώρα τα μεγέθη έχουν φτάσει σε τέτοιο σημείο που χρησιμοποιούνται σαν μονάδα μέτρησης τα Zettabytes (ZB) ή ακόμη και τα Yottabytes (YB). Η εξέλιξη του Internet Of Things (IoT) και των μέσων κοινωνικής δικτύωσης έχει οδηγήσει στην εκθετική αύξηση της συνολικής πληροφορίας.



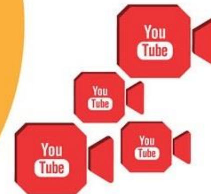

*Εικόνα 1 Αλλαγή του Όγκου Δεδομένων τα επόμενα χρόνια*

*Ταχύτητα (Velocity)*

Η ταχύτητα με την οποία δεχόμαστε τα δεδομένα και η ανάγκη για την επεξεργασία τους σε πραγματικό χρόνο. Εφαρμογές κινητής τηλεφωνίας που λαμβάνουν δεδομένα σε πραγματικό χρόνο, τα επεξεργάζονται και δίνουν αποτελέσματα καθώς και δεδομένα από αισθητήρες, όπου η άμεση ανταπόκριση είναι πολύ σημαντική, αποτελούν μερικά παραδείγματα. Οι εποχές που μαζεύονταν τα δεδομένα σε παρτίδες για επεξεργασία έχουν περάσει, τώρα αν δεν έχουμε real-time πρόσβαση και ανάλυση η ταχύτητα θεωρείται αργή.

*Ποικιλία (Variety)*

Μη δομημένοι ή ημιδομημένοι τύποι δεδομένων, όπως κείμενο, ήχος, βίντεο ή συνδυασμός αυτών χρειάζονται ανάλυση διαφορετική από τους δομημένους τύπους δεδομένων. Πολλές φορές σε μία γνωστή πηγή πληροφοριών ο τύπος των δεδομένων μπορεί να αλλάξει χωρίς προειδοποίηση και έτσι να παρουσιαστεί μεγαλύτερη πολυπλοκότητα. Η οργάνωση τέτοιων δεδομένων ώστε να έχουν κάποιο νόημα είναι ιδιαίτερα δύσκολη.



*Εικόνα 2 Ποικιλία των Big Data*

*Μεταβλητότητα (Variability)*

Η μεταβλητότητα των δεδομένων διαφέρει από την ποικιλία που διαθέτουν. Τα δεδομένα αλλάζουν σημασία με την πάροδο του χρόνου ή το περιβάλλον στο οποίο βρίσκονται. Στην επεξεργασία φυσικών γλωσσών από είναι πολύ συχνό. Μία λέξη μπορεί να έχει πολλές διαφορετικές ερμηνείες και δημιουργούνται καινούριες εκφράσεις που αντικαθιστούν τις παλιές. Αυτού του είδους η μεταβλητότητα των Big Data δημιουργεί πολλές προκλήσεις και δυσκολίες στη σωστή αξιοποίηση τους.

*Φιλαλήθεια (Veracity)*

Το να αντιλαμβάνεσαι την πληροφορία που σου δίνουν τα δεδομένα είναι σημαντικό, αλλά εξίσου σημαντικό είναι να μπορεί να κριθεί αυτή η πληροφορία ως ακριβής ή ως ατελής και ανακριβής. Όταν λαμβάνονται πολλά δεδομένα από διαφορετικές πηγές θα πρέπει να «καθαριστούν» και να πεταχτεί η πληροφορία που θεωρείται βλαβερή, ανούσια και θόρυβος. Ένα απλό παράδειγμα, ο έλεγχος της βάσης δεδομένων για τυχών χρήστες με ψευδή ονόματα και στοιχεία.

*Οπτικοποίηση (Visualization)*

Αυτή η ιδιότητα είναι πολύ σημαντική στη σημερινή εποχή. Η χρήση γραφημάτων και γραφικών αναπαραστάσεων για την απεικόνιση μεγάλης ποσότητας περίπλοκων

δεδομένων δίνει περισσότερο ανθρώπινο νόημα στις αναλύσεις μας. Η οπτικοποίηση των Big Data δίνει αποτελέσματα που είναι εύκολο να τα διαχειριστεί κάποιος.

*Εικόνα 3 Διαδρομές με ποδήλατο "Divvy" στο Chicago*

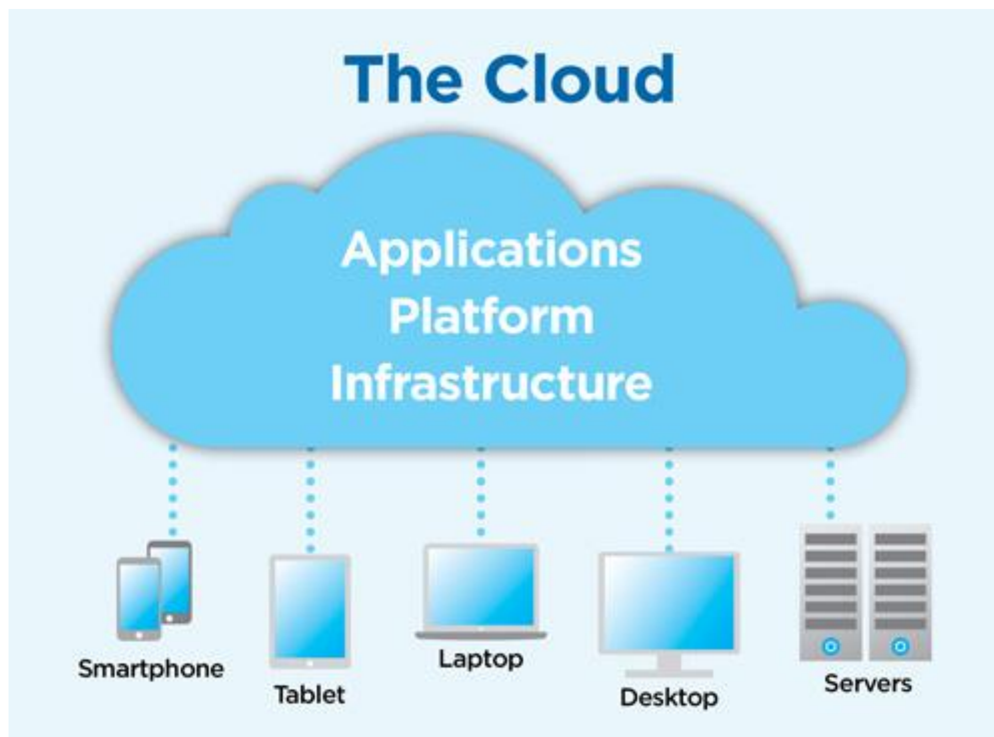
*Αξία (Value)*

Τα δεδομένα έχουν κρυφή αξία και πρέπει αυτή να ανακαλυφθεί μέσα από την επεξεργασία τους. Από όλα αυτά τα δεδομένα που συλλέγονται πρέπει να βρεθεί, για παράδειγμα, το εξάρτημα ενός συστήματος που πρόκειται να χαλάσει ή να γίνει μια πρόταση για τον καταναλωτή με βάση τα συναισθήματα ή τη συμπεριφορά του. Η μεγάλη πρόκληση για τα Big data είναι η αναγνώριση μοτίβων, η πρόβλεψη συμπεριφοράς και η λήψη συνειδητών αποφάσεων. Η εξόρυξη αυτής της αξίας είναι και ο απώτερος σκοπός ενός συστήματος ανάλυσης Big Data.

## 2.2 Cloud Computing

Οι εφαρμογές που αφορούν Big Data περιέχουν την αποθήκευση, την επεξεργασία και τη μεταφορά μεγάλου όγκου δεδομένων, κάτι που οι παραδοσιακές τεχνολογίες δεν μπορούν να επιτύχουν. Τη λύση στα περισσότερα από αυτά τα προβλήματα τη δίνει το Cloud Computing (Υπολογιστικό Νέφος), ένα νέο πεδίο πληροφορικής που παρέχει νέες προοπτικές σε τεχνολογίες δικτύωσης. Επίσημος ορισμός για το μοντέλο αυτό έχει δοθεί από το NIST (National Institute of Standards and Technology) [18] και ουσιαστικά αποτελεί μία τεχνολογία που δίνει την δυνατότητα στους χρήστες, είτε είναι μεμονωμένοι είτε ολόκληρες επιχειρήσεις και οργανισμοί, να αποθηκεύουν, επεξεργάζονται και να διαχειρίζονται τα δεδομένα τους σε ένα «νέφος» υπολογιστικών πόρων απομακρυσμένο από αυτούς και στο οποίο έχουν πολύ

εύκολη πρόσβαση. Ο καθένας μπορεί να λάβει τις υποδομές που χρειάζεται και τίποτα παραπάνω.




*Εικόνα 4 Αναπαράσταση Cloud Computing*

Η τεχνολογία αυτή θα μπορούσε να πει κανείς ότι πάει «χέρι-χέρι» με τα Big Data. Αρχικά, οι δυνατότητες επέκτασης αποθηκευτικού χώρου και επεξεργαστικής ισχύος είναι πολύ μεγάλες (στην αποθήκευση σχεδόν απεριόριστες), λύνοντας έτσι το πρόβλημα του τεράστιου όγκου πληροφοριών. Αντί να κάνεις μια εργασία σε έναν υπολογιστή για δέκα ώρες μπορείς να την κάνεις σε δέκα υπολογιστές για μία ώρα. Αυτό σημαίνει αποδοτικό κόστος υπηρεσίας, τόσο στην συντήρηση όσο και στην αναβάθμιση διότι μπορούν να χρησιμοποιηθούν υπολογιστικά συστήματα μέσης ισχύος και υπάρχει ευελιξία στην κατανομή πόρων, ανάλογα με το φόρτο εργασίας. Ο διαμοιρασμός, η εύκολη πρόσβαση στις πληροφορίες και η δημιουργία αντιγράφων ασφαλείας είναι επίσης μεγάλα πλεονεκτήματα καθώς η πληροφορία βρίσκεται σε πολλά σημεία ταυτόχρονα (data centers) και η πρόσβαση της γίνεται μέσω του διαδικτύου. Επίσης, έχουν αναπτυχθεί πολλά εργαλεία και τεχνικές που εκμεταλλεύονται το Υπολογιστικό Νέφος για την επεξεργασία Big Data, όπως για παράδειγμα το Map Reduce[11].

Σήμερα, η εξέλιξη του Cloud Computing είναι ραγδαία, και πάρα πολλές εταιρίες προφέρουν υπηρεσίες στο «νέφος» σε διάφορα επίπεδα. Με αυτό τον τρόπο ο πελάτης δεν χρειάζεται να ασχοληθεί καθόλου με τη συντήρηση και την απαρχαίωση του εξοπλισμού,

που σε διαφορετική περίπτωση θα έπρεπε να αγοράσει. Η προσβασιμότητα των ενοικιαζόμενων πόρων γίνεται, όπως αναφέρθηκε, μέσω του διαδικτύου οπότε οι εξωτερικές αναθέσεις (outsourcing) ξεφεύγουν από τα γεωγραφικά σύνορα. Ένα παράδειγμα αποτελεί το γεγονός ότι πολλές επιχειρήσεις από τις Ηνωμένες Πολιτείες Αμερικής βασίζονται σε εξωτερικούς συνεργάτες στην Ευρώπη για την προστασία των δεδομένων τους [20]. Αυτό συμβαίνει καθώς στην Ευρώπη οι νόμοι αλλά και η καλύτερη τεχνογνωσία πάνω στον τομέα καθιστούν την προστασία και την αποθήκευση των δεδομένων στο Cloud πολύ ασφαλέστερη.

Οι υπηρεσίες που προσφέρονται από το «νέφος» μπορούν να χωριστούν σε τρείς κατηγορίες[18]:

*Software as a Service (SaaS)*

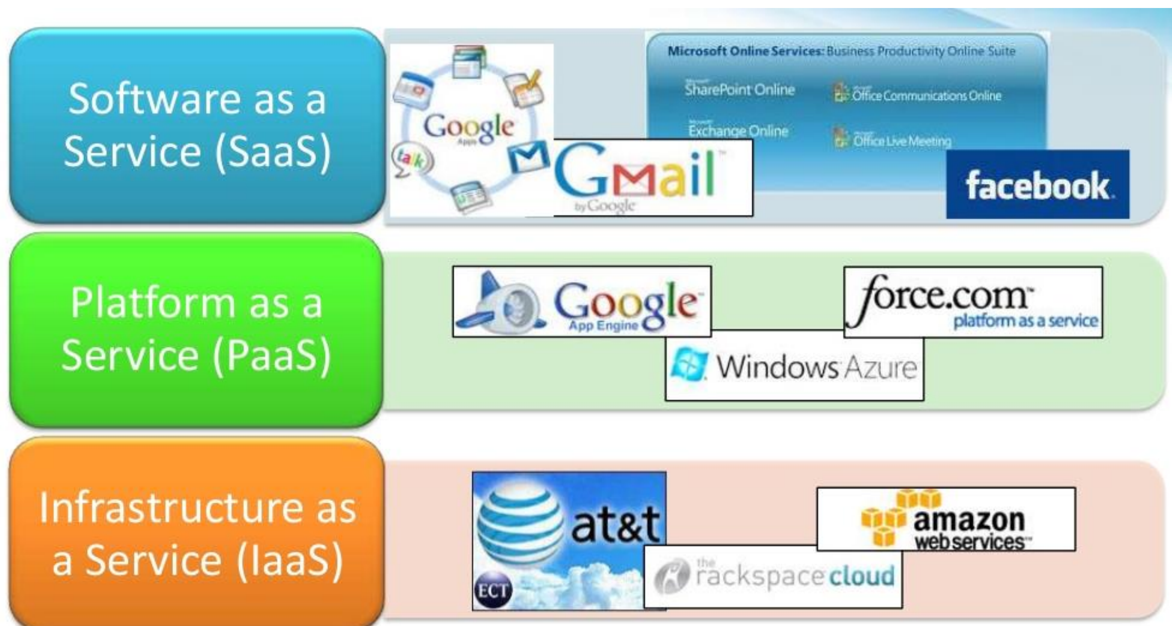
Δίνεται η δυνατότητα στον πελάτη να χρησιμοποιήσει τις εφαρμογές του παρόχου χρησιμοποιώντας τους πόρους του υπολογιστικού νέφους. Η πρόσβαση σε αυτές τις εφαρμογές γίνεται μέσω του Διαδικτύου και τρέχουν πάνω σε υποδομές της εταιρίας παροχής. Οι χρήστες έχουν είτε ελάχιστη πρόσβαση, κυρίως για ρυθμίσεις στις εφαρμογές, είτε δεν έχουν καθόλου.

*Platform as a Service (PaaS)*

Δίνεται η δυνατότητα στον πελάτη να αναπτύξει τις δικές του εφαρμογές ή να χρησιμοποιήσει άλλων πάνω στο υπολογιστικό νέφος που του παρέχεται. Και σε αυτή, την περίπτωση ο καταναλωτής δεν χειρίζεται ούτε ελέγχει τις υποδομές του cloud, αλλά διαθέτει κάποιο έλεγχο στο ποια προγράμματα μπορεί να εκτελέσει καθώς και στο περιβάλλον στο οποίο τα εκτελεί.

*Infrastructure as a Service (IaaS)*

Ο καταναλωτής έχει την ικανότητα διαχείρισης και επίβλεψης βασικών υπολογιστικών πόρων του cloud, όπως επεξεργαστική ισχύ, αποθηκευτικός χώρος και δικτύωση. Πάνω σε αυτές τις υποδομές μπορεί να τρέχει τις δικές του εφαρμογές ή αυτές άλλων. Ο πελάτης όμως δεν διαθέτει υλική πρόσβαση στους πόρους καθώς συνδέεται σε αυτούς μέσω κάποιου εικονικού περιβάλλοντος μέσω του διαδικτύου.




*Εικόνα 5 Παραδείγματα Μοντέλων Υπηρεσιών Νέφους*

Η εξάπλωση αυτή των Big data και του Cloud Computing έχει οδηγήσει σε διάδοση δυο θεμελιωδών τεχνολογιών για την αποθήκευση και την επεξεργασία των δεδομένων, το Apache Hadoop, μία από τις πιο σημαντικές πλατφόρμες λογισμικού που υποστηρίζει κατανεμημένες εφαρμογές που ασχολούνται με τεράστιο όγκο δεδομένων, και τις NoSQL βάσεις δεδομένων.

## 2.3 Βάσεις Δεδομένων

Οι βάσεις δεδομένων και τα συστήματα βάσεων δεδομένων αποτελούν ένα σημαντικό στοιχείο της καθημερινής ζωής στη σύγχρονη κοινωνία. Είναι πολύ απλό να σκεφτεί κάποιος μία καθημερινή δραστηριότητα του, που να σχετίζεται με κάποια αλληλεπίδραση σε κάποια βάση δεδομένων. Τέτοια παραδείγματα είναι, η κατάθεση ή ανάληψη χρημάτων από την τράπεζα, μία κράτηση σε ένα ξενοδοχείο καθώς και η αγορά κάποιου προϊόντος μέσω του διαδικτύου.

Ο ρόλος των βάσεων δεδομένων είναι κρίσιμος στη σημερινή εποχή, μια εποχή που χαρακτηρίζεται από τη χρήση των υπολογιστών και του διαδικτύου. Οπότε, είναι αξιόλογο να δοθεί ένας αυστηρότερος ορισμός για το τι είναι μία βάση δεδομένων [21]. Βάση δεδομένων (Database) είναι μία συλλογή από σχετιζόμενα δεδομένα. Με τον όρο δεδομένα εννοούμε γνωστά γεγονότα που μπορούν τα καταγραφούν και να έχουν κάποια υπονοούμενη σημασία.

Ωστόσο ο ορισμός αυτός κρύβει κάποιες σημαντικές ιδιότητες που χαρακτηρίζουν τις βάσεις δεδομένων

- Μια βάση δεδομένων αναπαριστά κάποια άποψη του πραγματικού κόσμου, που λέγεται και μικρόκοσμος (mini world) ή πεδίο Αναφοράς (Universe of Discourse, UoD). Οι αλλαγές σε αυτό τον μικρόκοσμο αντανακλώνται στη βάση δεδομένων.
- Μια βάση δεδομένων είναι μία λογικά συνεκτική συλλογή δεδομένων που έχει κάποια εγγενή σημασία. Μια τυχαία διευθέτηση δεδομένων δεν είναι σωστό να αναφέρεται ως βάση δεδομένων.
- Μια βάση δεδομένων σχεδιάζεται, χτίζεται και γεμίζει με δεδομένα για κάποιο σκοπό. Προορίζεται για μια συγκεκριμένη ομάδα χρηστών και για κάποιες προκαθορισμένες εφαρμογές για τις οποίες οι χρήστες ενδιαφέρονται.

Κάτι πολύ σημαντικό για τις βάσεις δεδομένων είναι ο τρόπος με τον οποίο μπορεί να δημιουργηθούν και να συντηρηθούν. Για παράδειγμα, ο κατάλογος καρτών που έχει μια βιβλιοθήκη είναι το μέσο δημιουργίας και συντήρησης μίας χειρόγραφης βάσης δεδομένων. Φυσικά, εμάς μας ενδιαφέρει όταν το μέσο αυτό είναι ένα σύστημα διαχείρισης βάσεων δεδομένων (database management system - DBMS)[21]. Ένα τέτοιο σύστημα είναι μία συλλογή από προγράμματα που επιτρέπουν στους χρήστες να δημιουργούν και να συντηρούν μία βάση δεδομένων. Το DBMS είναι ένα γενικής χρήσης σύστημα λογισμικού που διευκολύνει τις διαδικασίες ορισμού, κατασκευής, χειρισμού και διαμοιρασμού βάσεων δεδομένων για εφαρμογές. Άλλες σημαντικές λειτουργίες που μπορεί να παρέχει είναι η προστασία της βάσης καθώς και η μακροχρόνια συντήρηση της.

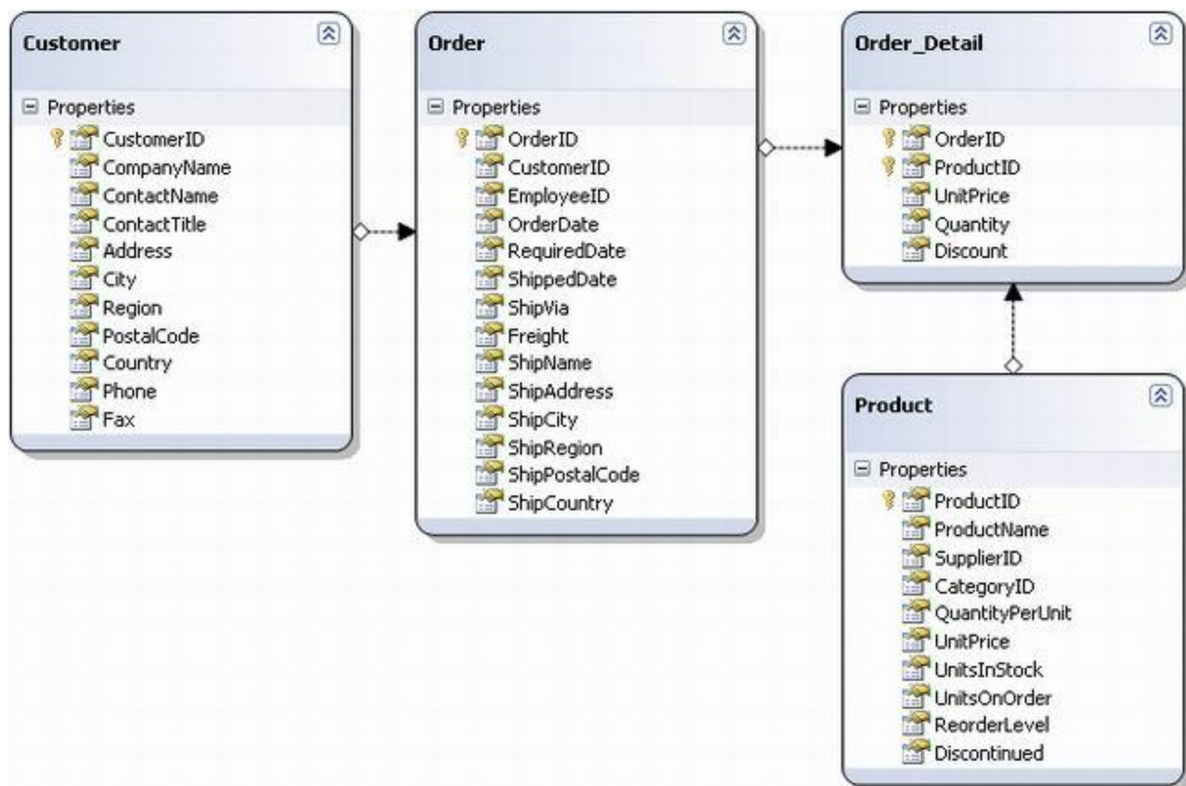
### 2.3.1 Σχεσιακές (SQL) Βάσεις δεδομένων

Ένα από τα πιο γνωστά μοντέλα βάσεων δεδομένων είναι το Σχεσιακό Μοντέλο. Το πρότυπο αυτό πρωτοπαρουσιάστηκε από τον Ted Codd της IBM Research το 1970[4] και λόγω της απλότητας και της μαθηματικής θεμελίωσης του, έγινε πολύ δημοφιλές και για αρκετά χρόνια κυρίαρχο στον τομέα αυτό. Οπότε, είναι σημαντικό να αναφέρουμε τα βασικά στοιχεία του Σχεσιακού Μοντέλου. Το μοντέλο αυτό παριστάνει την βάση δεδομένων ως μια συλλογή από σχέσεις. Χρησιμοποιείται η έννοια της μαθηματικής σχέσης σαν δομικό στοιχείο και η θεωρητική βάση περιέχει στοιχεία από τη θεωρία των συνόλων και τον κατηγορηματικό λογισμό πρώτης τάξης. Όπως αναφέρθηκε η βασική έννοια αυτού του πρότυπου είναι η *Σχέση*, για την οποία μπορούμε να πούμε ότι μοιάζει με ένα πίνακα τιμών

που κάθε γραμμή του πίνακα παριστάνει μια συλλογή από δεδομένα που σχετίζονται. Η ικανότητα ανάκτησης δεδομένων που έχουν μεταξύ τους κάποια συσχέτιση είναι βασική αρχή στις σχεσιακές βάσεις δεδομένων. Ένα απλό παράδειγμα είναι μία σχέση/πίνακας με όνομα ΦΟΙΤΗΤΗΣ και με στήλες που έχουν ονομασίες τύπου: Όνομα, Αριθμός Μητρώου, Έτος κ.λπ.

Αυτές οι βάσεις έχουν κάποιους πολύ σημαντικούς Κανόνες Ακεραιότητας (Integrity Rules) που εξασφαλίζουν την ακρίβεια και την προσβασιμότητα των δεδομένων τους [19].

- Οι σειρές ενός πίνακα πρέπει να είναι διακριτές, δηλαδή να μην υπάρχουν δύο ή παραπάνω που να είναι ίδιες, καθώς μπορεί να δημιουργηθούν προβλήματα για το ποια επιλογή είναι σωστή αφού φαινομενικά είναι όμοιες.
- Οι τιμές των στηλών δεν θα πρέπει να είναι επαναλαμβανόμενες ομάδες ή πίνακες.
- Εισάγεται η έννοια της τιμής null για την περίπτωση που μια τιμή λείπει από το συγκεκριμένο πεδίο. Η τιμή null δεν είναι ίδια με την «κενή» τιμή ή τη μηδενική τιμή.




*Εικόνα 6 Παράδειγμα Σχεσιακής Βάσης Δεδομένων*

Οι σχεσιακές βάσεις δεδομένων συνοδεύονται και από ένα σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων (relational database management system - RDBMS) που είναι υπεύθυνο, όπως έχει ήδη ειπωθεί, για τον τρόπο με τον οποίο τα δεδομένα αποθηκεύονται, συντηρούνται και ανακτώνται. Δημοφιλή RDBMS περιλαμβάνουν το DB2 και το Informix Dynamic Server (της IBM), την Oracle Rdb (της Oracle), τον SQL Server και την Access (της Microsoft) και συστήματα ανοικτού κώδικα όπως η MySQL και το PostgreSQL.

Ένας από τού κύριους λόγους που οι σχεσιακές βάσεις δεδομένων έγιναν πολύ δημοφιλείς ήταν η γλώσσα SQL (Structured Query Language). Η γλώσσα αυτή περιλαμβάνει χαρακτηριστικά σχεσιακής άλγεβρας καθώς και σχεσιακό λογισμό πλειάδων, και έχει σύνταξη φιλική και σχετικά φυσιολογική προς το χρήστη. Το γεγονός ότι η SQL αποτελεί πια πρότυπο γλώσσας δίνει στο χρήστη μια ευελιξία στην επιλογή του RDBMS καθώς ακολουθούνται οι ίδιοι κανόνες. Η SQL είναι μία πλήρης γλώσσα σχεσιακών βάσεων δεδομένων και για το λόγο αυτό ταυτίζεται και με τη φιλοσοφία του Σχεσιακού Μοντέλου πράγμα που οδήγησε στην ονομασία αυτών σε «SQL Βάσεις Δεδομένων». Πιο τεχνικά, η SQL διαθέτει εντολές για ορισμό δεδομένων, ερωτήσεις και ενημερώσεις. Δηλαδή είναι ταυτόχρονα Γλώσσα Ορισμού Δεδομένων (Data Definition Language - DDL) και Γλώσσα Χειρισμού δεδομένων (Data Manipulation Language - DML). Επιπλέον, περιλαμβάνει λειτουργίες για τον προσδιορισμό ασφάλειας και δικαιοδοσιών καθώς και για τον έλεγχο των δοσοληψιών.

## 2.3.2 NoSQL Βάσεις δεδομένων

Τα τελευταία χρόνια έχει παρατηρηθεί η άνοδος της δημοτικότητας κάποιων βάσεων δεδομένων που δεν χρησιμοποιούν το Σχεσιακό μοντέλο, όπως περιγράψαμε στην ενότητα 2.3.1 Σχεσιακές (SQL) Βάσεις δεδομένων. Ο όρος NoSQL (Not only SQL), όπως τον χρησιμοποιούμε σήμερα, είναι σχετικά πρόσφατος. Συγκεκριμένα, το 2009 χρησιμοποιήθηκε για πρώτη φορά από υποστηρικτές μη σχεσιακών βάσεων δεδομένων σε μία συνάντηση που είχαν στο San Francisco για τη συζήτηση και την ένδειξη ενδιαφέροντος στην αποθήκευση πληροφοριών σε κατανεμημένα και δομημένα περιβάλλοντα[6]. Φυσικά, μη σχεσιακές βάσεις δεδομένων υπήρχαν και πιο πριν από το 2009 απλά οι λόγοι που οδήγησαν τη σημερινή άνοδο των βάσεων αυτών φαίνεται ξεκάθαρα από τη συνεδρίαση αυτή. Πιο συγκεκριμένα, αυτοί που παρευρέθηκαν έψαχναν εναλλακτικές λύσεις στους τομείς που μία σχεσιακή βάση δεδομένων δεν αποτελούσε τη βέλτιστη επιλογή, ή όπως αναφέρθηκε πιο «σουρεαλιστικά»

σε περιοδικό πληροφορικής, «Οι υποστηρικτές του κινήματος NoSQL ήρθαν για να βρουν ένα σχέδιο που θα εκθρονίσει την τυραννία των αργών και ακριβών σχεσιακών βάσεων δεδομένων» [7].

Η εξέλιξη της τεχνολογίας, η ύπαρξη δεδομένων τεράστιου όγκου (Big Data) και η ευρύτατη χρήση του Cloud Computing έχει κάνει την ιδέα που αναπτύχθηκε το 2009 στο San Francisco να είναι πια αναγκαία. Χρειαζόμαστε να έχουμε αποδοτική αποθήκευση και ταχύτατη επεξεργασία των δεδομένων αυτών οπότε οδηγούμαστε στο να τα διαθέτουμε σε κατανεμημένη μορφή. Γι' αυτό το λόγο οι NoSQL βάσεις δεδομένων βγήκαν στο προσκήνιο.

*Εικόνα 7 Λογότυπα από 16 πολύ γνωστές NoSQL Βάσεις Δεδομένων*

Υπάρχουν διαφορετικές κατηγορίες μη σχεσιακών βάσεων δεδομένων που έχουν δημιουργηθεί για να ικανοποιούν συγκεκριμένες ανάγκες και δίνουν λύσεις σε διαφορετικά σενάρια χρήσης[2].

*Μοντέλο Κλειδιού-Τιμής (Key-Value Store)*

Δίνεται έμφαση στην απλότητα της μοντελοποίησης, στη μεγάλη ταχύτητα διαβάσματος και εγγραφής σε δεδομένα που δεν εμπλέκονται σε δοσοληψίες. Οι τιμές των δεδομένων μπορεί να είναι οποιοσδήποτε τύπος ψηφιακής πληροφορίας (κείμενο, αρχεία, κινούμενη εικόνα κ.λπ.) και η πρόσβαση σε αυτά γίνεται με ένα κλειδί. Η εφαρμογή έχει τον έλεγχο για το τι αποθηκεύεται και πώς, πράγμα που κάνει αυτό το μοντέλο πολύ ευέλικτο. Τα δεδομένα είναι κατανεμημένα, χωρισμένα και έχουν αντίγραφα (replicated) σε ένα σύμπλεγμα (cluster) και έτσι η δυνατότητα κλιμάκωσης (scalability) και διαθεσιμότητας (availability) είναι μεγάλη. Για το λόγο αυτό το μοντέλο αυτό συνήθως δεν υποστηρίζει συναλλαγές (transactions) δεδομένων αλλά είναι κατάλληλο για εφαρμογές που έχουν υψηλό

scaling και χρειάζονται υψηλή ταχύτητα. Σημαντικές βάσεις δεδομένων αυτής της κατηγορίας είναι οι Redis, Riak KV και Aerospike.

*Μοντέλο Έγγραφο-κεντρικών Δεδομένων (Document Oriented)*

Τα δεδομένα αποθηκεύονται σε μορφές JSON, XML και BSON[17] αρχείων. Μοιάζουν με το προηγούμενο μοντέλο, αλλά σε αυτή την περίπτωση η πληροφορία είναι ημιδομημένη σε αρχεία και τα αρχεία αυτά περιέχουν εγγραφές που περιγράφουν τον τύπο των δεδομένων και την τιμή της εν λόγω εγγραφής. Τα αρχεία ομαδοποιούνται σε «συλλογές» και μία έγγραφο-κεντρική βάση δεδομένων παρέχει μηχανισμούς για την αναζήτηση εγγραφών με ιδιαίτερα χαρακτηριστικά. Το μοντέλο αυτό είναι ιδιαίτερα ευέλικτο καθώς κάθε εγγραφή μπορεί να είναι ένα διαφορετικό αντικείμενο και οι περισσότερες βάσεις που ακολουθούν αυτό το μοντέλο έχουν ισχυρές μηχανές ερωτημάτων και δυνατότητες ευρετηρίου (indexing) που οδηγούν σε γρήγορη και αποτελεσματική ανάκτηση δεδομένων. Σημαντικές έγγραφο-κεντρικές βάσεις είναι η MongoDB και η CouchDB.

*Μοντέλο αποθήκευσης κατά στήλες (Column Oriented)*

Το μοντέλο αυτό μοιάζει με το RDBMS αλλά τα ονόματα και η μορφή των στηλών μπορεί να ποικίλει από γραμμή σε γραμμή. Αυτές οι βάσεις ομαδοποιούν τα δεδομένα των στηλών μαζί . Ένα ερώτημα μπορεί να ανακτήσει τα ζητούμενα δεδομένα με ένα operation γιατί μόνο οι στήλες που αφορούν το query ανακτώνται και αγνοούνται άχρηστα δεδομένα. Σε μία σχεσιακή βάση τα δεδομένα θα ήταν σε διαφορετικές γραμμές και κατά συνέπεια σε διαφορετικά σημεία του δίσκου οπότε θα χρειαζόντουσαν περισσότερα operations για την ανάκληση αλλά και περισσότερο χρόνο αναζήτησης (seek time) στο δίσκο. Επίσης, επειδή τα δεδομένα των στηλών μοιάζουν μεταξύ τους μπορεί να επιτευχθεί καλύτερη συμπίεση αυτών. Αν όμως τα ερωτήματα είναι τέτοιας μορφής έτσι ώστε να χρειάζονται δεδομένα από διαφορετικές στήλες τότε η απόδοση πέφτει. Οι πιο γνωστές βάσεις αυτής της κατηγορίας είναι η Cassandra, η HBase και η BigTable.

*Μοντέλο Γράφου (Graph)*

Αυτές οι βάσεις δεδομένων χρησιμοποιούν τα δομικά στοιχεία των γράφων για να αποθηκεύουν, χαρτογραφούν και να περιγράφουν σχέσεις στα δεδομένα. Όπως είναι αναμενόμενο το μοντέλο αυτό χρησιμοποιείται όταν υπάρχουν έντονες και πολυεπίπεδες εξαρτήσεις μεταξύ των δεδομένων, όπως για παράδειγμα στα κοινωνικά δίκτυα. Η προσπέλαση δεδομένων που σχετίζονται μεταξύ τους είναι πολύ αποδοτική. Βέβαια, άμεση

πρόσβαση στους κόμβους του γράφου με βάση κάποιο χαρακτηριστικό τους δεν υπάρχει. Η πιο δημοφιλείς βάσεις δεδομένων που ακολουθούν αυτό το μοντέλο είναι οι Neo4j και Titan.

*Συνδυασμός μοντέλων (Multi-Modal)*

Κάποιες NoSQL βάσεις δεδομένων συνδυάζουν τα πρότυπα που αναφέρθηκαν με αποτέλεσμα να έχουν μεγαλύτερη γκάμα εφαρμογών για τις οποίες προσφέρουν αποδοτική λύση. Γνωστό παραδείγματα είναι η OrientDB (graph, key-value store, document ).

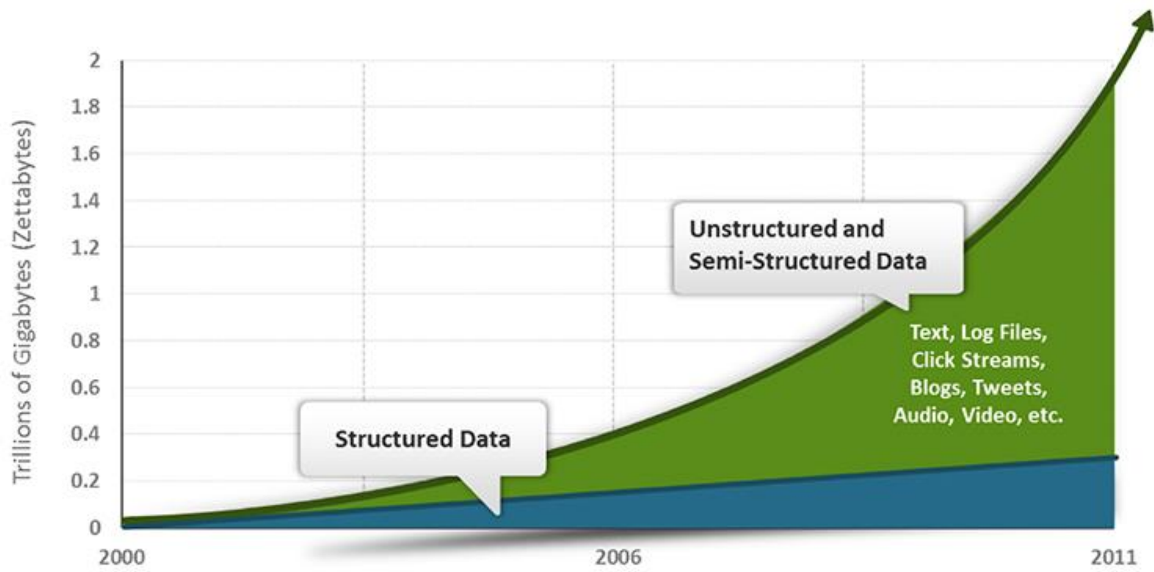
### 2.3.3 Σύγκριση NoSQL – SQL βάσεων

Έχοντας αναλύσει τις δύο διαφορετικές φιλοσοφίες που αφορούν τις βάσεις δεδομένων, σχεσιακές και μη σχεσιακές, εύκολα οδηγείται κανείς στη σύγκριση αυτών. Βέβαια, όταν μιλάμε για σύγκριση δεν θα μπούμε στην διαδικασία να χαρακτηριστεί κάποιος νικητής, διότι αυτή η προσέγγιση είναι λανθασμένη. Το κάθε μοντέλο είναι αποτελεσματικό σε διαφορετικά σενάρια χρήσης και προσφέρει εργαλεία και ιδιότητες που είναι χρήσιμα σε διαφορετικούς τομείς. Είναι στα χέρια του προγραμματιστή να κάνει τη χρήση τους αποτελεσματική ανάλογα με το πρόβλημα που πρέπει να λύσει. Αυτό δεν εμποδίζει όμως να δούμε τι προσφέρει και που διαφέρει το κάθε είδος.

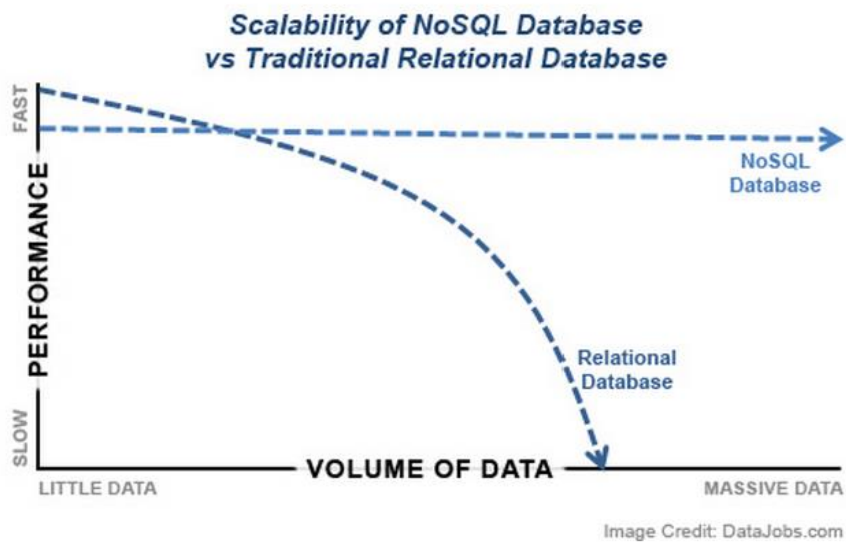
Η κύρια διαφορά είναι στο θέμα οργάνωσης. Οι SQL βάσεις δεδομένων είναι αυστηρά δομημένες όπως για παράδειγμα ένας τηλεφωνικός κατάλογος που έχει τηλεφωνικούς αριθμούς, ονόματα, διευθύνσεις κ.λπ. Αντίθετα, οι NoSQL βάσεις δεδομένων δεν έχουν τόσο αυστηρή οργάνωση στα δεδομένα τους, μερικές φορές απουσιάζει σχεδόν ολοκληρωτικά οποιαδήποτε δόμηση. Οι βάσεις αυτές μπορεί να συγκριθούν με φακέλους που περιέχουν σχετιζόμενες πληροφορίες, με την ειδοποιό διαφορά ότι τα δεδομένα που περιέχει κάθε φάκελος μπορεί να είναι οποιουδήποτε τύπου. Επίσης, στις σχεσιακές βάσεις δεδομένων κυρίαρχη είναι η έννοια του *Σχήματος (Schema)* που αφορά τη γενική διάταξη της βάσης, δηλαδή πως είναι οργανωμένα τα δεδομένα και τις σχέσεις που έχουν μεταξύ τους. Το Σχήμα σε αυτές τις βάσεις πρέπει να είναι αυστηρά ορισμένο πριν προστεθούν καινούργια δεδομένα κάτι που δεν ισχύει στις μη σχεσιακές βάσεις. Μία ακόμη σημαντική διαφορά είναι η κλιμακωσιμότητα (scalability) που προσφέρουν, δηλαδή πόσο και πως μπορούν να αυξήσουν τη δυνατότητα τους να επεξεργάζονται μεγαλύτερο μέγεθος δεδομένων. Αυτό που ισχύει γενικά, είναι ότι οι SQL βάσεις επιδέχονται κυρίως κατακόρυφη κλιμάκωση (vertical scaling) ενώ οι NoSQL οριζόντια κλιμάκωση (horizontal scaling)[10]. Φυσικά, υπάρχουν και

εξαιρέσεις καθώς έχουν δημιουργηθεί σχεσιακές βάσεις που δέχονται τέτοιου είδους κλιμάκωση αλλά δεν ξεφεύγουν από το δομικά αυστηρό χαρακτήρα τους.

Αυτό που προσφέρει η αυστηρή δόμηση των σχεσιακών βάσεων είναι ικανοποίηση των κριτηρίων ACID (Atomicity, Consistency, Isolation, Durability)[15] που είναι σημαντική αν είναι απαραίτητη η ακεραιότητα της βάσης και η συνοχή των δεδομένων. Επίσης, η μοντελοποίηση που ακολουθούν πολλές φορές βοηθάει στην επίλυση αρκετών προβλημάτων που έχουν παρόμοια δομή και τα δεδομένα δεν υφίστανται πολλές αλλαγές μετά την αποθήκευση τους. Από την άλλη πλευρά, είναι πολύ συχνά απαραίτητη η αποθήκευση και επεξεργασία πολύ μεγάλου μεγέθους δεδομένων που συχνά, λόγω του όγκου τους και της ποικιλία τους (variety), δεν έχουν κάποια ιδιαίτερη δομή. Για την ακρίβεια τα δομημένα δεδομένα που παράγονται σήμερα είναι λιγότερα και έχουν σχεδόν γραμμική αύξηση σε αντίθεση με τα άλλα. Και σαν να μην έφτανε αυτό, πρέπει το σύστημα μας να έχει και καλή απόδοση στον εκθετικά αυξανόμενο όγκο πληροφοριών. Η ευελιξία και η ταχύτητα που προσφέρουν οι μη σχεσιακές βάσεις σε αυτές τις περιπτώσεις αποδεικνύεται ιδιαίτερα χρήσιμη. Όλα αυτά σε συνδυασμό με την τεχνολογία του Cloud Computing, έχουν προσφέρει αποδοτικές λύσεις για την αποθήκευση και την επεξεργασία τεράστιων κομματιών πληροφορίας. Χρησιμοποιώντας οριζόντια κλιμάκωση είναι εφικτός ο εύκολος διαμοιρασμός των πληροφοριών αυτών σε πολλούς εξυπηρετητές με την χρήση υπολογιστικών συστημάτων μεσαίας ισχύος (commodity hardware). Η κλιμάκωση αυτού του είδους, εκτός του ότι δεν περιορίζεται τόσο πολύ όσο το vertical scaling προσφέρει οικονομικότερες λύσεις, καθώς τα μηχανήματα μεσαίας ισχύος είναι φθηνότερα, προσφέρει υψηλότερη διαθεσιμότητα γιατί τα δεδομένα είναι διαμοιρασμένα σε περισσότερα σημεία πρόσβασης και είναι εφικτή η προσκόμιση της απαραίτητης επεξεργαστικής ισχύος.

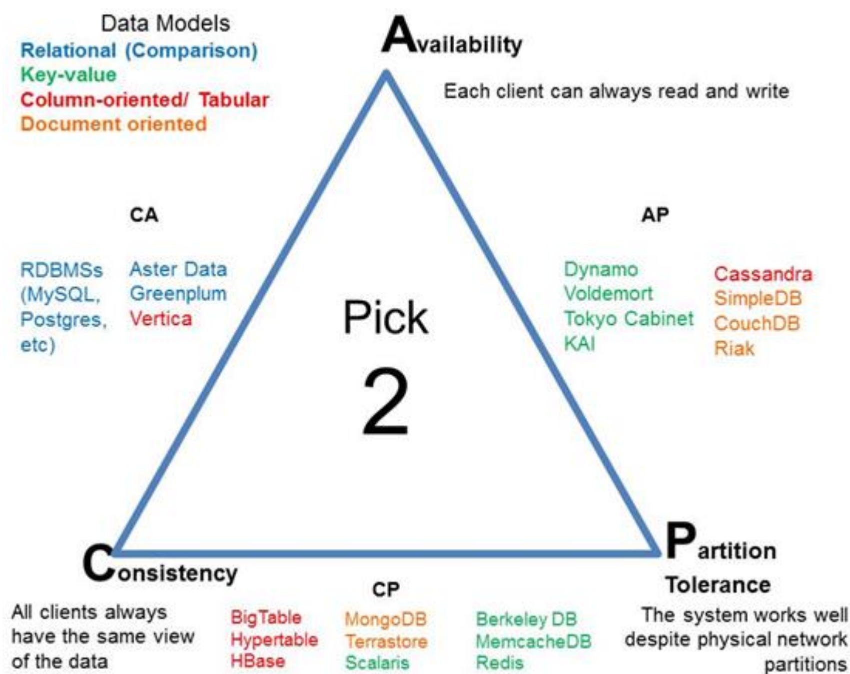



*Εικόνα 8 Παραγωγή Δεδομένων με την πάροδο των χρόνων*




*Εικόνα 9 Κλιμακωσιμότητα NoSQL vs RDBMS*

Είναι φανερό ότι σε κάθε περίπτωση πρέπει να θυσιαστεί κάποιο ή κάποια χαρακτηριστικά για να έχουμε το επιθυμητό αποτέλεσμα. Στην δική μας περίπτωση οι μη σχεσιακές βάσεις δεδομένων δεν ικανοποιούν πάντα τα κριτήρια ACID και προσφέρουν κατανεμημένους τρόπους αποθήκευσης, ευελιξία και ταχύτητα. Η διαπίστωση αυτή είναι πολύ σημαντική και μάλιστα έχει μελετηθεί και έχει άμεση σχέση με θεώρημα CAP (Consistency, Availability, Partition Tolerance)[13] που αναφέρεται στο γεγονός ότι πρέπει να επιλέξουμε δύο από τα τρία αυτά χαρακτηριστικά σε ένα σύστημα καθώς είναι αδύνατον να έχουμε και τα τρία.




*Εικόνα 10 Θεώρημα CAP*

Το ότι υπάρχουν πολλοί συνδυασμοί δείχνει ότι κάθε φορά πρέπει να γίνει σωστή ανάλυση του προβλήματος ώστε να επιλεχθούν οι κατάλληλες τεχνολογίες. Μία σημαντική σημείωση, είναι ότι το θεώρημα CAP έχει ανανεωθεί σε κάποια σημεία για να ανταποκρίνεται στα σημερινά δεδομένα και την αύξηση της χρήσης των κατανεμημένων συστημάτων[5].

Οι NoSQL βάσεις έχουν σχεδιαστεί «έχοντας στο μυαλό τους» τις ανάγκες των Big Data. Οι ανάγκες αυτές ήταν που δημιουργούσαν τα περισσότερα προβλήματα στις εταιρίες σε ότι αφορά την ανάλυση των δεδομένων τους. Οι χρήσιμες πληροφορίες βρίσκονται μέσα σε ένα πολύ μεγάλο όγκο δεδομένων που συνήθως δεν έχει κάποια χαρακτηριστική δομή και για να είναι βιώσιμη η ανάκτηση τους χρειαζόμαστε μεγάλη επεξεργαστική ισχύ και ταχύτητα σε ορισμένες εργασίες. Έτσι, για την αποτελεσματική και σωστή εξόρυξη και

ανάλυση των δεδομένων οι εταιρίες που προσφέρουν τέτοιες υπηρεσίες πολλές φορές επιλέγουν μη σχεσιακές βάσεις.

## 2.4 Εξωτερική Ανάθεση (Outsourcing)

Μέσα σε όλη αυτή την πληροφορία που διανέμεται, λαμβάνοντας υπόψη τον υψηλό ανταγωνισμό, οι εταιρίες χρειάζονται να επεξεργαστούν και να βγάλουν συμπεράσματα από αυτά τα δεδομένα. Αυτή η ανάγκη οδήγησε τους οργανισμούς να αναθέτουν τέτοιες εργασίες σε τρίτους. Το outsourcing τη σημερινή εποχή είναι πολύ διαδεδομένο και δεν εφαρμόζεται μονάχα από μεγάλες εταιρίες. Ιδιώτες συνεργάζονται με λογιστές για την κάλυψη των φορολογικών τους υποχρεώσεων και μικρομεσαίες επιχειρήσεις βασίζονται σε τρίτους για την παροχή εξοπλισμού και πρώτων υλών.

### 2.4.1 Χαρακτηριστικά Εξωτερικής Ανάθεσης

Για την καλύτερη κατανόηση της ευρείας εξάπλωσης αυτής της πρακτικής, θα αναφερθούν ορισμένα χαρακτηριστικά της που την κάνουν ιδιαίτερα δημοφιλή [23].

*Οικονομικοί Παράγοντες*

Μείωση των εξόδων για μια επιχείρηση καθώς δε χρειάζεται να κατανείμει ανθρώπινους πόρους (και κατά συνέπεια πληρωμή παραπάνω μισθών) για την υλοποίηση κάποιων εργασιών. Έτσι, μπορεί η επιχείρηση να διανείμει το ανθρώπινο δυναμικό και να επικεντρωθεί σε πιο βασικές εργασίες. Σημαντικός παράγοντας επίσης, είναι και η μείωση τον εξόδων που μπορεί να προέλθει από το outsourcing. Συχνά οι εξωτερικές επιχειρήσεις είναι αρκετά μεγαλύτερες από αυτές που τις προσλαμβάνουν και μπορεί να έχουν κατάλληλες εγκαταστάσεις και εργαλεία που να μη διαθέτει οποιοσδήποτε στην αγορά ή να είναι ακριβή η διαδικασία απόκτησης αυτών.

*Τεχνογνωσία*

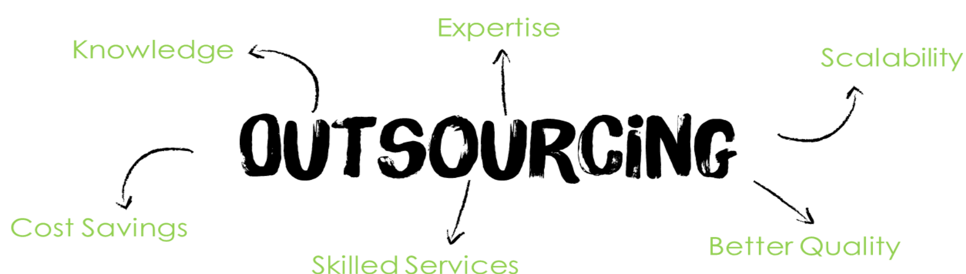
Πολλές από τις εργασίες χρειάζονται ειδική τεχνογνωσία (όπως για παράδειγμα η δημιουργία μιας δικτυακής εφαρμογής) που ο εκάστοτε οργανισμός δε διαθέτει καθώς είναι μακριά από το αντικείμενο του. Μπορεί να αποφευχθεί η απόκτηση της αυτής της τεχνογνωσίας, που θα να είναι και ασύμφορη, αν ανατεθεί η εργασία αυτή σε κάποιο τρίτο που διαθέτει την αρμόζουσα ειδίκευση και γνώση για να φέρει εις πέρας το έργο.

*Επίδοση*

Λόγω μεγέθους οι εταιρίες που παρέχουν υπηρεσίες μπορεί να έχουν περισσότερο ανθρώπινο δυναμικό, καλύτερες εγκαταστάσεις και εργαλεία με αποτέλεσμα να παρουσιάζουν καλύτερες επιδόσεις στην υλοποίηση εργασιών. Επίσης, οι οργανισμοί αυτοί προσπαθούν να πραγματοποιήσουν τη δουλεία που τους έχει ανατεθεί όσο το δυνατόν καλύτερα (συνέπεια στις προθεσμίες, ποιότητα) για να υπάρχει καλή σχέση πελάτη – παρόχου υπηρεσιών καθώς και για να διατηρηθεί η καλή φήμη της εταιρίας.

*Συμφωνίες*

Η συνεργασία μεταξύ των δύο οργανισμών που εμπλέκονται στο outsourcing προσφέρει και άλλα πλεονεκτήματα. Αρχικά, μεταφέρεται η ευθύνη του έργου σε κάποιον τρίτο, πράγμα που προσφέρει ευκολία στην επιχείρηση και μείωση της πίεσης καθώς δεν είναι υπεύθυνη για τη σωστή πραγμάτωση του. Επίσης, μοιράζεται το ρίσκο μεταξύ των δύο συνεργαζόμενων εταιριών και σε περίπτωση αποτυχίας η ζημία θα είναι μικρότερης έντασης στην κάθε μία.




*Εικόνα 11 Θετικά Στοιχεία Outsourcing*

Όπως είναι αναμενόμενο, το outsourcing περιέχει και κινδύνους καθώς η εταιρία βασίζεται σε κάποιον άλλο για την ικανοποίηση κάποιων λειτουργιών της. Τα ρίσκα αυτά κατηγοριοποιούνται ως εξής [23].

*Εμπιστοσύνη*

Πρέπει οι εξωτερικοί συνεργάτες να είναι έμπιστοι, τόσο στην ικανότητά τους να ολοκληρώσουν το έργο που τους ανατέθηκε, όσο και στο χειρισμό ευαίσθητων δεδομένων της εταιρίας που της παρέχουν υπηρεσίες.

*Ποιότητα*

Πολλές φορές είναι δύσκολο να βρεθεί οργανισμός με την κατάλληλη τεχνογνωσία και εμπειρία σε ένα συγκεκριμένο τομέα. Αυτό συνεπάγεται στο να μην μπορεί να ικανοποιηθεί μια αποδεκτή ποιότητα στην εργασία που έχει αναλάβει η αντίστοιχη εταιρία. Ένας άλλος παράγοντας που οδηγεί σε όχι τόσο ποιοτικό αποτέλεσμα είναι ότι μπορεί η αποτυχία εκπλήρωσης του έργου να είναι πιο κερδοφόρα για τον συνεργάτη που το έχει αναλάβει. Σε αυτή την περίπτωση εμπλέκονται και θέματα εμπιστοσύνης.

*Νομικά θέματα και Πνευματική Ιδιοκτησία*

Μία τέτοια συνεργασία θα εμπλέκει πολλές νομικές συμφωνίες για την προφύλαξη των συμφερόντων και των ευαίσθητων δεδομένων των εταιριών. Η διαδικασία της θεσμοθέτησης αυτών συχνά είναι πολύπλοκη και οικονομικά δύσκολη, ειδικά όταν κάθε επιχείρηση υπακούει σε διαφορετικό νομικό πλαίσιο. Ένα άλλο σημαντικό θέμα είναι το licensing, δηλαδή ποια εταιρία θα έχει τις πνευματικές άδειες για το έργο μετά την υλοποίηση του. Αυτό πρέπει να έχει συζητηθεί εκ των προτέρων, καθώς μπορεί η εξωτερική επιχείρηση να μεταπουλήσει το λογισμικό σε άλλους αν έχει τα πνευματικά δικαιώματα.

*Κρυφά Κόστη*

Εάν δε γίνει προσεκτική επιλογή μετά από έρευνα, είναι πιθανό κάποια από τα προτερήματα να στραφούν σε αρνητική κατεύθυνση (πχ τεχνογνωσία, εγκαταστάσεις). Τέτοια κόστη αφορούν και όλες τις κατηγορίες που αναφέρθηκαν ήδη. Αυτά δεν μπορούν να υπολογιστούν από πριν γιατί βασίζονται σε κάποια ενδεχόμενη μελλοντική αποτυχία ή ρήξη με τη συνεργαζόμενη εταιρία.

*Προστασία Δεδομένων*

Μια από τις σημαντικότερες επιπλοκές, είναι η ασφάλεια των δεδομένων που μεταβιβάζονται ανάμεσα στις επιχειρήσεις. Τα δεδομένα αυτά είναι ευκολότερο να υποκλαπούν καθώς δεν υφίστανται πια εντός μονάχα μιας εταιρίας. Υποκλοπή μπορεί να γίνει με διάφορους τρόπους. Ένας από αυτούς είναι η κακόβουλη χρήση των δεδομένων από την εξωτερική εταιρεία, δηλαδή να τα μεταχειριστεί προς όφελός της (εκβιασμός, πώληση των προσωπικών στοιχείων κ.λπ.). Ακόμα και αν ο εξωτερικός συνεργάτης δεν έχει σκοπό να χρησιμοποιήσει τα δεδομένα αυτά για κακόβουλο σκοπό, είναι δυνατόν κάποιος

εργαζόμενος εκ των έσω, από άγνοιά ή συνειδητά, να οδηγήσει σε διαρροή ευαίσθητων πληροφοριών. Υπάρχει επίσης πάντα το ενδεχόμενο εξωτερικής απειλής και έτσι είναι αναγκαίο ο εξωτερικός συνεργάτης να διαθέτει κατάλληλη προστασία προς τέτοιες επιθέσεις. Σύμφωνα με έρευνες[24], 63% των περιπτώσεων παραβίασης προσωπικών δεδομένων προέρχεται από outsourcing μεταξύ επιχειρήσεων.

### 2.4.2 Εξωτερική ανάθεση στην ανάλυση δεδομένων

Οι σημερινές εταιρίες έχουν αντιληφθεί πόσο πολύτιμη είναι η ανάλυση των δεδομένων για την απόκτηση χρήσιμων πληροφοριών στις επιχειρήσεις τους, τη βελτίωση στην κατανομή του κεφαλαίου τους αλλά και για να αυξήσουν τον αριθμό των πελατών τους καθώς και την ικανοποίηση τους. Παρόλα αυτά, οι περισσότερες από αυτές, σύμφωνα με μία μελέτη από το Forrester Research[8], διαχειρίζονται μόνο το 12% των δεδομένων που κατέχουν, γιατί είτε έχουν άγνοια για ένα μεγάλο ποσοστό είτε δεν γνωρίζουν πιο μέρος της πληροφορίας αυτής είναι χρήσιμο και πιο όχι.

Φυσικά για τους λόγους που αναφέρθηκαν στην προηγούμενη ενότητα πολλές εταιρίες επιλέγουν να αναθέσουν την εξόρυξη και ανάλυση δεδομένων σε κάποιο άλλο οργανισμό καθώς τα πλεονεκτήματα αυτής της κίνησης μπορεί να είναι πολύ κερδοφόρα. Οι ειδικοί σε αυτό τον τομέα είναι περιζήτητοι, σε σημείο που το Harvard Business Review ονόμασε αυτή την δουλειά «Την πιο ελκυστική του 21ου αιώνα»[9]. Και για το λόγο αυτό, σκοπός της παρούσας διπλωματικής εργασίας είναι να εξετάσουμε τις νέες τεχνολογίες που έχουν αναπτυχθεί τα τελευταία χρόνια για εξόρυξη και ανάλυση δεδομένων και να σχεδιαστεί ένα αποδοτικό, ευέλικτο σύστημα που θα μπορούσε να χρησιμοποιηθεί από τις εταιρίες που χρειάζονται αυτού του είδους τις υπηρεσίες.

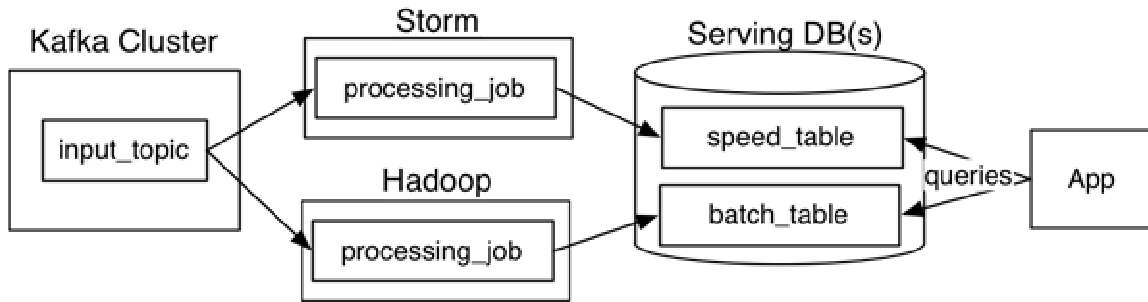
# 3. Ανάλυση Προβλήματος

Ο ρυθμός αύξησης των δεδομένων είναι πολύ μεγάλος και επρόκειτο να αυξηθεί. Πέρα όμως από την ανάπτυξη τους σε όγκο τα Big Data εξελίσσονται και στον τρόπο με τον οποίο επεξεργάζονται. Για το λόγο αυτό, τα πληροφοριακά συστήματα δεδομένων που αποθηκεύουν και επεξεργάζονται την πληροφορία σε μεγάλες παρτίδες, το γνωστό batch processing, δε θεωρούνται πια αρκετά. Έτσι, γίνεται λόγος και για ανάλυση σε πραγματικό χρόνο, δηλαδή να γίνονται ενέργειες για την άμεση εξόρυξη της πληροφορίας. Για την ανάλυση αυτών των «ροών πληροφορίας» χρησιμοποιείται ευρύτατα ο όρος stream processing. Φυσικά, το batch processing, που χαρακτηρίζει τις περισσότερες Big Data αρχιτεκτονικές, παρόλο που έχει ξεπεραστεί κατά μία έννοια, δε θεωρείται σε καμία περίπτωση άχρηστο διότι εφαρμόζεται εκτενέστατα από πολλά συστήματα. Οπότε, πρέπει να δημιουργηθεί μία αποδοτική αρχιτεκτονική που θα χρησιμοποιεί και τις δύο μεθόδους επεξεργασίας, batch και streaming.

Ας σκεφτούμε τώρα μία εταιρία σε ένα πολύ ανταγωνιστικό περιβάλλον η οποία θέλει να δημιουργήσει ένα σύστημα ανάλυσης της πληροφορίας που παράγει αυτή και η αγορά. Αυτό που χρειάζεται είναι μία πλατφόρμα που θα λαμβάνει όλα αυτά τα δεδομένα και θα τα χωρίζει. Κάποια από αυτά θα προορίζονται για αποθήκευση και μετά από εκεί αφού μαζευτούν αρκετά θα γίνεται η επεξεργασία τους σε μεγάλες παρτίδες και κάποια άλλα θα αναλύονται αμέσως ή με πολύ μικρή καθυστέρηση. Έτσι, θα κατέχει ανταγωνιστικό πλεονέκτημα καθώς όταν κάποια δεδομένα φαίνονται χρήσιμα η έρχονται από κάποια σημαντική πηγή μπορούν να αναλυθούν αμέσως. Αυτό καλούμαστε να δημιουργήσουμε σε αυτή τη διπλωματική εργασία, ένα τέτοιο σύστημα.

## 3.1 Lambda Architecture

Για το «πάντρεμα» του batch με το stream processing έγινε λόγος από τον Nathan Marz το 2011 με την δημιουργία του Lambda Architecture[14]. Αυτή η αρχιτεκτονική αποτελεί ουσιαστικά ένα υβριδικό μοντέλο που αποτελείται από ένα κομμάτι που κάνει αναλύσεις τύπου batch (batch layer) πάνω σε όλα τα διαθέσιμα δεδομένα, ένα υποσύστημα επεξεργασίας δεδομένων που μόλις έχουν φτάσει μέσα στο σύστημα (speed layer) και ένα τελευταίο κομμάτι, το service layer, που ουσιαστικά ενώνει τα δύο προηγούμενα προσφέροντας δυνατότητες αναζήτησης και συνδυασμού των αποτελεσμάτων των batch και speed layers.




*Εικόνα 12 Παράδειγμα συστήματος που χρησιμοποιεί Lambda Architecture*

Βέβαια, αυτή η αρχιτεκτονική διαθέτει και κάποιες δυσκολίες[12] στην υλοποίηση της που προσπαθούμε να αποφύγουμε στη δική μας ιδέα. Αρχικά, όταν δεν υπάρχει ένα κοινό εργαλείο για την πραγματοποίηση εφαρμογών τύπου batch και stream, η ιδέα πρέπει να υλοποιηθεί δύο φορές, μία με τις τεχνολογίες του batch layer και μία με αυτές του speed layer. Επίσης, το serving layer πρέπει να ρυθμιστεί σωστά ώστε να δέχεται και να επεξεργάζεται σωστά τύπους και δομές δεδομένων από διαφορετικές πηγές. Όλα αυτά προκαλούν μεγάλο προγραμματιστικό φόρτο και σε ότι αφορά την δημιουργία αλλά και τη συντήρηση. Η χρήση ενός μόνο εργαλείου για τις εργασίες stream καθώς και batch αλλά και η ύπαρξη μίας μόνο δομής με μεγάλες επεξεργαστικές δυνατότητες θα έλυνε αυτό το πρόβλημα.

## 3.2 Απαιτήσεις προβλήματος

Σημαντικό είναι να εξετάσουμε τις βασικές απαιτήσεις ενός τέτοιου συστήματος πριν ξεκινήσει η διαδικασία δημιουργίας του τελικού «προϊόντος».

Οποιαδήποτε αρχιτεκτονική συστήματος που αφορά τα Big Data χρειάζεται να έχει κάποια βασικά στοιχεία. Αρχικά, απαραίτητη είναι η ύπαρξη κάποιου κλιμακώσιμου χώρου αποθήκευσης που θα έχει τα δεδομένα ανά πάσα στιγμή διαθέσιμα προς κατανάλωση και επεξεργασία. Τέτοιες τεχνολογίες είναι για παράδειγμα κάποιο κατανεμημένο σύστημα αρχείων ή βάση δεδομένων. Η ύπαρξη κάποιας κατανεμημένης πλατφόρμας ανάλυσης, αναζήτησης και επεξεργασίας δεδομένων μεγάλης κλίμακας θεωρείται και αυτή απαραίτητη. Και τέλος δεν θα πρέπει να ξεχαστούν οποιαδήποτε εργαλεία χρειάζονται για τη διαχείριση πόρων και υπηρεσιών που χρησιμοποιούνται από τέτοια συστήματα.

Πέρα από τα γενικά στοιχεία που χρειάζονται να έχουν αυτά τα συστήματα καλό θα ήταν να αναλογιστεί κανείς και το περιβάλλον στο οποίο θα πρέπει να δουλεύουν. Το

Διαδίκτυο και οι επιχειρήσεις αποτελούν ένα εργοστάσιο συνεχούς παραγωγής πληροφορίας με μεγάλο ρυθμό και γι‘ αυτό το σύστημα μας θα περιμένουμε να απορροφά αυτή την πληροφορία γρήγορα, αποδοτικά και αξιόπιστα. Γιατί, πέρα από την ταχύτητα και το μέγεθος των δεδομένων πρέπει δοθεί σημασία και η αξία που περιέχεται σε αυτά. Απώλειες δεδομένων τη σημερινή εποχή μπορεί να προκαλέσουν μεγάλη ζημία σε εταιρίες και οργανισμούς, ίσως ακόμη να οδηγήσουν και στο κλείσιμο τους. Με άλλα λόγια, πρέπει η συμπεριφορά της αρχιτεκτονικής μας να λαμβάνει υπόψη τη μεγάλη διακίνηση πληροφορίας (throughput) και τις δυσκολίες που παρουσιάζονται λόγω αυτής. Ταυτόχρονα, θα πρέπει να υπάρχει και έλεγχος στην ροή των δεδομένων γιατί κάποια θα προορίζονται για αποθήκευση και κάποια άλλα για stream processing. Για να παραμείνει το επίπεδο του throughput σε επιθυμητά επίπεδα θα πρέπει τόσο η κατηγοριοποίηση των στοιχείων όσο και η αποθήκευση να γίνεται ιδιαίτερα γρήγορα αλλιώς θα υπάρχουν καθυστερήσεις σε ολόκληρη την αρχιτεκτονική, κάτι που όπως τονίσαμε επιδιώκουμε να αποφευχθεί.

## 3.3 Σημαντικότητα και Περιπλοκότητα

Για να καταλάβουμε πόσο σημαντικό είναι αυτό το πρόβλημα θα πρέπει να σκεφτούμε τη σημασία της ανάλυσης των δεδομένων σε πραγματικό χρόνο. Ας φανταστούμε ένα σενάριο όπου η Εταιρία Α ανανεώνει τα ευρετήρια αναζήτησης της κάθε μία ώρα καθώς περιμένει να μαζευτούν οι αντίστοιχες πληροφορίες, με άλλα λόγια κάνει batch processing. Τώρα, για παράδειγμα, συμβαίνει ένα γεγονός μεγάλης σημασίας και κάποιος ψάχνει στη μηχανή αναζήτησης της Εταιρίας Α γι’ αυτό το συμβάν θέλοντας να μάθει τα τελευταία νέα. Φυσικά όμως, δεν θα βρει τίποτα καθώς θα πάρει περίπου μία ώρα για να ανανεωθούν τα ευρετήρια και να έρθουν στις πιο δημοφιλής αναζητήσεις πληροφορίες γι’ αυτό το γεγονός. Αντίθετα, η Εταιρία Β αναλύει σωστά τις αναζητήσεις και δίνει τα αποτελέσματα που ζητούνται νωρίτερα. Η Εταιρία Α βρίσκεται σε μειονεκτική θέση. Αυτό το παράδειγμα, που δεν απέχει πολύ από την πραγματικότητα, δείχνει το ανταγωνιστικό πλεονέκτημα που μπορεί να έχει κάποιος από την άμεση εξαγωγή της πληροφορίας. Σκεπτόμενοι αυτό και συνδυάζοντας την ανάλυση δεδομένων σε real time με προβλήματα πού είναι περισσότερο βασισμένα στο να βρίσκεται λύση άμεσα, όπως η ανίχνευση ύποπτων οικονομικών κινήσεων την ώρα που συμβαίνουν, μπορεί εύκολα να καταλάβει κανείς γιατί το streaming analysis είναι ένα θέμα που συζητιέται τόσο πολύ.

Βέβαια, οι ανάγκες του stream processing είναι διαφορετικές από τα batch συστήματα. Δεν μπορούμε απλά να κάνουμε τα δεύτερα να τρέχουν πιο γρήγορα ή πιο συχνά. Απαιτούνται αρκετές σχεδιαστικές αλλαγές. Σε μία άπειρη ροή δεδομένων πρέπει να γνωρίζουμε ότι έχουμε όλα τα δεδομένα για ένα χρονικό διάστημα πράγμα που δεν είναι εύκολο. Επίσης, επειδή μιλάμε για εργασίες σε πραγματικό χρόνο τι θα γίνει αν τα δεδομένα μας έρθουν καθυστερημένα, είτε λόγω εξωτερικών παραγόντων είτε λόγω περιορισμένου throughput; Πέρα όμως από τις προκλήσεις μιας streaming αρχιτεκτονικής, υπάρχει δυσκολία και στον συνδυασμό ή την επέκταση της ώστε να πραγματοποιούνται και διεργασίες τύπου batch, πράγμα αποτελεί τον κύριο στόχο αυτής της εργασίας. Εδώ μιλάμε για διαφορετικά μεγέθη και χρονικά περιθώρια. Η συνολική αρχιτεκτονική πρέπει να μπορεί να αποθηκεύει σύνολα δεδομένων της τάξεως των Petabytes (PB), αλλά μία streaming εργασία θα δουλεύει σε μεγέθη που φτάνουν Megabytes (MB) έως Terabytes (TB) κάθε χρονική στιγμή. Ένα TB πληροφορίας ανά λεπτό είναι τεράστιος όγκος πληροφορίας και θα πρέπει επεξεργάζεται σε πολύ λίγο χρόνο!

| Metric | Sizes and units: Batch | Sizes and units: Streaming |
|---|---|---|
| Data sizes per job | TB to PB | MB to TB (in flight) |
| Time between data arrival and processing | Many minutes to hours | Microseconds to minutes |
| Job execution times | Minutes to hours | Microseconds to minutes |

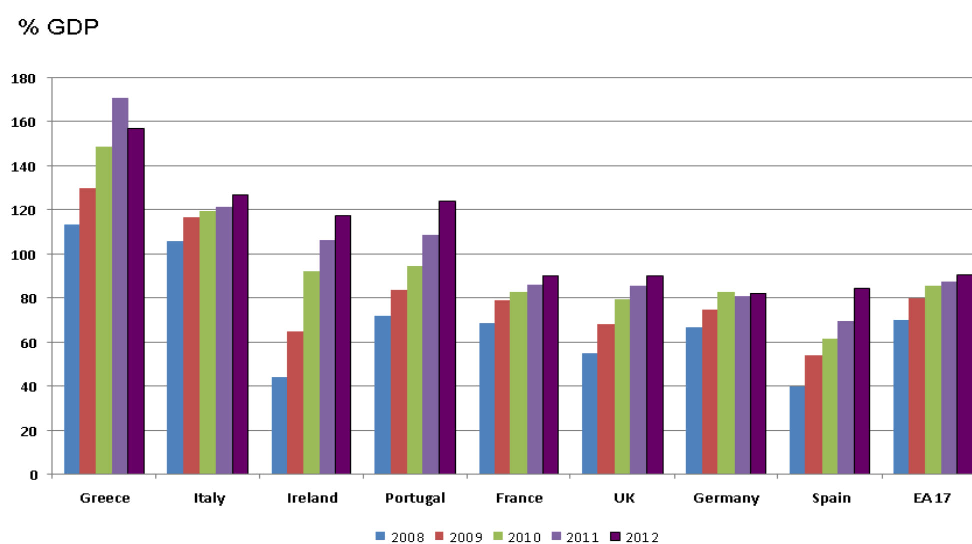
*Εικόνα 13 Διαφορές Μεγεθών σε Batch και Stream Processing*

## 3.4 Περιγραφή του προβλήματος προς αντιμετώπιση

Έχοντας αναφερθεί στο γενικό υπόβαθρο και στα χαρακτηριστικά του προβλήματος θα του προσδώσουμε μία πιο ειδική μορφή. Θα λάβουμε υπόψη την περίπτωση μίας εταιρίας που διαθέτει χαρτοφυλάκιο με πληροφορίες και χαρακτηριστικά άλλων επιχειρήσεων. Η εταιρία αυτή καλείται να λάβει κρίσιμες αποφάσεις με αυτές τις πληροφορίες και να βγάλει συμπεράσματα ώστε να προσφέρει ποιοτικές υπηρεσίες στους πελάτες που την συμβουλεύονται. Η εξαγωγή χρήσιμης πληροφορίας και η σωστή ανάλυση της θα οδηγήσει σε αξιόπιστα αποτελέσματα δίνοντας ανταγωνιστικό πλεονέκτημα και οδηγώντας στη λήψη σωστών αποφάσεων είτε σε θέματα επέκτασης είτε σε περίπτωση αδιεξόδου. Τα δεδομένα

διαθέτουν χρησιμότητα και τη στιγμή που δημιουργούνται αλλά και αργότερα, γι’ αυτό είναι απαραίτητη η επεξεργασία σε πραγματικό χρόνο καθώς και σε παρτίδες.

Αυτό το συγκεκριμένο πρόβλημα επιλέχθηκε καθώς αντιμετωπίζεται πολύ συχνά στη σημερινή εποχή. Οι τράπεζες χρειάζονται να αναλύουν συνεχώς τέτοια δεδομένα ώστε να πάρουν αποφάσεις που αφορούν την διαχείριση των «κόκκινων δανείων» και τελικά την αποφυγή καταστάσεων και ενεργειών που οδηγούν σε αυτά. Η οποιαδήποτε επένδυση με επιχειρηματικά κεφάλαια (Venture Capital) σε ακίνητα περιέχει ρίσκο και προϋποθέτει καλή ανάλυση της αγοράς για να υπάρχει κέρδος. Για να γίνει πιο κατανοητό αυτό το παράδειγμα μπορεί κανείς να σκεφτεί και τις μετοχές στο χρηματιστήριο. Ελέγχοντας τις τάσεις της αγοράς οι επενδυτές επιλέγουν να αγοράσουν μετοχές εταιριών που έχουν καλή προοπτική ανάπτυξης. Βέβαια, οι διαφορές στην χρηματιστηριακή αγορά με αυτή των ακινήτων είναι πολύ μεγάλες με τη δεύτερη να επηρεάζεται από περισσότερους εξωτερικούς παράγοντες που κάνουν την οποιαδήποτε ανάλυση πιο περίπλοκη. Ένα ακόμη πιο χειροπιαστό παράδειγμα, που σχετίζεται με αυτού του είδους το πρόβλημα, είναι η οικονομική κρίση του 2008 που αφορούσε τα στεγαστικά δάνεια. Η πληροφορία που έδειχνε ότι η οικονομία θα έφτανε σε αυτή την κατάσταση υπήρχε. Αυτό που δεν υπήρχε ήταν η κατάλληλη τεχνογνωσία και τα εργαλεία για την εξαγωγή της. Όλες αυτές οι περιπτώσεις δείχνουν την ανάγκη ύπαρξης επιπλέον επιπέδων ανάλυσης δεδομένων και υπολογιστικών συστημάτων που τα υλοποιούν.




*Εικόνα 14 Χρέος σε σχέση με το ΑΕΠ σε ορισμένες Ευρωπαϊκές χώρες μετά το 2008*

Για να προσομοιώσουμε το πρόβλημα που περιγράψαμε θα χρησιμοποιηθεί σαν δεδομένα εισόδου ένα μέρος της βάσης δεδομένων της εταιρίας Yelp. Το σύνολο των δεδομένων αυτών δίδεται για λόγους ακαδημαϊκούς από την ίδια την εταιρία με σκοπό να πραγματοποιηθεί έρευνα και ανάλυση καθώς και συγκριτική αξιολόγηση (benchmarking). Το δοσμένο Dataset περιέχει πληροφορίες για τοπικές επιχειρήσεις από διάφορες πόλεις καθώς και ανασκοπήσεις (reviews) και συμβουλές (tips) από χρήστες για κάποιες επιχειρήσεις σε πόλεις του Ηνωμένου Βασιλείου, της Γερμανίας, του Καναδά και των Ηνωμένων Πολιτειών. Τα αρχεία περιέχουν την πληροφορία σε μορφή JSON. Συνολικά διατίθενται δεδομένα που αφορούν περίπου 150.000 επιχειρήσεις, 1.000.000 tips καθώς και 4.200.000 reviews. Κάθε γραμμή των αρχείων αποτελεί ξεχωριστή οντότητα. Για να γίνει κατανοητή η μορφή των δεδομένων με τα οποία θα δουλέψουμε παρουσιάζεται μία εγγραφή JSON από κάθε αρχείο που θα χρησιμοποιηθεί.

*yelp_academic_dataset_business.json*

{"business_id":"0DI8Dt2PJp07XkVvIEIIcQ", "name":"Innovative Vapors", "neighborhood":"", "address":"227 E Baseline Rd, Ste  J2", "city":"Tempe", "state":"AZ", "postal_code":"85283", "latitude":33.3782141, "longitude":-111.936102, "stars":4.5, "review_count":17, "is_open":0, "attributes":["BikeParking: True", "BusinessAcceptsBitcoin: False",  "BusinessAcceptsCreditCards: True", "BusinessParking: {'garage': False, 'street': False, 'validated': False, 'lot': True, 'valet': False}", "DogsAllowed: False", "RestaurantsPriceRange2: 2","WheelchairAccessible: True"], "categories":["Tobacco Shops", "Nightlife", "Vape Shops", "Shopping"], "hours":["Monday 11:0-21:0", "Tuesday 11:0-21:0", "Wednesday 11:0-21:0", "Thursday 11:0-21:0","Friday 11:0-22:0","Saturday 10:0-22:0","Sunday 11:0-18:0"],"type": "business"}

*yelp_academic_dataset_review.json*

{"review_id":"IInSTHnXQ-qGpnUuXRf6pg", "user_id":"XnuuGtEI7E8syck0Qhj03w", "business_id":"LTICaCGZE14GuaUXUGbamg", "stars":5, "date":"2015-12-29", "text":"I cannot speak highly enough of Alex and the team at Cut and Taste. They've catered our events for the last two years and we couldn't be more pleased! \n\nThey always use the highest quality ingredients and are some of the best chefs in the country. The meat is always cooked perfectly and you can taste the care and thought that is put into every dish you are served. \n\nYou can always count on Cut and Taste to be on time, professional, and friendly. If you are looking for a company for catering give Cut and Taste a shot. You definitely won't be disappointed.", "useful":3, "funny":2,"cool":2, "type":"review"}

*yelp_academic_dataset_tip.json*

{"text":"Cool ranch Doritos locos tacos was pretty good","date":"2014-01-08", "likes":0,
"business_id":"GDnbt3isfhd57T1QqU6flg", "user_id":"Gfxb_D1k8tueAoGn4cseoA", "type":"tip"}

Στη συνέχεια, θα θεωρήσουμε ότι τα στοιχεία των επιχειρήσεων θα είναι ήδη αποθηκευμένα σε κάποιο σύστημα μαζί με κάποιο μέρος από τα reviews και τα tips. Για να προσομοιώσουμε την ροή δεδομένων με πραγματικό χρόνο θα χρησιμοποιήσουμε τα υπόλοιπα reviews και tips. Επειδή εισάγουμε όλες τις πληροφορίες των επιχειρήσεων στο σύστημα θεωρήθηκε ότι δεν θα μπορούν να εισαχθούν καινούριες επιχειρήσεις όσο θα γίνεται η αξιολόγηση της αρχιτεκτονικής. Επίσης, είναι συχνό το φαινόμενο να γίνεται κάποιου είδους άμεσης επεξεργασίας στις ροές δεδομένων πριν την αποθήκευση. Για να γίνει αυτό, χρησιμοποιήθηκε ένας αλγόριθμος ανάλυσης συναισθημάτων που ελέγχει αν το review ή το tip που έκανε κάποιος χρήστης για την επιχείρηση είχε θετικό η αρνητικό περιεχόμενο. Αυτό γίνεται πριν η πληροφορία μπει στο σύστημα και αρχίσει η επεξεργασία. Τα streaming δεδομένα θεωρούμε ότι έρχονται με σωστή χρονική σειρά όπως δηλαδή θα ερχόντουσαν και στην πραγματικότητα, αν δηλαδή δεν κάναμε προσομοίωση.

# 4. Τεχνική ανάλυση προτεινόμενης λύσης

Σε αυτή την ενότητα θα γίνει η περιγραφή της προτεινόμενης λύσης δίνοντας έμφαση στο τεχνικό κομμάτι της. Πιο συγκεκριμένα, θα αναλυθούν οι τεχνολογίες που θα χρησιμοποιηθούν στην τελική αρχιτεκτονική, πως συνδέονται μεταξύ τους και πως λειτουργεί ένα τέτοιο σύστημα χρησιμοποιώντας ένα συγκεκριμένο σετ δεδομένων.

## 4.1 Ανάλυση components του συστήματος

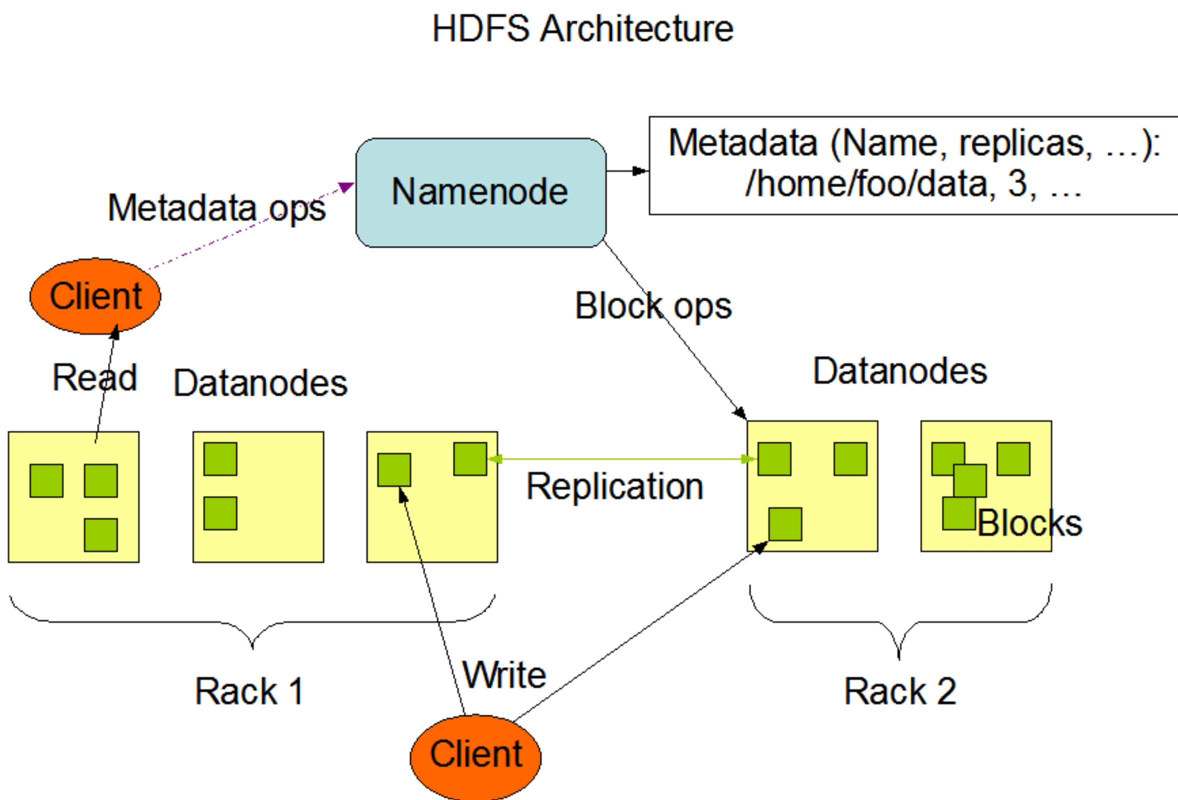
Πριν παρουσιαστεί η τελική αρχιτεκτονική θα πρέπει να αναλυθούν οι τεχνολογίες που θα χρησιμοποιηθούν και σχηματίζουν την τελική πλατφόρμα.

### 4.1.1 HDFS (Hadoop Distributed File System)

Το Hadoop Distributed File System (HDFS) ανήκει στην πλατφόρμα Apache Hadoop, που προσφέρει κατανεμημένους τρόπους αποθήκευσης και επεξεργασίας Big Data. Συγκεκριμένα, το HDFS είναι ένα σύστημα αρχείων κατανεμημένης μορφής που έχει σχεδιαστεί να τρέχει σε συστήματα μεσαίας ισχύος. Επίσης, έχει ορισμένα χαρακτηριστικά και στοιχεία που θα βοηθήσουν στην κάλυψη των αναγκών που έχουν δημιουργήσει τα Big Data. Τέτοια είναι, η ανοχή σε σφάλματα, καθώς το HDFS θα τρέχει σε πολλαπλούς υπολογιστές, η αποθήκευση και επεξεργασία τεράστιων κομματιών πληροφορίας, η μεγάλη διαθεσιμότητα των δεδομένων για τις εφαρμογές που τα χρησιμοποιούν και η υψηλή φορητότητα ώστε να μπορεί να χρησιμοποιείται σε πολλά και διαφορετικά περιβαλλοντα καθώς και η εύκολη μετακίνηση αυτών κοντά στα δεδομένα.

Σε ότι αφορά την αρχιτεκτονική του HDFS χρησιμοποιείται ένα μοντέλο master/slave. Ένα σύμπλεγμα υπολογιστών που χρησιμοποιεί HDFS διαθέτει ένα master server που ονομάζεται NameNode. Αυτός διαχειρίζεται το χώρο ονομάτων του συστήματος αρχείων και ελέγχει την πρόσβαση από τους πελάτες. Επίσης, υπάρχουν και πολλοί εξυπηρετητές που ονομάζονται DataNodes και διαχειρίζονται την αποθήκευση στο μηχάνημα που τρέχουν. Το HDFS προσφέρει τη δυνατότητα αποθήκευσης σε μορφή αρχείων, όμως εσωτερικά το κάθε αρχείο χωρίζεται σε κομμάτια και αυτά αποθηκεύονται σε ένα σύνολο DataNodes. Ο NameNode είναι υπεύθυνος για το άνοιγμα, το κλείσιμο, τη μετονομασία και τη διαχείριση καταλόγων καθώς και την αντιστοίχιση των κομματιών με τους DataNodes που τα περιέχουν. Οι DataNodes ικανοποιούν τα αιτήματα διαβάσματος και ανάγνωσης των

πελατών και ευθύνονται για τη δημιουργία, διαγραφή και κλωνοποίηση κομματιών σε περίπτωση που το ζητήσει ο NameNode. Το HDFS έχει δημιουργηθεί και χρησιμοποιεί Java, Έτσι κάθε μηχάνημα που υποστηρίζει Java μπορεί να τρέξει είτε ως DataNode είτε ως NameNode πράγμα που κάνει τη φορητότητα πολύ εύκολη.



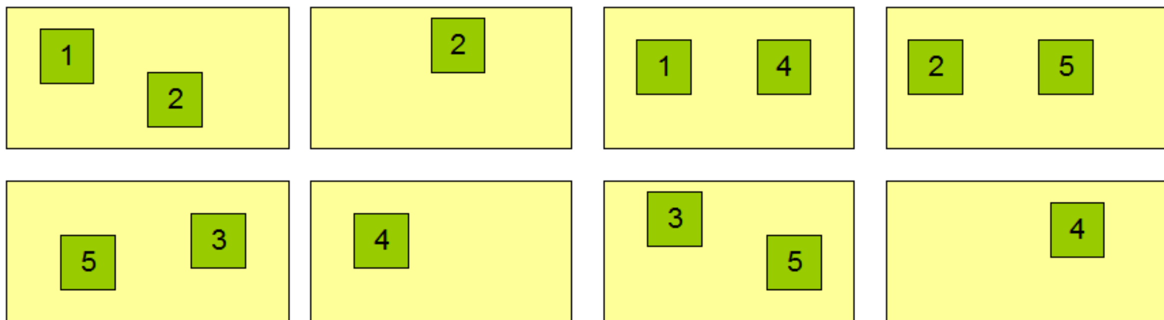

*Εικόνα 15 Αρχιτεκτονική HDFS*

Ένα ακόμη σημαντικό στοιχείο του HDFS είναι η «κλωνοποίηση» των δεδομένων ή αλλιώς replication. Τα κομμάτια των αρχείων που αποθηκεύονται στους DataNodes αντιγράφονται, μία η περισσότερες φορές σε άλλους DataNodes για να υπάρχει ανοχή στα σφάλματα καθώς και υψηλή διαθεσιμότητα των δεδομένων. Αποτελούν ουσιαστικά ένα είδος backup των δεδομένων και ο αριθμός των αντιγράφων μπορεί να ορισθεί από τους χρήστες αλλά και τις εφαρμογές. Τις αποφάσεις που αφορά το Replication τις παίρνει ο NameNode. Φυσικά, ένα αίτημα ανάγνωσης μπορεί να ικανοποιηθεί και από κάποιο αντίγραφο αλλά το HDFS προσπαθεί να το εκτελέσει χρησιμοποιώντας αυτό που βρίσκεται πιο κοντά στον αναγνώστη. Αντίστοιχα στην εγγραφή ή στην αλλαγή κάποιου κομματιού πρέπει να ενημερωθούν και οι κλώνοι του.

Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

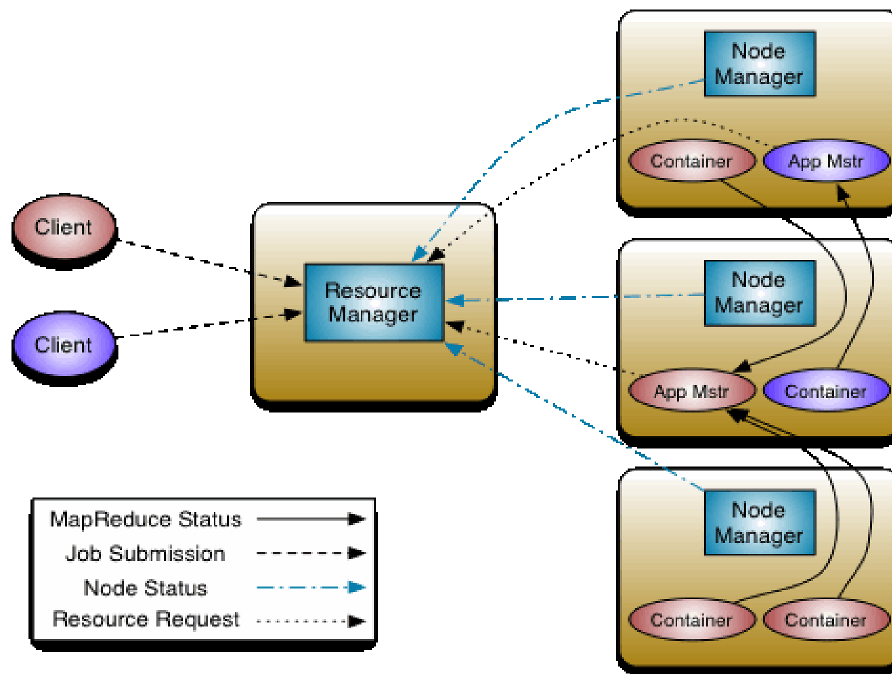
Datanodes




*Εικόνα 16 Replication στο HDFS*

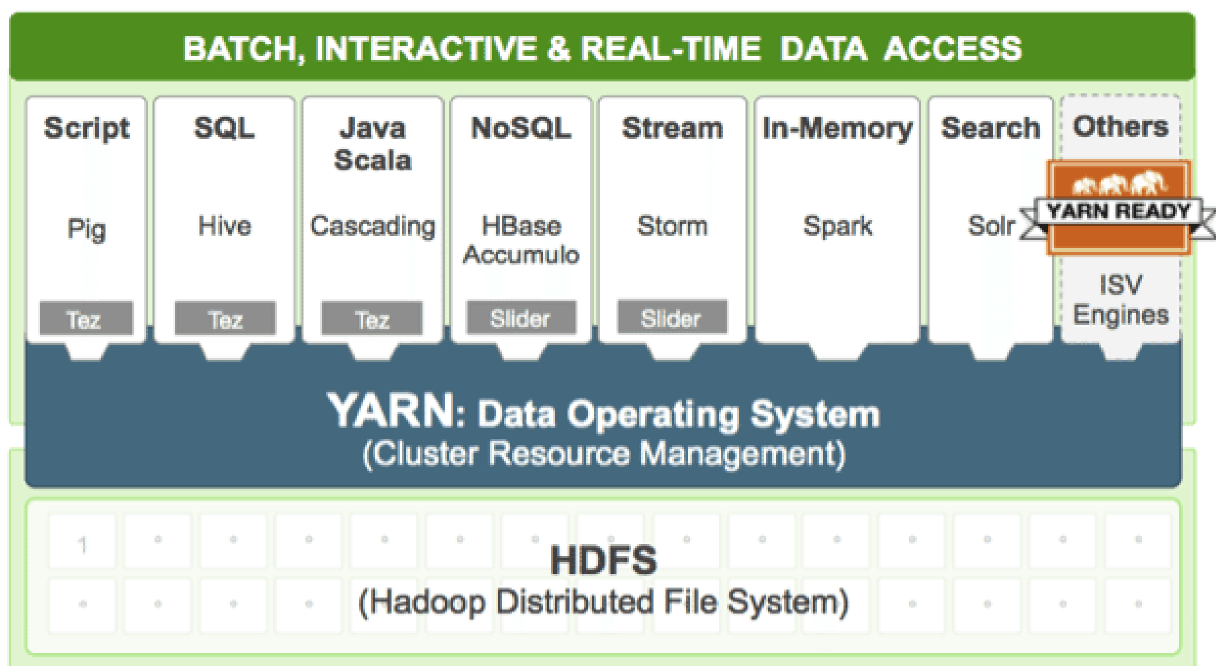
### 4.1.2 YARN (Yet Another Resource Negotiator)

Το YARN αποτελεί κι αυτό μέρος της πλατφόρμας Apache Hadoop όπως και το HDFS, και είναι βασικά ένας διαχειριστής πόρων και χρόνο προγραμματιστής εργασιών. Η βασική ιδέα που ακολουθεί είναι ο χωρισμός αυτών των δύο ιδιοτήτων σε διαφορετικούς δαίμονες (daemons). Για την ακρίβεια, υπάρχει ένας καθολικός Resource Manager (RM) και σε κάθε εργασία αντιστοιχίζεται ένας Application Master (AM). Ο RM είναι υπεύθυνος για την κατανομή των πόρων σε όλες τις εφαρμογές καθώς και για την αποδοχή αιτημάτων των εργασιών. Αντίστοιχα, κάθε κόμβος του cluster διαθέτει ένα Node Manager που έχει την ευθύνη των containers (αναπαράσταση κομματιών μνήμης που έχουν αποδοθεί στην εφαρμογή), παρακολουθώντας πόσο χρησιμοποιούν τους πόρους και αναφέροντας αυτά τα ποσά στον RM. Ο AM κάθε εργασίας ζητάει πόρους από τον RM και δουλεύει με τους Node Managers για την εκτέλεση και παρακολούθηση των εργασιών.




*Εικόνα 17 Αρχιτεκτονική Yarn*

Επίσης κάτι πολύ σημαντικό που έχει προσφέρει το YARN είναι η δυνατότητα χρήσης διαφορετικών τεχνολογιών πάνω σε δεδομένα που έχουν αποθηκευτεί στο HDFS. Με αυτό τον τρόπο μπορούν να δημιουργηθούν υπολογιστικά συστήματα που καλύπτουν μεγάλο εύρος εφαρμογών ταυτόχρονα, καθώς χρησιμοποιούνται μηχανές διαφορετικού είδους επεξεργασίας που έχουν πρόσβαση το ίδιο σύνολο δεδομένων την ίδια χρονική στιγμή.




*Εικόνα 18 Παράδειγμα συνδεσιμότητας YARN*

## 4.1.3 Apache Avro

Το Apache Avro είναι ένα σύστημα σειριοποίησης δεδομένων (data serialization) δηλαδή ένας τρόπος μετατροπής δομών δεδομένων σε μία μορφή που μπορεί να αποθηκευτεί, αποσταλεί αλλά και να επιστρέψει στην αρχική της μορφή εύκολα. Το Apache Avro προσφέρει την δυνατότητα δημιουργίας πολύπλοκων και πλούσιων τύπων δεδομένων τους οποίους τους μετατρέπει σε συμπαγείς, μικρές και γρήγορες δομές δεδομένων.

Το Avro βασίζεται πολύ στο σχήμα (schema) των δεδομένων, δηλαδή τη μορφή τους. Αυτό ορίζεται με κάποιο αρχείο JSON. Η πληροφορία που έχει σειριοποιηθεί με αυτό το πρότυπο και έχει αποθηκευτεί σε κάποιο αρχείο περιέχει, πέρα από τα δεδομένα, και το σχήμα που ακολουθούν, οπότε είναι εφικτή η επεξεργασία τους χωρίς τη δημιουργία κώδικα ή στατικών δομών. Αυτό συμβάλει στη δημιουργία συστημάτων ανάλυσης δεδομένων γενικού τύπου, που δεν εξαρτώνται από τη γλώσσα προγραμματισμού διότι χρησιμοποιούνται γενικοί μέθοδοι τύπου put/get για την πρόσβαση στα αντίστοιχα πεδία. Επίσης, επειδή το σχήμα βρίσκεται μαζί με την πληροφορία που περιγράφει, κατά την ανάγνωση και δεν χρειάζονται πληροφορίες για τον τύπο ή το μέγεθος των δεδομένων, έχουμε κωδικοποιήσεις μικρού μεγέθους που είναι εύκολα διαχειρίσιμες. Τέλος, μπορεί να χρησιμοποιηθεί διαφορετικό σχήμα για την σειριοποίηση και διαφορετικό για την μετατροπή στην αρχική μορφή. Το πρότυπο είναι έτσι φτιαγμένο και αναλαμβάνει τα πεδία που έχουν αλλαχθεί, προστεθεί ή απλά λείπουν, άρα δίνεται η δυνατότητα της εξέλιξης τους σχήματος κατά τη διάρκεια δημιουργίας ή συντήρησης ενός συστήματος.

**Avro vs...**

| Technologies | Protobuf | Thrift | Avro |
|---|---|---|---|
| Created | 2001 (2008) | 2007 | 2009 |
| Creator / Maintainer | Google / Google | Facebook / Apache | Doug cutting / Apache |
| Schema evolution | Field Tag | Field Tag | Schema |
| Static/Dynamic | Yes/No | Yes/No | Yes/Yes |
| Hadoop support | No | No | Yes |
| RPC | No | Yes | Yes |
| Used by | Google | Facebook, Cassandra | Hadoop, Liveperson |
| Lang support | Good | Great | Good |

*Εικόνα 19 Σύγκριση Apache Avro με παρόμοια πρότυπα*

## 4.1.4 Apache HBase

Η HBase αποτελεί μία μη σχεσιακή βάση (NoSQL) δεδομένων ανοιχτού λογισμικού που τρέχει πάνω από το HDFS και προσφέρει πρόσβαση , είτε για εγγραφή είτε για ανάγνωση σε μεγάλα σύνολα δεδομένων. Η HBase είναι επίσης και κατανεμημένη, πράγμα που της δίνει αρκετές δυνατότητες κλιμακωσιμότητας. Η φυσική ενσωμάτωση της στο Apache Hadoop την κάνει να λειτουργεί πολύ ευκολά μαζί με άλλες μηχανές πρόσβασης δεδομένων μέσω του YARN.

Η χρησιμότητας αυτής της βάσης φαίνεται κυρίως όταν χρειαζόμαστε τυχαία πρόσβαση στα δεδομένα μας, δηλαδή σε σενάρια πραγματικού χρόνου όπου τα δεδομένα αλλάζουν συνεχώς και η διαχείριση πολλαπλών reads/writes πρέπει να γίνεται γρήγορα. Ο σχεδιασμός της HBase της δίνει τη δυνατότητα να έχει πολύ μεγάλους πίνακες, της τάξεως των δισεκατομμυρίων γραμμών και εκατομμυρίων στηλών. Η πληροφορία που αποθηκεύεται μπορεί να διαθέτει πολλά δομικά επίπεδα και να μην υπακούει κάποιο συγκεκριμένο σχήμα πράγμα που την κάνει ιδανική στο να αποθηκεύει ημιδομημένα δεδομένα. Η αυτόματη κατανομή φορτίου των πινάκων, το replication (κυρίως μέσω του HDFS) και η υψηλή διαθεσιμότητα των δεδομένων που διαθέτει είναι όλα χαρακτηριστικά που δείχνουν την ανοχή της στα σφάλματα. Επίσης, το γεγονός ότι κάνει αναζητήσεις σχεδόν σε πραγματικό χρόνο και χρησιμοποιεί την κύρια μνήμη για caching την κάνει εξαιρετικά γρήγορη. Όλα αυτά σε συνδυασμό με την ευχρηστία της, λόγω της ευελιξίας στο μοντέλο των δεδομένων που δέχεται καθώς και η πρόσβαση της μέσω εύκολου API της Java την κάνει ένα ισχυρό εργαλείο.

*Εικόνα 20 Λογότυπο HBase*

Όσο αφορά το μοντέλο δεδομένων που ακολουθεί η HBase, αυτά αποθηκεύονται σε μορφή πινάκων που έχουν γραμμές και στήλες. Όμως, για να μην υπάρχει σύγχυση με τις σχεσιακές βάσεις δεδομένων θα ήταν καλύτερο να σκεφτούμε την HBase σαν μια δομή χάρτη (map) πολλών διαστάσεων. Στη συνέχεια θα αναλυθούν ορισμένες έννοιες που αφορούν το μοντέλο αποθήκευσης.

*Γραμμή (Row)*

Μία γραμμή στην HBase αποτελείται από ένα κλειδί γραμμής (row key) και από μία η περισσότερες στήλες. Οι γραμμές ταξινομούνται αλφαβητικά με βάση αυτό το κλειδί.

*Στήλη (Column)*

Μία στήλη είναι ο συνδυασμός μία οικογένειας στηλών και ενός αναγνωριστικού στήλης και χωρίζονται από τον χαρακτήρα «:».

*Οικογένεια Στηλών (Column Family)*

Είναι ουσιαστικά ένα σύνολο από στήλες μαζί με τις τιμές τους. Κάθε column family έχει δικές του ιδιότητες αποθήκευσης όπως για παράδειγμα πώς κωδικοποιούνται τα row keys ή πώς είναι συμπιεσμένα τα δεδομένα. Κάθε γραμμή σε ένα πίνακα έχει τις ίδιες οικογένειες στηλών, μπορεί όμως σε μία γραμμή να μην υπάρχει κάτι αποθηκευμένο σε κάποιο column family.

*Αναγνωριστικό Στήλης (Column Qualifier)*

Αυτό το στοιχείο προστίθεται σε ένα column family για να δείξει τη στήλη στην οποία βρίσκεται η πληροφορία που ζητείται. Αν έχουμε, για παράδειγμα, την οικογένεια στηλών `personal_data` παραδείγματα αναγνωριστικών είναι τα `e-mail` και `address` που δημιουργούν τις στήλες `personal_data:e-mail` και `personal_data:address`. Οι οικογένειες στηλών είναι συγκριμένες την ώρα της δημιουργίας του πίνακα, όμως τα αναγνωριστικά στηλών μεταβάλλονται και μπορεί να είναι πολύ διαφορετικά ανά γραμμή

*Κελί (Cell)*

Αποτελεί τον συνδυασμό της γραμμής, της οικογένειας στηλών και αναγνωριστικού στήλης. Ένα κελί περιέχει μία τιμή και μία χρόνο σφραγίδα που αναπαριστά την έκδοση της τιμής.

*Χρόνο σφραγίδα (Timestamp)*

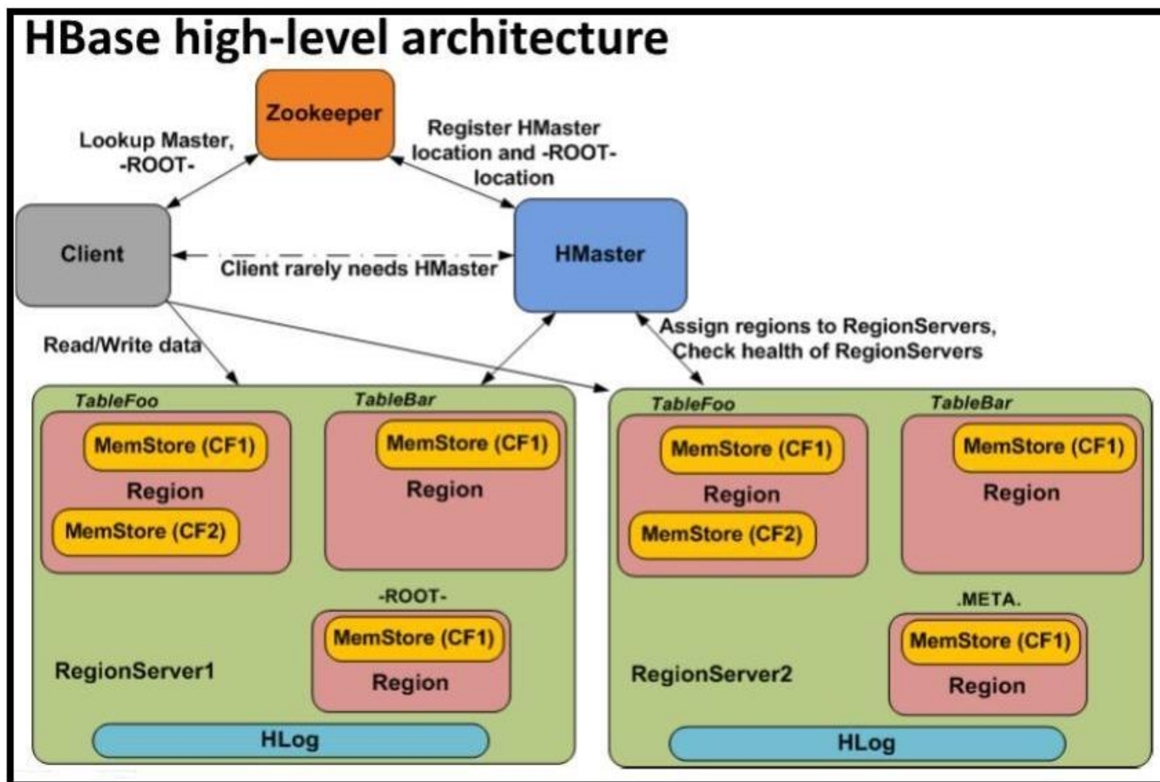
Μαζί με την τιμή γράφεται και μία χρόνο σφραγίδα που δείχνει τον χρόνο στον οποίο γράφτηκε η τιμή στον πίνακα. Αυτή μπορεί να αλλαχθεί κατά την εισαγωγή δεδομένων αν το επιθυμεί ο χρήστης.

COLUMN FAMILIES

| Row key | personal data | | professional data | |
|---|---|---|---|---|
| empid | name | city | designation | salary |
| 1 | raju | hyderabad | manager | 50,000 |
| 2 | ravi | chennai | sr.engineer | 30,000 |
| 3 | rajesh | delhi | jr.engineer | 25,000 |

*Εικόνα 21 Πίνακας στην HBase*

Κάθε πίνακας στην HBase έχει ένα πρωτεύον κλειδί και αυτό είναι το κλειδί γραμμής. Ο χώρος τιμών του κλειδιού αυτού χωρίζεται σε ακολουθιακά κομμάτια και ανατίθεται σε ένα Region, όπως είναι γνωστό στο οικοσύστημα της HBase. Η HBase διαθέτει RegionServers που έχουν ένα η περισσότερα Regions έτσι ώστε ο υπολογιστικός φόρτος να μοιράζεται ομοιόμορφα. Αν κάποια κλειδιά χρησιμοποιούνται περισσότερο το Region μπορεί να διασπαστεί κι άλλο για τη διατήρηση της ισορροπίας. Ο βασικός εξυπηρετητής της HBase (HMaster) και ο Zookeeper (βλ. 4.1.6 Apache ZooKeeper) είναι υπεύθυνοι για να δείχνουν πληροφορίες, σε ότι αφορά την τοπολογία του cluster, στους πελάτες. Εκεί συνδέονται οι πελάτες και λαμβάνουν τη λίστα των RegionServers, και τα Region τα οποία διαθέτει ο καθένας. Γι' αυτό το λόγο γίνεται άμεση επικοινωνία με τον RegionServer όταν χρειαζόμαστε κάποια πληροφορία. Βλέποντας πως δουλεύει η κύρια αρχιτεκτονική φαίνεται ξεκάθαρα και η υψηλή διαθεσιμότητα που προσφέρει αυτή η βάση δεδομένων. Η εύκολη πρόσβαση στα τοπολογικά στοιχεία του cluster καθώς και ο διαμοιρασμός της πληροφορίας σε πολλούς κόμβους συμβάλουν σε αυτό.



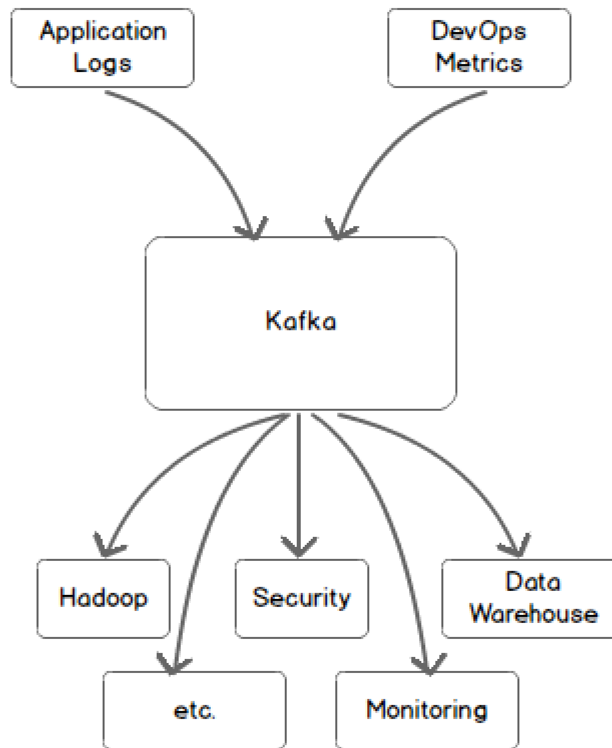

*Εικόνα 22 Αρχιτεκτονική HBase*

### 4.1.5 Apache Kafka

Το σύστημα της Apache Kafka αποτελεί μία κατανεμημένη πλατφόρμα σχεδιασμένη για streaming γραμμένη σε Scala και Java. Για την ακρίβεια αποτελεί ένα σύστημα που βασίζεται στην δημοσιοποίηση και την εγγραφή σε ροές πληροφοριών. Ένα τέτοιο σύστημα είναι ευρύτερα γνωστό ως «publish-subscribe» και αποτελεί ουσιαστικά μία ουρά μηνυμάτων. Κάποια άλλα χαρακτηριστικά της είναι, η υψηλή κλιμακωσιμότητα,, η ταχύτητα και η δυνατότητα αποθήκευσης μηνυμάτων με τέτοιο τρόπο ώστε σε περίπτωση αποτυχίας να μην υπάρχει απώλεια δεδομένων καθώς και η δυνατότητα άμεσης επεξεργασίας τους. Γι' αυτούς τους λόγους η Kafka αποτελεί ιδανική λύση τόσο για τη δημιουργία pipelines πραγματικού χρόνου που μεταφέρουν τα δεδομένα μεταξύ συστημάτων ή υπηρεσιών με αξιόπιστο τρόπο αλλά και για τη σχεδίαση εφαρμογών που επεξεργάζονται ή αντιδρούν με συγκεκριμένο τρόπο σε ροές δεδομένων.

Αρχικά πρέπει να ξεκαθαριστεί η έννοια του *topic* καθώς είναι πολύ σημαντική στο οικοσύστημα της Kafka. Όλα τα μηνύματα – αρχεία που εισέρχονται οργανώνονται σε topics. Όταν θέλει κάποιος να στείλει κάποια πληροφορία ή να διαβάσει πρέπει να το κάνει στο

αντίστοιχο topic. Ένας ακόμη σημαντικός όρος είναι το record. Στην Kafka δεν γίνεται σχεδόν ποτέ λόγος για «streams of data» αλλά αναφερόμαστε σε «streams of records». Με αυτή τη μορφή εισέρχονται τα δεδομένα στα διαφορετικά topics. Κάθε record περιέχει ένα κλειδί, μία τιμή (την πληροφορία μας) και μία χρόνο σφραγίδα. Τα τέσσερα κύρια API που διαθέτει, δίνουν τη δυνατότητα ανάπτυξης χρήσιμων εφαρμογών.




*Εικόνα 23 Η Kafka ως pipeline δεδομένων σε άλλα συστήματα*

*Producer API (Παραγωγός)*

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

*Consumer API (Καταναλωτής)*

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

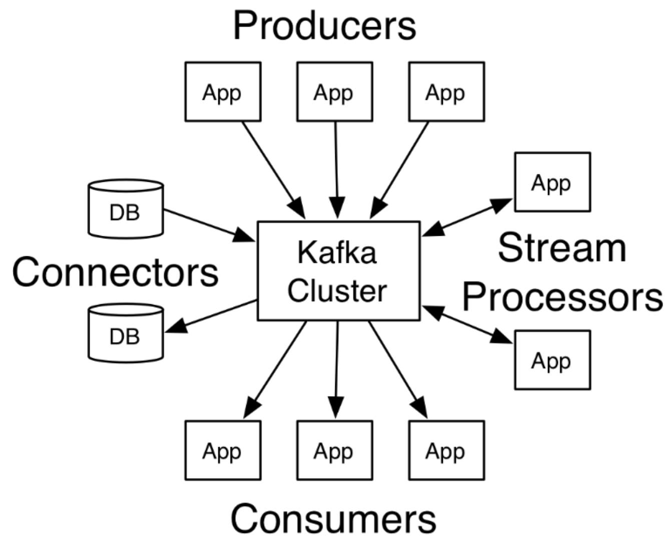
*Streams API*

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.

*Connector API*

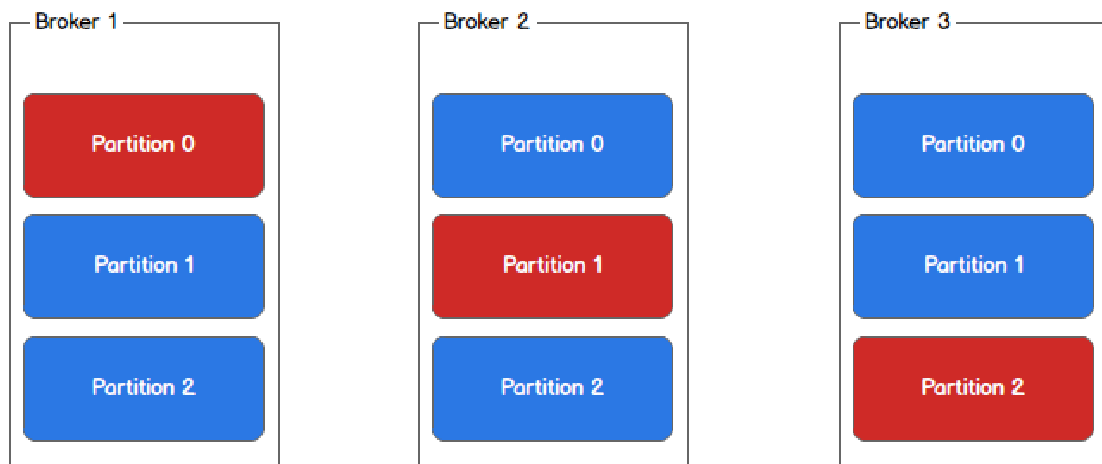
Επιτρέπει την δημιουργία επαναχρησιμοποιήσιμων καταναλωτών ή παραγωγών για την σύνδεση των topics της Kafka με άλλες εφαρμογές ή υπολογιστικά συστήματα.

Η ανάπτυξη εφαρμογών γίνεται κυρίως σε Java και Scala, αλλά η προσπάθεια ενσωμάτωσης της Kafka σε περισσότερα συστήματα έχει οδηγήσει στην υποστήριξη και άλλων γλωσσών προγραμματισμού όπως οι C/C++ και Python.




*Εικόνα 24 Τα βασικά API της Kafka*

Όπως ήδη ειπώθηκε η Kafka, όντας ένα κατανεμημένο σύστημα, τρέχει σε cluster και κάθε κόμβος του cluster ονομάζεται Kafka broker. Σε αυτό το σημείο πρέπει να αναλυθεί η ανατομία του topic της Kafka. Τα topic είναι χωρισμένα σε έναν αριθμό από κομμάτια (partitions). Αυτό συμβαίνει για να υπάρχει παραλληλισμός. Επειδή, επιτρέπεται η ύπαρξη πολλών καταναλωτών σε πάνω σε ένα topic, ο χωρισμός των δεδομένων και ο διαμοιρασμός τους σε διαφορετικούς brokers επιτρέπει σε διαφορετικούς consumers να διαβάζουν από το ίδιο topic παράλληλα. Υπάρχει και η δυνατότητα πολλοί καταναλωτές να διαβάζουν διαφορετικά partitions παράλληλα, έτσι δημιουργείται πολύ υψηλή διακίνηση και γρήγορος ρυθμός επεξεργασίας δεδομένων. Τα partitions του κάθε topic είναι replicated για να υπάρχει ανοχή σε σφάλματα. Οπότε τελικά, αυτό που κάνουν οι brokers είναι να κρατάνε έναν αριθμό από partitions και καθ’ ένα από αυτά θα έχει είτε την ιδιότητα του leader 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 σταμάτησε να λειτουργεί, ένα αντίγραφο θα πάρει την θέση του και δεν θα γίνει διακοπή των εργασιών που εκτελούνται.




*Εικόνα 25 Kafka Brokers*

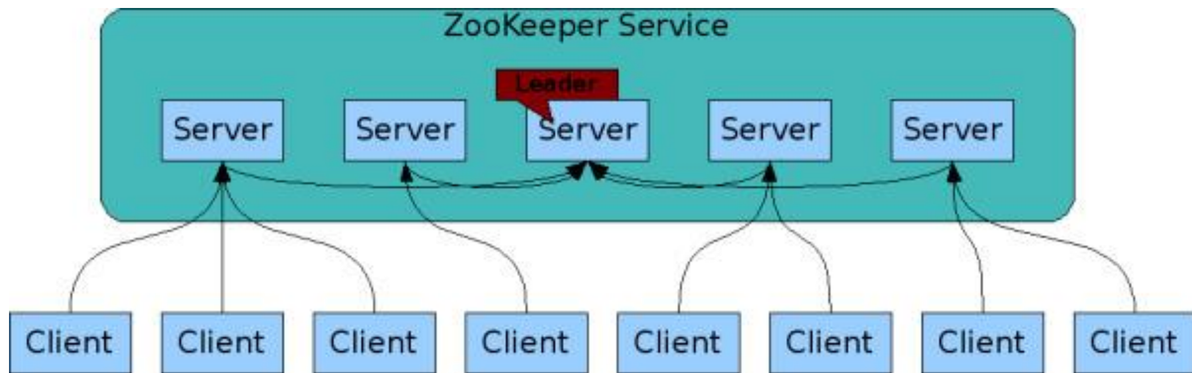
### 4.1.6 Apache ZooKeeper

Ο Apache ZooKeeper είναι μία κατανεμημένη υπηρεσία συντονισμού για κατανεμημένες εφαρμογές. Συντονίζει τους κόμβους κάποιου cluster και διατηρεί τα δεδομένα που μοιράζονται μεταξύ αυτών χρησιμοποιώντας διάφορες τεχνικές συγχρονισμού. Μερικές 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 ειπώθηκε αποτελεί κι αυτός ένα κατανεμημένο σύστημα. Ακολουθεί μοντέλο πελάτη-εξυπηρετητή όπου πελάτες είναι οι κόμβοι (για παράδειγμα τα μηχανήματα) που χρησιμοποιούνε την υπηρεσία και εξυπηρετητές οι κόμβοι που την προσφέρουν. Οι ZooKeeper servers ονομάζονται ensemble και μεταξύ αυτών υπάρχει ένας leader ο οποίος εκλέγεται μόλις ξεκινήσει η υπηρεσία. Οι πελάτες μπορούν να συνδεθούν σε κάποιον ZooKeeper server 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.

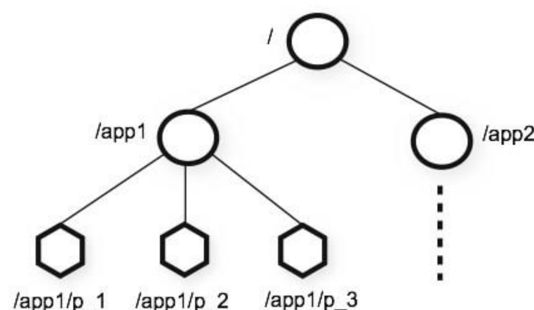
Το συνολικό σύστημα του ZooKeeper μοιάζει με ένα σύστημα αρχείων ιεραρχικού μοντέλου και αποτελείται από τα λεγόμενα znodes (ZooKeeper data nodes). Αυτά μοιάζουν με αρχεία συστήματος UNIX με τη διαφορά ότι μπορούν να έχουν και κόμβους παιδιά. Η

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.




*Εικόνα 26 Αρχιτεκτονική ZooKeeper*

Τα δεδομένα που είναι αποθηκευμένα στους znodes διαβάζονται ή 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 περνάνε όλα από τον leader. Όταν σταλθεί ένα τέτοιο αίτημα ο server που το λαμβάνει το στέλνει 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.




*Εικόνα 27 ZooKeeper Data Node (znode)*

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.

### 4.1.7 Apache Spark

Το Apache Spark αποτελεί μία υπολογιστική πλατφόρμα σχεδιασμένη να λειτουργεί σε cluster υπολογιστών. Μέσω αυτής μπορούν να δημιουργηθούν κατανεμημένες εφαρμογές γενικού σκοπού για την επεξεργασία μεγάλου όγκου δεδομένων χρησιμοποιώντας παραλληλισμό για την επίτευξη μεγάλων ταχυτήτων. Γενικά, το Spark αποτελεί μία επέκταση του προγραμματιστικού μοντέλου MapReduce αλλά διαθέτει πολύ περισσότερα εργαλεία που συμβάλουν στην υλοποίηση περισσότερων υπολογισμών. Ένα βασικό χαρακτηριστικό του είναι η δυνατότητα αποθήκευσης δεδομένων στην κύρια μνήμη του κάθε κόμβου του cluster στο οποίο δουλεύει. Αυτό, μειώνει κατά πολύ τις διαδικασίες εγγραφής και αναζήτησης στο δίσκο και δίνει τελικά πολύ μεγάλο προβάδισμα σε ταχύτητα έναντι του μοντέλου MapReduce στο οποίο βασίστηκε.

Η υλοποίηση του Spark έχει γίνει εξ' ολοκλήρου στη γλώσσα προγραμματισμού Scala αλλά προσφέρει API σε Java, Python και R, καθώς και Scala, για την ανάπτυξη εφαρμογών μαζί με μία μεγάλη ποικιλία βιβλιοθηκών. Ο πυρήνας του Spark αποτελεί τα θεμέλια όλης της πλατφόρμας καθώς προσφέρει κατανεμημένες υπηρεσίες δρομολόγησης και χρονοπρογραμματισμού εργασιών, διαχείριση μνήμης, αποκατάσταση βλαβών καθώς και λειτουργίες εισόδου/εξόδου (I/O) μέσω των APIs που αναφέρθηκαν. Πέρα όμως από αυτά υπάρχουν και εργαλεία υψηλότερου επιπέδου που λειτουργούν πάνω στον πυρήνα και μπορούν να χρησιμοποιηθούν σα βιβλιοθήκες. Αυτά τα στοιχεία είναι:

*Spark SQL*

Περιέχει υποστήριξη ερωτημάτων τύπου SQL και εισήγαγε τα DataFrames. Αυτή η δομή δεδομένων προορίζεται για δομημένα και ημιδομημένα δεδομένα και προσφέρει εύκολη πρόσβαση σε πληροφορίες διαφορετικού είδους, όπως JSON και Avro.

*Spark Streaming*

Χρησιμοποιεί την ταχύτητα δρομολόγησης του πυρήνα του Spark για να εκτελεί αναλύσεις ροών δεδομένων. Χωρίζει τα δεδομένα σε μικρές παρτίδες και πραγματοποιεί τις εργασίες

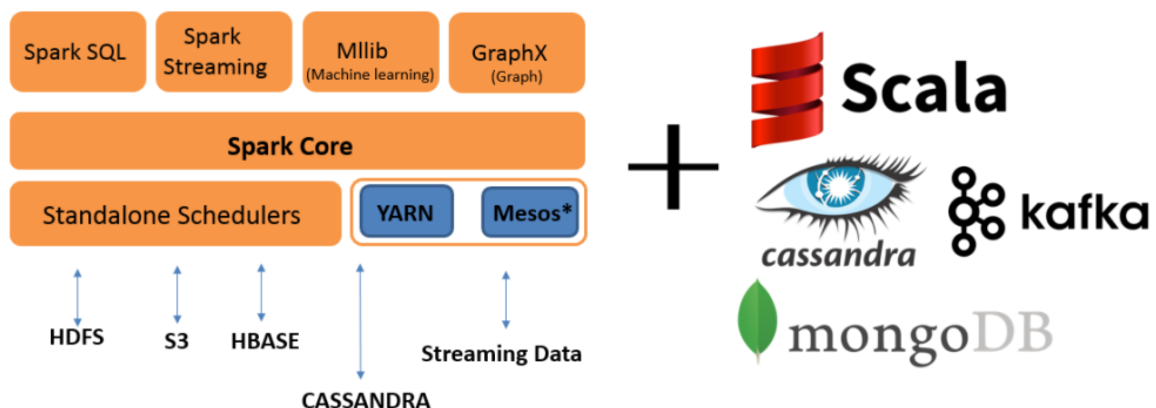
πάνω σε αυτές. Τα mini-batches αυτά επεξεργάζονται όπως οι κανονικές παρτίδες οπότε ο προγραμματισμός εργασιών streaming είναι ίδιος με αυτός των batch εφαρμογών.

*MLlib*

Αποτελεί μία βιβλιοθήκη που έχει συλλογή αλγορίθμων machine learning, υλοποιημένοι για να εκτελούνται κατανεμημένα. Επειδή το Spark χρησιμοποιεί αποδοτικά την κύρια μνήμη των κόμβων του cluster οι αλγόριθμοι αυτοί είναι πολύ πιο γρήγοροι από αντίστοιχα συστήματα που διαβάζουν από το σκληρό δίσκο όπως το Apache Mahout.

*GraphX*

Είναι μία κατανεμημένη πλατφόρμα επεξεργασίας γράφων. Εκμεταλλεύεται την ταχύτητα και τον παραλληλισμό του Spark καθώς και την μεγάλη ποικιλία σε αλγορίθμους που διαθέτει για το χειρισμό και την εκτέλεση υπολογισμών σε Γράφους.

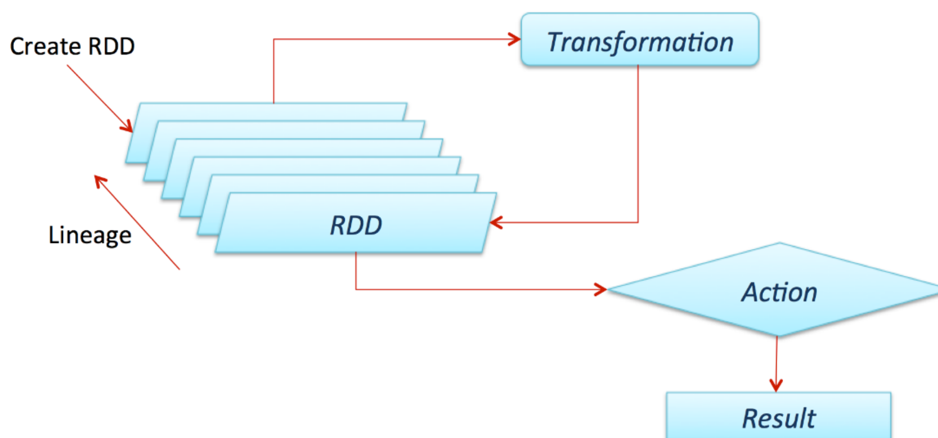



*Εικόνα 28 Οικοσύστημα Spark*

Το Spark απαιτεί κι ένα διαχειριστή του cluster καθώς και ένα κατανεμημένο σύστημα αποθήκευσης. Στην πρώτη κατηγορία υπάρχει έτοιμη ενσωματωμένη υλοποίηση (Standalone Scheduler) αλλά υποστηρίζεται επίσης το Hadoop YARN καθώς και το Apache Mesos. Οι επιλογές για κατανεμημένη αποθήκευση είναι αρκετές και εξαρτώνται και από την εφαρμογή. Τέτοια παραδείγματα είναι είτε συστήματα αρχείων όπως το HDFS , είτε βάσεις δεδομένων όπως η Cassandra και η HBase καθώς και υπηρεσίες cloud computing, όπως το Amazon S3.

Ένα πολύ σημαντικό στοιχείο του Spark είναι το Resilient Distributed Dataset (RDD). Αυτό αποτελεί μία κατανεμημένη αμετάβλητη (immutable) δομή αντικειμένων που χρησιμοποιείται εκτενέστατα από το Spark για την εκτέλεση εργασιών. Μέσα, η δομή μπορεί

να περιέχει τύπους δεδομένων που υποστηρίζουν οι γλώσσες Scala, Java και Python αλλά και αντικείμενα που έχει ορίσει ο χρήστης. Υπάρχουν δύο τρόποι δημιουργίας RDDs, είτε από μία υπάρχουσα δομή, για παράδειγμα έναν πίνακα, είτε απεικονίζοντας ένα σύνολο δεδομένων από κάποιο εξωτερικό αποθηκευτικό σύστημα. Σε κάθε περίπτωση το RDD θα διασπαστεί και θα διαμοιραστεί στους κόμβους του cluster για να μπορεί να γίνεται η επεξεργασία των κομματιών με τρόπο παράλληλο. Τα RDDs διαθέτουν δύο τρόπους επεξεργασίας: τους μετασχηματισμούς (transformations), που δημιουργούν ένα καινούριο RDD από κάποιο υπάρχον, και τις πράξεις (actions) που επιστρέφουν τις τιμές του RDD πίσω στο πρόγραμμα αφού γίνουν οι κατάλληλες επεξεργασίες. Τα transformations πραγματοποιούνται με οκνηρή αποτίμηση (lazy evaluation), δηλαδή δεν εκτελούν κανένα μετασχηματισμό μέχρις ότου συμβεί κάποιο action.

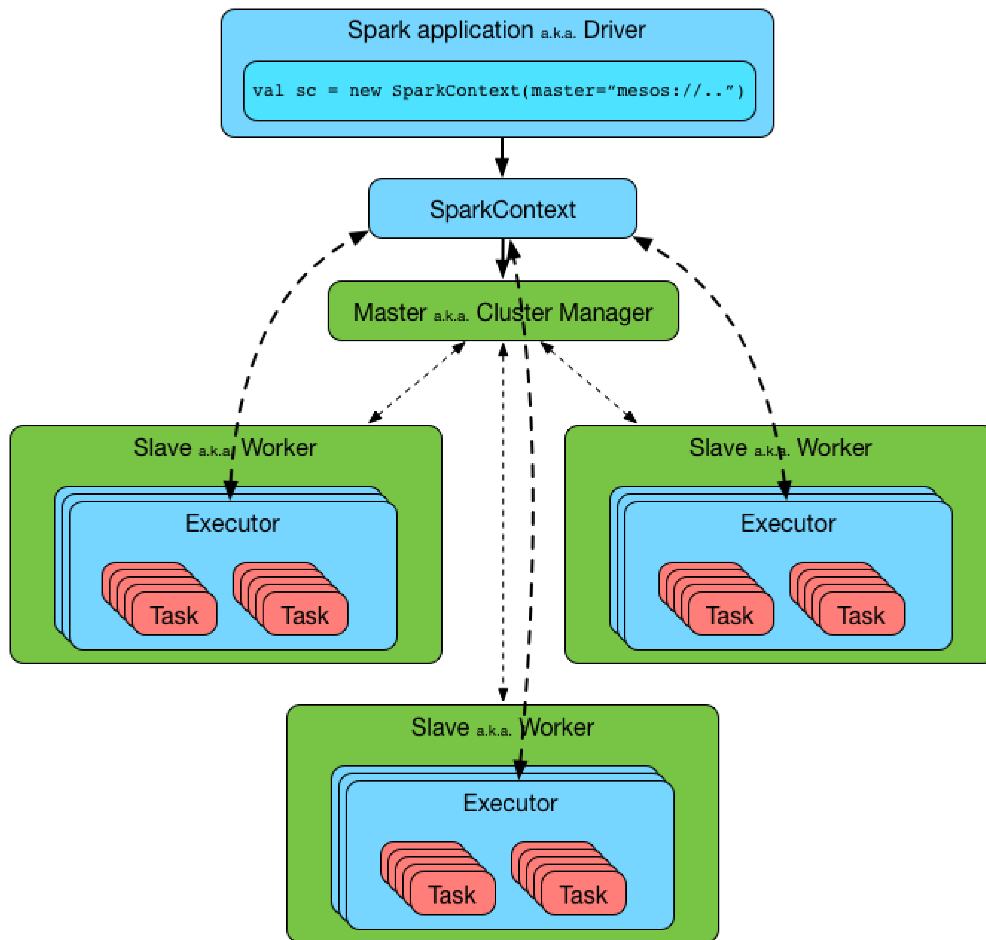



*Εικόνα 29 Ανατομία και πράξεις ενός RDD*

Τα RDD υλοποιούνται με τέτοιο τρόπο στο Spark ώστε να έχουν ιδιαίτερη αντοχή σε τυχόν σφάλματα. Συγκεκριμένα, παρέχονται μηχανισμοί ανάκτησης καθώς αποθηκεύεται η σειρά των μετασχηματισμών που έχουν εκτελεστεί αλλά και πληροφορίες για την πηγή που χρησιμοποιήθηκε για την άντληση δεδομένων. Έτσι σε περίπτωση προβλήματος μπορεί να «ξαναχτιστεί» το RDD που χάθηκε. Μια ακόμη πολύ σημαντική τεχνική που προσφέρεται είναι αυτή της προσωρινής αποθήκευσης στην κύρια μνήμη (caching). Αυτό κάνει τον κάθε κόμβο να αποθηκεύει προσωρινά το κομμάτι ή κομμάτια του RDD στη μνήμη του και όταν εκτελεστεί κάποιο action δε θα χρειαστούν να γίνουν οι αντίστοιχοι μετασχηματισμοί από την αρχή παρά μόνο την πρώτη φορά. Αυτός ο μηχανισμός είναι πολύ χρήσιμος στην επαναχρησιμοποίηση και στις επαναληπτικές αναλύσεις που γίνονται πάνω στα RDD.

Η αρχιτεκτονική του Spark ακολουθεί το μοντέλο master/slave, όπου σε αυτή την περίπτωση ονομάζονται master και worker. Κατά την εκκίνηση ενός Spark cluster ένα κόμβος παίρνει τον ρόλο του διαχειριστή cluster (τρέχοντας ένα από τα προγράμματα που αναφέραμε πιο πάνω) και αυτός αποτελεί τον master. Οπότε κατά συνέπεια, αναλαμβάνει την εποπτεία των πόρων που υφίστανται στο υπολογιστικό συγκρότημα. Στους υπόλοιπους κόμβους τρέχουν οι διεργασίες των workers και βρίσκονται σε συνεχή επικοινωνία με τον master ώστε να αναφέρουν τους διαθέσιμους υπολογιστικούς πόρους που υπάρχουν ή τα τυχόν προβλήματα που μπορεί να προκύψουν.

Όταν εκτελείται μία εφαρμογή με το Spark δημιουργείται μια διεργασία που ονομάζεται driver. Στον driver υπάρχει το αντικείμενο SparkContext που ενθυλακώνει όλες τις ρυθμίσεις και παραμέτρους της εργασίας προς εκτέλεση και συνδέεται με τον κόμβο «αρχηγό». Αυτός με τη σειρά του θα κάνει κατανομή πόρων μεταξύ των διαφόρων εφαρμογών και θα δημιουργήσει ένα είδος διεργασιών, στους κόμβους worker, που ονομάζονται executors. Οι executors αναλαμβάνουν το κομμάτι υπολογισμού που χρειάζεται για την πραγματοποίηση της εργασίας. Αφού λοιπόν δημιουργηθεί το μέσο με το οποίο θα εκτελεστεί η εργασία, ο driver μετατρέπει τον κώδικα σε tasks τα οποία θα ανατεθούν στους executors που είναι διαθέσιμοι. Πριν όμως φτάσουμε στην πραγματική δρομολόγηση αυτών των tasks ο driver καταστρώνει ένα λογικό πλάνο εκτέλεσης που έχει τη μορφή κατευθυνόμενου ακυκλικού γράφου (DAG) και συμβολίζει τις διαδικασίες που πρέπει να εκτελεστούν. Και αφού κάνει ορισμένες φυσικές βελτιστοποιήσεις στον τρόπο εκτέλεσης, δρομολογεί τα tasks στους executors για να ξεκινήσουν οι τελικοί υπολογισμοί. Τα αποτελέσματα της δουλείας μπορεί είτε να γυρίσουν πίσω στον driver είτε μπορεί να αποθηκευτούν, πχ RDD caching, για να χρησιμοποιηθούν από μελλοντικές δουλειές. Γενικά οι executors έχουν διάρκεια ζωής όσο και η εργασία πάνω στην οποία δουλεύουν, μόλις αυτή τελειώσει παύουν κι αυτοί να υπάρχουν. Αν για κάποιο λόγο αποτύχει ένα task κάποιου executor δεν σταματάει η συνολική εργασία που πραγματοποιείται, ούτε επαναλαμβάνονται όλα τα υπόλοιπα tasks. Χρειάζεται μόνο η επανεκκίνηση εκείνου που απέτυχε.



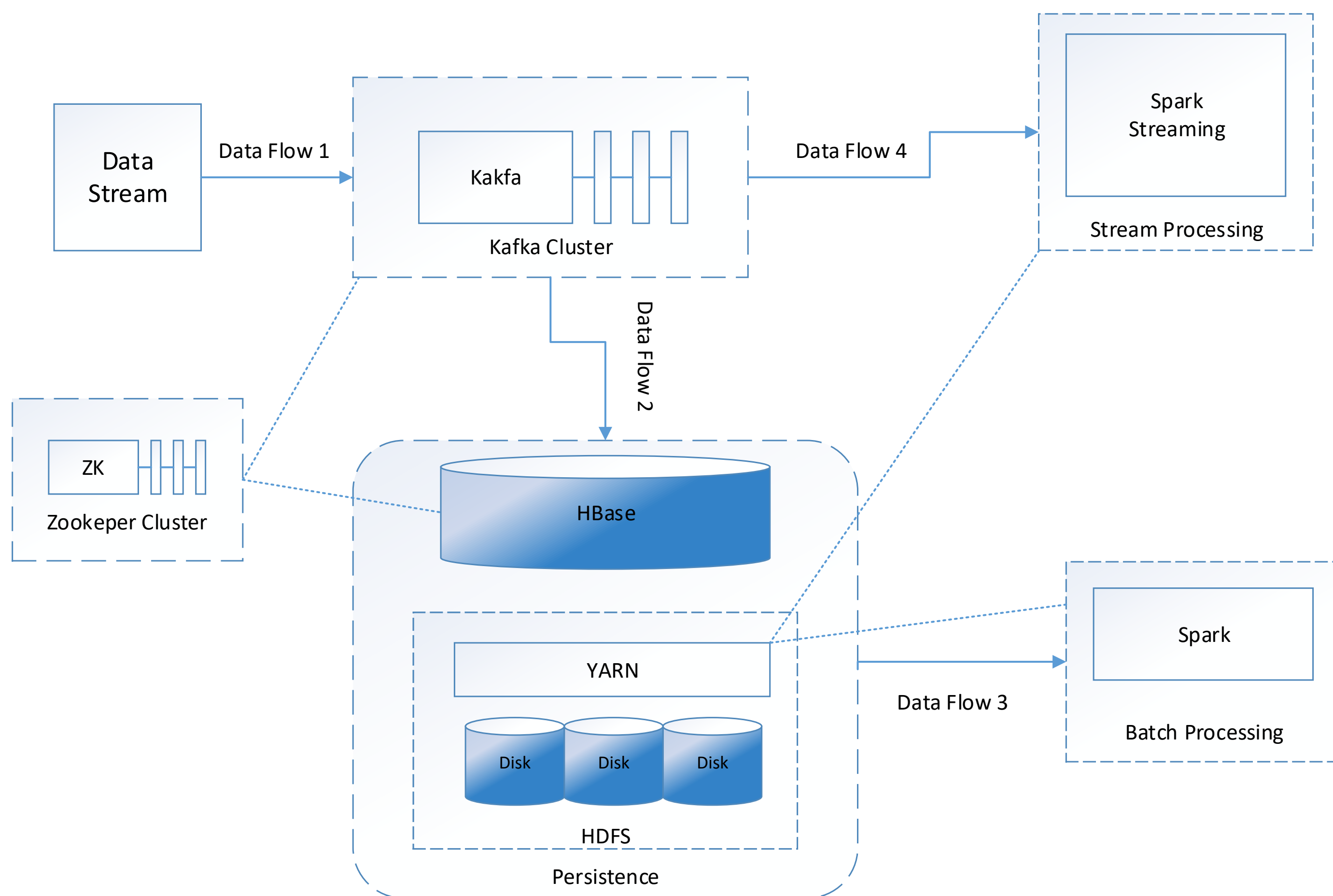

*Εικόνα 30 Αρχιτεκτονική Spark*

## 4.2 Αρχιτεκτονική

Αφού αναλύθηκαν όλες οι τεχνολογίες που θα χρησιμοποιηθούν στην αρχιτεκτονική που θα προταθεί ως λύση, ήρθε η ώρα να τις συνδέσουμε. Η τελική αρχιτεκτονική του πληροφοριακού συστήματος που δημιουργήθηκε παρουσιάζεται στην

Εικόνα 31 Αρχιτεκτονική Προτεινόμενης Λύσης.

**Data Stream**: Αποτελούν τα δεμένα εισόδου στο σύστημα μας. Αυτά εισέρχονται με μορφή συνεχόύε ροής δεδομένων και μπορεί να προέρχονται από διαφορετικές πηγές, είτε από το εξωτερικό περιβάλλον είτε από εξυπηρετητές που βρίσκονται στο ίδιο περιβάλλον. Παραδείγματα τέτοιων πηγών είναι οι μετρήσεις από αισθητήρες, ροή πληροφορίας από κοινωνικά δίκτυα καθώς και τα αρχεία καταγραφής ενός server. Το βασικό χαρακτηριστικό είναι ότι η παραγωγή τους γίνεται με γρήγορο ρυθμό




*Εικόνα 31 Αρχιτεκτονική Προτεινόμενης Λύσης*

**Data Flow 1:** Προφανώς η μοντελοποίηση των δεδομένων δεν θα είναι κάθε φορά ίδια, καθώς θα προέρχονται από διαφορετικές πηγές. Για την προώθηση της πληροφορίας στο επόμενο στάδιο θα φτιάχνονται Kafka producers που θα συνδέονται με τις ροές δεδομένων και θα τις στέλνουν στο cluster της Kafka με κωδικοποίηση Avro. Πέρα από την αποτελεσματικότητα στη συμπίεση, τη μεταφορά καθώς και στην εξαγωγή της πληροφορίας, το πρότυπο σειριοποίησης Avro αποτελεί συνδετικό κρίκο για τα κομμάτια της αρχιτεκτονικής μας. Τα υποσυστήματα δέχονται τον ίδιο τύπο πληροφορίας τον οποίον μπορούν μετατρέψουν στην αρχική του μορφή εύκολα και να διαχειριστούν όπως θέλουν. Επίσης, η δυναμικότητα των σχημάτων που προσφέρεται από το Avro είναι ιδανική για την εξέλιξη των ίδιων των δεδομένων που εισέρχονται στο σύστημα. Ακόμη και αν λαμβάνεται πληροφορία από την ίδια πηγή, με την πάροδο του χρόνου αυτή μπορεί να αλλάξει. Χρησιμοποιώντας το Avro, το σύστημα προσαρμόζεται εύκολα σε τέτοιες αλλαγές παραμένοντας λειτουργικό και ανταγωνιστικό.

**Kafka Cluster:** Η Apache Kafka αποτελεί τη ραχοκοκαλιά της αρχιτεκτονικής, καθώς οποιαδήποτε εισερχόμενη πληροφορία θα περνάει από αυτή. Θα χρησιμοποιηθεί σαν ένα κατανεμημένο και κλιμακώσιμο pipeline που θα χωρίζει την εισερχόμενη πληροφορία σε διαφορετικές ροές. Όταν περιγράψαμε το πρόβλημα, μιλήσαμε για την ανάγκη ύπαρξης συστημάτων που θα κάνουν και επεξεργασία τύπου batch καθώς και streaming. Οπότε μέσω της Kafka θα δημιουργούνται διαφορετικές κατηγορίες δεδομένων. Κάποιες θα προορίζονται για ανάλυση πραγματικού χρόνου και κάποιες για μεταφορά σε κάποιο αποθηκευτικό μέσω ώστε να πραγματοποιηθούν εκτενέστερες αναλύσεις. Για να το πετύχουμε αυτό θα δημιουργήσουμε διαφορετικά topics. Οι παραγωγοί θα αποστέλλουν τα δεδομένα στα διαφορετικά topic ανάλογα με την προέλευση τους. Για καλύτερη διαχείριση του όγκου δεδομένων η Kafka θα μπορούσε να τρέξει σε δικό της cluster.

**Data Flow 2:** Η ροή δεδομένων σε αυτό το σημείο προορίζεται για φύλαξη σε κάποιο αποθηκευτικό σύστημα. Χρησιμοποιώντας το Kafka Connect API μπορούμε να ορίσουμε σαν Kafka consumer άλλα συστήματα και να οδηγήσουμε την πληροφορία που προορίζεται για batch processing με εύκολο τρόπο σε κάποια κατανεμημένη αποθήκη. Αυτό που θα πρέπει τα υλοποιηθεί είναι το subscribe στα κατάλληλα topic του Kafka cluster έτσι ώστε η πληροφορία που εισέρχεται σε αυτά να μεταφέρεται για μακροπρόθεσμη αποθήκευση.

**Persistence:** Μία τέτοια αρχιτεκτονική χρειάζεται σίγουρα κάποιον αποθηκευτικό χώρο. Επειδή κάνουμε λόγο για Big Data και συνεχές ροές δεδομένων περιμένουμε αυτή η αποθήκη να είναι κλιμακώσιμη και να αποκρίνεται αποδοτικά και γρήγορα στη μεγάλη διακίνηση πληροφορίας. Γι' αυτούς τους λόγους έγινε η επιλογή του κατανεμημένου file system HDFS πάνω από το οποίο τρέχει η κατανεμημένη NoSQL βάση δεδομένων HBase. Η υλοποίηση του Kafka Connector για την HBase έχει υλοποιηθεί από μία πλατφόρμα που χρησιμοποιεί την Kafka στον πυρήνα της και ονομάζεται Confluent. Με αυτό τον connector δίνεται η δυνατότητα να ορίσουμε την HBase ως consumer σε Kafka topics. Έτσι, τα δεδομένα αποθηκεύονται σε κάποιο πίνακα της βάσης και κατά συνέπεια στο σύστημα αρχείων. Αυτό που πρέπει να τονιστεί είναι ότι ο συνδυασμός HBase-HDFS μας δίνει μεγάλη ελαστικότητα στον τρόπο αποθήκευσης και στη μορφή, ταχύτατη εισαγωγή και φύλαξη δεδομένων καθώς και χαμηλό κόστος ανά GB αποθηκευτικού χώρου, ικανοποιώντας έτσι πολλές από τις ανάγκες που δημιουργούν τα Big Data και τα streaming analytics.

**Data Flow 3:** Τα δεδομένα που έχουν αποθηκευτεί ζητούνται από το εργαλείο που θα εκτελεί τις διαδικασίες batch processing. Η ροή δεδομένων σε αυτό το σημείο γίνεται σε παρτίδες καθώς η πληροφορία έχει πάψει να είναι πραγματικού χρόνου εφόσον έχει αποθηκευτεί προηγουμένως.

**Batch Processing:** Τις αναλύσεις της μορφής batch θα τις κάνει το Spark. Το Spark έχει τη δυνατότητα να ενσωματωθεί σε πολλά συστήματα καθώς μπορεί να διαβάζει δεδομένα από πολλές πηγές. Σε αυτή την περίπτωση το Spark μπορεί να κάνει αναλύσεις βάσει των πινάκων της HBase, πού όπως έχει εξηγηθεί βρίσκεται η πληροφορία που προορίζεται για batch processing. Για την ακρίβεια, το Spark δημιουργεί RDD ή RDDs από τα δεδομένα της βάσης. Στη συνέχεια, μπορεί κανείς, και ανάλογα με την εφαρμογή, να χρησιμοποιήσει τη μεγάλη ποικιλία εργαλείων και βιβλιοθηκών που προσφέρονται μαζί με το Spark για να κάνει αναλύσεις και να εξάγει αποτελέσματα από αυτά τα δεδομένα. Παραδείγματα, αναλύσεων τέτοιων αποτελούν ερωτήματα τύπου SQL, machine learning, δημιουργία και επεξεργασία γράφων καθώς και υλοποίηση άλλων αλγορίθμων. Με αυτό τον τρόπο η αρχιτεκτονική είναι αρκετά ευέλικτη και κάνει «κλασσικές» αναλύσεις Big Data.

**Data Flow 4:** Η ροή δεδομένων σε αυτό το σημείο αποτελείται από την πληροφορία που προορίζεται για stream processing. Τα δεδομένα προέρχονται μόνο από τα topic εκείνα που είχαν δημιουργηθεί για το σκοπό αυτό.

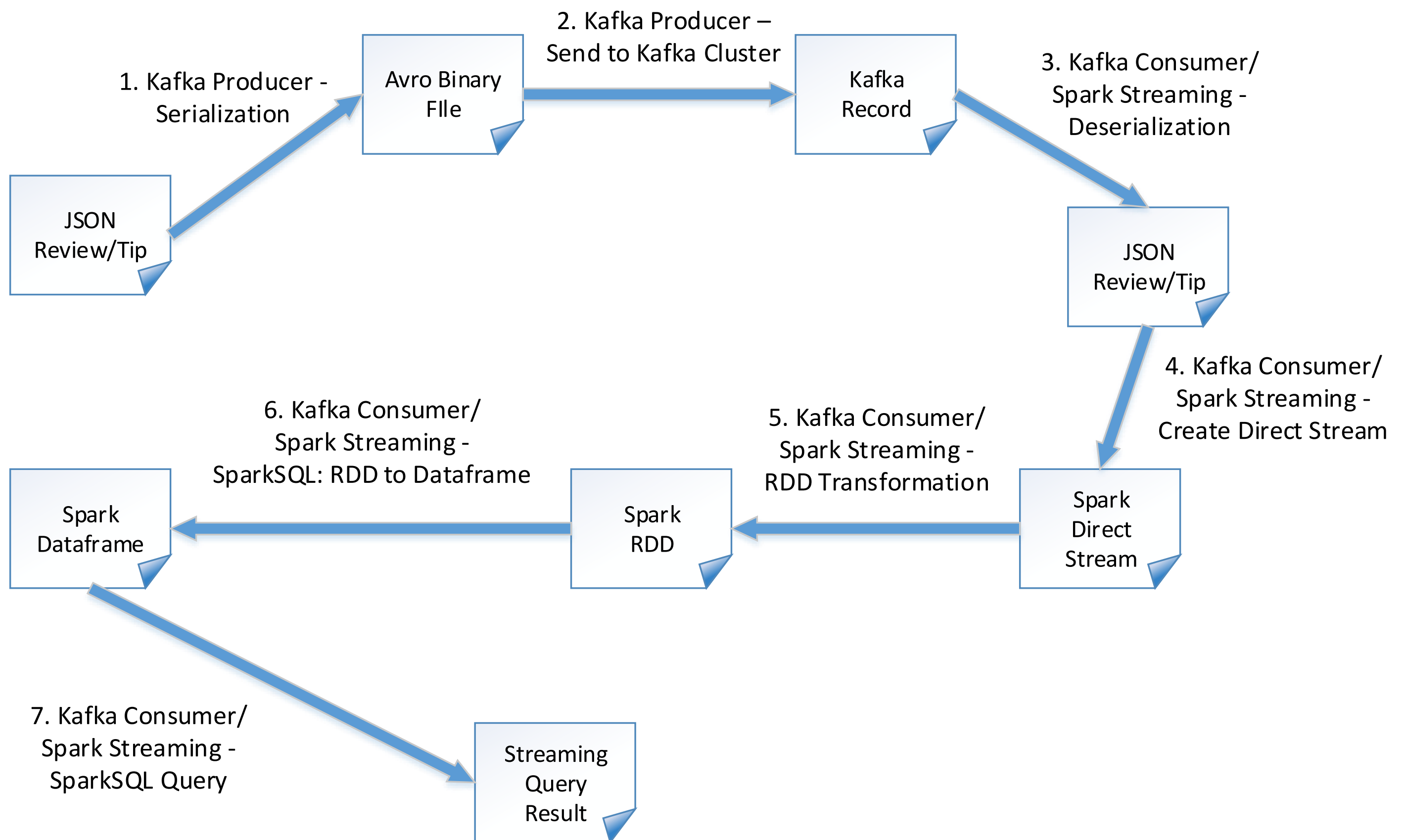
**Stream Processing:** Για εκτέλεση αναλύσεων πραγματικού χρόνου θα χρησιμοποιηθεί πάλι το Spark, αλλά πιο συγκεκριμένα ένα κομμάτι του που έχει φτιαχτεί γι’ αυτό το σκοπό, το Spark Streaming. Η ιδέα πίσω από αυτό είναι ότι αν μικρύνουμε αρκετά τις παρτίδες των δεδομένων θα έχουμε ουσιαστικά πολύ καλή προσέγγιση των αναλύσεων πραγματικού χρόνου. Έτσι τα δεδομένα πρέπει να μεταφερθούν από την Kafka στο Spark Steaming σε παρτίδες που θα προκύπτουν ανά μικρά χρονικά διαστήματα, του ενός λεπτού για παράδειγμα. Αυτές οι παρτίδες μετατρέπονται με τη σειρά τους σε ξεχωριστό RDD η κάθε μία και μπορούμε να χρησιμοποιήσουμε τα ίδια εργαλεία που είχαμε στη διάθεσή μας στο batch processing. Το Spark Streaming διαθέτει connector με την Kafka οπότε, μας δίνεται η δυνατότητα να φτιάξουμε ένα consumer που θα κάνει subscribe στα αντίστοιχα topic του

Kafka cluster και θα διαβάζει την πληροφορία που περιέχουν, αφού πρώτα μετατραπεί σε κατάλληλη μορφή, καθώς είχε σειριοποιηθεί με Avro.

**ZooKeeper Cluster:** Η Kafka και η HBase χρησιμοποιούνε τον Apache ZooKeeper για εργασίες που χρειάζονται κάποιο είδος ομοφωνίας και συντονισμό. Τέτοιες περιπτώσεις είναι η εκλογή αρχηγού ή η αποθήκευση κάποιων πληροφοριών για την κατάσταση κατανεμημένων δοσοληψιών. Για περισσότερη ανεξαρτησία και ανοχή σε σφάλματα ο ZooKeeper cluster θα μπορούσε να τρέξει σε δικά του dedicated μηχανήματα για να μην έρχεται σε αντιφατικές καταστάσεις με απαιτήσεις πόρων άλλων συστημάτων όπως η Kafka. Επίσης, έτσι αποφεύγονται πολλά προβλήματα που μπορεί να προκύψουν από σφάλματα υπηρεσιών που τρέχουν στα ίδια μηχανήματα.

**YARN:** Πέρα από το HDFS, το YARN θα χρησιμοποιηθεί και από το Spark, καθώς και το Spark Streaming, σαν διαχειριστής του cluster.

## 4.3 Κύκλος ζωής της πληροφορίας στο σύστημα

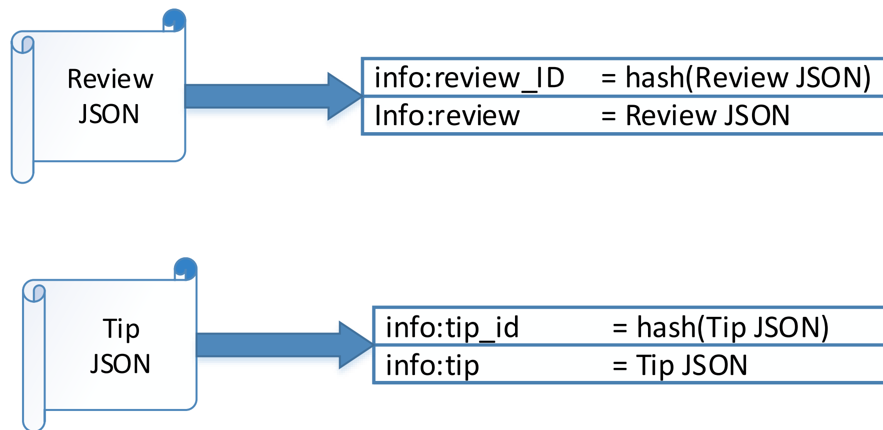



*Εικόνα 32 Κύκλος ζωής «πληροφορίας streaming»*

Στο παραπάνω σχήμα φαίνονται τα στάδια από τα οποία περνάει μία εγγραφή που δέχεται επεξεργασία σε real time.

Σε ότι αφορά την περίπτωση του batch processing, η πληροφορία αποθηκεύεται στην HBase με ένα συγκεκριμένο τρόπο. Αρχικά, δημιουργήσαμε τρείς πίνακες στη βάση με ονόματα «business», «reviews» και «tips» που αντιστοιχούν στα τρία διαφορετικά ήδη πληροφορίας που θα εισέρχονται στο σύστημα. Και η τρεις πίνακες για ευκολία θα έχουν μία οικογένεια στηλών που θα ονομάζεται «info». Για να εκμεταλλευτούμε τον παραλληλισμό του Apache Spark χωρίσαμε τους πίνακες αυτούς σε κομμάτια και το καθένα από αυτά θα βρίσκεται σε διαφορετικό Region Server, έτσι ώστε ο κάθε executor να λειτουργεί σε ένα διαφορετικό κομμάτι του πίνακα. Για τα business επιλέξαμε σαν κλειδί γραμμής το «business_id». Παρατηρώντας τις τιμές αυτού του πεδίου βλέπουμε ότι προέρχεται λογικά από κάποια συνάρτηση κατακερματισμού και οι χαρακτήρες που υπάρχουν είναι οι 1-9 , a-z, A-Z καθώς και -, _. Επειδή οι συναρτήσεις κατακερματισμού για πολλά δεδομένα δίνουν σχετικά ομοιόμορφη κατανομή μπορούμε να χωρίσουμε τον πίνακα ανάλογα με τον αρχικό

χαρακτήρα του «business_id». Για τα reviews όμως δεν μπορούμε να χρησιμοποιήσουμε σαν κλειδί γραμμής το ίδιο πεδίο καθώς μπορεί σε μία επιχείρηση να υπάρχουν πολλοί σχολιασμοί. Ένα πεδίο που θα μπορούσαμε να επιλέξουμε είναι το «review_id» αλλά δεν υπάρχει κάτι αντίστοιχο στα tips. Έτσι αυτό που κάναμε είναι να περάσουμε ολόκληρα τα review JSON και tip JSON από μία hash function και αυτό να αποτελέσει το row key για τον κάθε πίνακα αντίστοιχα.




*Εικόνα 33 Εγγραφές Review και Tip προς αποθήκευση στην HBase*

Η συνάρτηση κατακερματισμού που χρησιμοποιήθηκε είναι η SHA-1 με είσοδο όλο το αντίστοιχο JSON. Προφανώς, αν θελήσει κάποιος μπορεί να χρησιμοποιήσει άλλη. Στην περίπτωση της προηγούμενης σχηματικής αναπαράστασης το πεδίο info:review_id και info:tip_id θα ήταν sha1(Review JSON) και sha1(Tip JSON) αντίστοιχα. Επίσης, επιλέχθηκε και μία κοινή οικογένεια στηλών για απλότητα. Οι γραμμές του κάθε πίνακα της βάσης μας θα έχουν την εξής μορφή:

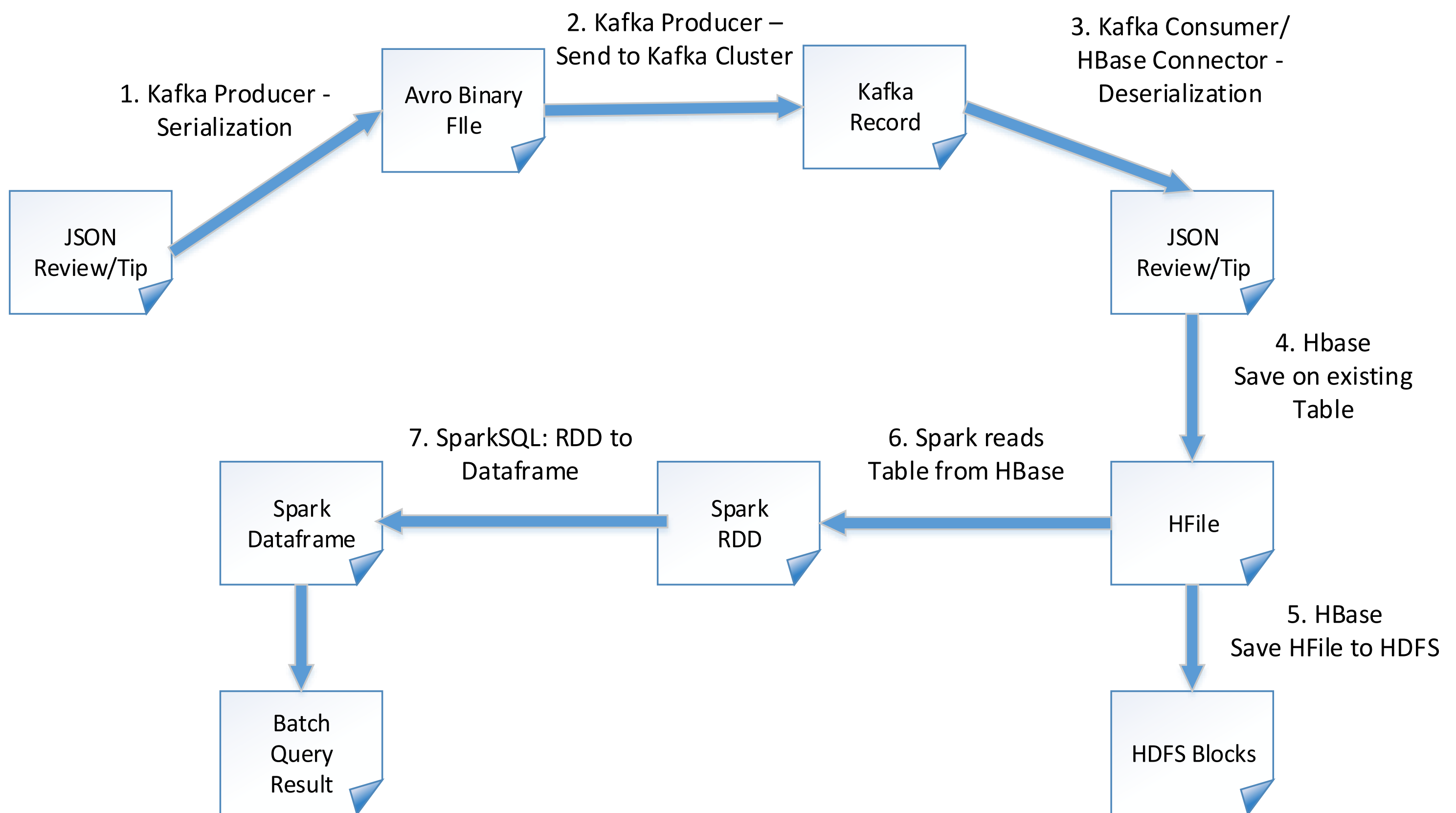
| business_id | Info | |
|---|---|---|
| | business_id | business_info |

| review_id | info | |
|---|---|---|
| | review_id | review |

| tip_id | info | |
|---|---|---|
| | tip_id | tip |

*Πίνακας 1 Τρόπος αποθήκευσης στοιχείων μίας επιχείρησης, ενός review και ενός tip*

Μετά την περιγραφή του τρόπου αποθήκευσης στην βάση δεδομένων ακολουθεί και ο κύκλος ζωής για τα δεδομένα που προορίζονται για επεξεργασία τύπου batch.




*Εικόνα 34 Κύκλος ζωής «πληροφορίας batch»*

# 5. Εργασίες προς εκτέλεση και Έναρξη συστήματος.

Ότι αναφέρεται σε αυτό το κεφάλαιο προϋποθέτει ότι το σύστημα και τα επιμέρους στοιχεία του έχουν εγκατασταθεί και όλα τρέχουν φυσιολογικά. Για περαιτέρω πληροφορίες σε αυτό το θέμα δείτε το Α. Διαδικασία εγκατάστασης λογισμικού.

Σε αυτό το κεφάλαιο θα γίνει λόγος για τις διεργασίες που καλείται το σύστημα να εκτελέσει καθώς και για τις εντολές που θα χρησιμοποιηθούν για να ξεκινήσουμε το σύστημα και να εκτελέσουμε τις εργασίες.

## 5.1 Εργασίες Batch και Stream Processing

Έχοντας ένα σχετικά μεγάλο σύνολο δεδομένων έγινε η προσπάθεια να σκεφτούμε εργασίες που το αποτέλεσμα τους μπορεί να ήταν σημαντικό σε κάποιο οργανισμό ή εταιρία. Θέλαμε δηλαδή να εξορύξουμε κάποια χρήσιμη πληροφορία. Επίσης, θέλουμε να πάρουμε μετρήσεις όταν το σύστημα εκτελεί εργασίες υψηλού υπολογιστικού φόρτου. Οπότε, στην περίπτωση batch processing σκεφτήκαμε δύο ερωτήματα.

Το πρώτο, το οποίο είναι πιο εύκολο υπολογιστικά, είναι να βρούμε σε κάθε πολιτεία τα ήδη καταστημάτων που υπάρχουν, δηλαδή, για παράδειγμα, πόσα εστιατόρια υπάρχουν στην πολιτεία Nevada των ΗΠΑ. Αυτή η πληροφορία είναι ιδιαίτερα χρήσιμη για κάποιον που θέλει να ανοίξει μία καινούρια επιχείρηση σε μία περιοχή . Προσφέρει πληροφορίες για τον ανταγωνισμό αλλά και για την προμήθεια-ζήτηση των υπηρεσιών που υπάρχουν εκεί. Αν μία περιοχή δεν έχει πολλά μαγαζιά με ρούχα, για παράδειγμα, ίσως θα ήταν καλό να ανοίξει μία τέτοιου είδους επιχείρηση. Φυσικά, θα πρέπει να γίνει περαιτέρω έρευνα αγοράς αλλά αυτό δεν είναι στα πλαίσια αυτής της διπλωματικής. Το ερώτημα αυτό δεν θεωρείται προγραμματιστικά πολύ δύσκολο καθώς όλη η πληροφορία που χρειαζόμαστε βρίσκεται σε ένα σημείο μόνο (στον πίνακα «business» της HBase). Από εδώ και πέρα σε αυτό θα αναφερόμαστε ως «query1»

Για το επόμενο ερώτημα, θεωρήσαμε χρήσιμο να ταξινομηθούν οι πόλεις ανάλογα με το πόσα καταστήματα έχουν περισσότερες θετικές κρητικές. Αν για παράδειγμα το Las Vegas έχει δύο επιχειρήσεις στις οποίες τα θετικά σχόλια ξεπερνάνε τα αρνητικά τότε αυτό δίνει 2 «Θετικούς Πόντους». Αντίστοιχα, μετράμε τις αρνητικές και τις ουδέτερες (ίδιος αριθμός θετικών και αρνητικών σχολίων) επιχειρήσεις. Εύκολα μπορεί να καταλάβει κανείς την σημαντικότητα αυτής της πληροφορίας καθώς οι καλές κριτικές των καταναλωτών μπορούν

να ανεβάσουν πολύ την φήμη ενός καταστήματος και να προσελκύσουν περισσότερη πελατεία. Αυτή η εργασία είναι υπολογιστικά δύσκολη καθώς η πληροφορία που χρειαζόμαστε βρίσκεται σε δύο πίνακες της βάσης («business» και «reviews») και απαιτείται μία πράξη τύπου SQL JOIN για να αντιστοιχηθούν οι επιχειρήσεις με τα επιμέρους reviews τους. Το JOIN δύο μεγάλων πινάκων δημιουργεί ένα ακόμα μεγαλύτερο από τον οποίο πρέπει να εξάγουμε και να ταξινομήσουμε την πληροφορία. Η αναφορά σε αυτό το ερώτημα παρακάτω θα γίνεται με την ονομασία «query2»

Τέλος, έχοντας δύο διαφορετικές ροές πληροφορίας, μία για τα reviews και μία για τα tips πρέπει να βρούμε δύο εργασίες streaming. Για τα reviews, θα μετριούνται ανά μικρά χρονικά διαστήματα της τάξεως του ενός λεπτού πως βαθμολογούν οι χρήστες, δηλαδή πόσες βαθμολογίες των πέντε, τεσσάρων κοκ αστεριών δόθηκαν σε αυτό το χρονικό διάστημα. Για τα tips θα μετράμε απλά τον αριθμό τους.

Μία ερώτηση που μπορεί να έχει κάποιος είναι πως θα κρίνουμε ποιες κριτικές είναι αρνητικές και ποιες θετικές. Για το λόγο αυτό το Εργαστήριο Distributed Knowledge and Media Systems του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής του ΕΜΠ μας έδωσε ένα πρόγραμμα που πραγματοποιεί Sentiment Analysis σε κείμενο. Έτσι πριν στείλουμε τα reviews και τα tips στο σύστημα θα δέχονται προ επεξεργασία ώστε να κρίνουμε αν είναι θετικά ή αρνητικά. Αυτή η διαδικασία θα δώσει μία μικρή καθυστέρηση στην αποστολή τους που κάνει την προσομοίωση να είναι πιο κοντά στην πραγματικότητα. Επίσης, είναι πολύ συχνό το φαινόμενο να γίνεται κάποια μορφή preprocessing στα δεδομένα την ώρα της δημιουργίας τους ώστε να χρησιμοποιηθεί η επιπλέον πληροφορία σε επόμενα στάδια.

## 5.2 Εργαλεία Μετρήσεων

Για την μέτρηση και την επίβλεψη των εργασιών batch processing καθώς και streaming του Spark χρησιμοποιήσαμε το UI που παρέχεται από την ίδια την εφαρμογή και ονομάζεται Spark History Server. Για την ενεργοποίηση αυτού, μπορεί κανείς να δει τον παρακάτω σύνδεσμο.

*https://spark.apache.org/docs/latest/monitoring.html*

Για την επίβλεψη διαφόρων μεγεθών της Apache Kafka την ώρα της λειτουργίας του συστήματος τα προγράμματα που χρησιμοποιήθηκαν ήταν το Kafka-monitor που έχει

αναπτυχθεί από την LinkedIn και το Kafka-manager της Yahoo. Οδηγίες εγκατάστασης και χρήσης βρίσκονται στους παρακάτω συνδέσμους.

https://github.com/linkedin/kafka-monitor

https://github.com/yahoo/kafka-manager

## 5.3 Προετοιμασία και έναρξη συστήματος

Εδώ θα γίνουν οι απαραίτητες ενέργειες για να ρυθμιστεί το σύστημα, και να τρέξουν οι ροές πληροφορίας, οι δουλείες streaming και οι εφαρμογές τύπου batch

### 5.3.1 Προετοιμασία Apache Kafka και Apache HBase

Στην αρχή πρέπει να φτιάξουμε τα κατάλληλα topic στον Kafka broker που τρέχει. Δημιουργήθηκαν topics με ονόματα «business», «reviews», «reviews_stream», «tips» και «tips_steam». Για την δημιουργία τους τρέξαμε την εντολή.

```
kafka-topics --create --zookeeper master:2181 --replication-factor
1 --partitions 1 --topic business
```

Αντίστοιχα για τα υπόλοιπα topic αλλάζουμε την παράμετρο ονόματος μετά το --topic.
Σε ότι αφορά την δημιουργία των πινάκων στην HBase αρχικά ανοίγουμε το shell με την εντολή

```
hbase shell
```

Και πληκτρολογούμε τις εντολές

```
create 'business','info', SPLITS => ['9','J','T','d','q']

create 'reviews','info', SPLITS => ['1','3','5','7','9','b','d']

create 'tips','info', SPLITS => ['3','7','b']
```

Προφανώς ο χωρισμός των πινάκων δεν είναι αναγκαστικός αλλά εξηγήθηκε η χρησιμότητα του στην ενότητα 4.3 Κύκλος ζωής της πληροφορίας στο σύστημα. Για την ένωση της Apache Kafka με την HBase βλ. Α.4 Εγκατάσταση Apache Kafka v0.10.2 – Πλατφόρμα Confluent v3.2.0

### 5.3.2 Εισαγωγή αρχικών δεδομένων στο σύστημα

Αρχικά πρέπει να βάλουμε στη βάση δεδομένων κάποιες εγγραφές ώστε οι εργασίες που θα εκτελεστούν να έχουν κάποιο νόημα. Για να μπούνε όλα τα business στην HBase τρέχουμε την εντολή

```
./Json_Producer_Consumer producer-business yelp_academic_dataset_business.json business
```

Το αρχείο Json_Producer_Consumer είναι ένα εκτελέσιμο πρόγραμμα που έχει γραφτεί σε Java και αποτελεί ουσιαστικά ένα producer της Kafka. Σαν πρώτη παράμετρο παίρνει τον τύπο του producer δηλαδή είτε «producer-business», είτε «producer-review» ή «producer-tip».

Για τον producer-business οι μεταβλητές που ακολουθούν είναι

`Input-file kafka-topic`

Για την εισαγωγή των tips και reviews οι εντολές είναι

```
./Json_Producer_Consumer producer-review yelp_academic_dataset_review_top.json reviews serious

./Json_Producer_Consumer producer-tip yelp_academic_dataset_tip_top.json tips serious
```

Αντίστοιχα μετά τον τύπο του producer που θέλουμε να τρέξει οι παράμετροι έχουν την εξής σημασία:

`Input-file kafka-topic Sentiment-analyser [delay (ms)]`

Sentiment-analyzer: Η ανάλυση συναισθημάτων των tip και των reviews γίνεται σε αυτό το στάδιο. Εδώ υπάρχουν δύο τιμές, το «serious», που είναι το κανονικό Sentiment Analysis που

περιγράψαμε, και το «dummy», που είναι μία πολύ απλή μορφή που είχε υλοποιηθεί για σκοπούς testing και debugging.

delay (ms): Μη υποχρεωτικό argument που βάζει καθυστέρηση χρόνου ανάμεσα στα records που στέλνονται στην Kafka.

Επίσης, τα αρχεία yelp_academic_dataset_review_top.json και yelp_academic_dataset_tip_top.json δεν αποτελούν όλα τα στοιχεία αλλά μόνο να πρώτα 300.000 tips και 1.500.000 reviews. Τα υπόλοιπα θα τα χρησιμοποιήσουμε για τη δημιουργία των ροών δεδομένων.

### 5.3.3 Έναρξη batch processing χωρίς Streaming

Ύστερα, θα τρέξουμε τις εργασίες batch που αναλύσαμε στην ενότητα 5.1 Εργασίες Batch και Stream Processing. Για τα συγκεκριμένα ερωτήματα έχει αναπτυχθεί ένα εκτελέσιμο jar αρχείο σε Scala με όνομα *hbasespark_2.11-1.0.jar*. Για να τρέξουμε την πρώτη εργασία εκτελούμε την παρακάτω εντολή στον κατάλογο που βρίσκεται αυτό το αρχείο.

```
spark-submit hbasespark_2.11-1.0.jar query1
```

Και αντίστοιχα για την δεύτερη όταν τελειώσει η πρώτη

```
spark-submit hbasespark_2.11-1.0.jar query2
```

### 5.3.4 Έναρξη Stream Processing Χωρίς Batch

Συνεχίζοντας, αφού έχουν τελειώσει και οι δύο batch εργασίες, θα δημιουργήσουμε τις δύο ροές πληροφορίας στέλνοντας τα υπόλοιπα reviews και tips. Μετά θα τρέξουμε την εφαρμογή του Spark Streaming που θα κάνει consume αυτές τις ροές. Κάθε λεπτό θα εκτελεί τις εργασίες που έχουμε αναφέρει για τον αριθμό των tips και reviews που έχουν ληφθεί σε αυτό το χρονικό παράθυρο.

Επειδή θέλουμε οι ροές των δεομένων να τρέχουνε στο παρασκήνιο θα τρέξουμε τις εντολές

```
nohup ./Json_Producer_Consumer producer-review yelp_academic_dataset_review_bottom.json reviews serious > reviews_stream.out &

nohup ./Json_Producer_Consumer producer-tip yelp_academic_dataset_tip_bottom.json tips serious > tips_stream.out &
```

Στην συνέχεια θέλουμε να ξεκινήσουμε την εφαρμογή που αναπτύξαμε και κάνει consume αυτές τις ροές και έχει το όνομα *SparkStream-assembly-1.0.jar.* Για να την τρέξουμε και αυτή στο παρασκήνιο θα χρησιμοποιήσουμε την εντολή

```
nohup spark-submit --executor-cores 2 SparkStream-assembly-1.0.jar > stream.spark.out &
```

Στην εντολή αυτή είπαμε στους executors να έχουν δύο πυρήνες διότι ένας χρειάζεται για να τρέχει ο receiver του αντίστοιχου stream και ο δεύτερος για την επεξεργασία των δεδομένων.

### 5.3.5 Λειτουργία και των δύο τεχνικών

Για να ελέγξουμε την πλατφόρμα μας σε αυτό που έχουμε ονομάσει «πλήρη λειτουργία», δηλαδή batch και stream processing μαζί, αφού έχουμε κάνει τα βήματα 5.3.1, 5.3.2, 5.3.3, και 5.3.4 με αυτή τη σειρά, τρέχουμε πάλι την εντολή

```
spark-submit hbasespark_2.11-1.0.jar query1
```

Και αντίστοιχα για την δεύτερη όταν τελειώσει η πρώτη

```
spark-submit hbasespark_2.11-1.0.jar query2
```

# 6. Μετρήσεις

Για να αξιολογήσουμε το σύστημα που έχουμε δημιουργήσει θα λάβουμε μετρήσεις από διαφορετικά υποσυστήματα του. Αυτά είναι η Apache Kafka, οι διαθέσιμοι πόροι που υπάρχουν στο cluster αλλά και οι ίδιες διεργασίες batch και streaming που εκτελούνται στο σύστημα.

Προσπαθούμε να αξιολογήσουμε το σύστημα όσο το δυνατόν εκτενέστερα και για αυτό θα το εξετάσουμε από διάφορες σκοπιές, όταν δουλεύουν κάποια από τα επιμέρους κομμάτια του καθώς και όταν δουλεύουν όλα ταυτόχρονα. Για το λόγο αυτό θα ξεχωρίσουμε τις εξής περιπτώσεις:

- Όταν δουλεύουν μόνο οι διαδικασίες batch (ξεχωριστά η μία με την άλλη) και διαβάζουν δεδομένα από την βάση. Η Kafka και το Spark Streaming είναι ανενεργά προς το παρόν. *(Στάδιο 1)*
- Όταν δουλεύει μόνο το κομμάτι επεξεργασίας ροής δεδομένων δηλαδή οι ροές δεδομένων εισόδου, η Kafka, που στέλνει τα δεδομένα στην βάση και στο Spark Streaming καθώς και το ίδιο το Spark Streaming που τελικά επεξεργάζεται αυτές τις ροές. Σε αυτό το σενάριο το σύστημα θα τρέχει και να επεξεργάζεται αυτές τις ροές για κάποια ώρα (στη συγκεκριμένη περίπτωση το αφήσαμε 50 λεπτά) ώστε να πάρουμε αντιπροσωπευτικές μετρήσεις όταν η ροή δεδομένων είναι σταθερή. *(Στάδιο 2)*
- Όταν ενώ τρέχει το κομμάτι του Stream Processing (Στάδιο 2) τρέχουμε την «εύκολη» batch εργασία a.k.a. query1. *(Στάδιο 3)*
- Όταν ενώ τρέχει το κομμάτι του Stream Processing (Στάδιο 2) τρέχουμε τη «δύσκολη» batch εργασία. a.k.a. query2. *(Στάδιο 4)*

Αρχικά θα ελέγξουμε τις διεργασίες batch processing στα στάδια 1, 3 και 4 καθώς στο 2 δεν υφίστανται.

Query1 & Query2 Alone

| App ID | App Name | Started | Completed | Duration | Spark User | Last Updated |
|---|---|---|---|---|---|---|
| application_1504293660326_0010 | SparkQuery | 2017-09-02 19:02:23 | 2017-09-02 19:09:08 | 6.7 min | root | 2017-09-02 19:09:19 |
| application_1504293660326_0009 | SparkQuery | 2017-09-02 19:00:06 | 2017-09-02 19:01:33 | 1.4 min | root | 2017-09-02 19:01:33 |

Query1 Along with Stream Processing

| App ID | App Name | Started | Completed | Duration | Spark User | Last Updated |
|---|---|---|---|---|---|---|
| application_1504293660326_0014 | SparkQuery | 2017-09-02 13:09:15 | 2017-09-02 13:10:44 | 1.5 min | root | 2017-09-02 13:10:44 |

Query2 Along with Stream Processing

| App ID | App Name | Started | Completed | Duration | Spark User | Last Updated |
|---|---|---|---|---|---|---|
| application_1504293660326_0015 | SparkQuery | 2017-09-02 13:21:44 | 2017-09-02 13:30:14 | 8.5 min | root | 2017-09-02 13:30:14 |

*Εικόνα 35 Batch Processing Jobs Times*

Αμέσως μπορούμε να δούμε το χρόνο που πήρε η εκτέλεση αυτών των εργασιών. Στο πρώτο στάδιο η πρώτη όντας η ευκολότερη διήρκησε 1.4 min και η επόμενη 6.7 min καθώς είναι υπολογιστικά πιο δύσκολη. Οπότε όπως προβλέψαμε μία εργασία τύπου SQL JOIN θα προσομοιώσει σχετικά καλά αυτό που επιδιώξαμε, να ασκήσουμε δηλαδή πίεση στους υπολογιστικούς πόρους του συστήματος ώστε να γίνει έλεγχος για το πως ανταποκρίνεται η πλατφόρμα σε αυτές τις καταστάσεις.

Για το Στάδιο 3 ο χρόνος για την εύκολη εργασία επηρεάστηκε ελάχιστα από 1.4 min σε 1.5 min που δεν αποτελεί ουσιαστική διαφορά. Αντίθετα, στο στάδιο 4 που αφορά το query2 βλέπουμε ότι ο χρόνος αυξήθηκε κατά 2 λεπτά περίπου που δείχνει ότι το σύστημα είχε μεγάλο υπολογιστικό φόρτο.

Στην συνέχεια θα εξετάσουμε τους executors αυτών των εργασιών για να δούμε πως εκτελέστηκαν οι εργασίες στα διαφορετικά στάδια.

Query1 Executors Alone

| Executor ID | Address | Status | Cores | Complete Tasks | Total Tasks | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|
| driver | 147.102.19.1:34437 | Active | 0 | 0 | 0 | 0.0 B | 0.0 B |
| 1 | tweetcluster88:44208 | Active | 1 | 317 | 317 | 555.3 KB | 1.3 MB |
| 2 | tweetcluster87:43729 | Active | 1 | 96 | 96 | 886.1 KB | 552.5 KB |

Query1 Executors With Stream Processing

| Executor ID | Address | Status | Cores | Complete Tasks | Total Tasks | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|
| driver | 147.102.19.1:49565 | Active | 0 | 0 | 0 | 0.0 B | 0.0 B |
| 1 | tweetcluster87:40671 | Active | 1 | 92 | 92 | 890.9 KB | 529.3 KB |
| 2 | tweetcluster88:33619 | Active | 1 | 321 | 321 | 570.6 KB | 1.3 MB |

*Εικόνα 36 Executors για Query1 σε διαφορετικά στάδια υπολογιστικού φόρτου*

Εξετάζοντας τους executors της πρώτης εργασίας φαίνεται ότι ο τρόπος εκτέλεσης δεν επηρεάστηκε καθόλου είτε η διαδικασία τρέχει μόνη της είτε μαζί με εφαρμογές streaming. Παρατηρείται ότι υπάρχει παραλληλισμός καθώς και οι δυο workers δουλεύανε ταυτόχρονα. Γίνεται read καθώς και write, αφού έχουμε διάβασμα από την HBase και γράψιμο του τελικού αποτελέσματος σε αρχείο.

Οι ίδιες μετρήσεις για την δεύτερη batch εργασία δίνουν τα παρακάτω αποτελέσματα.

Query2 Executors Alone

| Executor ID | Address | Status | Cores | Complete Tasks | Total Tasks | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|
| driver | 147.102.19.1:50514 | Active | 0 | 0 | 0 | 0.0 B | 0.0 B |
| 1 | tweetcluster88:38641 | Active | 1 | 821 | 821 | 7 MB | 27.2 MB |
| 2 | tweetcluster87:43395 | Active | 1 | 8 | 8 | 0.0 B | 7 MB |

Query2 Executors With Stream Processing

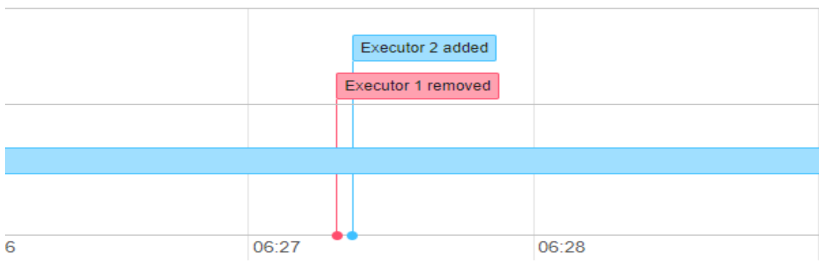
| Executor ID | Address | Status | Cores | Complete Tasks | Total Tasks | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|
| driver | 147.102.19.1:56626 | Active | 0 | 0 | 0 | 0.0 B | 0.0 B |
| 1 | tweetcluster88:36379 | Dead | 1 | 29 | 29 | 0.0 B | 27.3 MB |
| 2 | tweetcluster88:54941 | Active | 1 | 817 | 817 | 0.0 B | 34.5 MB |

*Εικόνα 37 Executors για Query1 σε διαφορετικά στάδια υπολογιστικού φόρτου*

Αρχικά, μπορούμε να δούμε διαφορές με την πρώτη διεργασία που εξετάσαμε προηγουμένως. Τα μεγέθη των read και write διαφέρουν καθώς το δεύτερο πρόγραμμα διάβασε πολύ περισσότερα δεδομένα. Οι περισσότερες πράξεις I/O καθυστερούν σημαντικά τους πόρους του συστήματος καθώς ο επεξεργαστής και η κύρια μνήμη πρέπει να περιμένουν τον αργό σκληρό δίσκο.

Όταν συγκρίνουμε την ίδια δουλειά στα δύο διαφορετικά στάδια αυτό που βλέπουμε είναι ότι στην περίπτωση όπου έτρεχε μόνη της η διαδικασία εκτέλεσης ήταν ομαλή αλλά στην περίπτωση της ταυτόχρονης εκτέλεσης (μαζί με της διεργασίες του Spark Streaming) ένας executor πέθανε και λογικά ο άλλος έπρεπε να συνεχίζει το έργο του γιατί τα 34.5 MB που γράφτηκαν ισούται με το άθροισμα των MB που έγραψαν οι executors όταν η λειτουργία της διεργασίας διεξήχθη ομαλά στο πρώτο στάδιο.

Ψάχνοντας παραπάνω για το πότε σταμάτησε να λειτουργεί ο executor που αναγράφεται, βρέθηκε το εξής χρονοδιάγραμμα εκτέλεσης:




*Εικόνα 38 Κομμάτι χρονοδιαγράμματος εκτέλεσης του query2 όταν γίνεται και stream processing ταυτόχρονα*

Παρατηρώντας αυτό, καταλαβαίνει κανείς ότι η εργασία αυτή, σε αυτό το σενάριο εκτέλεσης, δεν είχε παραλληλισμό. Λειτουργούσε ένας μόνο worker που πέθανε και τον αντικατέστησε άλλος αφού έλαβε και την δουλειά που είχε να κάνει.

Για πληρότητα θα αναφερθούμε και στα αποτελέσματα της εκτέλεσης αυτών των διεργασιών τα οποία γράφονται σε μορφή csv και αποθηκεύονται στο HDFS και έχουν αυτή τη μορφή.

| state | category | categoryCount |
|---|---|---|
| AZ | Restaurants | 9754 |
| AZ | Shopping | 7219 |
| BW | Restaurants | 1650 |
| BW | Food | 607 |
| EDH | Shopping | 768 |
| EDH | Food | 766 |

*Πίνακας 2 Δείγμα αποτελέσματος query1*

| city | PositiveCompanies | NegativeCompanies | NeutralCompanies |
| --- | --- | --- | --- |
| Las Vegas | 7456 | 1330 | 297 |
| Toronto | 5636 | 847 | 248 |
| Phoenix | 4845 | 979 | 209 |
| Scottsdale | 2570 | 339 | 94 |
| Charlotte | 2437 | 415 | 109 |

*Πίνακας 3 Δείγμα αποτελέσματος query2*

Στην συνέχεια θα προχωρήσουμε στην εποπτεία των πόρων του cluster στα διάφορα στάδια που περιγράψαμε.

Query1 & Query2 Alone (Same Metrics)

| Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total |
| --- | --- | --- | --- | --- |
| 1 | 16 | 3 | 3.63 GB | 8 GB |

| Memory Reserved | VCores Used | VCores Total | VCores Reserved | Active Nodes |
| --- | --- | --- | --- | --- |
| 0 B | 3 | 8 | 0 | 2 |

Streaming Jobs Only

| Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total |
| --- | --- | --- | --- | --- |
| 1 | | 3 | 3.63 GB | 8 GB |

| Memory Reserved | VCores Used | VCores Total | VCores Reserved | Active Nodes |
| --- | --- | --- | --- | --- |
| 0 B | 3 | 8 | 0 | 2 |

Query1 + Streaming Jobs

| Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total |
| --- | --- | --- | --- | --- |
| 2 | 12 | 6 | 7.25 GB | 8 GB |

| Memory Reserved | VCores Used | VCores Total | VCores Reserved | Active Nodes |
| --- | --- | --- | --- | --- |
| 0 B | 6 | 8 | 0 | 2 |

Query2 + Streaming Jobs

| Apps Running | Apps Completed | Containers Running | Memory Used | Memory Total |
| --- | --- | --- | --- | --- |
| 2 | 13 | 5 | 5.88 GB | 6.63 GB |

| Memory Reserved | VCores Used | VCores Total | VCores Reserved | Active Nodes |
| --- | --- | --- | --- | --- |
| 1.38 GB | 5 | 7 | 1 | 2 |

*Εικόνα 39 Οι πόροι του Cluster σε διάφορα στάδια*

Βλέπουμε ότι γενικά το χρησιμοποιείται το 35-40% των πόρων στις καταστάσεις που τρέχουν είτε οι batch διεργασίες είτε το stream processing μόνα τους. Υπάρχει ακόμη χώρος για να τρέξουμε κι άλλες εφαρμογές σε εκείνα τα στάδια.

Όταν εκτελούμε την πρώτη εργασία batch μαζί με την επεξεργασία των streams βλέπουμε ότι η πόροι του cluster έχουν μειωθεί σημαντικά. Βέβαια, οι αριθμοί που προκύπτουν είναι απολύτως λογικοί καθώς αν προστεθούν οι τιμές του πίνακα «Query1 Alone» και «Streaming Jobs Only» έχουμε τις αναγραφόμενες τιμές. Η κύρια μνήμη φτάνει ίσα ίσα για να εκτελεστούν και οι δύο δουλείες ταυτόχρονα. Αυτό βέβαια φαίνεται να μην επηρέασε τη batch εργασία καθόλου όπως είδαμε. Θα δούμε στην πορεία πως επηρεάστηκε το Spark Streaming.

Αντίθετα, στην τελευταία περίπτωση όπου είχαμε την επιπλοκή με τον νεκρό worker βλέπουμε ότι η εφαρμογή έκανε reserve έναν πυρήνα και ένα ποσοστό μνήμης που σημαίνει ότι για κάποια δουλειά, τη χρονική στιγμή που ζήτησε πόρους, το cluster δεν μπορούσε να τους παρέχει και τους κράτησε για να τους δώσει αργότερα όταν θα ελευθερωθούν κι άλλοι. Η εργασία αυτή όπως αναλύσαμε και είδαμε, είναι περισσότερο memory και CPU intensive από την προηγούμενη που ήταν υπολογιστικά ευκολότερη. Είναι πολύ πιθανό αυτή η παύση λειτουργίας του executor που μελετήσαμε προηγουμένως να οφείλεται σε αυτή την έλλειψη μνήμης.

Το κομμάτι συνεχούς επεξεργασίας των ροών δεδομένων είναι πολύ σημαντικό και γι' αυτό πρέπει να παρθούν οι κατάλληλες μετρήσεις ώστε να δούμε πως ανταποκρίνεται στα στάδια στα οποία είναι ενεργό.

Αρχικά, θα δούμε τους executors που είναι υπεύθυνοι για την επεξεργασία των δεδομένων των ροών.

Για την εξέλιξη των εργασιών του Spark Streaming δημιουργήθηκαν δύο worker που λειτουργούσαν παράλληλα όπως φαίνεται παρακάτω

| Executor ID | Address | Status | Cores | Complete Tasks | Total Tasks | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|
| driver | 147.102.19.1:54066 | Active | 0 | 0 | 0 | 0.0 B | 0.0 B | 0.0 B |
| 1 | tweetcluster87:43859 | Active | 1 | 17476 | 17476 | 18.6 MB | 0.0 B | 64.3 KB |
| 2 | tweetcluster88:56627 | Active | 1 | 22916 | 22916 | 14.2 MB | 67 B | 5.8 KB |

*Εικόνα 40 Executors στις για τις εργασίες streaming*

Οι διαδικασίες που αφορούν το streaming πρέπει να είναι συνεχώς ανοιχτές ώστε να λαμβάνουν την πληροφορία από τις ροές δεδομένων. Αν αφήναμε τα προγράμματα να τρέξουν κι άλλο θα βλέπαμε μεγαλύτερα νούμερα. Ο παραλληλισμός που προσφέρει το Spark, δίνει την δυνατότητα να λαμβάνονται και να επεξεργάζονται περισσότερες από μία ροή πληροφορίας ταυτόχρονα.

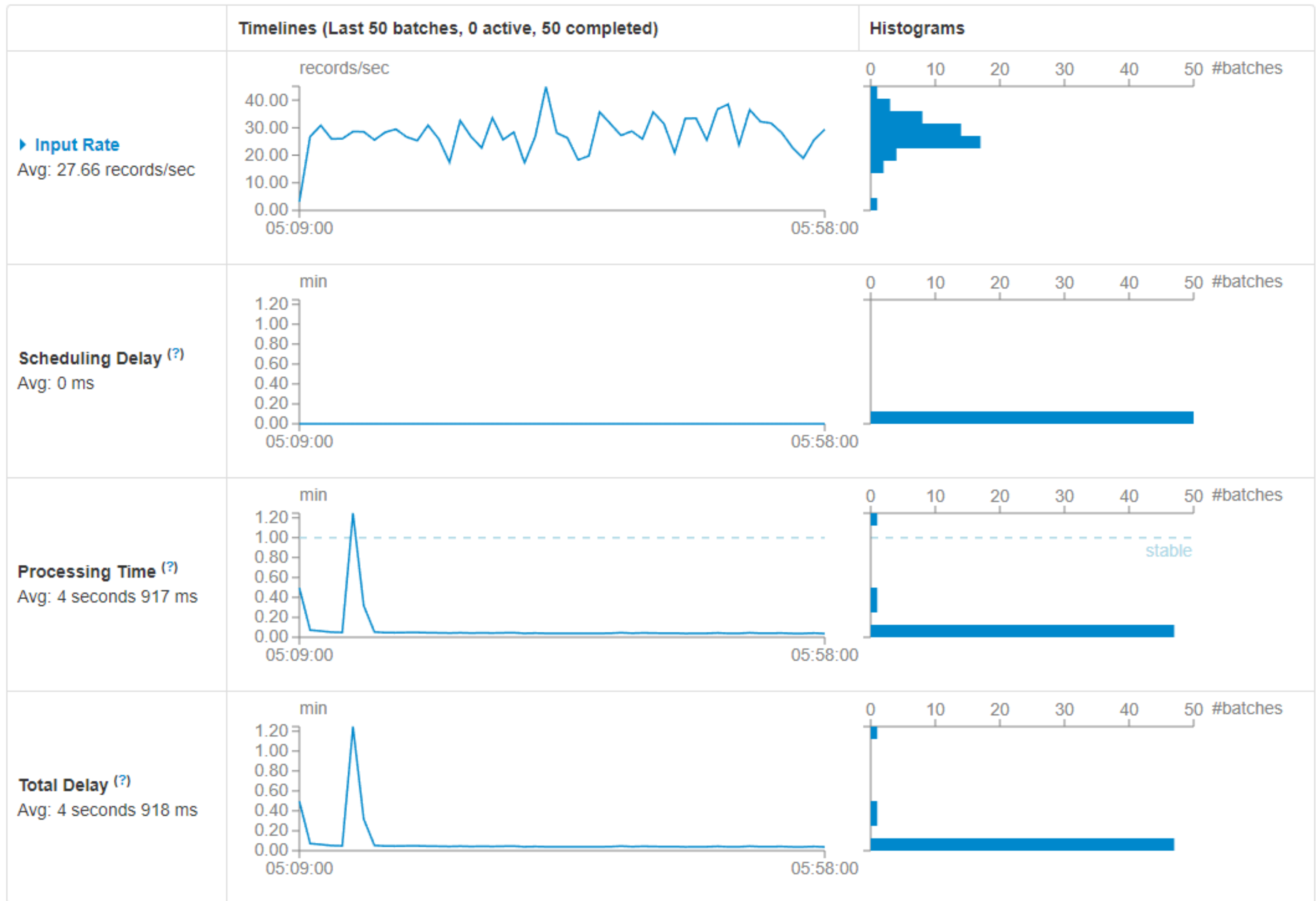
Η εργασίες αυτές γράφουν τα αποτελέσματα τους για την κάθε παρτίδα σε μορφή csv και αποθηκεύουν το αρχείο στο HDFS. Μια τυχαία παρτίδα review και tip έχουν την παρακάτω μορφή:

| stars | starsCount |
| --- | --- |
| 1 | 34 |
| 2 | 26 |
| 3 | 19 |
| 4 | 75 |
| 5 | 93 |

| TipCount |
| --- |
| 1757 |

*Πίνακας 4 Πίνακες αποτελεσμάτων δουλείας streaming*

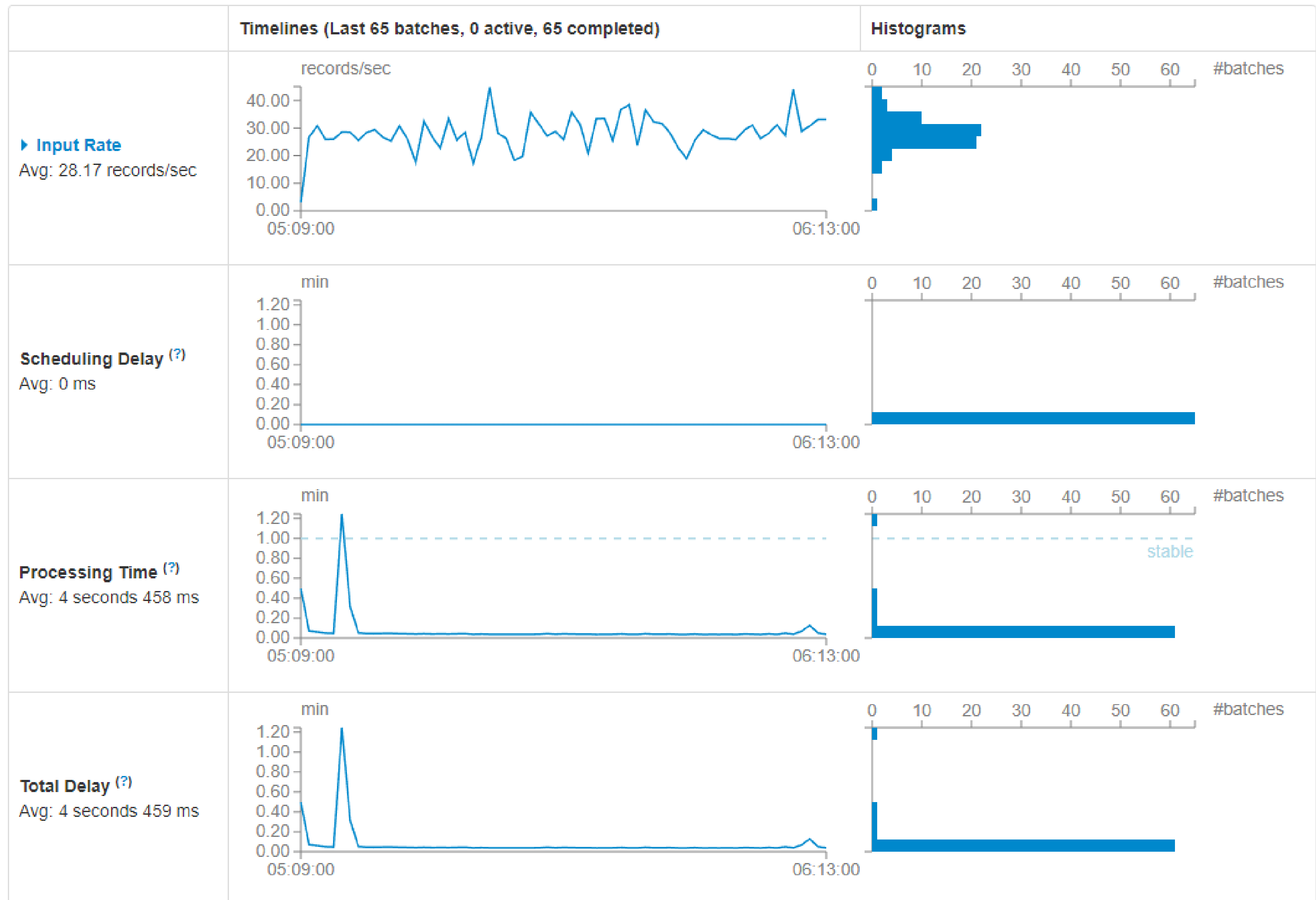
Οι γενικές μετρήσεις που αφορούν το Spark Streaming είναι οι παρακάτω:




*Εικόνα 41 Γενικά στοιχεία Spark Streaming όταν έχουμε μόνο streaming job*

Αυτό που παρατηρείται είναι η σχετικά μεγάλη διακύμανση στις πόσες εγγραφές διαβάζονται και αυτό οφείλεται κυρίως στο ότι για κάθε διαφορετικό review και tip ο Sentiment Analyzer θα κάνει διαφορετικό χρόνο για να βγάλει συμπέρασμα. Επίσης, οι οποιεσδήποτε καθυστερήσεις στην εκπόνηση των εργασιών οφείλονται μόνο στον χρόνο επεξεργασίας για την εύρεση του αποτελέσματος καθώς ο χρονοπρογραμματισμός είναι FIFO by default. Η παρατήρηση που πρέπει να γίνει αφορά στη σχετικά μεγάλη καθυστέρηση στην έναρξη της εφαρμογής. Η δημιουργία των workers, η διαπραγμάτευση για τους υπολογιστικούς πόρους που θα χρειαστούν, η αποστολή του απαραίτητου κώδικα στους executors και όλες οι άλλες διαδικασίες που γίνονται κατά την έναρξη της εφαρμογής δημιουργούν επιπλέον

καθυστέρηση της τάξεως των 40s η οποία μετά εξαφανίζεται. Αυτή η υψηλή τιμή στο διάγραμμα Total Delay που είναι 1.2 min δεν μπορούσε να προσδιοριστεί από που προέκυψε και όντας ασυνήθιστα υψηλή σε σχέση με τις άλλες, θα αγνοηθεί καθώς αποτελεί 1 τιμή από τις 50 που πάρθηκαν (πάνω πάνω φαίνεται ότι έχουν επεξεργαστεί 50 παρτίδες).




Εικόνα 42 Γενικά στοιχεία Spark Streaming μετά το τέλος του query1

Η μόνη επίδραση της εκτέλεσης της «εύκολης» batch εργασίας βρίσκεται στο processing time. Αυτό που προκάλεσε ήταν την αύξηση στην καθυστέρηση της επεξεργασίας της παρτίδας, καθώς αυτό συνέβαινε την ίδια χρονική στιγμή με τη batch εργασία. Αυτό είναι λογικό καθώς όπως είδαμε, οι πόροι του cluster και ιδιαίτερα η κύρια μνήμη είχαν σχεδόν

εξαντληθεί. Μέχρι τότε, οι καθυστερήσεις επεξεργασίας ήταν 2-3s και εκείνη τη χρονική στιγμή πήγε στα 13s. Μετά βέβαια το σύστημα επανήλθε στην κατάσταση ισορροπίας που βρισκόταν. Πέρα από αυτό, η συμπεριφορά της εφαρμογής δεν άλλαξε.

*Εικόνα 43 Γενικά στοιχεία Spark Streaming μετά το τέλος του query2*

Και σε αυτή την περίπτωση το μόνο που αλλάζει είναι η καθυστέρηση στην επεξεργασία των παρτίδων. Αυτή τη φορά η αύξηση της καθυστέρησης είναι μικρότερη από την προηγούμενη περίπτωση όμως διαρκεί περισσότερο καθώς η εργασία που εκτελείται χρειάζεται περισσότερο χρόνο για να τελειώσει. Από καθυστερήσεις της τάξεως των 2-3s μεταπηδήσαμε στα 5-6s. Φυσικά, μετά το τέλος του batch job το σύστημα επανήλθε αμέσως στην προηγούμενη κατάσταση του.

Από ότι φαίνεται, στις ταυτόχρονες εκτελέσεις, το κέντρο επεξεργασίας του συστήματος μας, δηλαδή ο spark cluster, χρειαζόταν περισσότερους πόρους. Αυτό, που πρέπει να τονιστεί είναι ότι παρόλα αυτά το Spark Streaming συνέχιζε να καταναλώνει records με τον ρυθμό που είχε όταν δεν είχαμε ταυτόχρονη εκτέλεση batch και stream processing, που σημαίνει ότι δεν είχαμε καμία επίπτωση στη διακίνηση της πληροφορίας.

Τέλος, για να ολοκληρώσουμε τις μετρήσεις του συστήματος θα λάβουμε μετρήσεις από την Kafka για τα στάδια στα οποία είναι ενεργή, δηλαδή όπου πραγματοποιούνται εργασίες τύπου streaming. Πρέπει να αναφερθεί ότι η Kafka αποτελεί τη ραχοκοκαλιά της αρχιτεκτονικής που έχουμε υλοποιήσει και γι' αυτό το λόγο ο broker τρέχει σε ξεχωριστό μηχάνημα, και όχι σε αυτά που βρίσκονται οι workers του Spark.

Για τον broker της Kafka έχουμε συνολικά:

Streaming Jobs Only

Combined Metrics

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 53.44 | 60.92 | 62.53 | 58.21 |
| Bytes in /sec | 19k | 21k | 22k | 21k |
| Bytes out /sec | 25k | 26k | 32k | 30k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

Right After Completion of Query1 + Streaming Jobs

Combined Metrics

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 54.40 | 61.26 | 62.32 | 59.14 |
| Bytes in /sec | 20k | 21k | 22k | 21k |
| Bytes out /sec | 27k | 36k | 33k | 31k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

Right After Completion of Query2 + Streaming Jobs

Combined Metrics

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 55.46 | 59.32 | 61.02 | 60.27 |
| Bytes in /sec | 20k | 21k | 21k | 21k |
| Bytes out /sec | 28k | 36k | 33k | 32k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

*Εικόνα 44 Μετρήσεις Kafka Broker σε διαφορετικά σενάρια*

Μπορούμε να πάρουμε μία πρώτη εντύπωση από την κατάσταση του συστήματος μας αλλά το πιο ενδιαφέρον συμπέρασμα που μπορεί να βγει είναι ότι δεν υπάρχει καμία απώλεια μηνυμάτων.

Για πιο αναλυτικές μετρήσεις θα κοιτάξουμε τα ίδια μεγέθη στα topics που αφορούν το streaming.

Metrics — reviews

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 4.22 | 4.58 | 4.38 | 4.26 |
| Bytes in /sec | 3.9k | 3.9k | 4k | 4k |
| Bytes out /sec | 3.9k | 3.9k | 4k | 4k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

Metrics — reviews_stream

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 4.38 | 4.28 | 4.33 | 4.37 |
| Bytes in /sec | 3.9k | 3.9k | 3.9k | 3.9k |
| Bytes out /sec | 6.4k | 9.1k | 8.1k | 7.6k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

Metrics — tips

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 22.83 | 16.35 | 23.25 | 23.74 |
| Bytes in /sec | 6.7k | 5.3k | 6.9k | 7k |
| Bytes out /sec | 6.7k | 5.3k | 6.9k | 7k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

Metrics — tips_stream

| Rate | Mean | 1 min | 5 min | 15 min |
|---|---|---|---|---|
| Messages in /sec | 23.31 | 16.95 | 22.77 | 24.07 |
| Bytes in /sec | 5.9k | 4.7k | 5.8k | 6.1k |
| Bytes out /sec | 9.8k | 11k | 12k | 12k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

*Εικόνα 45 Μετρήσεις των topics μόνο όταν έχουμε streaming jobs*

| Metrics | reviews | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 4.25 | 4.43 | 4.31 | 4.30 |
| Bytes in /sec | 3.9k | 3.9k | 3.9k | 4k |
| Bytes out /sec | 3.9k | 3.9k | 3.9k | 4k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

| Metrics | reviews_stream | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 4.37 | 4.40 | 4.32 | 4.33 |
| Bytes in /sec | 3.9k | 3.7k | 3.8k | 3.8k |
| Bytes out /sec | 6.7k | 5.4k | 7.2k | 7.5k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

| Metrics | tips | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 23.48 | 25.86 | 26.69 | 25.46 |
| Bytes in /sec | 6.9k | 7.4k | 7.6k | 7.4k |
| Bytes out /sec | 6.9k | 7.3k | 7.6k | 7.4k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

| Metrics | tips_stream | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 23.89 | 25.88 | 26.64 | 25.62 |
| Bytes in /sec | 6k | 6.4k | 6.5k | 6.3k |
| Bytes out /sec | 10k | 15k | 13k | 12k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

*Εικόνα 46 Μετρήσεις των topics όταν έχουμε streaming job και ολοκληρώθηκε το query1*

| Metrics | reviews | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 4.25 | 4.25 | 4.26 | 4.27 |
| Bytes in /sec | 3.9k | 4.1k | 4.1k | 4k |
| Bytes out /sec | 4k | 4.1k | 4.1k | 4k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

| Metrics | reviews_stream | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 4.36 | 4.31 | 4.31 | 4.30 |
| Bytes in /sec | 3.9k | 3.8k | 3.8k | 3.9k |
| Bytes out /sec | 6.9k | 8.8k | 8k | 7.8k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

| Metrics | tips | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 23.75 | 25.95 | 25.93 | 25.65 |
| Bytes in /sec | 6.9k | 7.4k | 7.4k | 7.4k |
| Bytes out /sec | 7k | 7.4k | 7.4k | 7.4k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

| Metrics | tips_stream | | | |
|---|---|---|---|---|
| **Rate** | **Mean** | **1 min** | **5 min** | **15 min** |
| Messages in /sec | 24.17 | 28.58 | 26.59 | 25.96 |
| Bytes in /sec | 6.1k | 6.9k | 6.5k | 6.4k |
| Bytes out /sec | 10k | 8.4k | 11k | 12k |
| Bytes rejected /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed fetch request /sec | 0.00 | 0.00 | 0.00 | 0.00 |
| Failed produce request /sec | 0.00 | 0.00 | 0.00 | 0.00 |

*Εικόνα 47 Μετρήσεις των topics όταν έχουμε streaming job και ολοκληρώθηκε το query2*

Στα topic reviews και tips ότι πληροφορία εισέρχεται την καταναλώνει αμέσως ο Kafka-HBase connector και πάει προς αποθήκευση στην HBase. Ο ρυθμός εισαγωγής αυτών των μηνυμάτων στην ουρά είναι ίδιος με το ρυθμό κατανάλωσης. Επίσης, παρατηρούμε ότι τα tips μπαίνουν πολύ πιο γρήγορα στην Kafka από τα reviews και αυτό οφείλεται στο γεγονός ότι ο Sentiment Analyzer συμπεραίνει περιπου 5x φορές πιο γρήγορα για τα tips καθώς το κείμενο που τα συνοδεύει είναι συνήθως μικρότερο. Ο μέσος αριθμός των records/sec που εισέρχονται στο Spark Streaming (οι μετρήσεις του Spark Streaming είναι πιο πάνω) είναι ίδιος με το άθροισμα των messages/sec των topics reviews_stream και tips_stream. Άρα και σε γι' αυτόν τον consumer ο ρυθμός εισαγωγής μηνυμάτων είναι ίδιος με τον ρυθμό κατανάλωσης τους. Αυτό είναι πολύ σημαντικό καθώς ο consumer δεν μένει πίσω στην κατανάλωση των records.  Η τιμή που σχετίζεται με αυτό λέγεται consumer lag.

| | | |
|---|---|---|
| connect-hbase-sink | KF | reviews: (100% coverage, 90 lag)<br>tips: (100% coverage, 279 lag) |
| connect-hbase-sink | KF | reviews: (100% coverage, 196 lag)<br>tips: (100% coverage, 1191 lag) |

*Εικόνα 48 Consumer Lag κατά τη διάρκεια λειτουργίας*

Το consumer lag είναι η διαφορά του τελευταίου μηνύματος (offset) που καταγράφτηκε σε ένα topic Kafka από το τελευταίο μήνυμα που έχει διαβάζει ο consumer. Το lag είναι φυσιολογικό να υπάρχει καθώς πάντα υφίσταται χρονική διαφορά από την ώρα της εισόδου ενός record στην ουρά μέχρι την κατανάλωση του. Αυτό που πρέπει να συμβαίνει για να μην αυξάνεται συνεχώς το lag είναι, ο ρυθμός εισαγωγής στην Kafka να είναι σχεδόν ίδιος με το ρυθμό κατανάλωσης. Αυτό ικανοποιείται στην περίπτωση μας καθώς σε όλη τη διάρκεια της εκτέλεσης αυτής της προσομοίωσης το lag του HBase consumer παραμένει ανάμεσα στις τιμές της προηγούμενης εικόνας περίπου.

Η συμπεριφορά της Kafka παρέμεινε ίδια σε όλα τα στάδια λειτουργίας του συστήματος. Ακόμη και σε κάποιο πιο memory intensive και CPU intensive σενάρια η Kafka συνεχίζει να αποδίδει το ίδιο καλά και να μην έχει καμία απώλεια δεδομένων.

# 7. Αξιολόγηση και επίλογος

Έχοντας λάβει μετρήσεις για τα κομμάτια της αρχιτεκτονικής σε διάφορα σενάρια χρήσης τους συστήματος ήρθε η ώρα να βγάλουμε κάποια συμπεράσματα.

## 7.1 Αξιολόγηση Αρχιτεκτονικής

*Throughput*

Ο ρυθμός παραγωγής και μετάδοσης της πληροφορίας αποδείχτηκε πολύ καλός. Τα δεδομένα είτε προορίζονταν για αποθήκευση άμεσα είτε για επεξεργασία, καταναλώνονταν αμέσως από το αντίστοιχο σύστημα. Η διακίνηση των πληροφοριών από την αρχή μέχρι το τέλος δεν φάνηκε να συναντάει επιπλέον καθυστερήσεις ακόμη και όταν προσπαθήσαμε να ελέγξουμε το σύστημα σε περιόδους υψηλής πίεσης.

*Streaming Data Loss*

Με τις μετρήσεις είδαμε ότι η Apache Kafka είχε μηδενικές απώλειες δεδομένων σε όλα τα σενάρια χρήσης. Τα δεδομένα μεταφερόντουσαν χωρίς καμία αποτυχία στα υπόλοιπα υποσυστήματα. Οι απώλειες πληροφοριών μίας πλατφόρμας streaming είναι πολύ σημαντικές και ζημιογόνες καθώς κάνουν την εξόρυξη χρήσιμης πληροφορίας δύσκολη ή ακόμη και αδύνατη.

*Perfomance Under Pressure*

Όταν πήραμε μετρήσεις, την ώρα που το σύστημα εκτελούσε διεργασίες πάνω στις ροές δεδομένων και ταυτόχρονα δρομολογήθηκαν προς εκτέλεση και δουλειές που θα διάβαζαν κομμάτι της βάσης δεδομένων και θα εκτελούσαν batch analytics, η συμπεριφορά και η απόδοση ήταν πολύ καλές.

Αυτό που παρατηρήσαμε είναι ότι αν η εργασία batch είναι υπολογιστικά εύκολη τότε ίσως να μην επηρεαστεί καθόλου ο χρόνος και τρόπος εκτέλεσης της καθώς οι υπολογιστικοί πόροι του συστήματος θα είναι αρκετοί . Αντίθετα, αν είναι πιο δύσκολη τότε ενδεχομένως να καθυστερήσει η ολοκλήρωση της. Βέβαια, αυτό δεν είναι πρόβλημα καθώς τέτοιου είδους αναλύσεις δεν βασίζονται στην ταχύτητα τους.

Από την άλλη πλευρά τα data stream analytics πρέπει να γίνονται σε πραγματικό χρόνο αλλιώς χάνουν την αξία τους. Την ώρα λοιπόν της εκτέλεσης των προηγούμενων

εργασιών η επεξεργασία των ροών δεδομένων έδειξε να καθυστερεί λίγο παραπάνω χωρίς όμως αυτό να είναι προβληματικό καθώς αυτοί οι χρόνοι έμειναν σε λογικά επίπεδα, αρκετά μικρότερα από το παράθυρο χρόνου των αναλύσεων (1 λεπτό) και επανήλθαν στην αρχική τους τιμή αμέσως μετά το τέλος των εργασιών batch.

*Data Mining Capabilities*

Με την χρήση του Apache Spark έχουμε πολύ μεγάλες δυνατότητες εξόρυξης δεδομένων. Οι αλγόριθμοι και οι βιβλιοθήκες που διατίθενται επιτρέπουν την εκτέλεση πολύπλοκων εργασιών όπως ανάλυση γράφων, Machine Learning κτλ. Στη δική μας περίπτωση πραγματοποιήσαμε με ευκολία πολύπλοκα ερωτήματα που εξήγαν χρήσιμη πληροφορία από ένα μεγάλο σύνολο δεδομένων σε σχετικά μικρό χρόνο. Οι δουλειές τρέχουν με τρόπο παράλληλο σε περισσότερα μηχανήματα πράγμα που κάνει την εκτέλεση τους γρήγορη.

*Scalability*

Το τελικό σύστημα που υλοποιήθηκε έχει πολύ μεγάλες δυνατότητες κλιμακωσιμότητας καθώς αποτελείται από επιμέρους συστήματα με υψηλό scalability. Το σημαντικό είναι ότι αυτή η κλιμακωσιμότητα είναι οριζόντιας μορφής. Μπορούμε να προσθέσουμε μηχανήματα με σχετική ευκολία στο HDFS cluster για να έχουμε παραπάνω χωρητικότητα και περισσότερους Region Servers στην HBase καθώς και περισσότερους υπολογιστικούς πόρους στο Spark Cluster. Τέτοιες ενέργειες θα οδηγήσουν σε μεγαλύτερα επίπεδα παραλληλισμού. Ακόμα, και στην Kafka μπορούμε να ανοίξουμε περισσότερους brokers για έχουμε μηχανισμούς replication και partitioning των topic ώστε να διαβάζουν ταυτόχρονα περισσότεροι consumers.

*Availability*

Τα υποσυστήματα που έχουν χρησιμοποιηθεί είναι φτιαγμένα έτσι ώστε να προσφέρουν όσο το δυνατόν περισσότερο uptime. Το HDFS κάνει replicate τα αρχεία του σε περίπτωση που χαθεί ένας DataNode και δίνει την δυνατότητα ύπαρξης δεύτερου master που είναι σε standby mode. Στην HBase μπορούμε να έχουμε περισσότερους από ένα master και αν κάποιος RegionServer κλείσει, μεταφέρει τα δεδομένα του σε κάποιον που βρίσκεται σε λειτουργία. Η Kafka έχει αυτόματο μηχανισμό που αν κάποιο partition που είναι leader πάψει να υπάρχει για κάποιο λόγο, τότε ένα replica παίρνει την θέση του ως αρχηγός. Εφόσον, τα επιμέρους κομμάτια του συστήματος προσφέρουν μεγάλη διαθεσιμότητα το ίδιο θα κάνει και το τελικό σύστημα.

## 7.2 Περαιτέρω Βελτιώσεις και Επίλογος

Το σύστημα αυτό σχεδιάστηκε, υλοποιήθηκε και μελετήθηκε εκτενώς και για την παρούσα διπλωματική αποτελεί ικανοποιητική λύση στο πρόβλημα που είχε τεθεί. Θα μπορούσαν όμως να υπάρχουν βελτιώσεις. Το βασικό «πρόβλημα» που αντιμετωπίστηκε, και αυτό φαίνεται με τις μετρήσεις που πραγματοποιήθηκαν, είναι οι υπολογιστικοί πόροι που διατέθηκαν για τη δημιουργία του και κυρίως το γεγονός ότι η ποσότητα τους ήταν οριακή για την εκπόνηση των μετρήσεων υπό υψηλή πίεση. Φυσικά, αυτό δεν αποτελεί δεν αποτελεί ιδιαίτερο εμπόδιο καθώς οι δυνατότητες οριζόντιας κλιμάκωσης είναι πολύ μεγάλες. Για να ανταπεξέλθει το σύστημα αυτό σε πιο ρεαλιστικές καταστάσεις, σε ροές δεδομένων που ανέρχονται στα MB/s ή GB/s θα χρειαστούμε περισσότερα μηχανήματα, εξοπλισμένα με ισχυρούς επεξεργαστές και αρκετή κύρια μνήμη. Σημαντικές αλλαγές στο σύστημα θα μπορούσαν να προσφέρουν και οι τεχνικές του replication και partitioning που διαθέτει η Apache Kafka και δεν χρησιμοποιήθηκαν σε αυτή την εργασία καθώς δεν παραχωρήθηκαν αρκετά μηχανήματα. Αν η Kafka διαθέτει το δικό της αποκλειστικό cluster και χρησιμοποιεί τους μηχανισμούς αυτούς, υπάρχει δυνατότητα παραλληλισμού στην κατανάλωση των μηνυμάτων από πολλούς consumers κάτι που θα δώσει ακόμη μεγαλύτερο throughput στο σύστημα.

Συνοψίζοντας, η τελική πλατφόρμα που δημιουργήθηκε ικανοποιεί τα στοιχεία που χρειάζεται μία πλατφόρμα ανάλυσης δεδομένων σε πραγματικό χρόνο. Προσφέρει, επίσης, και επιπλέον δυνατότητες batch processing και data warehousing, ενισχύοντας κατά πολύ τις δυνατότητες data mining και γενικής επεξεργασίας Big Data. Η παράλληλη χρήση των τεχνολογιών αυτών φάνηκε να γίνεται με ομαλό τρόπο και χωρίς να εμποδίζεται η κύρια λειτουργία τους. Η υψηλή διακίνηση δεδομένων, η διαθεσιμότητα καθώς και οι προδιαγραφές κλιμάκωσης που φαίνεται να διαθέτει το σύστημα το καθιστά ένα αξιόπιστο και ισχυρό εργαλείο.

# Βιβλιογραφία


[1] Ashley DeVan. *The 7 V's of Big Data.* https://www.impactradius.com/blog/7-vs-big-data.

[2] Basho.com. *NoSQL Databases Explained.* http://basho.com/resources/nosql-databases/

[3] Danny Sullivan. *Google now handles at least 2 trillion searches per year.* http://searchengineland.com/google-now-handles-2-999-trillion-searches-per-year-250247

[4] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 1970; 13(6) http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[5] Eric Brewer. *CAP Twelve Years Later: How the "Rules" Have Changed* https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

[6] Eric Evans. *NOSQL 2009.* http://blog.sym-link.com/2009/05/12/nosql_2009.html

[7] Eric Lai. No to SQL? Anti-database movement gains steam. *Computerworld* 2009; p1. http://www.computerworld.com/article/2526317/database-administration/no-to-sql--anti-database-movement-gains-steam.html

[8] Forrester Research. *THE FORRESTER WAVE™: BIG DATA HADOOP SOLUTIONS, Q1 2014* https://www.forrester.com/The+Forrester+Wave+Big+Data+Hadoop+Solutions+Q1+2014/-/E-PRE6807

[9] HBR, Thomas H. Davenport, D.J. Patil. *Data Scientist: The Sexiest Job of the 21st Century.* https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

[10] IBM, David Beaumont. *How to explain vertical and horizontal scaling in the cloud.* https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-cloud/

[11] J. Dean, S. Ghemawat. MapReduce: A Flexible Data Processing Tool. *Communications of the ACM* 2010; 53(1): p72-77

[12] Jay Kreps. *Questioning the Lambda Architecture.* https://www.oreilly.com/ideas/questioning-the-lambda-architecture

[13] Julian Browne. *Brewer's CAP Theorem.* http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[14] Michael Hausenblas & Nathan Bijnens. *Lambda Architecture*. http://lambda-architecture.net/

[15] Microsoft. *ACID properties*. https://msdn.microsoft.com/en-us/library/aa480356.aspx?f=255&MSPPError=-2147217396

[16] MG Siegler. *Eric Schmidt: Every 2 Days We Create As Much Information As We Did Up To 2003*. https://techcrunch.com/2010/08/04/schmidt-data/

[17] MongoDB. *JSON and BSON*. https://www.mongodb.com/json-and-bson

[18] NIST. *The NIST Definition of Cloud Computing*. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[19] Oracle. *A Relational Database Overview*. https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

[20] Pablo Valerio. US Firms Looking To Europe For Data Protection. *Network Computing* 2016;http://www.networkcomputing.com/cloud-infrastructure/us-firms-looking-europe-data-protection/129277031

[21] R. Elmasri, S.B. Navathe. *Fundamentals of Database Systems*, 6th ed. Athens: Εκδόσεις Δίαυλος; 2011

[22] Vrettos Moulos. *Outsourcing Information Security Management*. London: Kings College of London; 2008

[23] Warren C. Axelrod. *Outsourcing Information Security*; p27-69. Norwood: Artech House; 2004

[24] Warwick Ashford. *Bad outsourcing decisions cause 63% of data breaches*. http://www.computerweekly.com/news/2240178104/Bad-outsourcing-decisions-cause-63-of-data-breaches

[25] Wikipedia. *Big data*. https://en.wikipedia.org/wiki/Big_data

# Παράρτημα

## Α. Διαδικασία εγκατάστασης λογισμικού.

Παρακάτω θα δοθούν οι απαραίτητες οδηγίες για την εγκατάσταση των απαιτούμενων λογισμικών που χρησιμοποιεί η τελική αρχιτεκτονική μας καθώς και οποιεσδήποτε ρυθμίσεις για τη σύνδεση μεταξύ αυτών. Σε διαφορετικούς υπολογιστές με διαφορετικά λειτουργικά συστήματα μπορεί να προκύψουν προβλήματα ή να χρειάζονται ελαφρώς διαφορετικές ρυθμίσεις. Σε κάθε περίπτωση η διαδικασία που περιγράφεται θα αφορά μία συγκεκριμένη έκδοση λογισμικού. Τα προγράμματα αυτά αναβαθμίζονται συνεχώς οπότε θα πρέπει να συμβουλευτείτε το εγχειρίδιο εγκατάστασης και χρήσης.

Τα μηχανήματα στα οποία έγινε η εγκατάσταση των προγραμμάτων είχαν όλα εγκατεστημένα την έκδοση 1.8 της Java, έτρεχαν κάποιο λειτουργικό σύστημα τύπου Linux/Unix και είχαν όλα δημόσια IP. Αν δεν διαθέτουν δημόσια IP θα πρέπει να γραφτεί κάποιο NAT script που δεν θα καλυφθεί σε αυτό τον οδηγό εγκατάστασης. Επίσης, για να μην υπάρχουν προβλήματα δικαιοδοσίας όλες η εγκαταστάσεις έγιναν μέσω του χρήστη «root»

## Α.1 Εγκατάσταση Apache Hadoop v2.7.3

Στην διάθεση μας έχουμε τρία μηχανήματα. Το πρώτο πράγμα που θα κάνουμε είναι να ενημερώσουμε τα αρχεία hosts των μηχανημάτων μας ώστε να μην χρειάζεται να θυμόμαστε τις IP τους απ' έξω. Το αρχείο hosts υπάρχει σε κάθε υπολογιστή, και στην ουσία αποτελεί μια τοπική βάση DNS την οποία συμβουλεύεται πρώτη κάθε φορά που προσπαθεί να κάνει resolve ένα όνομα σε διεύθυνση IP.

Έτσι ψάχνουμε στην τοποθεσία

```
/etc/hosts
```

Αφού ανοίξουμε αυτό το αρχείο συμπληρώνουμε και στα τρία μηχανήματα

```
<master_ip> master

<slave1_ip> slave1

<slave2_ip> slave2
```

Στην συνέχεια σε κάθε μηχάνημα ενημερώνουμε το αρχείο hostname ώστε να αντιστοιχεί σε αυτά τα ονόματα τρέχοντας την παρακάτω εντολή ξεχωριστά σε κάθε ένα ανάλογα με την IP που θέσαμε στο hosts

```
echo ‘master’ > /etc/hostname
```

```
echo ‘slave1’ > /etc/hostname
```

```
echo ‘slave2’ > /etc/hostname
```

Το Hadoop για να λειτουργήσει θα πρέπει οι υπολογιστές που αποτελούν το cluster να μπορούν να συνδέονται μεταξύ τους με την χρήση ssh χωρίς κωδικό. Αυτό επιτυγχάνεται με τη χρήση ζεύγους ιδιωτικού/δημόσιου κλειδιού. Η βασική ιδέα της τεχνικής αυτής είναι ότι ο κάθε χρήστης μπορεί να έχει ένα ιδιωτικό κλειδί (αρχείο) το οποίο έρχεται ζεύγος με ένα δημόσιο κλειδί. Ο χρήστης τοποθετεί το δημόσιο κλειδί στους υπολογιστές τους οποίους θέλει να έχει πρόσβαση, και με την χρήση του ιδιωτικού μπορεί να συνδέεται σε αυτούς χωρίς κωδικό. Η πιστοποίηση του χρήστη γίνεται καθώς μόνο ο χρήστης έχει στην κατοχή του το ιδιωτικό κλειδί. Οπότε στον master τρέχουμε

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

Η εντολή δημιουργεί ένα ιδιωτικό κλειδί id_rsa και ένα δημόσιο κλειδί id_rsa.pub.

Τρέχουμε

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Η παραπάνω εντολή βάζει το δημόσιο κλειδί id_rsa.pub στον κατάλογο με τα αποδεκτά δημόσια κλειδιά του χρήστη root του master μηχανήματος. Με αυτόν τον τρόπο, όποιος έχει στην κατοχή του το ιδιωτικό κλειδί id_rsa μπορεί να συνδεθεί στο root@master. Τώρα, αρκεί να μεταφέρουμε τον κατάλογο .ssh στον χρήστη root και των μηχανημάτων slave1 και slave2.

```
scp -r .ssh/ root@slave1:~/

scp -r .ssh/ root@slave2:~/
```

Στην συνέχεια κατεβάζουμε το installation package, σε όλα τα μηχανήματα, από την επίσημη ιστοσελίδα http://hadoop.apache.org/ και το αποσυμπιέζουμε σε κατάλογο της επιλογής μας. Στη συγκεκριμένη περίπτωση το αποσυμπιέσαμε στον κατάλογο /opt. Κάνουμε edit το

~/.bashrc όλων των κόμβων του cluster ώστε να κάνουμε export τις παρακάτω μεταβλητές περιβάλλοντος. Προσθέτουμε τις παρακάτω γραμμές στο τέλος του bashrc.

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/

export HADOOP_INSTALL=/opt/hadoop-2.6.3/

export PATH=$PATH:$HADOOP_INSTALL/bin

export PATH=$PATH:$HADOOP_INSTALL/sbin

export HADOOP_HOME=$HADOOP_INSTALL

export HADOOP_MAPRED_HOME=$HADOOP_INSTALL

export HADOOP_COMMON_HOME=$HADOOP_INSTALL

export HADOOP_HDFS_HOME=$HADOOP_INSTALL

export HADOOP_YARN_HOME=$HADOOP_INSTALL

export HADOOP_CONF_DIR=$HADOOP_INSTALL/etc/hadoop
```

Το configuration του Hadoop βρίσκεται στο path: $HADOOP_INSTALL/etc/hadoop. Στον κατάλογο αυτό θα επεξεργαστούμε τα εξής αρχεία και στον master και στους slave:

core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!--

  Licensed under the Apache License, Version 2.0 (the "License");

  you may not use this file except in compliance with the License.

  You may obtain a copy of the License at


    http://www.apache.org/licenses/LICENSE-2.0


  Unless required by applicable law or agreed to in writing,
software
```

```
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
  See the License for the specific language governing permissions
and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
	<property>
		<name>fs.default.name</name>
		<value>hdfs://master:9000</value>
	</property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0
```

```
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
     <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication.</description>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/opt/hdfsnames</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/opt/hdfsdata</value>
```

```
  </property>
  <property>
    <name>dfs.blocksize</name>
    <value>64m</value>
    <description>Block size</description>
  </property>
  <property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
  </property>
  <property>
    <name>dfs.support.append</name>
    <value>true</value>
  </property>
</configuration>
```

yarn-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>master</value>
    <final>true</final>
```

```
    <description>host is the hostname of the resource manager.
    </description>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>master:8025</value>
    <final>true</final>
    <description>host is the hostname of the resource manager and
    port is the port on which the NodeManagers contact the Resource
Manager.
    </description>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>master:8030</value>
    <final>true</final>
    <description>host is the hostname of the resourcemanager and
port is the port
    on which the Applications in the cluster talk to the Resource
Manager.
    </description>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>master:8050</value>
    <final>true</final>
```

```
    <description>the host is the hostname of the ResourceManager and the port is the port on
    which the clients can talk to the Resource Manager. </description>
  </property>
  <property>
    <name>yarn.nodemanager.address</name>
    <value>0.0.0.0:0</value>
    <description>the nodemanagers bind to this port</description>
  </property>
  <property>
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/app-logs</value>
    <description>directory on hdfs where the application logs are moved to </description>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
    <description>Shuffle service that needs to be set for Map Reduce to run </description>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
```

```
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>master:8088</value>
    <description>The http address of the RM web application
</description>
  </property>
  <property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
  </property>
  <property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>4096</value>
  </property>
  <property>
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>4</value>
  </property>
  <property>
    <name>yarn.scheduler.minimum-allocation-mb</name>
    <value>128</value>
  </property>
  <property>
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>2048</value>
```

```
  </property>
  <property>
    <name>yarn.scheduler.minimum-allocation-vcores</name>
    <value>1</value>
  </property>
  <property>
    <name>yarn.scheduler.maximum-allocation-vcores</name>
    <value>2</value>
  </property>
</configuration>
```

mapred-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
    <description>No description</description>
  </property>
  <property>
    <name>mapreduce.map.java.opts</name>
    <value>-Xmx728m</value>
    <description>No description</description>
```

```
</property>
<property>
<name>mapreduce.map.memory.mb</name>
  <value>1024</value>
  <description>No description</description>
</property>
<property>
  <name>mapreduce.reduce.java.opts</name>
  <value>-Xmx728m</value>
  <description>No description</description>
</property>
<property>
<name>mapreduce.reduce.memory.mb</name>
  <value>1024</value>
  <description>No description</description>
</property>
<property>
  <name>mapreduce.map.cpu.vcores</name>
  <value>1</value>
  <description>No description</description>
</property>
<property>
  <name>mapreduce.reduce.cpu.vcores</name>
  <value>1</value>
  <description>No descriprion</description>
```

```
  </property>
  <property>
    <name>mapreduce.task.timeout</name>
    <value>0</value>
  </property>
</configuration>
```

Τα αρχεία mapred-site.xml και yarn-site.xml εξαρτώνται από τις δυνατότητες των μηχανημάτων του cluster και έτσι δεν έχουν συγκεκριμένες τιμές. Ανάλογα με την επεξεργαστική ισχύ και την μνήμη που έχει το κάθε μηχάνημα αυτά τα αρχεία μπορεί να αλλάζουν. Ορισμένοι οδηγοί που δείχνουν ποια μορφή πρέπει να έχουν αυτά τα αρχεία είναι οι παρακάτω:

https://hortonworks.com/blog/how-to-plan-and-configure-yarn-in-hdp-2-0/

https://www.alexjf.net/blog/distributed-systems/hadoop-yarn-installation-definitive-guide/

Το τελευταίο αρχείο που θα πειραχτεί είναι το αρχείο slaves στο master μόνο. Αυτό βρίσκεται στην ίδια τοποθεσία που βρίσκονταν και τα άλλα αρχεία. Εκεί θα βάλουμε απλά τα ονόματα που ορίσαμε στο αρχείο hosts ώστε να ξέρει ο master ποιους slaves έχει. Στην δική μας περίπτωση

```
slave1
slave2
```

Τέλος αυτό που μένει είναι να ξεκινήσουμε το HDFS και το YARN. Αρχικά συνδεόμαστε στο master και ΜΟΝΟ την πρώτη φορά κάνουμε format

```
hdfs namenode -format
```

Και στην συνέχεια ξεκινάμε το HDFS δίνοντας από το master την εξής εντολή

```
start-dfs.sh
```

Για την εκκίνηση του YARN τρέχουμε στο master την εντολή

```
start-yarn.sh
```

Χρησιμοποιώντας την εντολή jps μπορούμε να δούμε τι τρέχει στο master και στους slave.

master

```
root@master ~ # jps

30130 NameNode

30340 SecondaryNameNode

30522 ResourceManager

31743 Jps
```

Ο Namenode τρέχει στο master (όπως πρέπει) καθώς και ο ResourceManager του YARN

slave1

```
root@slave1 ~ # jps

6582 DataNode

7259 Jps

6735 NodeManager
```

Ο DataNode του HDFS και ο NodeManager του YARN τρέχουν στους slave.

Το σύστημα έχει ξεκινήσει και στα παρακάτω url μπορεί κανείς να βλέπει την κατάσταση του cluster

http://<master_IP>:50070: Το url αυτό δείχνει την κατάσταση του NameNode, καθώς και τα περιεχόμενα του αποθηκευτικού συστήματος.

http://<master_IP>:8088: Αυτό είναι το url για το endpoint του YARN και δείχνει την κατάσταση των resources του cluster και των εφαρμογών που τρέχουν σε αυτό.

Για να κλείσουμε το Hadoop τρέχουμε από το master

```
stop-yarn.sh

stop-dfs.sh
```

## A.2 Εγκατάσταση Apache ZooKeeper v3.4.9

Αρχικά κατεβάζουμε από την επίσημη ιστοσελίδα, https://zookeeper.apache.org/, το installation package και το αποσυμπιέζουμε σε κατάλογο της επιλογής μας. Στην δική μας περίπτωση /usr/local/zookeeper/.

Κάνουμε edit to ~/.bashrc ώστε να κάνουμε export τις παρακάτω μεταβλητές περιβάλλοντος. Προσθέτουμε τις παρακάτω γραμμές στο τέλος του bashrc

```
export ZOOKEEPER_HOME="/usr/local/zookeeper/zookeeper-3.4.9/"

export PATH="$ZOOKEEPER_HOME/bin:$PATH"
```

Στην συνέχεια ορίζουμε τις εξής ρυθμίσεις στο αρχείο $ZOOKEEPER_HOME/conf/zoo.cfg

```
tickTime=2000

dataDir=/var/zookeeper/

dataLogDir=/usr/local/zookeeper/zookeeper-3.4.9/logs

clientPort=2181

initLimit=10

syncLimit=5

server.1=master:2888:3888

server.2=slave1:2888:3888

server.3=slave2:2888:3888
```

Όπου master, slave1, slave2 είναι τα μηχανήματα που ανήκουν στο ZooKeeper ensemble. Κάθε μηχάνημα πρέπει να γνωρίζει για τα άλλα και αυτό το επιτυγχάνουμε γράφοντας αυτές τις γραμμές στην μορφή **server.id=host:port:port.**

Οπότε το επόμενο πράγμα που πρέπει να κάνουμε είναι να δώσουμε IDs στους ZooKeeper servers. Το αναγνωριστικό διακομιστή μπορεί να οριστεί δημιουργώντας ένα αρχείο με όνομα myid, ένα για κάθε server, το οποίο βρίσκεται στον κατάλογο δεδομένων αυτού του διακομιστή, όπως καθορίζεται από την παράμετρο του αρχείου ρυθμίσεων dataDir που ορίσαμε στο προηγούμενο αρχείο. Το αρχείο myid αποτελείται από μια μόνο γραμμή που περιέχει μόνο το to id του μηχανήματος. Έτσι το myid του διακομιστή 1 θα περιέχει το κείμενο "1" και τίποτα άλλο. Η ταυτότητα πρέπει να είναι μοναδική μέσα στο σύνολο.

Στη συγκεκριμένη περίπτωση το θα δημιουργήσουμε σε κάθε μηχάνημα στον κατάλογο /var/zookeeper/ ένα αρχείο με όνομα «myid» όπου στο master θα περιέχει μόνο τον αριθμό «1» , στον slave1 τον αριθμό «2» και στον slave2 τον αριθμό «3».

Τέλος αυτό μου μένει είναι να ξεκινήσουμε σε κάθε μηχάνημα τον ZooKeeper με την εντολή

```
zkServer.sh start
```

Σε κάθε κόμβο μπορούμε να ελέγξουμε την κατάσταση του ZooKeeper server με την εντολή

```
zkServer.sh status
```

Επίσης με την εντολή jps βλέπει κανείς την υπηρεσία του ZooKeeper που τρέχει και έχει όνομα QuorumPeerMain.

```
root@master ~ # jps

30130 NameNode

2323 QuorumPeerMain

30340 SecondaryNameNode

30522 ResourceManager

2415 Jps
```

## A3. Εγκατάσταση Apache HBase v1.2.4

Για την εγκατάσταση της HBase σε πλήρως κατανεμημένη μορφή πρέπει να έχει προηγηθεί η εγκατάσταση του Hadoop, καθώς χρησιμοποιείται το HDFS σαν file system, καθώς και η εγκατάσταση του ZooKeeper.

Κατεβάζουμε το installation package, σε όλα τα μηχανήματα, από την επίσημη ιστοσελίδα https://hbase.apache.org/ και το αποσυμπιέζουμε σε κατάλογο της επιλογής μας. Στη συγκεκριμένη περίπτωση το αποσυμπιέσαμε στον κατάλογο /usr/local/hbase/. Κάνουμε edit το ~/.bashrc όλων των μηχανημάτων ώστε να κάνουμε export τις παρακάτω μεταβλητές περιβάλλοντος. Προσθέτουμε τις παρακάτω γραμμές στο τέλος του bashrc.

```
export HBASE_HOME="/usr/local/hbase/hbase-1.2.4/"

export PATH="$HBASE_HOME/bin:$PATH"
```

Για να ελέγξουμε αν έγινε η αρχική εγκατάσταση τρέχουμε την εντολή

```
hbase version
```

Για την σωστή ρύθμιση της HBase κάνουμε edit τα αρχεία $HBASE_HOME/conf/hbase-env.sh και $HBASE_HOME/conf/hbase-site.xml.

hbase-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/

export HBASE_CLASSPATH=${HADOOP_CONF_DIR}

export HBASE_PID_DIR=/var/hbase/pids

export HBASE_MANAGES_ZK=false
```

Οι μεταβλητές αυτές υπάρχουν ήδη (αν δεν υπάρχουν προσθέστε τις).

hbase-site.xml (master)

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!--

/**

 *

 * Licensed to the Apache Software Foundation (ASF) under one

 * or more contributor license agreements.  See the NOTICE file

 * distributed with this work for additional information

 * regarding copyright ownership.  The ASF licenses this file

 * to you under the Apache License, Version 2.0 (the

 * "License"); you may not use this file except in compliance

 * with the License.  You may obtain a copy of the License at

 *

 *     http://www.apache.org/licenses/LICENSE-2.0

 *
```

```
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
-->
<configuration>
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:9000/hbase</value>
</property>
<property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
</property>
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>master,slave1,slave2</value>
    <description>Comma separated list of servers in the ZooKeeper Quorum.
    For example, "host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".
```

```
    By default this is set to localhost for local and pseudo-distributed modes
    of operation. For a fully-distributed setup, this should be set to a full
    list of ZooKeeper quorum servers. If HBASE_MANAGES_ZK is set in hbase-env.sh
    this is the list of servers which we will start/stop ZooKeeper on.
    </description>
</property>
<property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/var/zookeeper</value>
</property>
<property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2181</value>
 </property>
 <property>
      <name>hbase.rpc.timeout</name>
      <value>60000</value>
 </property>
<property>
  <name>hbase.client.scanner.timeout.period</name>
  <value>60000</value>
</property>
```

```
<property>
  <name>hbase.cells.scanned.per.heartbeat.check</name>
  <value>10000</value>
</property>

</configuration>
```

hbase-site.xml (slaves)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
/**
 *
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
software
 * distributed under the License is distributed on an "AS IS" BASIS,
```

```
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions
and
 * limitations under the License.
 */
-->
<configuration>
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:9000/hbase</value>
</property>
<property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
</property>
</configuration>
```

Τέλος τον master node πρέπει στο αρχείο HBASE_HOME/conf/regionservers να ορίσουμε τους regionservers της hbase. Σε αυτή την περίπτωση το αρχείο αυτό θα γράφει

```
slave1
slave2
```

Αυτό που μένει να γίνει είναι να ξεκινήσουμε την HBase από τον master με την εντολή

```
start-hbase.sh
```

Με την εντολή jps μπορεί κανείς να δει τα daεmons που τρέχουν σε master και slaves και αφορούν την HBase.

master

```
root@master ~ # jps

30130 NameNode

2323 QuorumPeerMain

30340 SecondaryNameNode

6808 HMaster

7128 Jps

30522 ResourceManager
```

slave

```
root@slave1 ~ # jps

8720 QuorumPeerMain

6582 DataNode

16827 HRegionServer

17483 Jps

6735 NodeManager
```

Το σύστημα έχει ξεκινήσει και στο παρακάτω url μπορεί κανείς να βλέπει την κατάσταση του cluster πάνω στο οποίο τρέχει η HBase.

http://<master_IP>:16010: Το url αυτό δείχνει την κατάσταση των RegionServers, καθώς και γενικά στοιχεία για τους κόμβους αλλά και για την ίδια την HBase.

Σε περίπτωση που θέλουμε να κλείσουμε την HBase τρέχουμε την παρακάτω εντολή από τον master.

```
stop-hbase.sh
```

## A.4 Εγκατάσταση Apache Kafka v0.10.2 – Πλατφόρμα Confluent v3.2.0

Η Kafka για να λειτουργήσει σωστά χρειάζεται ένα ZooKeeper cluster και γι' αυτό χρειάζεται να προηγηθεί η εγκατάσταση του ZooKeeper. Επίσης, θα χρειαστεί να συνδέσουμε την Kafka με την HBase ώστε να μπορεί να διαβάζει από εκεί δεδομένα και έτσι επιλέχθηκε η πλατφόρμα Confluent που έχει στον πυρήνα της την Kafka αλλά διαθέτει επιπλέον βιβλιοθήκες καθώς και connectors για τη σύνδεση με άλλα εργαλεία.

Κατεβάζουμε το installation package, στο master καθώς θα έχουμε μόνο έναν Kafka Server, από την επίσημη ιστοσελίδα https://www.confluent.io/ και το αποσυμπιέζουμε σε κατάλογο της επιλογής μας. Στη συγκεκριμένη περίπτωση το αποσυμπιέσαμε στον κατάλογο /opt. Κάνουμε edit to ~/.bashrc ώστε να κάνουμε export τις παρακάτω μεταβλητές περιβάλλοντος. Προσθέτουμε τις παρακάτω γραμμές στο τέλος του bashrc

```
export CONFLUENT_HOME="/opt/confluent-3.2.0/"

export PATH="$CONFLUENT_HOME/bin:$PATH"
```

Στην συνέχεια, πριν τρέξουμε τον Kafka broker και το schema registry πρέπει να βάλουμε στις ρυθμίσεις τη διεύθυνση του ZooKeeper cluster.

Για την Kafka ψάχνουμε το αρχείο $CONFLUENT_HOME/etc/kafka/server.properties και αλλάζουμε την παρακάτω γραμμή

```
zookeeper.connect=master:2181,slave1:2181,slave2:2181
```

Για το schema-registry ψάχνουμε το αρχείο $CONFLUENT_HOME/etc/schema-registry/schema-registry.properties και αλλάζουμε την παρακάτω γραμμή

```
kafkastore.connection.url=master:2181,slave1:2181,slave2:2181
```

Η εντολή που τρέχει τον Kafka broker από το $CONFLUENT_HOME είναι

```
./bin/kafka-server-start ./etc/kafka/server.properties
```

Αλλά το process θα τρέχει συνεχώς και θα παράγει πληροφορίες για τον broker. Οπότε θέλουμε να τρέχει στο background και για λόγους debugging το output θα φυλάσσεται. Άρα τελικά θα χρησιμοποιήσουμε την εντολή

```
nohup ./bin/kafka-server-start ./etc/kafka/server.properties >
kafka-broker.out &
```

Αντίστοιχα για το schema registry θα χρησιμοποιήσουμε την εντολή

```
nohup ./bin/schema-registry-start ./etc/schema-registry/schema-registry.properties > schema-registry.out &
```

Για να τρέξουμε τον connector της HBase με την Kafka κατεβάζουμε το installation package του Stream Reactor από την επίσημη ιστοσελίδα

http://docs.datamountaineer.com/en/latest/install.html#install

και το αποσυμπιέζουμε σε κατάλογο της επιλογής μας. Στη συγκεκριμένη περίπτωση το αποσυμπιέσαμε στον κατάλογο $CONFLUENT_HOME/stream-reactor/.

Οι παρακάτω εντολές εκτελούνται από το $CONFLUENT_HOME/stream-reactor/.

Για να τρέξουμε τον connector στο background και να έχουμε και log για debugging τρέχουμε την εντολή

```
nohup ./stream-reactor/stream-reactor-0.2.5-3.2.0/bin/start-connect.sh > connector.out &
```

Και στην συνέχεια φτιάχνουμε ένα sink connector για την HBase ακολουθώντας τις οδηγίες του παρακάτω συνδέσμου.

http://docs.datamountaineer.com/en/latest/hbase.html

Τέλος τρέχουμε την εντολή

```
./stream-reactor/stream-reactor-0.2.5-3.2.0/bin/cli.sh create hbase-sink < ./stream-reactor/stream-reactor-0.2.5-3.2.0/conf/hbase-sink.properties
```

Για να ελέγξουμε αν ο connector είναι ενεργός γράφουμε

```
./stream-reactor/stream-reactor-0.2.5-3.2.0/bin/cli.sh ps
```

Και στο αποτέλεσμα βλέπουμε το κείμενο «hbase-sink» που είναι το όνομα που δώσαμε στο sink connector.

Με την εντολή jps μπορούμε να δούμε τα προγράμματα που τρέχουν.

```
root@master ~ # jps
9793 Jps
30130 NameNode
8402 SupportedKafka
2323 QuorumPeerMain
30340 SecondaryNameNode
8807 SchemaRegistryMain
6808 HMaster
30522 ResourceManager
9518 ConnectDistributed
```

Η Apache Kafka τρέχει ως «SupportedKafka», το Schema registry ως «SchemaRegistryMain» και ο connector ως « ConnectDistributed ».

## A5. Εγκατάσταση Apache Spark v2.1.0

Το Spark για να λειτουργήσει σωστά χρειάζεται να εγκατασταθεί πρώτα η γλώσσα προγραμματισμού Scala. Για τη συγκεκριμένη έκδοση του Spark επιλέχθηκε η έκδοση 2.12.1 της Scala.

Οπότε κατεβάζουμε το installation package, σε όλα τα μηχανήματα, από την επίσημη ιστοσελίδα https://www.scala-lang.org/download/ και το αποσυμπιέζουμε σε κατάλογο της επιλογής μας. Στη συγκεκριμένη περίπτωση το αποσυμπιέσαμε στον κατάλογο /usr/local/scala/. Το ίδιο κάνουμε και για το installation package του Spark από την ιστοσελίδα https://spark.apache.org/. Την αποσυμπίεση την κάναμε στον κατάλογο /opt.

Στη συνέχεια κάνουμε edit to ~/.bashrc όλων των κόμβων του cluster ώστε να κάνουμε export τις παρακάτω μεταβλητές περιβάλλοντος. Προσθέτουμε τις παρακάτω γραμμές στο τέλος του bashrc

```
export PATH=$PATH:/usr/local/scala/scala-2.12.1/bin

export SPARK_HOME=/opt/spark-2.1.0-bin-hadoop2.7

export PATH=$PATH:$SPARK_HOME/bin
```

Για να δούμε αν η Scala και το Spark εγκαταστάθηκαν σωστά εκτελούμε τις εντολές

```
scala -version

spark-shell
```

Η δεύτερη εντολή θα ανοίξει το shell του Spark.

ΣΗΜΑΝΤΙΚΟ: Επειδή το Spark θα επικοινωνεί με άλλα συστήματα όπως η HBase και η Kafka είναι πολύ πιθανόν να χρειαστεί να μεταφερθούνε κάποιες βιβλιοθήκες από αυτά στον κατάλογο $SPARK_HOME/jars

## B. Κώδικας εφαρμογών

Ακολουθεί ο κώδικας των Kafka Producers σε Java και των διεργασιών τύπου batch και Streaming σε Scala.

### B.1. Kafka Producers

Run.jana

```
import gr.ntua.cep.sentiment.SentimentAnalysisException;
import gr.ntua.cep.sentiment.SentimentAnalysisImp;

import java.io.IOException;

/**
 * Pick what producer you want to run. This lets us
 * have a single executable as a build target.
 */
public class Run {
    public static void main(String[] args) throws IOException,
InterruptedException, SentimentAnalysisException {
        if (args.length < 1 ) {
            throw new IllegalArgumentException("Must have either 'producer-
business','producer-tip' or 'producer-review' as argument");
        }
        switch (args[0]) {
            case "producer-business":
                ProducerBusiness.main(args);
                break;
            case "producer-tip":
                SentimentAnalysisImp analyser =
SentimentAnalysisImp.getInstance();
                ProducerTip.main(args,analyser);
                break;
            case "producer-review":
                SentimentAnalysisImp analyser1 =
SentimentAnalysisImp.getInstance();
                ProducerReview.main(args,analyser1);
                break;
            default:
                throw new IllegalArgumentException("Don't know how to do " +
args[0]);
        }
    }
}
```

ProducerBusiness.java

```
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import java.io.*;
import java.util.Properties;

/*
*  This is the code of the Kafka producer for business topic
* */

public class ProducerBusiness {
```

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

        if (args.length != 3) {
            System.out.println("Please provide command line arguments for
Producer: input_file topic_name");
            System.exit(1);
        }
        // Kafka parameters
        String topic = args[2];
        String url = "http://147.102.19.1:8081";
        Properties props = new Properties();
        props.put("bootstrap.servers", "147.102.19.1:9092");
        props.put("acks", "all");
        props.put("retries", "0");
        props.put("batch.size", "16384");
        props.put("auto.commit.interval.ms", "1000");
        props.put("linger.ms", "0");
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
"io.confluent.kafka.serializers.KafkaAvroSerializer");
        props.put("block.on.buffer.full", "true");
        props.put("schema.registry.url", url);

        KafkaProducer<String, GenericRecord> producer = new KafkaProducer<String,
GenericRecord>(props);
        // Parsers for the Dsta schema
        Schema.Parser parser = new Schema.Parser();
        Schema schema = parser.parse(Schemas.bussiness_schema);

        FileInputStream fis;
        BufferedReader br = null;
        try {
            fis = new FileInputStream(args[1]);
            br = new BufferedReader(new InputStreamReader(fis));

            String line = null;
            ObjectMapper mapper = new ObjectMapper();
            while ((line = br.readLine()) != null) {
                JsonNode businessJson =  mapper.readTree(line);
                String business_id = businessJson.path("business_id").asText();
                // Send data to topic for Persistence Storage
                GenericRecord business = new GenericData.Record(schema);
                business.put("business_id",business_id);
                business.put("business_info", businessJson.toString());
                ProducerRecord<String, GenericRecord> data = new
ProducerRecord<String, GenericRecord>(topic, business);
                producer.send(data);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        producer.close();
    }
}
```

ProducerReview.java

```
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import gr.ntua.cep.sentiment.SentimentAnalysisException;
import gr.ntua.cep.sentiment.SentimentAnalysisImp;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.io.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Properties;

/*
*  This is the code of the Kafka producer for reviews and reviews_stream topics
* */

public class ProducerReview {

    public static void main(String[] args, SentimentAnalysisImp analyser) throws
IOException, InterruptedException, SentimentAnalysisException {


        if (args.length != 4 && args.length != 5) {
            System.out.println(args.length+"");
            System.out.println("Please provide command line arguments for
Producer: input_file topic_name \"dummy\"|\"serious\"  [time_delay(ms)]");
            System.exit(1);
        }
        // Kakfa parameters
        String topic = args[2];
        String topic_stream = args[2] + "_stream";
        String url = "http://147.102.19.1:8081";
        Properties props = new Properties();
        props.put("bootstrap.servers", "147.102.19.1:9092");
        props.put("acks", "all");
        props.put("retries", "0");
        props.put("batch.size", "16384");
        props.put("auto.commit.interval.ms", "1000");
        props.put("linger.ms", "0");
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
"io.confluent.kafka.serializers.KafkaAvroSerializer");
        props.put("block.on.buffer.full", "true");
        props.put("schema.registry.url", url);

        // Parsers for the Dsta schema
        KafkaProducer<String, GenericRecord> producer = new KafkaProducer<String,
GenericRecord>(props);
        Schema.Parser parser_stream = new Schema.Parser();
        Schema schema_stream = parser_stream.parse(Schemas.review_stream_schema);
        Schema.Parser parser = new Schema.Parser();
        Schema schema = parser.parse(Schemas.review_schema);

        FileInputStream fis;
        BufferedReader br = null;
        try {
            fis = new FileInputStream(args[1]);
            br = new BufferedReader(new InputStreamReader(fis));

            String line = null;
            ObjectMapper mapper = new ObjectMapper();
```

```
            while ((line = br.readLine()) != null) {

                String hashcode = sha1(line);
                JsonNode reviewJson =  mapper.readTree(line);
                String reviewText = reviewJson.path("text").asText();
                double reviewsStarts = reviewJson.path("stars").asDouble();
                String sentiment_analysis="neutral";
                // We choose either "dummy" or "serious" Sentiment analyzer
                switch (args[3]){
                    case "dummy":
                        if  (reviewsStarts>=3) {
                            sentiment_analysis = "positive";
                        } else {
                            sentiment_analysis = "negative";
                        }
                        break;
                    case "serious":
                        int sentiment = analyser.findSentiment(reviewText);

                        switch (sentiment){
                            case 1:
                                sentiment_analysis = "positive";
                                break;
                            case 0:
                                sentiment_analysis = "negative";
                                break;
                            default:
                                sentiment_analysis = "neutral";
                        }
                        break;
                    default :
                        System.out.println("Choose dummy or serious sentiment
analysis");

                        System.exit(1);
                }

                // Send data to topic for Persistence Storage
                ((ObjectNode) reviewJson).put("sentiment",sentiment_analysis);
                GenericRecord review = new GenericData.Record(schema);
                review.put("review_id",hashcode);
                review.put("review", reviewJson.toString());
                ProducerRecord<String, GenericRecord> data = new
ProducerRecord<String, GenericRecord>(topic, review);
                producer.send(data);

                //Send data to topic for Stream Analytics
                GenericRecord review_stream = new
GenericData.Record(schema_stream);
                review_stream.put("review",reviewJson.toString());
                ProducerRecord<String, GenericRecord> data_stream = new
ProducerRecord<String, GenericRecord>(topic_stream, review_stream);
                producer.send(data_stream);

                if (args.length == 5) {
                    Thread.sleep(Integer.parseInt(args[4]));
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        producer.close();
    }

    public static String sha1(String input) throws NoSuchAlgorithmException {
```

```java
        MessageDigest mDigest = MessageDigest.getInstance("SHA1");
        byte[] result = mDigest.digest(input.getBytes());
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < result.length; i++) {
            sb.append(Integer.toString((result[i] & 0xff) + 0x100,
16).substring(1));
        }
        return sb.toString();
    }

}
```

ProducerTip.java

```java
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import gr.ntua.cep.sentiment.SentimentAnalysisException;
import gr.ntua.cep.sentiment.SentimentAnalysisImp;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericRecord;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.io.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Properties;

/*
*  This is the code of the Kafka producer for tips and tips_stream topics
* */

public class ProducerTip {

    public static void main(String[] args, SentimentAnalysisImp analyser) throws
IOException, InterruptedException, SentimentAnalysisException {

        if (args.length != 4 && args.length != 5) {
            System.out.println(args.length+"");
            System.out.println("Please provide command line arguments for
Producer: input_file topic_name \"dummy\"|\"serious\" [time_delay(ms)]");
            System.exit(1);
        }
        String topic = args[2];
        String topic_stream = args[2] + "_stream";
        String url = "http://147.102.19.1:8081";
        Properties props = new Properties();
        props.put("bootstrap.servers", "147.102.19.1:9092");
        props.put("acks", "all");
        props.put("retries", "0");
        props.put("batch.size", "16384");
        props.put("auto.commit.interval.ms", "1000");
        props.put("linger.ms", "0");
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
"io.confluent.kafka.serializers.KafkaAvroSerializer");
        props.put("block.on.buffer.full", "true");
        props.put("schema.registry.url", url);

        // Parsers for the Dsta schema
        KafkaProducer<String, GenericRecord> producer = new KafkaProducer<String,
GenericRecord>(props);
        Schema.Parser parser_stream = new Schema.Parser();
```

```
        Schema schema_stream = parser_stream.parse(Schemas.tip_stream_schema);
        Schema.Parser parser = new Schema.Parser();
        Schema schema = parser.parse(Schemas.tip_schema);

        FileInputStream fis;
        BufferedReader br = null;
        try {
            fis = new FileInputStream(args[1]);
            br = new BufferedReader(new InputStreamReader(fis));

            String line = null;
            ObjectMapper mapper = new ObjectMapper();
            while ((line = br.readLine()) != null) {

                // Send data to topic for Persistence Storage
                String hashcode = sha1(line);
                JsonNode tipJson =  mapper.readTree(line);
                String tipText = tipJson.path("text").asText();
                double reviewsStarts = tipJson.path("stars").asDouble();
                // We choose either "dummy" or "serious" Sentiment analyzer
                String sentiment_analysis="neutral";
                switch (args[3]){
                    case "dummy":
                        if  (reviewsStarts>=3) {
                            sentiment_analysis = "positive";
                        } else {
                            sentiment_analysis = "negative";
                        }
                        break;
                    case "serious":
                        int sentiment = analyser.findSentiment(tipText);

                        switch (sentiment){
                            case 1:
                                sentiment_analysis = "positive";
                                break;
                            case 0:
                                sentiment_analysis = "negative";
                                break;
                            default:
                                sentiment_analysis = "neutral";
                        }
                        break;
                    default :
                        System.out.println("Choose dummy or serious sentiment
analysis");

                        System.exit(1);
                }

                // Send data to topic for Persistence Storage
                ((ObjectNode) tipJson).put("sentiment",sentiment_analysis);
                GenericRecord tip = new GenericData.Record(schema);
                tip.put("tip_id",hashcode);
                tip.put("tip", tipJson.toString());
                ProducerRecord<String, GenericRecord> data = new
ProducerRecord<String, GenericRecord>(topic, tip);
                producer.send(data);

                //Send data to topic for Stream Analytics
                GenericRecord tip_stream = new GenericData.Record(schema_stream);
                tip_stream.put("tip",tipJson.toString());
                ProducerRecord<String, GenericRecord> data_stream = new
ProducerRecord<String, GenericRecord>(topic_stream, tip_stream);
                producer.send(data_stream);

                if (args.length == 5) {
                    Thread.sleep(Integer.parseInt(args[4]));
                }
```

```
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        producer.close();
    }

    public static String sha1(String input) throws NoSuchAlgorithmException {
        MessageDigest mDigest = MessageDigest.getInstance("SHA1");
        byte[] result = mDigest.digest(input.getBytes());
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < result.length; i++) {
            sb.append(Integer.toString((result[i] & 0xff) + 0x100,
16).substring(1));
        }
        return sb.toString();
    }
}
```

Schemas.java

```
public class Schemas {

    public static String bussiness_schema = "{\"namespace\": \"avro.business\",
\"type\": \"record\", " +
            "\"name\": \"businessJson\"," +
            "\"fields\": [" +
            "{\"name\": \"business_id\", \"type\": \"string\"}," +
            "{\"name\": \"business_info\", \"type\": \"string\"}" +
            "]}";

    public static String review_stream_schema = "{\"namespace\":
\"avro.review.stream\", \"type\": \"record\", " +
            "\"name\": \"reviewStream\"," +
            "\"fields\": [" +
            "{\"name\": \"review\", \"type\": \"string\"}" +
            "]}";

    public static String tip_stream_schema = "{\"namespace\": \"avro.tip.stream\",
\"type\": \"record\", " +
            "\"name\": \"tipStream\"," +
            "\"fields\": [" +
            "{\"name\": \"tip\", \"type\": \"string\"}" +
            "]}";
    public static String review_schema = "{\"namespace\": \"review.avro\",
\"type\": \"record\", " +
            "\"name\": \"reviewJson\"," +
            "\"fields\": [" +
            "{\"name\": \"review_id\", \"type\": \"string\"}," +
            "{\"name\": \"review\", \"type\": \"string\"}" +
            "]}";
    public static String tip_schema = "{\"namespace\": \"tip.avro\", \"type\":
\"record\", " +
            "\"name\": \"tipJson\"," +
            "\"fields\": [" +
            "{\"name\": \"tip_id\", \"type\": \"string\"}," +
            "{\"name\": \"tip\", \"type\": \"string\"}" +
            "]}";
}
```

Για να γίνει σωστά το compile αυτού του maven project χρειαζόμαστε και το αρχείο pom

producer.pom

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>file-producer</groupId>
    <artifactId>producer-test</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <repositories>

        <repository>
            <id>confluent</id>
            <url>http://packages.confluent.io/maven/</url>
        </repository>
        <repository>
            <id>Internal repository</id>
            <url>file://${basedir}/lib</url>
        </repository>

        <!-- further repository entries here -->

    </repositories>

    <dependencies>

        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.11</artifactId>
            <!-- For CP 3.2.0 -->
            <version>0.10.2.0-cp1</version>
        </dependency>

        <dependency>
            <groupId>io.confluent</groupId>
            <artifactId>kafka-avro-serializer</artifactId>
            <!-- For CP 3.2.0 -->
            <version>3.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro</artifactId>
            <version>1.8.1</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-core</artifactId>
            <version>2.9.0.pr2</version>
        </dependency>
        <dependency>
            <groupId>com.sentiment</groupId>
            <artifactId>sentiment</artifactId>
            <version>1.0</version>
        </dependency>
        <!-- further dependency entries here -->

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>2.6</version>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>Run</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.skife.maven</groupId>
                <artifactId>really-executable-jar-maven-plugin</artifactId>
                <version>1.5.0</version>
                <configuration>
                    <!-- value of flags will be interpolated into the java
invocation -->
                    <!-- as "java $flags -jar ..." -->
                    <!--<flags></flags>-->

                    <!-- (optional) name for binary executable, if not set will
just -->
                    <!-- make the regular jar artifact executable -->
                    <programFile>Json_Producer_Consumer</programFile>
                </configuration>

                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>really-executable-jar</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

## B.2 Batch Applications

HbaseSpark.scala

```
import org.apache.hadoop.hbase.client._
import org.apache.hadoop.hbase._
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.spark._

/*
    Batch processing using Spark
* */
object HbaseSpark {

  def main(args: Array[String]) {
    if (args.length == 0) {
      println("dude, I need at least one parameter")
    }
    val sparkConf = new SparkConf().setAppName("SparkQuery").setMaster("yarn")
    val sc = new SparkContext(sparkConf)
    val conf = HBaseConfiguration.create()
    val name1 = "business"
    val name2 = "reviews"
    val name3 = "tips"

    //hbase parameters
    System.setProperty("user.name", "root")
    System.setProperty("HADOOP_USER_NAME", "root")
    conf.set("hbase.zookeeper.quorum", "147.102.19.1,147.102.19.88,147.102.19.32")
    conf.setInt("hbase.zookeeper.property.clientPort", 2181)
    conf.set("hbase.rpc.timeout", "1800000")
    conf.set("hbase.client.scanner.timeout.period", "1800000")
    conf.set("zookeeper.znode.parent", "/hbase")

    conf.set(TableInputFormat.INPUT_TABLE, name1)
    //connection to hbase
    val conn = ConnectionFactory.createConnection(conf);
    val admin = conn.getAdmin();
    val table = TableName.valueOf(name1)
    if (!admin.isTableAvailable(table)) {
      val tableDesc = new HTableDescriptor(table)
      admin.createTable(tableDesc)
    }
    //read table business from hbase and put it in an RDD
    val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
classOf[ImmutableBytesWritable], classOf[Result])
    val data = hBaseRDD.map(kv => MyFunctions.navMapToMap(kv._2.getMap))
    val data_business = data.map(x =>
MyFunctions.BusinessToString(x)).flatMap(identity)
    conn.close()

    //choose either "query1" or "query2"
    args(0) match {
      case "query1" =>  MyQueries.top_categories_by_state(data_business)
      case "query2" =>  {

        // query2 chosen
        conf.set(TableInputFormat.INPUT_TABLE, name2)
        val conn2 = ConnectionFactory.createConnection(conf);
        //connection to hbase
        val admin2 = conn2.getAdmin();
        val table2 = TableName.valueOf(name2)
        if (!admin2.isTableAvailable(table2)) {
          val tableDesc2 = new HTableDescriptor(table2)
          admin2.createTable(tableDesc2)
```

```
        }
        //read table reviews from hbase
        val hBaseRDD2 = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
classOf[ImmutableBytesWritable], classOf[Result])
        val data2 = hBaseRDD2.map(kv => MyFunctions.navMapToMap(kv._2.getMap))
        val data_review = data2.map(x =>
MyFunctions.ReviewToString(x)).flatMap(identity)
        conn2.close()
        MyQueries.most_positive_cities(data_business, data_review)
      }
      case whoa => println("Choose either query1 or query2")
    }

    sc.stop()

  }

}
```

## MyQueries.scala

```
import org.apache.spark.rdd.RDD
import org.apache.spark.sql._

/*
 Queries on RDDs
* */
object MyQueries {

  //Query1
  def top_categories_by_state (business_rdd: RDD[String]): Unit = {
    val sqlContext =
SparkSession.builder.config(business_rdd.sparkContext.getConf).getOrCreate()
    val df_table = sqlContext.read.json(business_rdd)
    df_table.createOrReplaceTempView("df_table")
    val CategoryCountByStateQuery = """SELECT
                                   | tempTable.state, tempTable.category,
COUNT(tempTable.category) categoryCount
                                   | FROM (SELECT state, EXPLODE(categories)
category FROM df_table ) tempTable
                                   | GROUP BY tempTable.state, tempTable.category
                                   | ORDER BY tempTable.state, categoryCount DESC
                                   | """.stripMargin
    val CategoryCountByState = sqlContext.sql(CategoryCountByStateQuery)
    //CategoryCountByState.show()

CategoryCountByState.coalesce(1).write.option("header",true).csv("/data/Categories
ByState")
  }

  // Query2
  def most_positive_cities (business_rdd: RDD[String], review_rdd: RDD[String]):
Unit = {
    val sqlContext =
SparkSession.builder.config(business_rdd.sparkContext.getConf).getOrCreate()
    val business = sqlContext.read.json(business_rdd).cache()
    val review = sqlContext.read.json(review_rdd).cache()
    business.createOrReplaceTempView("business")
    review.createOrReplaceTempView("review")
    val PositiveCitiesQuery = """SELECT
                             |  tempTable.city,
                             |  COUNT(CASE WHEN
tempTable.PositiveCount>tempTable.NegativeCount THEN 1 END) AS PositiveCompanies,
                             |  COUNT(CASE WHEN
tempTable.PositiveCount<tempTable.NegativeCount THEN 1 END) AS NegativeCompanies,
```

```
                             |  COUNT(CASE WHEN
tempTable.PositiveCount=tempTable.NegativeCount THEN 1 END) AS NeutralCompanies
                             |  FROM (
                             |    SELECT business.name, business.city,
                             |    COUNT(CASE WHEN review.sentiment='positive' THEN
1 END) AS PositiveCount,
                             |    COUNT(CASE WHEN review.sentiment='negative' THEN
1 END) AS NegativeCount
                             |    FROM business, review
                             |    WHERE business.business_id = review.business_id
                             |    GROUP BY business.city, business.name
                             |  ) tempTable
                             |  GROUP BY tempTable.city
                             |  ORDER BY PositiveCompanies DESC
                             | """.stripMargin
    val PositiveCities = sqlContext.sql(PositiveCitiesQuery)
    //PositiveCities.show()

PositiveCities.coalesce(1).write.option("header",true).csv("/data/PositiveCities")
  }

}
```

## MyFunctions.scala

```
import org.apache.hadoop.hbase.protobuf.ProtobufUtil
import org.apache.hadoop.hbase.client.Scan
import org.apache.hadoop.hbase.util.{Base64, Bytes}

import scala.collection.JavaConverters._
import scala.collection.mutable.ArrayBuffer

/*
Useful function that we used
* */

object MyFunctions {

  def navMapToMap(navMap: MyTypes.HBaseRow): MyTypes.value = {
    mapAsScalaMapConverter(navMap).asScala.map(cf => (cf._1,
mapAsScalaMapConverter(cf._2).asScala
      .map(col => (col._1, mapAsScalaMapConverter(col._2).asScala.map(elem =>
(elem._1.toLong, elem._2))))))
  }

  def BusinessToString(NavMap: MyTypes.value): ArrayBuffer[String] ={
    // Map(CF -> Map(column qualifier -> Map(timestamp -> value)))
    val buf = ArrayBuffer.empty[String]
    NavMap.map(cf => cf._2.filter{ case (col,_) => Bytes.toString(col) ==
"business_info" }.map(col => col._2.map(elem => buf += Bytes.toString(elem._2)) ))
    return buf
  }

  def ReviewToString(NavMap: MyTypes.value): ArrayBuffer[String] ={
    // Map(CF -> Map(column qualifier -> Map(timestamp -> value)))
    val buf = ArrayBuffer.empty[String]
    NavMap.map(cf => cf._2.filter{ case (col,_) => Bytes.toString(col) == "review"
}.map(col => col._2.map(elem => buf += Bytes.toString(elem._2)) ))
    return buf
  }

  def convertScanToString (scan : Scan ) : String = {
    val proto = ProtobufUtil.toScan(scan)
    return Base64.encodeBytes(proto.toByteArray)
  }

}
```

MyTypes.scala

```
import scala.collection.mutable._

/*
 The type of an HBase when read from Hadoop
* */

object MyTypes {

  type HBaseRow = java.util.NavigableMap[Array[Byte],
java.util.NavigableMap[Array[Byte], java.util.NavigableMap[java.lang.Long,
Array[Byte]]]]
  type value = Map[Array[Byte], Map[Array[Byte], Map[Long, Array[Byte]]]]

}
```

Για να γίνει σωστά compile αυτό το sbt project χρειαζόμαστε και το αρχείο build.sbt

build.sbt

```
name := "HbaseSpark"

version := "1.0"

scalaVersion := "2.11.8"

libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.hbase" % "hbase" % "1.2.4"
libraryDependencies += "org.apache.hbase" % "hbase-client" % "1.2.4"
libraryDependencies += "org.apache.hbase" % "hbase-common" % "1.2.4"
libraryDependencies += "org.apache.hbase" % "hbase-server" % "1.2.4"
libraryDependencies += "org.apache.spark" % "spark-sql_2.11" % "2.1.0"
libraryDependencies += "org.json4s" % "json4s-core_2.11" % "3.2.11"
libraryDependencies += "org.json4s" % "json4s-jackson_2.11" % "3.2.11"
```

## B.3 Streaming Application

SparkStream.scala

```
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import io.confluent.kafka.serializers.KafkaAvroDeserializer
import org.apache.avro.generic.GenericRecord
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark._

/*
  Stream Processing using Spark
* */

object SparkStream {

  def main(args: Array[String]): Unit = {

    val sparkConf = new SparkConf().setMaster("yarn").setAppName("SparkStream")
    sparkConf.set("spark.streaming.concurrentJobs", "2")

    //set window time for stream batches
    val ssc = new StreamingContext(sparkConf, Seconds(60))

    // Kafka parameters
    val kafkaParams1 = Map[String, Object](
      "bootstrap.servers" -> "147.102.19.1:9092",
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[KafkaAvroDeserializer],
      "group.id" -> "review",
      "auto.offset.reset" -> "latest",
      "enable.auto.commit" -> (false: java.lang.Boolean),
      "schema.registry.url" -> "http://147.102.19.1:8081"
    )

    val kafkaParams2 = Map[String, Object](
      "bootstrap.servers" -> "147.102.19.1:9092",
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[KafkaAvroDeserializer],
      "group.id" -> "tip",
      "auto.offset.reset" -> "latest",
      "enable.auto.commit" -> (false: java.lang.Boolean),
      "schema.registry.url" -> "http://147.102.19.1:8081"
    )

    val topics_review = Array("reviews_stream")
    val topics_tip = Array("tips_stream")

    // Dstream for tips
    val stream_tip = KafkaUtils.createDirectStream[String, GenericRecord](
      ssc,
      PreferConsistent,
      Subscribe[String, GenericRecord]( topics_tip, kafkaParams2)
    )

    //Dstream for reviews
    val stream_review = KafkaUtils.createDirectStream[String, GenericRecord](
      ssc,
      PreferConsistent,
      Subscribe[String, GenericRecord]( topics_review, kafkaParams1)
    )

    // Process those Dstreamms
```

```
    val dstream_tip =stream_tip.map (record => record.value())
    val dstream_review = stream_review.map (record => record.value())

    TipStream.TipStream(dstream_tip)
    ReviewStream.ReviewStream(dstream_review)

    ssc.start()
    ssc.awaitTermination()

  }

}
```

ReviewStream.scala

```
import org.apache.avro.generic.GenericRecord
import org.apache.spark.streaming.dstream.DStream

/*
  Process Review Ddata stream
* */
object ReviewStream {

  def ReviewStream(review_dstream: DStream[GenericRecord]): Unit ={
    review_dstream.foreachRDD( (rdd,time) => {
      val review = rdd.map( avroRecord => avroRecord.get("review").toString)
      MyQueries.reviewStars(review,time)
    })
  }

}
```

TipStream.scala

```
import org.apache.avro.generic.GenericRecord
import org.apache.spark.streaming.dstream.DStream

/*
  Process Tip data stream
* */
object TipStream {

  def TipStream(tip_dstream: DStream[GenericRecord]): Unit ={
    tip_dstream.foreachRDD( (rdd,time) => {
      val tip = rdd.map( avroRecord => avroRecord.get("tip").toString)
      MyQueries.tipsNum(tip,time)
    })
  }
```

MyQueries.scala

```
import org.apache.spark.rdd.RDD
import org.apache.spark.sql._
import org.apache.spark.streaming.Time

object MyQueries {

  // process of the Tip data stream
  def tipsNum (tips: RDD[String], timestamp : Time): Unit = {
    val sqlContext =
SparkSession.builder.config(tips.sparkContext.getConf).getOrCreate()
    val df_table = sqlContext.read.json(tips).cache()
    //df_table.show()
    if (!df_table.take(1).isEmpty) {
      df_table.createOrReplaceTempView("df_table")
      val tipsQuery =
        """SELECT
          | COUNT(*) as TipCount
          | FROM df_table
          | """.stripMargin
      val TipsCount = sqlContext.sql(tipsQuery)

TipsCount.coalesce(1).write.option("header",true).option("nullValue","null").csv("
/data/TipStream_"+ timestamp)
    }
  }

  //process of the Review data stream
  def reviewStars (reviews: RDD[String], timestamp : Time): Unit = {
    val sqlContext =
SparkSession.builder.config(reviews.sparkContext.getConf).getOrCreate()
    val df_table = sqlContext.read.json(reviews).cache()
    //df_table.show()
    if (!df_table.take(1).isEmpty) {
      df_table.createOrReplaceTempView("df_table")
      val reviewsQuery =
        """SELECT
          | stars, COUNT(stars) as StartsCount
          | FROM df_table
          | GROUP BY stars
          | ORDER BY stars
          | """.stripMargin
      val ReviewsStarsCount = sqlContext.sql(reviewsQuery)

ReviewsStarsCount.coalesce(1).write.option("header",true).option("nullValue","null
").csv("/data/ReviewStream_"+ timestamp)
    }
  }

}
```

Για να γίνει σωστά compile αυτό το sbt project χρειαζόμαστε και το αρχείο build.sbt

build.sbt

```
name := "SparkStream"

version := "1.0"

scalaVersion := "2.11.8"


libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0" % "provided"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0" %
"provided"
libraryDependencies += "org.apache.spark" % "spark-sql_2.11" % "2.1.0" %
"provided"
libraryDependencies += "org.apache.avro" % "avro" % "1.7.7" % "provided"
libraryDependencies += "org.apache.spark" % "spark-streaming-kafka-0-10_2.11" %
"2.1.0"
libraryDependencies += "io.confluent" % "kafka-avro-serializer" % "3.2.0"
libraryDependencies += "com.databricks" % "spark-avro_2.11" % "3.2.0"

resolvers ++= Seq(
  Resolver.sonatypeRepo("public"),
  "Confluent Maven Repo" at "http://packages.confluent.io/maven/"
)

assemblyMergeStrategy in assembly := {
  case PathList("META-INF", xs @ _*) => MergeStrategy.discard
  case x => MergeStrategy.first
}

assemblyExcludedJars in assembly := {
  val cp = (fullClasspath in assembly).value
  cp filter { f =>
    f.data.getName.contains("jackson") ||
    f.data.getName.contains("slf4j") ||
    f.data.getName.contains("avro-1.7.7") ||
    f.data.getName.contains("lz4") ||
    f.data.getName.contains("log4j") ||
    f.data.getName.contains("spark-tags") ||
    f.data.getName.contains("commons-compress") ||
    f.data.getName.contains("scala-") ||
    f.data.getName.contains("scala-xml") ||
    f.data.getName.contains("xz") ||
    f.data.getName.contains("paranamer") ||
    f.data.getName.contains("metrics-core") ||
    f.data.getName.contains("zookeeper") ||
    f.data.getName.contains("snappy-java") ||
    f.data.getName.contains("netty")

  }
}
```