ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη και σύγκριση τεχνολογιών εξόρυξης γνώσης από δεδομένα ροής σε πραγματικό χρόνο – Η περίπτωση της Streaming SQL

Διπλωματική Εργασία

του

Γράβα Χριστόφορου

Θεσσαλονίκη , Νοέμβριος 2020

Μελέτη και σύγκριση τεχνολογιών εξόρυξης γνώσης
από δεδομένα ροής σε πραγματικό χρόνο –
Η περίπτωση της Streaming SQL

Γράβας Χριστόφορος

Πτυχίο Τηλεπληροφορικής & Διοίκησης , 2007

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Ευαγγελίδης Γεώργιος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 3/11/2020

Ευαγγελίδης Γεώργιος	Κολωνιάρη Γεωργία	Γεωργιάδης Χρήστος
.....	.....	.....

Γράβας Χριστόφορος

.....

Περίληψη

Στην παρούσα διπλωματική εργασία παρουσιάζεται μια μελέτη των τεχνολογιών Apache Kafka, KSQL, Apache Flink, Apache Samza, Apache Spark, και το πρότυπο Lambda Architecture. Στη συνέχεια γίνεται μια σύγκριση μεταξύ τους, παρουσιάζοντας την καταλληλότερη για πιο αποτελεσματική επεξεργασία δεδομένων ροής. Για την εξαγωγή συμπερασμάτων καθώς και για την πειραματική μελέτη/σύγκρισή τους έγινε χρήση ροών δεδομένων μεγάλης κλίμακας. Για την ανάλυση των παραπάνω τεχνολογιών χρησιμοποιήθηκε η Streaming SQL η οποία αποτελεί γλώσσα ερωτημάτων που επεκτείνει την SQL με ικανότητα επεξεργασίας ροών δεδομένων σε πραγματικό χρόνο, προσθέτει την ικανότητα χειρισμού ροών δεδομένων, οι οποίες αποτελούν άπειρες ακολουθίες πλειάδων που δεν είναι όλες διαθέσιμες ταυτόχρονα. Για το λόγο αυτό, οι πράξεις πάνω σε αυτά τα δεδομένα πρέπει να είναι μονοτονικές και τα ερωτήματα πάνω στις ροές δεδομένων είναι γενικά "συνεχή" , εκτελούνται για μεγάλες χρονικές περιόδους και επιστρέφουν βαθμιαία αποτελέσματα.

Η Streaming SQL, χρησιμοποιείται συνήθως στο πλαίσιο ενός συστήματος διαχείρισης ροών δεδομένων (Data Stream Management System - DSMS), για εφαρμογές που περιλαμβάνουν αλγοριθμικές συναλλαγές, δεδομένα παιχνιδιών, αναλύσεις δεδομένων αγοράς, παρακολούθηση δικτύου, ανίχνευση, παρακολούθηση και πρόληψη ηλεκτρονικής απάτης, αναλύσεις clickstream και συμμόρφωση σε πραγματικό χρόνο.

Λέξεις Κλειδιά: SQL Streaming, Apache Streaming processing frameworks, Apache Kafka, KSQL, Apache Spark, Spark SQL, Apache Samza, Apache Flink, Lambda Architecture, Apache Storm.

Abstract

This master thesis presents a study of Apache Kafka technologies and KSQL, Apache Flink, Apache Samza, Apache Spark, and the Lambda Architecture standard. They were then compared to each other, presenting the most suitable for more efficient processing of flow data. Large-scale data flows were used to draw conclusions as well as for experimental study / comparison. Streaming SQL was used to analyze the above technologies, which is a query language that extends SQL with the ability to process data in real time, adding the ability to handle data, which are infinite sequences of blocks that are not all available at the same time. Because the feeds are infinite, the operations on this data must be monotonous and the queries on the feeds are generally "continuous" and run for long periods of time and return gradual results.

Streaming SQL is commonly used in a Data Stream Management System (DSMS) for applications including algorithmic transactions, game data, market data analysis, network monitoring, detection, monitoring and prevention of fraud, clickstream analysis and real-time compliance.

Keywords: SQL Streaming, Apache Streaming processing frameworks, Apache Kafka, KSQL, Apache Spark, Spark SQL, Apache Samza, Apache Flink, Lambda Architecture, Apache Storm.

Πίνακας περιεχομένων

Πίνακας εικόνων

Πίνακας Γραφημάτων

1 Τι είναι το stream processing;

Το stream processing ή αλλιώς η επεξεργασία δεδομένων ροής είναι μια τεχνολογία που εφαρμόζεται σε μεγάλους όγκους δεδομένων (Big Data). Χρησιμοποιείται για την υποβολή ερωτημάτων σε συνεχούς ροής δεδομένα και την ανίχνευση συνθηκών, εντός μικρού χρονικού διαστήματος από τη στιγμή λήψης αυτών των δεδομένων. Η χρονική περίοδος ανίχνευσης κυμαίνεται από λίγα χιλιοστά του δευτερολέπτου μέχρι και μερικά λεπτά. Για παράδειγμα, με την επεξεργασία δεδομένων ροής, μπορείτε να λάβετε μια ειδοποίηση όταν η θερμοκρασία ενός υγρού έχει φτάσει στο σημείο πήξης, διερευνώντας ακολουθίες δεδομένων που προέρχονται από έναν αισθητήρα θερμοκρασίας.

Η επεξεργασία δεδομένων ροής αναφέρεται επίσης με πολλά άλλα ονόματα, όπως ανάλυση σε πραγματικό χρόνο (real-time analysis και real-time streaming analytics), ανάλυση ροής (streaming analytics, σύνθετη επεξεργασία συμβάντων (Complex Event Processing), και επεξεργασία συμβάντων (event processing). Παρόλο που κάποιοι όροι έχουν σημαντικές διαφορές, τα σύγχρονα εργαλεία που χρησιμοποιούμε, όπως τα διάφορα frameworks , συγκλίνουν στον όρο επεξεργασία ροής (stream processing).

1.1 Ποια η χρησιμότητα του stream processing;

Η εμφάνιση του μεγάλου όγκου δεδομένων τα τελευταία χρόνια , γνωστά με τον όρο Big Data, καθόρισαν την αξία της πληροφορίας που προέρχεται από την επεξεργασία δεδομένων. Δεν έχουν όμως όλες οι πληροφορίες την ίδια σημαντικότητα και σε αυτό παίζει ρόλο ο χρόνος στον οποίο προκύπτουν, καθώς πολλές από αυτές τις πληροφορίες πρέπει να αξιοποιηθούν τη στιγμή που προκύπτουν τα δεδομένα, διότι μετά από μικρό χρονικό διάστημα, ολίγων δευτερολέπτων ίσως και ολίγων λεπτών, ενδέχεται να έχουν χάσει τη σημασία τους. Το stream processing επιτρέπει ταχύτατα σενάρια επεξεργασίας δεδομένων, παρέχοντας πληροφορίες πολύ γρήγορα, συχνά σε διάστημα χιλιοστών του δευτερολέπτου έως και ενός δευτερολέπτου από την δημιουργία αυτών των δεδομένων.

Οι προκλήσεις που έχει να αντιμετωπίσει και που δίνει λύσεις το stream processing είναι πολλές και διαφορετικές:

- Επεξεργασία τεράστιων ποσοτήτων συμβάντων ροής.
- Ανταπόκριση σε πραγματικό χρόνο στις μεταβαλλόμενες συνθήκες της αγοράς.
- Απόδοση και επεκτασιμότητα καθώς οι όγκοι δεδομένων αυξάνουν σε μέγεθος και πολυπλοκότητα.
- Ταχεία ενοποίηση με υπάρχουσες υποδομές και πηγές δεδομένων: Δεδομένα εισόδου (π.χ. δεδομένα αγοράς, δεδομένα εισόδου χρήστη, αρχεία, δεδομένα ιστορικού από DWH) και δεδομένα εξόδου (π.χ. συναλλαγές, ειδοποιήσεις μέσω email, πίνακες εργαλείων, αυτοματοποιημένες αντιδράσεις).
- Γρήγορη ανάπτυξη εφαρμογών λόγω της γρήγορης αλλαγής τοπίου και των απαιτήσεων της αγοράς.
- Μεγαλύτερη παραγωγικότητα προγραμματιστών σε όλα τα στάδια του κύκλου ζωής ανάπτυξης εφαρμογών προσφέροντας καλή υποστήριξη εργαλείων και ευέλικτη ανάπτυξη.
- Ανάλυση και παρακολούθηση δεδομένων ροής , συνεχή επεξεργασία ερωτημάτων, αυτοματοποιημένες ειδοποιήσεις και αντιδράσεις.
- Οπτικοποίηση δεδομένων.

Επίσης, κάποιοι περαιτέρω λόγοι χρήσης του stream processing είναι και οι παρακάτω:

1. **Ατέρμονες ροές δεδομένων (Never-ending streams).**

Ορισμένα δεδομένα προέρχονται ως μια ατελείωτη ροή συμβάντων. Για να κάνουμε μαζική επεξεργασία αυτών, πρέπει να τα αποθηκεύσουμε, να διακόψουμε κάποια στιγμή τη συλλογή δεδομένων και να επεξεργαστούμε τα δεδομένα που έχουμε συλλέξει. Στη συνέχεια, πρέπει να δημιουργήσουμε την επόμενη παρτίδα δεδομένων και να αρχίσουμε να ανησυχούμε για τη συγκέντρωση των δεδομένων αυτών σε διαφορετικές πλειάδες. Το stream processing δεν χρειάζεται να αντιμετωπίσει τα θέματα που αναφέρθηκαν παραπάνω καθώς χειρίζεται ατέρμονες ροές δεδομένων. Εφαρμόζοντας την τεχνική αυτή μπορούμε να εντοπίσουμε μοτίβα, να ελέγξουμε αποτελέσματα, να εντοπίσουμε πολλαπλά επίπεδα εστίασης και φυσικά να επεξεργαστούμε εύκολα δεδομένα που προκύπτουν από πολλές ροές ταυτόχρονα.

Η επεξεργασία ροής εναρμονίζεται τέλεια με δεδομένα χρονοσειρών και ανίχνευση προτύπων με την πάροδο του χρόνου. Για παράδειγμα, έστω ότι προσπαθούμε να

εντοπίσουμε τη διάρκεια περιήγησης ενός χρήστη σε μια ατέρμονη ροή δεδομένων, μέσω εντοπισμού μιας ακολουθίας, είναι πολύ δύσκολο να το κάνουμε με παρτίδες, καθώς κάποια περίοδος συνεδρίας θα χωριστεί σε δύο παρτίδες. Την συγκεκριμένη περίπτωση η επεξεργασία ροής μπορεί να τη χειριστεί εύκολα.

Αναλύοντας περαιτέρω το προηγούμενο παράδειγμα συμπεραίνουμε, ότι τα δεδομένα χρονοσειρών είναι οι πιο συνεχείς σειρές δεδομένων. Τέτοια δεδομένα λαμβάνουμε από αισθητήρες κίνησης, αισθητήρες υγείας, αρχεία καταγραφής συναλλαγών, αρχεία καταγραφής δραστηριοτήτων κ.λπ. Σχεδόν όλα τα δεδομένα IoT είναι δεδομένα χρονοσειρών.

2. Εξοικονόμηση υπολογιστικών πόρων

Η επεξεργασία δεδομένων σε δέσμες εφαρμόζει συσσώρευση δεδομένων και στη συνέχεια τα επεξεργάζεται, σε αντίθεση με την επεξεργασία ροής η οποία τα επεξεργάζεται καθώς εισέρχονται, επομένως κατανέμει την επεξεργασία με την πάροδο του χρόνου. Ως εκ τούτου, η επεξεργασία ροής μπορεί να λειτουργήσει με πολύ λιγότερους υπολογιστικούς πόρους από την επεξεργασία σε δέσμες. Επιπλέον, η επεξεργασία ροής επιτρέπει επίσης κατά προσέγγιση επεξεργασία ερωτημάτων μέσω συστηματικής αποφόρτισης και εφαρμόζεται πιο αποτελεσματικά σε περιπτώσεις χρήσης όπου αρκούν οι κατά προσέγγιση απαντήσεις ερωτημάτων.

3. Λιγότερη αναγκαιότητα αποθήκευσης δεδομένων

Μερικές φορές τα δεδομένα είναι τεράστια και δεν είναι καν δυνατή η αποθήκευσή τους. Η επεξεργασία ροής επιτρέπει να χειριζόμαστε μεγάλα δεδομένα και να διατηρεί μόνο τα χρήσιμα κομμάτια.

4. **Εφαρμογή σε μεγάλη γκάμα δεδομένων**

Υπάρχουν πολλά διαθέσιμα δεδομένα ροής (π.χ. συναλλαγές πελατών, δραστηριότητες, επισκέψεις σε ιστότοπους) και με το πέρασμα των χρόνων θα αυξηθούν γρηγορότερα με την χρήση του IoT (και με την εμφάνιση όλο και περισσότερο ειδών αισθητήρων). Το stream processing αποτελεί ένα πολύ πιο φυσικό μοντέλο για αυτές τις περιπτώσεις χρήσης.

1.2 Ποιος χρησιμοποιεί το stream processing;

Σε γενικές γραμμές, η επεξεργασία ροής είναι χρήσιμη σε περιπτώσεις όπου μπορούμε να εντοπίσουμε ένα πρόβλημα και έχουμε μια λογική απάντηση για να βελτιώσουμε το αποτέλεσμα. Επίσης, παίζει βασικό ρόλο σε οργανισμούς και επιχειρήσεις που ασχολούνται κατά κύριο λόγο με δεδομένα.

Μερικά παραδείγματα εφαρμογής, όπως αναφέρονται στο άρθρο του Srinath Perera (Perera, 2018):

- Φροντίδα ασθενών.
- Αλγοριθμικές συναλλαγές, παρακολούθηση χρηματιστηριακών αγορών.
- Παρακολούθηση γραμμής παραγωγής.
- Βελτιστοποίηση εφοδιαστικής αλυσίδας.
- Προώθηση και διαφήμιση με γνώμονα τις κινήσεις των καταναλωτών.
- Παραβιάσεις, παρακολούθηση της κυκλοφορίας και εντοπισμός επιθέσεων απάτης (π.χ. Uber).
- Εφαρμογές έξυπνων συσκευών (π.χ. έξυπνα αυτοκίνητα).
- Έξυπνα δίκτυα (π.χ. πρόβλεψη φορτίου και ανίχνευση άτυπης λειτουργίας, 4 δισεκατομμύρια συμβάντα, σε εύρος 100Ks).
- Παρακολούθηση υπολογιστικών συστημάτων και δικτύων υπολογιστών.
- Προγνωστική συντήρηση.
- Παρακολούθηση Geofencing, διαφόρων οχημάτων καθώς και παρακολούθησης άγριας ζωής (π.χ. TFL - Transport for London).
- Ανάλυση αθλητικών δεδομένων σε πραγματικό χρόνο (π.χ. ανάλυση ποδοσφαιρικών αγώνων).

1.3 Πώς εφαρμόζουμε το stream processing;

Η απάντηση έγκειται στην πολυπλοκότητα του αλγορίθμου που θέλουμε να εφαρμόσουμε, στην κλιμάκωση της επεξεργασίας, στην αξιοπιστία και την ανοχή σε σφάλματα κτλ.

Το stream processing λαμβάνει διάφορα συμβάντα από μια ροή δεδομένων, τα αναλύει, και δημιουργεί νέα συμβάντα σε νέες ροές. Για να συμβεί αυτό χρειαζόμαστε συμβάντα, τα λεγόμενα events, τα οποία ουσιαστικά αποτελούν μια γραμμή δεδομένων. Όπως αναφέρθηκε και προηγουμένως, μια πηγή συμβάντων μπορεί να αποτελέσει ένας αισθητήρας θερμοκρασίας περιβάλλοντος, ο οποίος ωθεί τα συμβάντα σε εμάς ή κάποιο κομμάτι κώδικα που περιοδικά αντλεί συμβάντα από την πηγή. Ένα χρήσιμο εργαλείο για αυτή τη διεργασία είναι μια ουρά μηνυμάτων, την οποία μπορούμε να σκεφτούμε ως μια δεξαμενή η οποία κρατάει τα δεδομένα προς επεξεργασία.

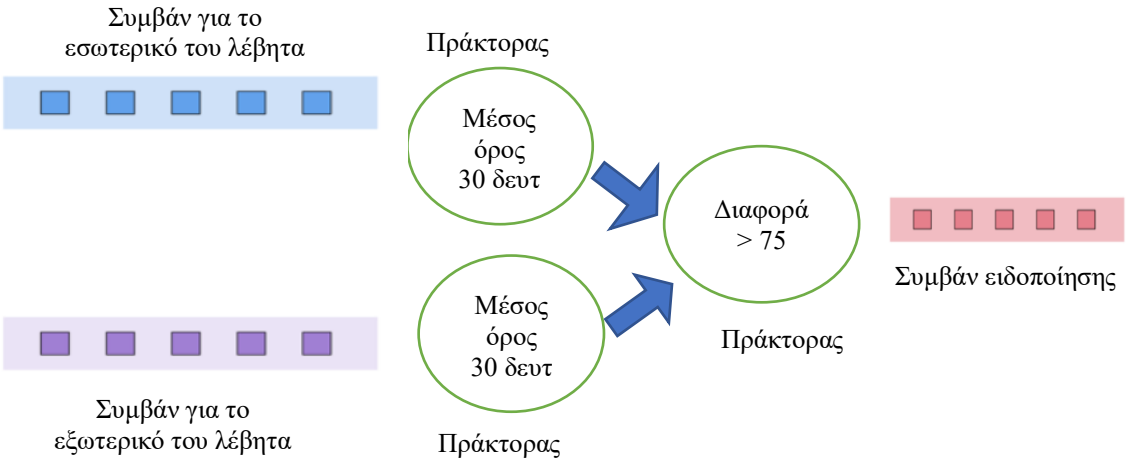
Αφού έχουμε κρατήσει σε κάποιο αποθηκευτικό μέσο τα δεδομένα μας, θα πρέπει να τα εκμεταλλευτούμε. Παλαιότερα θα χρειαζόταν να συντάξουμε κώδικα για να συνδέσουμε την επεξεργασία των δεδομένων με την πηγή των συμβάντων και στη συνέχεια να επεξεργαστούμε ένα-ένα τα συμβάντα. Για την επεξεργασία αυτή πλέον χρειαζόμαστε έναν actor ή αλλιώς agent ο οποίος θα δέχεται διαδοχικά τα συμβάντα (βλ. Εικόνα 1).

Εικόνα 1: Agent
πηγή: https://wso2.com/files/field/image/introduction-to-streaming-analytics-1.png

Για να κατανοήσουμε περαιτέρω την ιδέα αυτή ας εξετάσουμε ένα λέβητα που έχει έναν αισθητήρα για τη μέτρηση της θερμοκρασίας του. Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να εντοπίσουμε πότε η ροή θερμοκρασίας από το λέβητα είναι μεγαλύτερη

από 350ο C. Αυτή η νέα ροή δεδομένων μπορεί να χρησιμοποιηθεί από έναν άλλο πράκτορα για να δημιουργήσει μια ακόμη ροή. Στην πραγματικότητα, οι agents μπορεί να λειτουργήσουν πιο περίπλοκα σε λειτουργίες όπου θα δέχονται πολλές ροές δεδομένων για να δημιουργήσουν μια σύνθετη ροή δεδομένων. Όταν συνδέουμε πολλαπλές ροές με πράκτορες, λαμβάνουμε ως αποτέλεσμα ένα διάγραμμα επεξεργασίας. Στη συνέχεια τα συμβάντα ρέουν μέσα από τη διεργασία του διαγράμματος αυτού και παράγουν νέα συμβάντα. (WSO2, 2019)

Ας υποθέσουμε ότι υπολογίζουμε τον μέσο όρο λειτουργίας 30 δευτερολέπτων σε μια ροή θερμοκρασίας μέσα στο λέβητα και έξω από το λέβητα και θέλουμε να δημιουργήσουμε ένα συμβάν όταν οι τιμές των δύο θερμοκρασιών έχουν μεγάλη διαφορά. Το γράφημα για τον υπολογισμό αυτού του ερωτήματος θα έχει ως εξής (βλ Εικόνα 2):



Εικόνα 2: Γράφημα παραγωγής νέας ροής δεδομένων από πολλαπλά events
πηγή: https://wso2.com/files/field/image/introduction-to-streaming-analytics-2.png

Στο παραπάνω διάγραμμα παρατηρούμε ότι οι πράκτορες θα πρέπει να διατηρούν τιμές οι οποίες προέρχονται από προηγούμενα συμβάντα (π.χ. άθροισμα και πλήθος μετρήσεων). Σε περίπτωση που το σύστημα χάσει τα δεδομένα αυτά θα αποτύχει και γι' αυτό το λόγο χρειαζόμαστε ένα πρότυπο το οποίο θα μας βοηθήσει να αποθηκεύσουμε και

να ανακτήσουμε αυτά τα δεδομένα ροής. Τη λειτουργία αυτή μας την προσφέρει ένας επεξεργαστής δεδομένων ροής.

Οι επεξεργαστές ροής δεδομένων μας επιτρέπουν να ελέγξουμε τους πράκτορες που συμμετέχουν, να τους ενοποιήσουμε συνδέοντας τους με τις πηγές παραγωγής δεδομένων. Οι επεξεργαστές επίσης, συλλέγουν τα δεδομένα, στη συνέχεια τα διαμοιράζουν στους πράκτορες ελέγχοντας ταυτόχρονα τη σωστή σειρά διαμοιρασμού, συλλέγουν τα δεδομένα, ελέγχουν αστοχίες και είναι υπεύθυνοι για την κλιμάκωση υψηλού φορτίου δεδομένων.

Για την παραγωγή τώρα μιας εφαρμογής η οποία εφαρμόζει stream processing χρειαζόμαστε frameworks που είναι κατάλληλα για αυτό. Τα frameworks που πρόκειται να αναλύσουμε στα επόμενα κεφάλαια είναι τα εξής:

- Apache Kafka
- Apache Shamza
- Apache Spark
- Apache Flink
- Apache Storm

Τα παραπάνω framewrorks εφαρμόζουν το πρότυπο Lambda Architecture και την Streaming SQL.

Ο Nathan Marz, περίπου το 2011, παρήγαγε τον όρο Lambda Architecture (LA) με βάση την εμπειρία του σε κατανεμημένα συστήματα επεξεργασίας δεδομένων στο Backtype και στο Twitter. Το LA στοχεύει στην ικανοποίηση των αναγκών για ένα στιβαρό σύστημα που είναι ανεκτικό σε σφάλματα, τόσο έναντι βλαβών υλικού όσο και ανθρώπινων λαθών, είναι σε θέση να εξυπηρετεί ένα ευρύ φάσμα φορτίων εργασίας και περιπτώσεων χρήσης, στις οποίες απαιτούνται ενημερώσεις χαμηλού λανθάνοντος χρόνου.

Η αρχιτεκτονική Lambda είναι μια αρχιτεκτονική επεξεργασίας δεδομένων που έχει σχεδιαστεί για να χειρίζεται τεράστιες ποσότητες δεδομένων, εκμεταλλευόμενο τις μεθόδους επεξεργασίας δεδομένων δέσμης και ροής. Το συγκεκριμένο πρότυπο αρχιτεκτονικής προσπαθεί να εξισορροπήσει τον λανθάνοντα χρόνο, την απόδοση και την ανοχή σφαλμάτων χρησιμοποιώντας την επεξεργασία δεδομένων δέσμης για να παρέχει ολοκληρωμένες και ακριβείς προβολές των δεδομένων αυτών, ενώ ταυτόχρονα χρησιμοποιεί και επεξεργασία ροής σε πραγματικό χρόνο για την παροχή προβολής δεδομένων στο διαδίκτυο. Οι δύο αυτοί τρόποι επεξεργασίας ενοποιούνται και παρουσιάζεται το τελικό αποτέλεσμα.

Από το 2016, μια νέα ιδέα με το όνομα Streaming SQL έκανε την εμφάνισή της η οποία δίνει τη δυνατότητα στους χρήστες να θέσουν ερωτήματα SQL σε δεδομένα ροής. Από το 2018, οι περισσότεροι επεξεργαστές ροής υποστηρίζουν τη δυνατότητα υποβολής ερωτημάτων στα δεδομένα που επεξεργάζονται.

2 Δεδομένα Ροής – Streams

Με τον όρο δεδομένα αναφερόμαστε σε δυο ειδών μορφής δεδομένα: τα δεδομένα πεπερασμένου μεγέθους (bounded data) και τα απεριόριστου μεγέθους (unbounded data). Ο τρόπος με τον οποίο μπορούμε να αλληλοεπιδράσουμε επάνω σε αυτά τα δεδομένα είναι η εφαρμογή ερωτημάτων. Στο σημείο αυτό κάνουν την εμφάνισή τους δύο βασικές έννοιες: οι πίνακες και τα δεδομένα ροής. Οι πίνακες είναι μια ολιστική προβολή συνόλου δεδομένων που έχουν προκύψει από χρονικά στιγμιότυπα τα οποία χειριζόμαστε με τη βοήθεια της SQL. Οι ροές δεδομένων από την άλλη αποτελούν μια προβολή στοιχείων δεδομένων που εξελίσσονται στην πάροδο του χρόνου. Με τον όρο stream αναφερόμαστε σε μια άπειρη ακολουθία δεδομένων που διατίθενται σε μια ροή χρόνου.

Κάθε στοιχείο δεδομένων σχετίζεται με μια ή περισσότερες χρονικές σημάνσεις (timestamps) που μπορούν να υποδεικνύουν τη χρονική στιγμή δημιουργίας αυτής της ροής δεδομένων, την εγκυρότητα του περιεχομένου, καθώς και τη χρονική στιγμή που είναι διαθέσιμη για επεξεργασία. Πιο αναλυτικά, η σημασιολογία της χρονικής σήμανσης (timestamp) έγκειται στο γεγονός ότι μέσα από αυτούς τους χρόνους προκύπτουν σημαντικά συμπεράσματα για την σειρά των δεδομένων καθώς και για το συγχρονισμό τους. Ιδανικά, ο χρόνος παραγωγής των δεδομένων και ο χρόνος επεξεργασίας θα πρέπει να είναι μηδενικός, πολλές αστάθειες όμως στο χρονισμό αυτό μπορεί να δημιουργήσουν πολλά και μεταβαλλόμενα προβλήματα. Σε επόμενη ενότητα θα αναφερθούμε εκτενέστερα στη σημασία που παίζει ο χρόνος στα δεδομένα ροής.

Τα δεδομένα ροής μπορεί να είναι δομημένα ή αδόμητα. Ένα δομημένο stream, περιέχει στοιχεία τα οποία ακολουθούν μια συγκεκριμένη μορφή η οποία μας επιτρέπει περεταίρω μοντελοποίηση αυτής της ροής. Σε αντίθεση με τα μη-δομημένα streams τα οποία μπορεί να περιέχουν αυθαίρετο περιεχόμενο που προκύπτει συχνά από το συνδυασμό ροών από πολλαπλές πηγές (συχνά συναντώνται και με τον όρο event showers ή events clouds (Doblander C, 2014)) .

Υπάρχουν τρεις μεγάλες κατηγορίες μοντέλων για τις δομημένες ροές δεδομένων, οι οποίες διαφοροποιούνται με τον τρόπο τον οποίο σχετίζονται και πως επηρεάζονται τα δεδομένα ροής μεταξύ τους:

- το περιστροφικό μοντέλο (turnstile model),
- το μοντέλο cash register,
- και το μοντέλο χρονοσειρών (time series model).

Το πιο γενικό μοντέλο είναι το turnstile model, στο οποίο η ροή δεδομένων αναπαρίσταται ως ένα διάνυσμα από στοιχεία, όπου το κάθε στοιχείο S_i στη ροή είναι μία αύξηση ή μείωση σε ένα στοιχείο του υποκείμενου διανύσματος. Το μέγεθος του διανύσματος είναι και ο κύριος τομέας (domain) της ροής των στοιχείων. Το μοντέλο αυτό χρησιμοποιείται παραδοσιακά στα συστήματα βάσης δεδομένων, στα οποία έχουμε εισαγωγή, διαγραφή και ενημέρωση δεδομένων μιας βάσης.

Στο μοντέλο cash register, τα στοιχεία που αποτελούν τη ροή δεδομένων είναι μόνο προσθήκες στο υποκείμενο διάνυσμα με τη διαφορά ότι τα στοιχεία αυτά δεν μπορούν να φύγουν ποτέ από το διάνυσμα. Η λειτουργία αυτή είναι πανομοιότυπη με τη λειτουργία καταγραφής ιστορικού σχέσεων στις βάσεις δεδομένων.

Τέλος, το μοντέλο χρονοσειρών (timeline model) συμπεριφέρεται σε κάθε στοιχείο S_i σαν να είναι ένα καινούριο και ανεξάρτητο διάνυσμα εισαγωγής τιμών. Αυτό έχει ως αποτέλεσμα, το υποκείμενο μοντέλο να αυξάνεται συνεχώς χωρίς κανένα περιορισμό. Το μοντέλο αυτό χρησιμοποιείται συχνά σε μηχανές επεξεργασίας ροής δεδομένων λόγω του ότι κάθε στοιχείο μπορεί να επεξεργαστεί ως μια ξεχωριστή οντότητα.

Με βάση όλα τα παραπάνω συμπεραίνουμε ότι η συγκέντρωση μιας ροής δεδομένων από ενημερώσεις που προκύπτουν σε μια ροή χρόνου αποτελούν ένα πίνακα, ενώ η παρατήρηση των αλλαγών που γίνονται σε ένα πίνακα κατά τη ροή του χρόνου αποδίδουν μια ροή δεδομένων. Η ιδέα αυτή είναι ένας από τους λόγους της μεγάλης επιτυχίας του Apache Kafka και αποτελεί το οικοσύστημα το οποίο έχει χτιστεί γύρω από αυτή την ιδέα.

3 SQL και η επέκταση της σε Streaming SQL

Όπως αναφέρθηκε και στην προηγούμενη ενότητα τα δεδομένα ροής – streams – είναι δεδομένα πίνακα σε κίνηση. Σκεφτείτε έναν ατέρμων πίνακα όπου συνεχώς εμφανίζονται νέα δεδομένα με την πάροδο του χρόνου. Ένα stream είναι ένας τέτοιος πίνακας. Μία εγγραφή ή μια σειρά δεδομένων σε ένα stream ονομάζεται συμβάν (event). Το συμβάν αυτό, έχει μια μορφή , και συμπεριφέρεται όπως μια γραμμή βάσης δεδομένων. Για να επεξεργαστούμε τα δεδομένα αυτά εφαρμόζουμε ερωτήματα SQL τα οποία μας δίνουν διάφορες πληροφορίες. Στην περίπτωση των δεδομένων ροής που είναι αποθηκευμένα δεδομένα σε πίνακα έχουμε την StreamSQL , η οποία με τη σειρά της εφαρμόζει και αυτή ερωτήματα στα δεδομένα μόνο που τα ερωτήματα αυτά είναι ατέρμονα και τα παράγωγα αυτών των ερωτημάτων αποτελούν streams.

Η StreamSQL είναι μια γλώσσα που εφαρμόζει ερωτήματα και επεκτείνει την SQL με τη δυνατότητα να επεξεργάζεται δεδομένα ροής σε πραγματικό χρόνο. Η SQL προορίζεται κυρίως για χειρισμό σχέσεων, δηλ. πίνακες, οι οποίοι είναι σε γραμμές. Η StreamSQL προσθέτει τη δυνατότητα χειρισμού ροών δεδομένων, οι οποίες αποτελούν ατέρμονες πλειάδες δεδομένων που δεν είναι όλες διαθέσιμες το ίδιο χρονικό διάστημα. Το γεγονός ότι οι ροές αυτών των δεδομένων είναι ατέρμονες, μας υποχρεώνει να εφαρμόσουμε μονοτονικές λειτουργίες, τα ερωτήματα θα πρέπει να είναι «συνεχή», να εκτελούνται για μεγάλα χρονικά διαστήματα και να επιστρέφουν σταδιακά αποτελέσματα.

3.1 StreamSQL και Σχεσιακή Άλγεβρα

Η SQL είναι μια ισχυρή γλώσσα για την αναζήτηση δεδομένων σε πίνακες. Έχει σχεδιαστεί ως ένα σύνολο ανεξάρτητων εντολών: προβολή (SELECT), φιλτράρισμα (WHERE), σύνδεση (JOIN) και ομαδοποίηση (GROUP BY), οι οποίες μπορούν να συνδυαστούν για τη δημιουργία πολύ ισχυρών ερωτημάτων. Μεταξύ αυτών, οι δύο πρώτες, SELECT και WHERE, αποτελούν λειτουργίες που εφαρμόζονται σχεδόν με τον ίδιο τρόπο και στη StreamSQL.

Όταν μιλάμε για ροές δεδομένων στη StreamSQL, είναι σημαντικό να κρατάμε κατά νου τη θεωρητική βάση της SQL και τη σχεσιακή άλγεβρα. Η σχεσιακή άλγεβρα είναι ένας μαθηματικός τρόπος που μας βοηθάει να περιγράψουμε τις σχέσεις ανάμεσα

στα δεδομένα μέσα από αναφορές σε ονοματοδοτημένες πλειάδες δεδομένων. Χρησιμοποιώντας κλασικούς όρους βάσεων δεδομένων, σχέση είναι ένας φυσικός πίνακας βάσης δεδομένων, είτε κάτι παρόμοιο με αυτό, αποτελέσματα ενός ερωτήματος που έχουμε θέσει στην SQL κ.ο.κ . Είναι δηλαδή ένα σύνολο γραμμών οι οποίες συμπεριλαμβάνουν ονοματοποιημένα δεδομένα σε στήλες.

Μια από τις πιο σημαντικές πτυχές της σχεσιακής άλγεβρας είναι η ιδιότητα κλειστότητας η οποία ορίζει ότι η εφαρμογή οποιουδήποτε τελεστή σχεσιακής άλγεβρας σε οποιαδήποτε σχέση πάντα παράγει μια νέα σχέση. Με άλλα λόγια οι σχέσεις είναι το βασικό εργαλείο της σχεσιακής άλγεβρας και όλοι οι τελεστές τις χρησιμοποιούν σαν είσοδο για να παράξουν μια νέα έξοδο αποτελεσμάτων. Στη StreamSQL έχουμε να αντιμετωπίσουμε μια πολύ έντονη σχέση, αυτή της ροής δεδομένων με τους πίνακες. Είναι πολύ σημαντικό σε αυτό το σημείο να αναφερθούμε και πάλι στο γεγονός ότι πίνακες και ροές δεδομένων είναι δυο διαφορετικές έννοιες.

3.2 Σχέσεις μεταβαλλόμενου χρόνου

Το κλειδί για την φυσική ενσωμάτωση των ροών δεδομένων στην SQL αποτελεί η επέκταση των ήδη υπαρχουσών σχέσεων ώστε να αντιπροσωπεύουν σύνολα δεδομένων στην πάροδο του χρόνου και όχι σύνολα δεδομένων συγκεκριμένων χρονικών στιγμών. Δηλαδή, αντί για σχέσεις σε χρονικά σημεία χρειαζόμαστε σχέσεις μεταβαλλόμενου χρόνου.

Σκεφτείτε ένα ανεπεξέργαστο σύνολο δεδομένων (dataset) το οποίο περιέχει συμβάντα χρηστών. Κατά την χρονική πάροδο και καθώς οι χρήστες δημιουργούν και άλλα συμβάντα, το dataset συνεχίζει να μεγαλώνει και να επεκτείνεται. Αν παρατηρήσουμε τα δεδομένα μια συγκεκριμένη χρονική στιγμή, θα λάβουμε μια κλασική σχέση. Αν όμως τα παρατηρήσουμε μέσα από την ολιστική τους εξέλιξη στο πέρασμα του χρόνου τότε θα λάβουμε μια σχέση μεταβαλλόμενου χρόνου.

Οι σχέσεις μεταβαλλόμενου χρόνου είναι σαν ένα τρισδιάστατο πίνακα ο οποίος εκτείνεται σε άξονες x ,y και z. Καθώς η σχέση αλλάζει, νέα στιγμιότυπα προστίθενται στον άξονα z. Για την καλύτερη κατανόηση αυτού θα αναφέρουμε ένα παράδειγμα με dataset όπου διατηρούμε τα στοιχεία και τα scores διαφόρων χρηστών. Τα δεδομένα αυτά έχουν προκύψει μέσα από ένα πολύ γνωστό παιχνίδι κινητών συσκευών. Ας υποθέσουμε λοιπόν ότι εκτελούμε την εξής εντολή

```
12:07 >  SELECT * FROM UserScores;
```

και το σύνολο των δεδομένων μας έχει την εξής μορφή κατά τη χρονική στιγμή 12:07:

Name	Score	Time
Yannis	7	12:01
Paul	3	12:03
Yannis	1	12:03
Yannis	4	12:07

Παρατηρούμε ότι έχει γίνει καταγραφή τεσσάρων scores κατά την πάροδο του χρόνου. Υποθέτουμε ότι έχουμε λάβει μόνο αυτά τα δεδομένα για τη σχέση αυτή. Η στήλη Time περιέχει χρονικές σημάνσεις (timestamps) οι οποίες δείχνουν τη στιγμή που έφτασε η εγγραφή στο σύστημα μας. Μέσα από την χρονική αναπαράσταση των παραπάνω σχέσεων αντιλαμβανόμαστε την εξέλιξη των δεδομένων κατά την πάροδο του χρόνο και αν θέλαμε να λάβουμε και τα δεδομένα μετά τη χρονική στιγμή 12:07 θα γράφαμε τα εξής:

```
12:07 >  SELECT TVR * FROM UserScores;
```

Με το TVR δηλώνουμε ότι θέλουμε να λάβουμε τη σχέση μεταβαλλόμενου χρόνου λαμβάνοντας τα εξής αποτελέσματα:

[-inf , 12 : 01)			[12 : 01 , 12 : 03)		
Name	Score	Time	Name	Score	Time
			Yannis	7	

[12:03 , 12:07)		[12 : 07 , now)
Name | Score | Time		Name | Score | Time
Yannis | 7 | 12:01		Yannis | 7 | 12:01
Paul | 3 | 12:03		Paul | 3 | 12:03
Yannis | 1 | 12:03		Yannis | 1 | 12:03
		Yannis | 4 | 12:07

Για μεγαλύτερη ευκολία παρουσίασης των αποτελεσμάτων τοποθετήσαμε τα δεδομένα σε ένα σχεσιακό πίνακα δυο διαστάσεων έχοντας έτσι τη δυνατότητα να παρατηρήσουμε ότι οι σχέσεις μεταβαλλόμενου χρόνου εμπεριέχουν μια δόση κλασικών σχέσεων κάθε μία από τις οποίες αποτυπώνει τα δεδομένα σε ένα συγκεκριμένο εύρος χρόνου. Αυτό που είναι σημαντικό για τον ορισμό των σχέσεων μεταβαλλόμενου χρόνου είναι ότι αποτελούν μια ακολουθία κλασικών σχέσεων που κάθε μία υπάρχει ανεξάρτητα από τα δικά τους χρονικά όρια και καταγράφουν ένα στιγμιότυπο κατά το οποίο η σχέση δεν άλλαξε. Έτσι, μπορούμε να εφαρμόσουμε ένα σχεσιακό τελεστή και να λάβουμε το ίδιο αποτέλεσμα αν τον εφαρμόζαμε και σε μια κλασσική ακολουθιακή σχέση.

Πηγαίνοντας ένα βήμα παραπέρα, παρατηρούμε ότι το αποτέλεσμα της χρήσης σχεσιακών τελεστών σε μια συχνότητα σχέσεων, όπου η κάθε μια σχετίζεται χρονικά, θα παράγει πάντα μια άλλη συχνότητα σχέσεων με τα ίδια χρονικά διαστήματα. Ο ορισμός αυτός μας δίνει δύο σημαντικές ιδιότητες:

- Όλοι οι σχεσιακοί τελεστές της κλασικής σχεσιακής άλγεβρας συνεχίζουν να συμπεριφέρονται με τον ίδιο τρόπο ακόμα και όταν εφαρμόζονται σε σχέσεις μεταβαλλόμενου χρόνου.
- Η ιδιότητα της κλειστότητας της σχεσιακής άλγεβρας παραμένει η ίδια ακόμα κι όταν εφαρμόζεται σε σχέσεις μεταβαλλόμενου χρόνου.

Συμπερασματικά, κρατάμε το γεγονός ότι όλοι οι κανόνες της κλασικής σχεσιακής άλγεβρας διατηρούν τις ιδιότητες τους ακόμα κι όταν εφαρμοστούν σε σχέσεις μεταβαλλόμενου χρόνου.

Ας εφαρμόσουμε τώρα φιλτράρισμα στο dataset του προηγούμενου παραδείγματος και ας παρατηρήσουμε τα επιστρεφόμενα αποτελέσματα, συντάσσοντας ένα ερώτημα με τη χρήση του τελεστή WHERE.

```
12:07> SELECT TVR * FROM UserScores WHERE Name = "Yannis";
```

[-inf , 12 : 01)			[12 : 01 , 12 : 03)		
Name	Score	Time	Name	Score	Time
			Yannis	7	12:01

[12:03 , 12:07)			[12 : 07 , now)		
Name	Score	Time	Name	Score	Time
Yannis	7	12:01	Yannis	7	12:01
Yannis	1	12:03	Yannis	1	12:03
			Yannis	4	12:07

Όπως θα παρατηρήσετε, η εφαρμογή αυτής της σχέσης έχει τα ίδια αποτελέσματα με την προηγούμενη μόνο που αυτή έχει φιλτραριστεί ως προς τα δεδομένα του χρήστη Yannis. Παρόλο που η χρονικά μεταβαλλόμενη σχέση καταγράφει την προστιθέμενη διάσταση του χρόνου που απαιτείται για την καταγραφή της εξέλιξης αυτών των δεδομένων με την πάροδο του χρόνου, το ερώτημα συμπεριφέρεται ακριβώς όπως θα περίμενε κανείς εφαρμόζοντας την SQL.

Αν θέλουμε να περιπλέξουμε περισσότερο την κατάσταση, ας δοκιμάσουμε μια σχέση η οποία θα ομαδοποιήσει τα δεδομένα μας ως προς το σύνολο των πόντων που συγκέντρωσε ο κάθε παίκτης.

```
12:07> SELECT TVR Name,
         SUM(Score) as Total,
         MAX(Time) as Time FROM UserScores GROUP BY Name;
```

```
--- ------------------------------------
|       [ -inf , 12 : 01)     |   |     [ 12 : 01 , 12 : 03)    |
------------------- |------------------
| Name | Score | Time |   | Name | Score | Time |
|      |       |      |   | Yannis |   7    | 12:01 |
|      |       |      |   |        |        |       |
------------------- |------------------

--- ------------------------------------
|       [ 12:03 , 12:07)     |    |     [ 12 : 07 , now)    |
------------------- |------------------
| Name | Score | Time |   | Name | Score | Time |
| Yannis |  8      | 12:03 |   | Yannis |    12   | 12:07 |
| Paul   |  3      | 12:03 |   | Paul   |    3    | 12:03 |
|        |         |       |   |        |         |       |
------------------- |------------------
```

Και πάλι, η χρονικά μεταβαλλόμενη έκδοση αυτού του ερωτήματος συμπεριφέρεται ακριβώς όπως θα περίμενε κανείς, με κάθε κλασική σχέση στην ακολουθία να περιέχει απλώς το άθροισμα των βαθμολογιών για κάθε χρήστη. Ανεξάρτητα από το πόσο περίπλοκο είναι ένα ερώτημα που θα μπορούσαμε να εκτελέσουμε, τα αποτελέσματα είναι πάντοτε πανομοιότυπα με την εφαρμογή αυτού του ερωτήματος σε αντίστοιχες κλασικές σχέσεις που συνθέτουν σχέσεις εισαγωγής δεδομένων και διακύμανσης χρόνου.

Οι χρονικές σχέσεις όμως είναι περισσότερο θεωρητικές κατασκευές παρά πρακτικές αναπαραστάσεις δεδομένων. Οι σχέσεις αυτές θα μπορούσαν να γίνουν αρκετά

μεγάλες και δυσκίνητες για μεγάλα σύνολα δεδομένων που αλλάζουν συχνά στην πάροδο του χρόνου. Για να δούμε πώς συνδέονται ρεαλιστικά με την επεξεργασία ροής σε πραγματικό περιβάλλον, θα πρέπει να διερευνήσουμε το πως σχετίζονται οι σχέσεις μεταβαλλόμενου χρόνου με τις ροές δεδομένων και τους πίνακες.

3.3 Ροές δεδομένων και Πίνακες

Ποια η σχέση των ροών δεδομένων με τους πίνακες; Για τη σύγκριση αυτή ας υποθέσουμε πάλι ότι ομαδοποιούμαι τα δεδομένα του προηγουμένου παραδείγματος με τους πίνακες βαθμολογιών παικτών ενός παιχνιδιού για κινητές συσκευές, εφαρμόζοντας αυτή τη φορά το παρακάτω ερώτημα:

```
12:07> SELECT TVR Name,
          SUM(Score) as Total,
          MAX(Time) as Time FROM UserScores GROUP BY Name;
```

```
--- ------------------------------------
|       [ -inf , 12 : 01)     |   |     [ 12 : 01 , 12 : 03)    |
------------------- |------------------
| Name | Score | Time |   | Name | Score | Time |
|      |       |      |   | Yannis |   7    | 12:01 |
|      |       |      |   |        |        |       |
------------------- |-------------------

--- ------------------------------------
|       [ 12:03 , 12:07)     |    |     [ 12 : 07 , now)    |
------------------- |------------------
| Name | Score | Time |   | Name | Score | Time |
| Yannis |  8      | 12:03 |   | Yannis |    12   | 12:07 |
| Paul   |  3      | 12:03 |   | Paul   |    3    | 12:03 |
|        |         |       |   |        |         |       |
------------------- |-------------------
```

Παρατηρούμε ότι η παραπάνω αναπαράσταση δεδομένων δείχνει το πλήρες ιστορικό της σχέσης στην πάροδο του χρόνου. Οι σχέσεις μεταβαλλόμενου χρόνου είναι

ουσιαστικά μια ακολουθία κλασικών σχέσεων όπου καθεμιά από αυτές καταγράφει ένα στιγμιότυπο σε ένα συγκεκριμένο χρονικό σημείο και οι κλασικές σχέσεις είναι ανάλογες με τους πίνακες. Παρατηρώντας μια χρονικά μεταβαλλόμενη σχέση ως ένα πίνακα βλέπουμε το στιγμιότυπο μιας σχέσης (point-in-time) για τη χρονική στιγμή που παρατηρούμε τα δεδομένα μας.

Για παράδειγμα, εάν επρόκειτο να παρατηρήσουμε την προηγούμενη ομαδοποιημένη σχέση μεταβαλλόμενου χρόνου ως ένα πίνακα στις 12:01, θα λάβουμε τα ακόλουθα (θα χρησιμοποιήσουμε τη λέξη TABLE ώστε το ερώτημα να μας επιστρέψει έναν πίνακα) :

```
12:01> SELECT TABLE Name,
          SUM(Score) as Total,
          MAX(Time) as Time FROM UserScores GROUP BY Name;
```

Name	Score	Time
Yannis	7	12:01

Παρατηρώντας τώρα τα δεδομένα μας για τη χρονική στιγμή 12:07 λαμβάνουμε τα παρακάτω αποτελέσματα:

```
12:07> SELECT TABLE Name,
          SUM(Score) as Total,
          MAX(Time) as Time FROM UserScores GROUP BY Name;
```

Name	Score	Time
Yannis	12	12:07
Paul	3	12:03

Το σημαντικό εδώ είναι ότι υπάρχει ακριβής εφαρμογή των σχέσεων μεταβαλλόμενου χρόνου στην SQL. Η έκδοση SQL 2011 μας δίνει τη δυνατότητα των χρονικών πινάκων (temporal tables), ουσιαστικά μας παρέχει σχέσεις μεταβαλλόμενου χρόνου, με την εντολή AS OF SYSTEM TIME, λαμβάνοντας έτσι ένα χρονικό στιγμιότυπο με σχέση μεταβαλλόμενου χρόνου. Για παράδειγμα, ας δημιουργήσουμε ένα ερώτημα για τη χρονική στιγμή 12:07 σε σχέση με τη χρονική στιγμή 12:03.

```
12:07> SELECT TABLE Name,
          SUM(Score) as Total,
          MAX(Time) as Time FROM UserScores GROUP BY Name
          AS OF SYSTEM TIME '12:03';
```

Name	Score	Time
Yannis	8	12:03
Paul	3	12:03

Μέσα από το παραπάνω παράδειγμα αντιλαμβανόμαστε ότι οι πίνακες μας δίνουν τη δυνατότητα να καταγράψουμε ένα στιγμιότυπο μεταβαλλόμενου χρόνου σε δεδομένη χρονική στιγμή. Το ίδιο συμβαίνει και σε real-time εφαρμογές που χρησιμοποιούν πίνακες. Μερικές εφαρμογές χρησιμοποιούν τους πίνακες για να διατηρήσουν περισσότερες πληροφορίες ιστορικού καταγράφοντας έτσι το πλήρες ιστορικό των δεδομένων στην πάροδο του χρόνου.

Οι ροές δεδομένων είναι ελαφρώς διαφορετικά εργαλεία καθώς καταγράφουν και την εξέλιξη του πίνακα στην πάροδο του χρόνου. Αντί να αποτυπώνουν ολιστικά στιγμιότυπα ολόκληρης της σχέσης κάθε φορά που αλλάζει, καταγράφουν τη συχνότητα αλλαγής η οποία έχει ως αποτέλεσμα τα στιγμιότυπα μιας σχέσης μεταβαλλόμενου χρόνου. Ας το δούμε μέσα από το παρακάτω παράδειγμα:

Ας καλέσουμε και πάλι τα αποτελέσματα του ερωτήματος TVR:

```
12:07> SELECT TVR Name,
          SUM(Score) as Total,
          MAX(Time) as Time FROM UserScores GROUP BY Name;
```

[-inf , 12 : 01)			[12 : 01 , 12 : 03)		
Name	Score	Time	Name	Score	Time
			Yannis	7	12:01

[12:03 , 12:07)			[12 : 07 , now)		
Name	Score	Time	Forename	Score	Time
Yannis	8	12:03	Yannis	12	12:07
Paul	3	12:03	Paul	3	12:03

Ας παρατηρήσουμε τώρα τη σχέση μεταβαλλόμενου χρόνου που έχουμε σαν μια ροή δεδομένων που υπάρχει σε διαφορετικά χρονικά σημεία. Προχωρώντας βήμα – βήμα θα συγκρίνουμε την αρχική μορφή του πίνακα TVR με την εξέλιξη της ροής δεδομένων για συγκεκριμένα χρονικά διαστήματα. Για να τα συγκρίνουμε θα πρέπει να χρησιμοποιήσουμε τις παρακάτω εντολές:

- **STREAM** : παρόμοια με την εντολή TABLE που είδαμε προηγουμένως. Μας επιστρέφει μια ροή δεδομένων για κάθε συμβάν αποτυπώνοντας την εξέλιξη της σχέσης μεταβαλλόμενου χρόνου καθώς περνάει ο χρόνος.

- **Sys.Undo** : εντολή που χρησιμοποιείται για διαχείριση αναίρεσης ενεργειών των δυναμικών πινάκων στο πρότυπο Apache Flink. Στο πρότυπο αυτό, τα παραδοσιακά συστήματα βάσης δεδομένων χρησιμοποιούν αρχεία καταγραφής για την αναδημιουργία πινάκων σε περίπτωση διαφόρων σφαλμάτων. Υπάρχουν διαφορετικές τεχνικές καταγραφής, όπως η καταγραφή UNDO, REDO και UNDO / REDO. Με λίγα λόγια, τα αρχεία καταγραφής UNDO διατηρούν την προηγούμενη τιμή ενός τροποποιημένου στοιχείου για να μπορέσουν να κάνουν αναίρεση, τα αρχεία καταγραφής REDO καταγράφουν τη νέα τιμή ενός τροποποιημένου στοιχείου για να επαναλάβουν τις χαμένες αλλαγές των ολοκληρωμένων συναλλαγών και τα αρχεία καταγραφής UNDO / REDO καταγράφουν την παλιά και τη νέα τιμή του ένα αλλαγμένο στοιχείο για να μπορέσουμε να αναιρέσουμε τις ατελείς συναλλαγές και να επαναλάβετε τις χαμένες αλλαγές των ολοκληρωμένων συναλλαγών.

Ξεκινώντας λοιπόν, από το στιγμιότυπο 12:01 έχουμε τα εξής:

```
12:01> SELECT Table Name,
               SUM(Score) as Total,
               MAX(Time) as Time
               FROM UserScores GROUP BY Name;
```

Name	Total	Time
Yannis	7	12:01

```
12:01> SELECT STREAM Name,
               SUM(Score) as Total,
               MAX(Time) as Time
               Sys.Undo as Undo
               FROM UserScores GROUP BY Name;
```

```
- - - - - - - - - - - - - - - - - - - - - - -
|  Name  | Total  | Time | Undo |
- - - - - - - - - - - - - - - - - - - - - - -
| Yannis |    7     | 12:01 |          |
```

..... [12:01, 12:01] .....

Ο πίνακας καθώς και τα δεδομένα ροής φαίνονται σχεδόν πανομοιότυπα σε αυτό το σημείο. Η μόνη διαφορά που παρατηρείται είναι στη στήλη Undo, ενώ τα αποτελέσματα του πρώτου πίνακα είναι πλήρης μέχρι τις 12:01 (υποδηλώνεται από την τελικές παύλες [- - - - - -] που κλείνουν το κάτω άκρο της σχέσης), τα αποτελέσματα ροής από την άλλη στον δεύτερο πίνακα παραμένουν ατελή , όπως υποδηλώνεται από τις τελείες [……..] που εμφανίζονται για να δηλώσουν την ανοιχτή ουρά της σχέσης όσο και από το χρονικό διάστημα επεξεργασίας των δεδομένων που έχει παρατηρηθεί μέχρι στιγμής, αφήνοντας ανοιχτό το ενδεχόμενο να ακολουθήσουν κι άλλα δεδομένα .

Αν εκτελεστεί σε πραγματική υλοποίηση, το ερώτημα με τη χρήση της εντολής STREAM, το σύστημα μας θα περιμένει σίγουρα να ακολουθήσουν και άλλα δεδομένα. Έτσι, αν περιμέναμε μέχρι τη χρονική στιγμή 12:03, τρεις νέες γραμμές δεδομένων θα εμφανιζόταν. Συγκρίνοντας τα αποτελέσματα θα είχαμε το παρακάτω αποτέλεσμα.

```
12:03> SELECT TABLE Name,
                SUM(Score) as Total,
                MAX(Time) as Time
                FROM UserScores GROUP BY Name;
```

```
    - - - - - - - - - - - - - - - - - -
    |  Name  | Total  | Time |
    | Yannis |    8     | 12:03 |
    | Paul   |    3     | 12:03 |

    - - - - - - - - - - - - - - - - - -
```

```
12:01> SELECT STREAM Name,
                SUM(Score) as Total,
                MAX(Time) as Time
                Sys.Undo as Undo
                FROM UserScores GROUP BY Name;
                -------------------------
                |  Name | Total  | Time | Undo |
                -------------------------
                | Yannis |   7    | 12:01 |      |
                | Paul   |   3    | 12:03 |      |
                | Yannis |   7    | 12:03 | undo |
                | Yannis |   8    | 12:03 |      |
                ..... [12:01, 12:03] .....
```

Στο σημείο αυτό παρατηρούμε ένα ενδιαφέρον σημείο που αξίζει να αναφερθούμε. Γιατί υπάρχουν τρεις νέες σειρές στη ροή (Paul 3 και Yannis undo-7 and 8) όταν το αρχικό σύνολο δεδομένων μας περιείχε μόνο δύο σειρές (Paul 3 και Yannis 1) για τη συγκεκριμένη χρονική περίοδο; Η απάντηση έγκειται στο γεγονός ότι εδώ παρατηρούμε τη ροή των αλλαγών σε μια συγκέντρωση των αρχικών εισόδων. Συγκεκριμένα, για το χρονικό διάστημα από 12:01 έως 12:03, η ροή πρέπει να καταγράψει δύο σημαντικές πληροφορίες σχετικά με την αλλαγή στη συνολική βαθμολογία του παίκτη Yannis λόγω της άφιξης της νέας τιμής 1:

- Η προηγούμενη αναφορά περιείχε σύνολο 7 βαθμών και ήταν λανθασμένη.
- Το νέο σύνολο είναι 8 πόντοι.

Και αυτό ακριβώς μας επιτρέπει να κάνουμε η ειδική στήλη Sys.Undo : διάκριση μεταξύ κανονικών γραμμών και σειρών που αποτελεί ανάκληση μιας τιμής που αναφέρθηκε προηγουμένως.

Ένα ιδιαίτερα ωραίο χαρακτηριστικό των ερωτημάτων STREAM είναι ότι μπορείτε να αρχίσετε να βλέπετε πώς όλα αυτά σχετίζονται με τον κόσμο των κλασικών πινάκων

Online Transaction Processing (OLTP). Το αποτέλεσμα που επιστρέφει η εντολή STREAM αυτού του ερωτήματος ουσιαστικά καταγράφει μια ακολουθία λειτουργιών INSERT και DELETE που θα μπορούσαμε να χρησιμοποιήσουμε για να υλοποιήσουμε αυτήν τη σχέση με την πάροδο του χρόνου σε ένα περιβάλλον OLTP (και στην πραγματικότητα, οι ίδιοι οι πίνακες OLTP είναι ουσιαστικά μεταβαλλόμενες χρονικές σχέσεις κατά την πάροδο του χρόνου μέσω ροών INSERT, UPDATE και DELETE).

Τώρα, στην περίπτωση που δεν ενδιαφερόμαστε για τις αναιρέσεις που συμβαίνουν στη ροή, το ερώτημα STREAM θα έμοιαζε με αυτό:

```
12:01> SELECT STREAM Name,
          SUM(Score) as Total,
          MAX(Time) as Time
          ~~Sys.Undo as Undo~~
          FROM UserScores GROUP BY Name;
```

Name	Total	Time
Yannis	7	12:01
Paul	3	12:03
Yannis	8	12:03

.....[12:01, 12:03] .....

Ωστόσο, υπάρχει σαφής αξία για την κατανόηση της εμφάνισης της πλήρους ροής, οπότε θα επιστρέψουμε στη στήλη Sys.Undo για το τελευταίο μας παράδειγμα. Μιλώντας για αυτό, αν περιμέναμε άλλα τέσσερα λεπτά έως τις 12:07, θα δεχόμασταν δύο επιπλέον σειρές στο ερώτημα STREAM, ενώ το ερώτημα TABLE θα συνέχιζε να εξελίσσεται όπως πριν:

```
12:07> SELECT TABLE Name,
          SUM(Score) as Total,
          MAX(Time) as Time
          FROM UserScores GROUP BY Name;
```

Name	Total	Time
Yannis	12	12:07
Paul	3	12:03

```
12:01> SELECT STREAM Name,
          SUM(Score) as Total,
          MAX(Time) as Time
          Sys.Undo as Undo
          FROM UserScores GROUP BY Name;
```

Name	Total	Time	Undo
Yannis	7	12:01	
Paul	3	12:03	
Yannis	7	12:03	undo
Yannis	8	12:03	
Yannis	8	12:07	undo
Yannis	12	12:07	

..... [12:01, 12:07] .....

Με το παραπάνω παράδειγμα γίνεται ξεκάθαρο ότι τα δεδομένα ροής που αναλύσαμε, εφαρμόζοντας σε αυτά σχέσεις μεταβαλλόμενου χρόνου, είναι μια διαφορετική και πολύ δυνατή μορφή δεδομένων σε σχέση με τα δεδομένα που αποθηκεύονται σε πίνακες και τα οποία αποτυπώνουν ένα στιγμιότυπο μιας ολόκληρης σχέσης σε ένα συγκεκριμένο χρονικό σημείο. Ενδιαφέρον είναι να αναφέρουμε ότι η απόδοση των stream δεδομένων έχει περισσότερα κοινά χαρακτηριστικά με την πρωτότυπη TVR που βασίζεται σε πίνακα.

```
12:07> SELECT TVR Name, SUM(Score) as Total, MAX(Time) as
       Time FROM UserScores GROUP BY Name;
```

[-inf , 12 : 01)				[12 : 01 , 12 : 03)		
Name	Score	Time		Name	Score	Time
				Yannis	7	12:01

12:03, 12:07)				[12 : 07 , now)		
Name	Score	Time		Name	Score	Time
Yannis	8	12:03		Yannis	12	12:07
Paul	3	12:03		Paul	3	12:03

Πράγματι, μπορούμε να πούμε με ευκολία ότι το ερώτημα STREAM παρέχει απλώς μια εναλλακτική απόδοση ολόκληρου του ιστορικού δεδομένων που υπάρχει στο αντίστοιχο ερώτημα TVR βάσει πίνακα. Το πιο σημαντικό στοιχείο της STREAM είναι η συνοπτικότητά της. Καταγράφει μόνο τις αλλαγές μεταξύ κάθε στιγμιότυπου σχέσης point-in-time στο TVR. Αυτό που αποδίδει η TVR ακολουθία πινάκων είναι η σαφήνεια που μας παρέχει. Αποτυπώνει την εξέλιξη της σχέσης με την πάροδο του χρόνου σε μια μορφή που υπογραμμίζει τη φυσική της σχέση με τις κλασικές σχέσεις και με αυτόν τον τρόπο παρέχει έναν απλό και σαφή ορισμό της σχεσιακής σημασιολογίας στο πλαίσιο της ροής δεδομένων καθώς και την πρόσθετη διάσταση του χρόνου που προσφέρει η ροή δεδομένων.

Μια άλλη σημαντική πτυχή των ομοιοτήτων μεταξύ των αποδόσεων STREAM και των TVR βάσει πίνακα είναι το γεγονός ότι είναι ουσιαστικά ισοδύναμες στα συνολικά δεδομένα που κωδικοποιούν. Αυτό φτάνει στον πυρήνα της δυαδικότητας ροής δεδομένων / πίνακα που οι υποστηρικτές του λένε εδώ και καιρό: οι ροές και οι πίνακες είναι πραγματικά μόνο δύο διαφορετικές πλευρές του ίδιου νομίσματος. Οι ροές και οι πίνακες εφαρμόζονται σε σχέσεις που ποικίλλουν με βάσει το χρόνο αφήνοντας ξεκάθαρο το γεγονός ότι μια πλήρης χρονικά μεταβαλλόμενη σχέση είναι τόσο πίνακας όσο και ροή δεδομένων ταυτόχρονα. Οι πίνακες και οι ροές είναι απλά διαφορετικές φυσικές εκδηλώσεις της ίδιας έννοιας, ανάλογα με το πλαίσιο.

Στο σημείο αυτό, είναι σημαντικό να θυμηθούμε ότι αυτή η δυαδικότητα ροής δεδομένων / πίνακα ισχύει μόνο εφόσον και οι δύο εκδόσεις κωδικοποιούν τις ίδιες πληροφορίες, δηλαδή, όταν έχουμε πίνακες ή ροές δεδομένων πλήρους πιστότητας. Σε πολλές περιπτώσεις, ωστόσο, η πλήρης πιστότητα δεν είναι πρακτική. Όπως αναφέρθηκε νωρίτερα, η κωδικοποίηση του πλήρους ιστορικού μιας σχέσης που ποικίλει χρονικά, ανεξάρτητα από το αν είναι σε μορφή ροής ή πίνακα, μπορεί να είναι μια «ακριβή» υπολογιστικά διαδικασία για μια μεγάλη πηγή δεδομένων. Είναι πολύ συνηθισμένο οι εκδηλώσεις ροής και πίνακα ενός TVR να έχουν ποικίλες απώλειες. Οι πίνακες συνήθως κωδικοποιούν μόνο την πιο πρόσφατη έκδοση ενός TVR. Εκείνοι που υποστηρίζουν προσωρινή πρόσβαση συχνά συμπιέζουν το κωδικοποιημένο ιστορικό σε συγκεκριμένα στιγμιότυπα χρονικής στιγμής ή / και σε συλλογές άχρηστων δεδομένων που είναι παλαιότερες από κάποιο χρονικό όριο. Ομοίως, οι ροές κωδικοποιούν συνήθως μια περιορισμένη διάρκεια της εξέλιξης ενός TVR, συχνά ένα σχετικά πρόσφατο τμήμα αυτής της ιστορίας. Οι μόνιμες ροές όπως αυτές που παράγονται από την Kafka παρέχουν τη δυνατότητα κωδικοποίησης του συνόλου ενός TVR, αλλά και πάλι αυτό είναι σχετικά ασυνήθιστο, με δεδομένα παλαιότερα από κάποιο όριο που συνήθως απορρίπτονται μέσω μιας διαδικασίας συλλογής άχρηστων δεδομένων.

Το βασικό σημείο εδώ είναι ότι οι ροές και οι πίνακες είναι απολύτως όμοιοι ο ένας με τον άλλον, και αποτελούν έναν έγκυρο τρόπο κωδικοποίησης μιας χρονικά μεταβαλλόμενης σχέσης. Στην πράξη, είναι συνηθισμένο οι φυσικές εκδηλώσεις ροής / πίνακα ενός TVR να έχουν απώλειες με κάποιο τρόπο. Αυτές οι ροές μερικής πιστότητας και οι πίνακες αποδίδουν μείωση των συνολικών κωδικοποιημένων πληροφοριών για κάποιο όφελος, συνήθως μειωμένο κόστος πόρων. Και αυτοί οι τύποι αντιστάθμισης είναι

σημαντικοί, διότι συχνά μας επιτρέπουν να κατασκευάζουμε αγωγούς που εφαρμόζονται σε πηγές δεδομένων πραγματικά μεγάλης κλίμακας. Αλλά περιπλέκουν επίσης την κατάσταση και απαιτούν μια βαθύτερη κατανόηση για να χρησιμοποιηθούν σωστά.

3.4 Νέες τεχνικές της StreamSQL

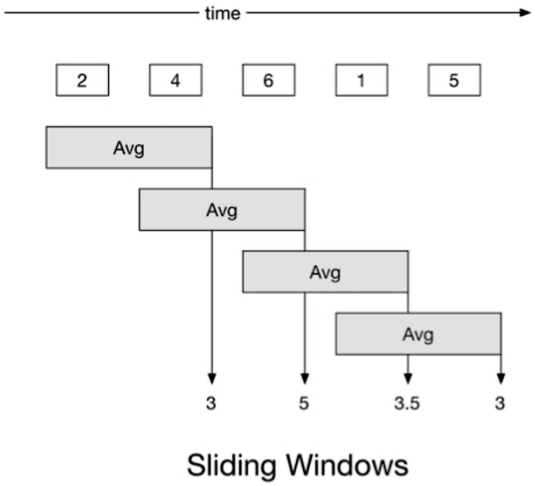
Κάποιες από τις τεχνικές λεπτομέρειες που ήρθε να προσθέσει η StreamSQL είναι οι εξής: (WSO2, 2019)

- **Windows**: Τα παράθυρα – Windows – αποτελούν ένα από τα βασικά και δομικά στοιχεία σχεδόν όλων των συστημάτων επεξεργασίας ροής δεδομένων. Ορίζουν δέσμες δεδομένων από μια ατέρμονη ροή δεδομένων και χρησιμοποιούνται για την εκτέλεση υπολογισμών που θα ήταν αδύνατο να πραγματοποιηθούν σε μια ατέρμονη ροή δεδομένων. Για παράδειγμα ο υπολογισμός της μέσης τιμής όλων των στοιχείων μιας ροής αριθμητικών δεδομένων.

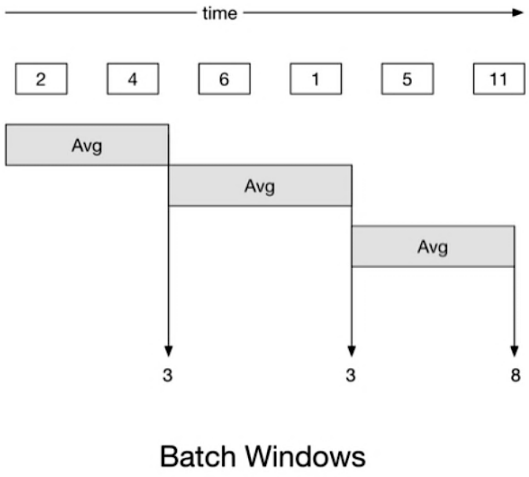
  Οι πιο συνηθισμένοι τύποι παραθύρων είναι τα παράθυρα με βάση το πλήθος και το χρόνο. Τα πρώτα καθορίζουν το μέγεθός τους ως προς τον αριθμό των στοιχείων που περιέχουν, ενώ τα δεύτερα καθορίζουν το μέγεθός τους σε όρους χρονικού πλαισίου και περιλαμβάνουν όλα τα στοιχεία με χρονική σήμανση (timestamp) που περιλαμβάνονται σε αυτό χρονικό πλαίσιο.

  Στο παράδειγμα του λέβητα, που αναφέρθηκε σε προηγούμενη ενότητα, αν θέλαμε να παρακολουθήσουμε τα δεδομένα που μπορούμε να λάβουμε θα εφαρμόζαμε εντολές φιλτραρίσματος και προβολής των δεδομένων ροής, επίσης θα χρησιμοποιούσαμε μια μορφή λειτουργικής μνήμης. Τα δεδομένα ροής χρησιμοποιούν αυτή τη λειτουργική μνήμη σα παράθυρο. Μπορείτε να φανταστείτε τα παράθυρα αυτά σα να παρατηρούμε τα δεδομένα μας σε κάτοψη. Τα συμβάντα παραθύρου διατηρούνται στη μνήμη και είναι διαθέσιμα για περαιτέρω επεξεργασία. Η πιο απλή μορφή παραθύρου είναι το παράθυρο συρόμενου μεγέθους (sliding length window). Η έννοια του «συρόμενου» αναφέρεται για να δώσει έννοια στο γεγονός ότι το παράθυρο θα πυροδοτήσει μια έξοδο για κάθε συμβάν.

Επίσης, υπάρχει και το παράθυρο δέσμης (batch window), στο οποίο το παράθυρο θα πυροδοτήσει κάποιο συμβάν αφού τελειώσει η προηγούμενη δέσμη δεδομένων (βλ. Εικόνες 3 , 4) (WSO2, 2019)



Εικόνα 3:Sliding Windows
Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/



Εικόνα 4: Batch Windows

Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/

Η λειτουργική μνήμη που χρησιμοποιείται για τις παραπάνω διεργασίες καθορίζεται είτε από τον αριθμό των συμβάντων, δημιουργώντας έτσι την έννοια των length windows, είτε από τον χρόνο που πέρασε από τη στιγμή δημιουργίας της δέσμης των δεδομένων μέχρι και την ολοκλήρωσή της, όπου βάσει αυτού έχουμε την έννοια των time windows. (WSO2, 2019)

Το κριτήριο αυτό δημιουργεί τους παρακάτω τέσσερις συνδυασμούς:

- *Sliding length window* – διατηρεί τα τελευταία Ν συμβάντα και πυροδοτείτε για κάθε νέο συμβάν.
- *Batch length window* – διατηρεί τα τελευταία Ν συμβάντα και πυροδοτεί καινούργια για κάθε Ν συμβάντα.
- *Sliding time window* – διατηρεί ενεργά τα συμβάντα για τα τελευταία Ν χρονικά βήματα και πυροδοτείτε για κάθε νέο συμβάν.
- *Batch time window* - διατηρεί ενεργά τα συμβάντα για τα τελευταία Ν χρονικά βήματα και πυροδοτείτε στο τέλος μιας χρονικής περιόδου.

Τα παραπάνω αποτελούν τα τέσσερα πιο κοινά σενάρια. Τα παράθυρα μας δίνουν τη δυνατότητα να διατηρούμε μια λειτουργική μνήμη και να παρατηρούμε αποτελεσματικά προγενέστερα δεδομένα στον άξονα του χρόνου. Μερικά από τα παραδείγματα εφαρμογής των παραπάνω είναι τα εξής:

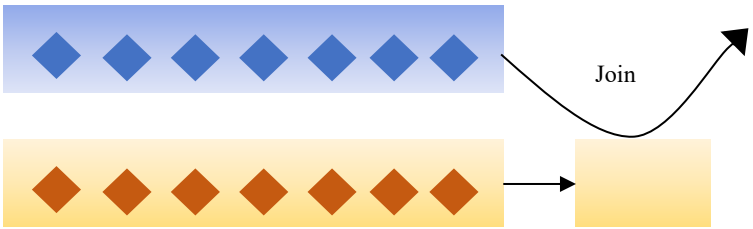
- Αφαίρεση θορύβου σε περιπτώσεις κινητού μέσου όρου ή διάμεσου.
- Οι πολλαπλοί κινητοί μέσοι (π.χ. στο 1 λεπτό, 5 λεπτά και 15 λεπτά) συμπεριφέρονται ως δείκτες που προσδιορίζουν μια τάση των δεδομένων και συχνά χρησιμοποιείται σε προβλέψεις μετοχών.
- Ανάλυση χρηματιστηριακών αγορών, όπου χρησιμοποιείται η μέθοδος αυτή για τον εντοπισμό αλλαγής τάσεων μέσα από τον εντοπισμό σημείων διασταύρωσης κινητών μέσων όρων.
- Περαιτέρω επεξεργασία για τον εντοπισμό ανωμαλιών στα δεδομένα που προκύπτουν.
- Η διακύμανση σε πρόσφατα δεδομένα παραθύρου μπορεί να χρησιμοποιηθεί για την πρόβλεψη νέων τάσεων και επικείμενων τιμών.

- **JOIN**: Εάν θέλουμε να χειριστούμε δεδομένα από διαφορετικούς πίνακες, χρησιμοποιούμαι την εντολή JOIN της SQL. Ομοίως δρούμε και στην περίπτωση που θέλουμε να χειριστούμε δεδομένα από διαφορετικές ροές δεδομένων. Μια ροή δεδομένων μπορεί να συνδεθεί με μια σχέση για την παραγωγή μιας νέας ροής δεδομένων. Κάθε πλειάδα στη ροή δεδομένων συνδέεται με την τρέχουσα τιμή της σχέσης με αποτέλεσμα να επιστρέφει το μηδέν ή περισσότερες πλειάδες δεδομένων.

Για την ένωση πολλαπλών δεδομένων, το πρώτο που θα πρέπει να λάβουμε υπόψη είναι ότι τα δεδομένα θα πρέπει να είναι ευθυγραμμισμένα στο χρόνο καθώς τα λαμβάνουμε, σε άλλη περίπτωση θα έχει διαφορετικές χρονικές σημάνσεις (timestamps). (WSO2, 2019)

Ο δεύτερος παράγοντας που θα πρέπει να ληφθεί σοβαρά είναι το γεγονός ότι αφού τα δεδομένα είναι ατέρμονα, οι ενώσεις τους θα πρέπει να είναι οριοθετημένες, διαφορετικά και η ένωση τους θα είναι ατέρμονη.

Τελευταίος παράγοντας που θα πρέπει να ληφθεί υπόψη είναι το γεγονός ότι χρειαζόμαστε από τις ενώσεις να παράγουμε συνεχή δεδομένα καθώς τα δεδομένα μας είναι ατέρμονα. Όπως θα δείτε και στην εικόνα 5 (WSO2, 2019), την περίπτωση αυτή τη χειριζόμαστε διατηρώντας ένα παράθυρο και συλλέγοντας συμβάντα από την κάτω ροή δεδομένων, η οποία κρατάει παθητική στάση στην ένωση αυτή, και στη συνέχεια προχωράμε στη ένωση των νέων συμβάντων με την επάνω ροή δεδομένων.



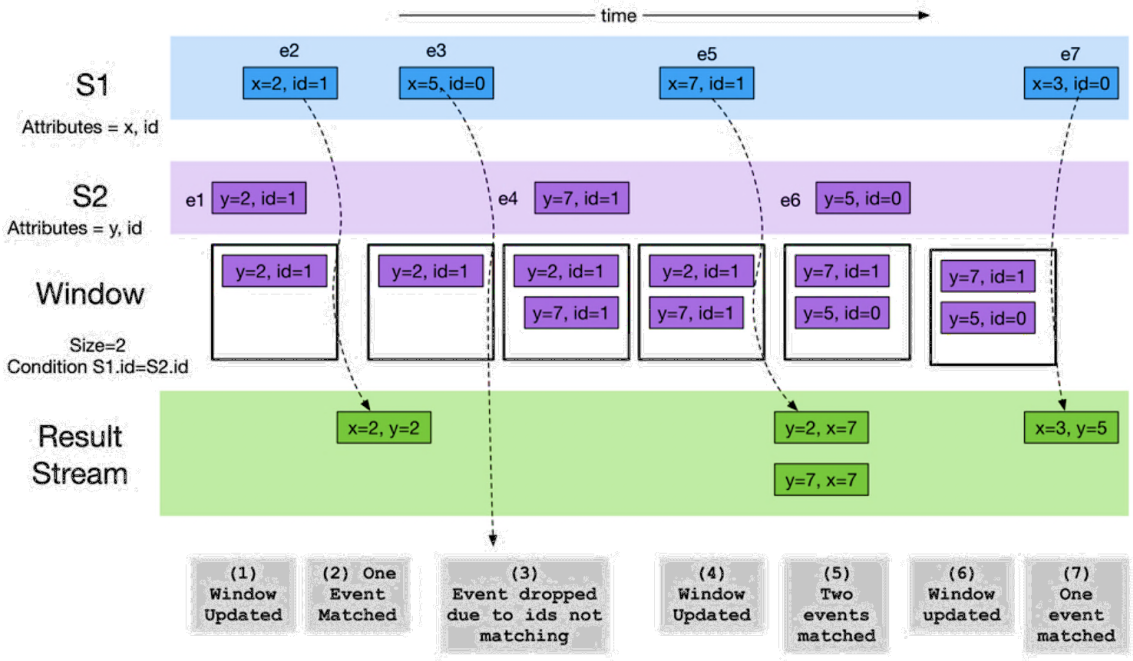
Εικόνα 5:Ενοποίηση δυο ροών δεδομένων (Join)

Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/

Ας δούμε και το παρακάτω παράδειγμα κώδικα. Ας υποθέσουμε ότι έχουμε ένα stream S1 με χαρακτηριστικά x και id, καθώς και ένα stream S2 με χαρακτηριστικά y και id. Η ενοποίηση τους θα γίνει με βάση το id. Το ερώτημα ενοποίησης σε StreamSQL θα έχει την εξής μορφή: (WSO2, 2019)

```
Select x, y From S1 as s1 join S2 as s2 on
  s1.id=s2.id insert into JoinedStream
```

Ας δούμε τώρα πως θα ενεργήσει το ερώτημα αυτό πάνω στα streams μας (βλ. Εικόνα 6) (WSO2, 2019).



Εικόνα 6:Ενοποίηση streams με βάσει χαρακτηριστικό τους id

Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/

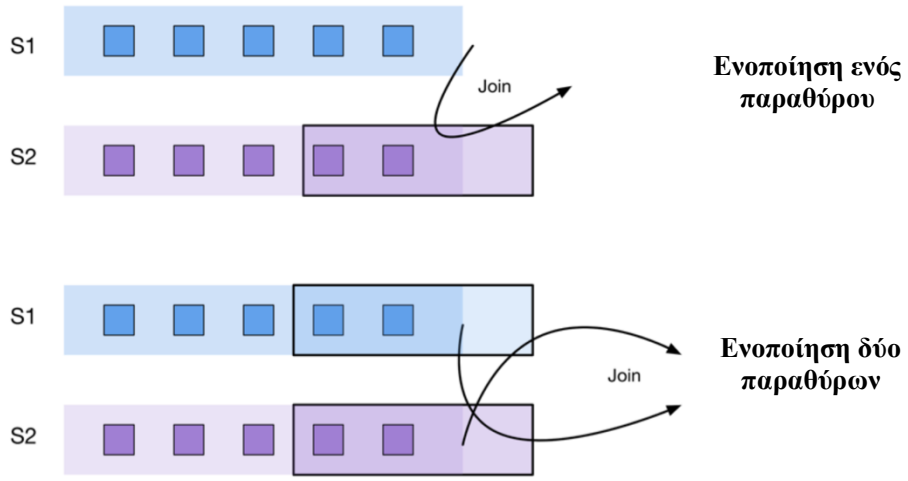
Όπως μπορείτε να δείτε και στην Εικόνα 5, το μέγεθος του παραθύρου έχει οριστεί με την τιμή 2, γεγονός που σημαίνει ότι μπορούν να διατηρηθούν δύο συμβάντα μόνο εντός παραθύρου.

Παρατηρώντας τη διεργασία στη ροή του χρόνου βλέπουμε τα εξής:

- Τη στιγμή που λαμβάνει χώρα το e1, συλλέγεται από το παράθυρο.
- Τη στιγμή που λαμβάνει χώρα το e2 ενοποιείται με το e1 και παράγουν ένα νέο συμβάν.
- Το συμβάν e3, το οποίο λαμβάνει χώρα στο ενεργό stream, απορρίπτεται καθώς δεν ταυτίζεται το id του με κανένα συμβάν στο παράθυρο.
- Το συμβάν e4, διατηρείται στο παράθυρο καθώς βρίσκεται στο stream S2, το οποίο κρατάει μια παθητική στάση στη διεργασία ενοποίησης.
- Το συμβάν e5, ταιριάζει με τα συμβάντα στο παράθυρο και έτσι παράγονται δυο συμβάντα ως έξοδος.
- Το συμβάν e6, διατηρείται στο παράθυρο και το παλιότερο συμβάν e1 αφαιρείται καθώς το μέγεθος του παραθύρου έχει την τιμή 2.

- Το συμβάν e7, ταιριάζει με τα συμβάντα και παράγονται δυο συμβάντα ως έξοδος.

Για να δουλέψει σωστά το παραπάνω μοντέλο θα πρέπει τουλάχιστον μια ροή δεδομένων να πρέπει να ενοποιηθεί με το παράθυρο. Αυτό μας οδηγεί στις εξής δύο περιπτώσεις (βλ. Εικόνα 7).



Εικόνα 7:Περιπτώσεις ενοποίησης

Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/

Στην πρώτη περίπτωση στην οποία αναφερθήκαμε στο προηγούμενο παράδειγμα, ένα stream S2 οριοθετήθηκε από το παράθυρο και τα συμβάντα διατηρήθηκαν στο παράθυρο αυτό καθώς ελέγχονταν για ταίριασμα με συμβάντα του δεύτερου stream S1. Επίσης, μόνο τα συμβάντα του stream S2 πυροδότησαν ένα αποτέλεσμα. (WSO2, 2019)

Στη περίπτωση της ενοποίησης σε δύο παράθυρα θα διατηρηθούν συμβάντα μέσα στο παράθυρο τα οποία θα λάβουν χώρα και από τα δύο streams. Ερχόμενο ένα νέο συμβάν θα γίνει έλεγχος για ταίριασμα με βάση το χαρακτηριστικό που έχουμε θέσει και έτσι θα πυροδοτηθεί ένα νέο συμβάν ως αποτέλεσμα.

- **Σχέση 'Happens after' & Μοτίβο**: Τι θα συμβεί στην περίπτωση που ένα event A ακολουθείται από ένα event B; Για παράδειγμα θα θέλαμε να δούμε τι θα συμβεί αν η θερμοκρασία ενός χώρου αυξηθεί κατά 30% μέσα στα επόμενα 10 λεπτά. Για

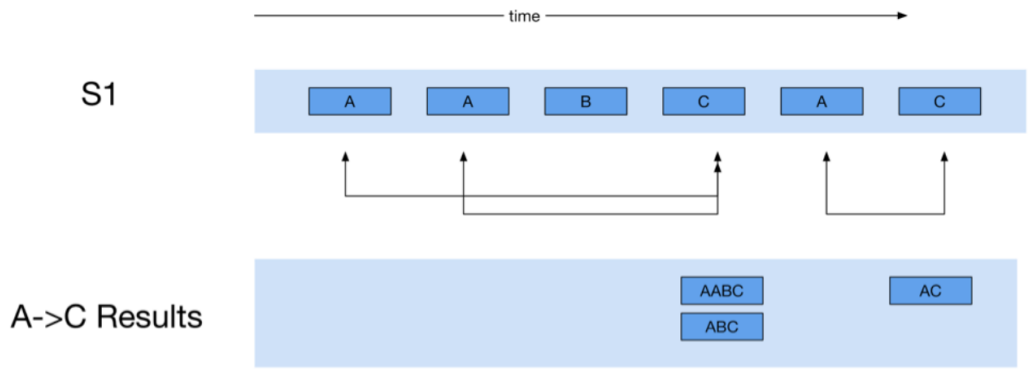
να παρακολουθήσουμε αυτή την αλλαγή θα πρέπει να καταγράψουμε τις θερμοκρασίες για τα τελευταία 10 λεπτά και να τις συγκρίνουμε με κάθε νέο συμβάν. Το ερώτημα αυτό έρχεται να το λύσει η σχέση 'happens-after'.

Η StreamSQL, όπως αναφέρθηκε και πριν, επεκτείνει το σχεσιακό μοντέλο της SQL προσθέτοντας την έννοια του χρόνου στα συμβάντα και αποδέχεται ότι κάθε δεδομένο υπάρχει σε συγκεκριμένη χρονική στιγμή. Διατηρώντας αυτή την οπτική είναι εύκολο να σκεφτούμε σχετικά με τη σειρά των συμβάντων. Τέτοια σενάρια η StreamSQL τα χειρίζεται χρησιμοποιώντας τη σχέση "happens-after", η οποία αντιπροσωπεύεται από το σύμβολο "->". Η λειτουργία ξεκινά ζητώντας στον κινητήρα ροής να αναζητήσει την πρώτη συνθήκη του ερωτήματος και όταν ολοκληρωθεί, αρχίζει να ψάχνει τη δεύτερη συνθήκη, η οποία είναι ένα συμβάν όπως το παραπάνω παράδειγμα με την αύξηση θερμοκρασίας 30% από τη θερμοκρασία του πρώτου συμβάντος.

```
Select bid, ts as T From BoilerStream as b1 ->
BoilerStream[(T - b1.T)/b1.T > 0.3 ] insert into
BoilerIncreasedStream
```

Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/

Στην εικόνα 8, βλέπουμε πως πρόκειται να εκτελεστεί το ερώτημα αυτό.



Εικόνα 8:Happens-after

Πηγή: https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/

Στο ερώτημα αυτό θα καταγραφεί το αποτέλεσμα που θα έχει η περίπτωση που μετά το event A ακολουθήσει το event C. Τα παραγόμενα αποτελέσματα είναι οι συνδυασμοί των ‘AABC’, ‘ABC’ και ‘AC’.

Η σχέση ‘happens-after’ μπορεί να χρησιμοποιηθεί για τις παρακάτω περιπτώσεις:

- να ανιχνεύσει την άνοδο ή την πτώση μιας τιμής,
- να εντοπίσει ανώτατος ή κατώτατες τιμές συνόλων,
- να αναγνωρίσει μοτίβα γραφήματος (π.χ. χρηματιστηριακές αγορές).
- Επιπλέον, μπορεί να χρησιμοποιηθεί για τον εντοπισμό και χαρτογράφηση ανθρώπινης συμπεριφοράς (π.χ. ο τρόπος που περπατάμε ή τρέχουμε)
- και για έλεγχο της πληρότητας μιας εργασίας (π.χ. Η απάντηση σε ένα μήνυμα ηλεκτρονικού ταχυδρομείου έρχεται εντός μίας ώρας;).

Η StreamSQL διαθέτει μοτίβα τα οποία είναι πιο ισχυρά από μια απλή σχέση ‘Happens-after’ και μας δίνουν τη δυνατότητα να συντάξουμε ερωτήματα με συνδυασμό απλών τελεστών. (WSO2, 2019)

Περίπτωση 1: Ένα μοτίβο ερωτήματος μπορεί να περιέχει πολλαπλές περιπτώσεις ελέγχων.

Για παράδειγμα A → B → C.

Περίπτωση 2: Κάθε στοιχείο ενός ερωτήματος μπορεί να αναφερθεί ως προαιρετικό καθώς και με οποιοδήποτε αριθμό αναφοράς.

Για παράδειγμα:
A → B? → C : μετά το event A ακολουθεί προαιρετικά το event B και μετά το event C.

A → B* → C : μετά το event A ακολουθεί οποιοδήποτε πλήθος του event B και μετά το event C.

Για να κατανοήσουμε τις δυνατότητες που μας προσφέρουν οι παραπάνω λειτουργίες ας φέρουμε ως παράδειγμα ένα ποδοσφαιρικό αγώνα όπου η μπάλα, τα παπούτσια κάθε παίκτη και τα γάντια του τερματοφύλακα διαθέτουν αισθητήρες που εκπέμπουν ένα συμβάν το οποίο περιλαμβάνει μια χρονική σήμανση, x, y, z συντεταγμένες (x, y, z) , ταχύτητες (vx, vy, vz) και επιταχύνσεις (ax , ay, az).

Εφαρμόζοντας τα παραπάνω πρότυπα έχουμε τη δυνατότητα να περιγράψουμε τον τρόπο με τον οποίο η streamSQL μπορεί να ανιχνεύσει την κατοχή της μπάλας από κάθε παίκτη. Με αυτό τον τρόπο θα μπορούμε να δούμε αν είναι αποδοτικά τα αποτελέσματα που θα λάβουμε.

Το πρώτο ερώτημα εντοπίζεται όταν κλωτσήθηκε η μπάλα. Η κλωτσιά εντοπίστηκε ως συμβάν όταν ένας παίκτης και η μπάλα ήταν σε απόσταση ενός μέτρου ο ένας από τον άλλο και η ταχύτητα της μπάλας έπειτα αυξήθηκε σε περισσότερα από 55m / s-2. Αυτό το καταφέραμε με τη σύνδεση των streams του παίκτη και των streams της μπάλας.

Στη συνέχεια, για να εντοπίσουμε την κατοχή της μπάλας του παίκτη 1 πρέπει πρώτα να αναζητήσουμε την ύπαρξη κλωτσιάς από άλλο παίκτη, ακολουθούμενο από μια ή περισσότερες κλωτσιές από τον παίκτη 1 και στη συνέχεια ακολουθούμενο από μια κλωτσιά από έναν άλλο παίκτη. Αν και εδώ έχουμε να κάνουμε με μια περίπλοκη συνθήκη, εφαρμόζοντας τη δυνατότητα εφαρμογής μοτίβου της streamSQL μπορούμε να ανιχνεύσουμε τις δύο παραπάνω συνθήκες μόνο με δύο ερωτήματα.

Ακριβώς όπως και οι κανονικές εκφράσεις, έτσι και τα μοτίβα της streamSQL μπορούν να υλοποιηθούν εύκολα και αποδοτικά στις ροές δεδομένων. Με τα παραπάνω παραδείγματα, παρατηρήσαμε πάνω 100.000 events ανά δευτερόλεπτο.

Με τη δυνατότητα χρήσης των μοτίβων η streamSQL απευθείας υπερβαίνει σε απόδοση την τυπική SQL και επιτρέπει την εφαρμογή της σε μια ευρεία κατηγορία αποτελεσματικών περιπτώσεων χρήσης πραγματικού κόσμου, όπως οι εφαρμογές του Internet of Things (IoT).

Πέρα από τις παραπάνω τεχνολογίες ενδιαφέρον έχει να αναφερθούν και τα παρακάτω μοτίβα ενεργειών που εφαρμόζονται σε δεδομένα ροής με τη βοήθεια της StreamingSQL.

- **Προεπεξεργασία:** Η προεπεξεργασία γίνεται συχνά ως τρόπος προβολής από τη μία ροή δεδομένων στην άλλη ή μέσω φιλτραρίσματος. Οι πιθανές λειτουργίες περιλαμβάνουν:
    - Φιλτράρισμα και κατάργηση ορισμένων συμβάντων,
    - Αναδιαμόρφωση μιας ροής με κατάργηση, μετονομασία ή προσθήκη νέων χαρακτηριστικών σε μια ροή,
    - Διαχωρισμός και συνδυασμός χαρακτηριστικών σε μια ροή,
    - Μετασχηματισμός χαρακτηριστικών.

Για παράδειγμα, από μια ροή δεδομένων του Twitter, ενδέχεται να επιλέξουμε να εξαγάγουμε τα πεδία: συντάκτης, χρονική σήμανση, τοποθεσία και στη συνέχεια να τα φιλτράρουμε με βάση την τοποθεσία του συντάκτη του tweet.

- **Ειδοποιήσεις και κατώφλια:** Αυτό το μοτίβο εντοπίζει μια συνθήκη και δημιουργεί ειδοποιήσεις με βάση τη συνθήκη. (π.χ. Συναγερμός σε υψηλή θερμοκρασία). Οι ειδοποιήσεις αυτές βασίζονται σε μια απλή τιμή ή και σε πιο περίπλοκες συνθήκες, όπως ρυθμός αύξησης κ.λπ.

Για παράδειγμα, δεδομένα που προκύπτουν από την καθημερινή κίνηση TFL (Transport for London) στην πόλη του Λονδίνου, μπορούμε να ενεργοποιήσουμε μια ειδοποίηση ταχύτητας όταν το λεωφορείο υπερβεί ένα δεδομένο όριο ταχύτητας.

- **Συσχέτιση δεδομένων, απόντα συμβάντα και εσφαλμένα δεδομένα.**
  Το μοτίβο αυτό έχει πολλά κοινά σημεία με την εντολή Join που αναφέραμε παραπάνω, και σε αυτή την περίπτωση συνδυάζουμε πολλές ροές. Επιπλέον, συσχετίζουμε τα δεδομένα στην ίδια ροή. Αυτό συμβαίνει επειδή διαφορετικοί αισθητήρες δεδομένων μπορεί να στέλνουν συμβάντα με διαφορετικούς ρυθμούς και πολλές περιπτώσεις χρήσης απαιτούν αυτόν τον θεμελιώδη χειριστή.

Ακολουθούν μερικά πιθανά σενάρια.

- Αντιστοίχιση δύο ροών δεδομένων που στέλνουν συμβάντα σε διαφορετικές ταχύτητες.
- Εντοπισμός συμβάντος που λείπει από μια ροή δεδομένων (π.χ. εντοπισμός ενός αιτήματος πελάτη που δεν έχει απαντηθεί εντός 1 ώρας από τη λήψη του.)
- Εντοπισμός λανθασμένων δεδομένων (π.χ. Ανίχνευση αποτυχημένων αισθητήρων χρησιμοποιώντας ένα σύνολο αισθητήρων που παρακολουθούν επικαλυπτόμενες περιοχές και χρήση αυτών των περιττών δεδομένων για εύρεση εσφαλμένων αισθητήρων και αφαίρεση των δεδομένων τους από περαιτέρω επεξεργασία).

- **Αλληλεπίδραση με βάσεις δεδομένων:** Συχνά πρέπει να συνδυάσουμε τα δεδομένα σε πραγματικό χρόνο με τα δεδομένα ιστορικού που είναι αποθηκευμένα σε έναν δίσκο. Για παράδειγμα:
  - Τη χρονική στιγμή που πραγματοποιείται μια συναλλαγή, αναζητήστε την ηλικία χρησιμοποιώντας το αναγνωριστικό πελάτη από τη βάση δεδομένων πελατών που θα χρησιμοποιηθεί για τον εντοπισμό απάτης.
  - Έλεγχος ύπαρξης μιας συναλλαγής στις μαύρες ή λευκές λίστες μιας βάσης δεδομένων.

- **Εντοπισμός τάσεων**
  - Συχνά συναντάμε δεδομένα χρονοσειρών. Η ανίχνευση μοτίβων από δεδομένα χρονοσειρών και η ανάλυση τους είναι κοινές περιπτώσεις χρήσης.

Ακολουθούν μερικά από τα παραδείγματα εντοπισμού τάσεων.

- Άνοδος, πτώση τιμών
- Στροφή (εναλλαγή από άνοδο σε πτώση τιμών)
- Ακραίες τιμές
- Πολύπλοκες τάσεις όπως τριπλό κάτω μέρος κ.λπ.

Οι τάσεις αυτές είναι χρήσιμες σε μια μεγάλη ποικιλία περιπτώσεων χρήσης όπως:

- Χρηματιστήρια και αλγοριθμικές συναλλαγές.
- Εφαρμογή SLA (Συμφωνία επιπέδου υπηρεσίας), Αυτόματη κλιμάκωση και Εξισορρόπηση φορτίου.
- Προβλεπόμενη συντήρηση (π.χ. να υπολογίσουμε ότι ένας σκληρός δίσκος θα γεμίσει μέσα στην επόμενη εβδομάδα).

- **Ανίχνευση και μετάβαση σε λεπτομερή ανάλυση**

  Η κύρια ιδέα του προτύπου είναι να εντοπίσει μια κατάσταση που υποδηλώνει κάποια ανωμαλία και να την αναλύσει περαιτέρω χρησιμοποιώντας δεδομένα ιστορικού. Αυτό το μοτίβο χρησιμοποιείται σε περιπτώσεις όπου δεν μπορούμε να αναλύσουμε όλα τα δεδομένα με πλήρη λεπτομέρεια. Αντ ' αυτού, αναλύουμε τις ανώμαλες περιπτώσεις με πλήρη λεπτομέρεια.

  Ακολουθούν μερικά παραδείγματα.

  - Χρησιμοποιούμε βασικούς κανόνες για την ανίχνευση της απάτης (π.χ. συναλλαγές μεγάλων χρηματικών ποσών) και, στη συνέχεια, τραβάμε όλες τις συναλλαγές που έγιναν μέσω αυτής της πιστωτικής κάρτας για μεγαλύτερο χρονικό διάστημα (π.χ. δεδομένα 3 μηνών) από μια παρτίδα και εκτελούμε μια λεπτομερή ανάλυση.
  - Ενώ παρακολουθούμε τον καιρό, εντοπίζουμε συνθήκες όπως υψηλή θερμοκρασία ή χαμηλή ατμοσφαιρική πίεση σε μια δεδομένη περιοχή και, στη συνέχεια, ξεκινάμε μια τοπική πρόβλεψη υψηλής ανάλυσης σε αυτήν την περιοχή.
  - Μπορούμε να εντοπίσουμε έμπιστους πελάτες, για παράδειγμα μέσω των δαπανών άνω των 1000€ μέσα σε ένα μήνα και, στη συνέχεια, εκτελούμε ένα λεπτομερές μοντέλο για να αποφασίσουμε τις δυνατότητες προσφοράς μιας συμφωνίας.

- **Χρήση μοντέλων**

Η ιδέα είναι να εκπαιδεύσουμε ένα μοντέλο (συχνά ένα μοντέλο Machine Learning) και στη συνέχεια να το χρησιμοποιήσετε με Realtime για να λάβουμε αποφάσεις. Για παράδειγμα, μπορούμε να δημιουργήσουμε ένα μοντέλο χρησιμοποιώντας την R, να το εξαγάγουμε ως PMML (Predictive Model Markup Language) και να το χρησιμοποιήσουμε εντός του αγωγού πραγματικού χρόνου.

- **Εκτέλεση του ίδιου ερωτήματος σε δεδομένα δέσμης και πραγματικού χρόνου.**

  Αυτό το μοτίβο εκτελεί το ίδιο ερώτημα τόσο σε Realtime όσο και σε δεδομένα σε δέσμες. Χρησιμοποιείται συχνά για να καλύψει το κενό που δημιουργείτε στα δεδομένα λόγω επεξεργασίας δεδομένων δέσμης. Για παράδειγμα, εάν η μαζική επεξεργασία διαρκεί 15 λεπτά, τα αποτελέσματα δεν θα έχουν δεδομένα για τα τελευταία 15 λεπτά.

  Η ιδέα αυτού του μοτίβου ονομάζεται Αρχιτεκτονική Λάμδα (Lambda Architecture) και σκοπός του είναι η χρήση στοιχείων ανάλυσης πραγματικού χρόνου για να καλύψουμε το κενό. Η Αρχιτεκτονική Λάμδα αναλύεται στο κεφάλαιο 5.

4 Πλατφόρμες ροής δεδομένων

Ενώ η streaming SQL αποτελεί ένα αντικείμενο ενεργής έρευνας εδώ και τρεις δεκαετίες, η επεξεργασία ροής δεδομένων απέκτησε τα τελευταία χρόνια την προσοχή της αγοράς και πολλά συστήματα επεξεργασίας έχουν υιοθετήσει κάποια χαρακτηριστικά από τη λειτουργικότητα της SQL. Παρακάτω θα δούμε μερικές από αυτές όπως τα Apache Spark, Apache Samza, Apache Flink, Apache Kafka και την KSQL.

4.1 Apache Spark

Το Apache Spark είναι ένα γρήγορο και γενικής χρήσης σύμπλεγμα υπολογιστικών συστημάτων. Αρχικά αναπτύχθηκε στο Πανεπιστήμιο της Καλιφόρνια, στο AMPLab του Berkley , η βασική κωδικοποίηση του Spark αργότερα δωρίστηκε στο Apache Software Foundation, το οποίο έκτοτε το έχει διατηρήσει. Το Spark παρέχει μια διεπαφή για τον προγραμματισμό ολόκληρων ομάδων με έμμεσο παραλληλισμό δεδομένων και ανοχή σφαλμάτων.

Το Spark Streaming είναι μια επέκταση του πυρήνα API Spark που επιτρέπει την επεκτασιμότητα, υψηλής απόδοσης, ανεκτική σε σφάλματα επεξεργασίας ζωντανών ροών δεδομένων. Τα δεδομένα μπορούν να απορροφηθούν από πολλές πηγές όπως τα Kafka, Flume, Kinesis ή θύρες TCP και μπορούν να υποβληθούν σε επεξεργασία με τη χρήση σύνθετων αλγορίθμων που εκφράζονται με λειτουργίες υψηλού επιπέδου, όπως χαρτογράφηση (map), μείωση (reduce), σύνδεση (connect) και παραθύρου (window). Τέλος, τα επεξεργασμένα δεδομένα μπορούν να προωθηθούν σε συστήματα αρχείων, βάσεις δεδομένων και ζωντανούς πίνακες ελέγχου. Στην πραγματικότητα, μπορείτε να εφαρμόσετε αλγόριθμους μηχανικής εκμάθησης και επεξεργασίας γραφημάτων του Spark σε ροές δεδομένων.

Το Apache Spark μπορεί να θεωρηθεί ως μια ολοκληρωμένη λύση για επεξεργασία εκμεταλλευόμενη όλα τα επίπεδα της αρχιτεκτονικής Lambda. Περιέχει Spark Core που περιλαμβάνει API υψηλού επιπέδου και βελτιστοποιημένους μηχανισμούς που υποστηρίζει γραφήματα, Spark SQL για SQL καθώς και δομημένη επεξεργασία δεδομένων, και Spark Streaming που επιτρέπει την επεκτάσιμη επεξεργασία δεδομένων

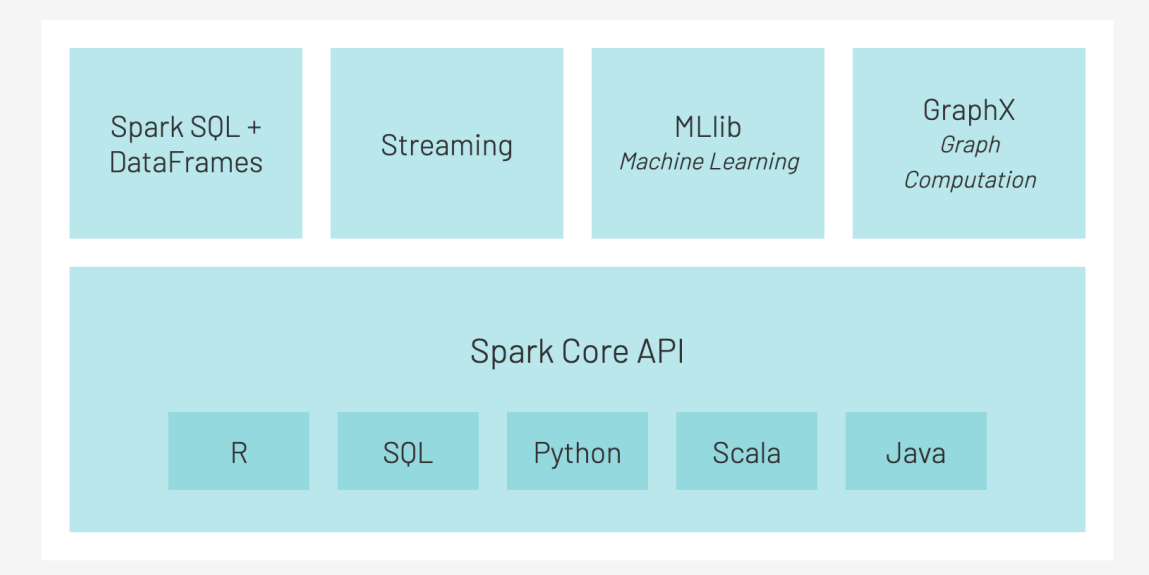
ροής πραγματικού χρόνου με υψηλή απόδοση και ανοχή στα λάθη. Σίγουρα, η επεξεργασία δεδομένων δέσμης που χρησιμοποιεί το Spark μπορεί να είναι αρκετά ακριβή υπολογιστικά και μπορεί να μην ταιριάζει σε όλα τα σενάρια και τους όγκους δεδομένων, αλλά, πέρα από αυτό, είναι μια αξιοπρεπής αντιστοίχιση για την υλοποίηση της αρχιτεκτονικής Lambda.

Επίσης, παρέχει μια αφαίρεση υψηλού επιπέδου που ονομάζεται διακριτική ροή ή DStream, η οποία αντιπροσωπεύει μια συνεχή ροή δεδομένων. Τα DStreams μπορούν να δημιουργηθούν είτε από ροές δεδομένων εισόδου από πηγές όπως Kafka, Flume και Kinesis, είτε εφαρμόζοντας λειτουργίες υψηλού επιπέδου σε άλλες DStreams. Εσωτερικά, ένα DStream αντιπροσωπεύεται ως ακολουθία RDDs. (Wissem Inoubli, 2018)

Εικόνα 9: Η ροή δεδομένων στο Spark streaming
πηγή : https://spark.apache.org/docs/latest/img/streaming-flow.png

Το Spark μπορεί να αναπτυχθεί με διάφορους τρόπους, παρέχει εγγενείς συνδέσεις για τις γλώσσες προγραμματισμού Java, Scala, Python και R και υποστηρίζει SQL, ροή δεδομένων, μηχανική εκμάθηση και επεξεργασία γραφημάτων. Θα το συναντήσουμε σε τράπεζες, εταιρείες τηλεπικοινωνιών, εταιρείες παιχνιδιών, κυβερνήσεις και όλους τους

μεγάλους τεχνολογικούς γίγαντες όπως η Apple, το Facebook, η IBM και η Microsoft.



Εικόνα 10: Το οικοσύστημα του Apache Spark
πηγή: https://databricks.com/spark/about

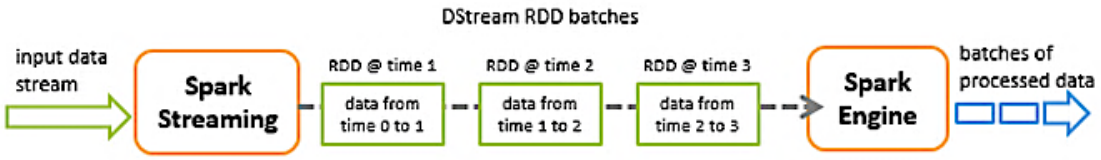
Οι βασικές λειτουργίες του Spark Streaming (Apache, 2020) **είναι οι ακόλουθες:**

1. Διακριτές ροές (DStreams)
2. Εισερχόμενα DStreams και Αποδέκτες
3. Μετασχηματισμοί σε DStreams
4. Δεδομένα εξόδου σε DStreams
5. DataFrame και Λειτουργίες SQL
6. Λειτουργίες MLlib
7. Caching / Προσωρινή αποθήκευση
8. Σημεία ελέγχου
9. Συσσωρευτές, μεταβλητές εκπομπής και σημεία ελέγχου

Πιο αναλυτικά :

1. **Διακριτές ροές (DStreams)**: Το Discretized Stream ή αλλιώς DStream είναι η βασική αφαίρεση που παρέχεται από το Spark Streaming. Αντιπροσωπεύει μια συνεχή ροή δεδομένων, είτε τη ροή δεδομένων εισαγωγής που λαμβάνεται από την πηγή, είτε τη επεξεργασμένη ροή δεδομένων που δημιουργείται μετασχηματίζοντας τη ροή εισόδου. Εσωτερικά, ένα DStream αντιπροσωπεύεται από μια συνεχή σειρά RDDs, η

οποία είναι η αφηρημένη απεικόνιση του Spark από ένα αμετάβλητο, κατανεμημένο σύνολο δεδομένων. Κάθε RDD σε ένα DStream περιέχει δεδομένα από ένα συγκεκριμένο διάστημα, όπως φαίνεται στην εικόνα 11 (McDonald, 2015). Κάθε λειτουργία που εφαρμόζεται σε μια ροή DSt μεταφράζεται σε λειτουργίες των υποκείμενων RDD.



Εικόνα 11: Dstreams RDD δέσμες
πηγή: https://mapr.com/blog/spark-streaming-hbase/assets/blogimages/sparkstream2-blog.png

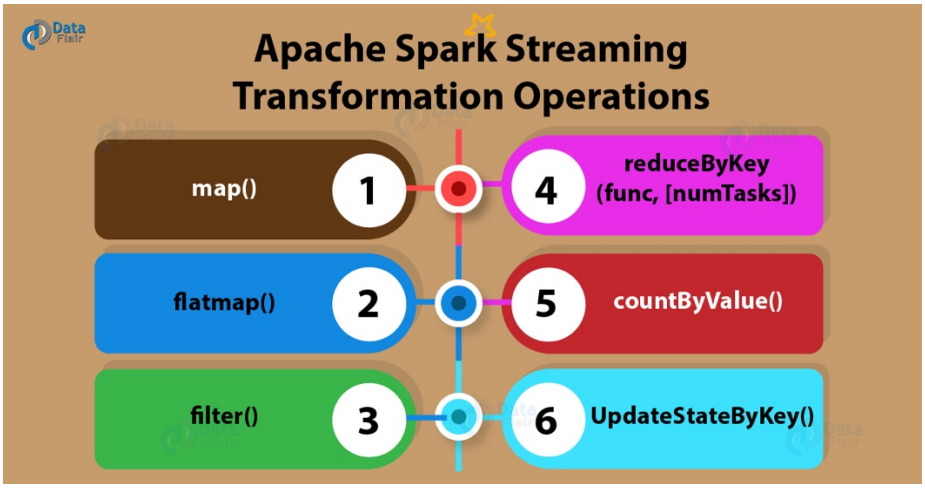
2. **Εισερχόμενα DStreams και Αποδέκτες :** Τα εισερχόμενα DStreams είναι DStreams που αντιπροσωπεύουν τη ροή δεδομένων εισόδου που λαμβάνονται από πηγές ροής. Κάθε είσοδος DStream σχετίζεται με ένα αντικείμενο δέκτη που λαμβάνει τα δεδομένα από μια πηγή και τα αποθηκεύει στη μνήμη του Spark για επεξεργασία. Το Spark Streaming παρέχει δύο κατηγορίες ενσωματωμένων πηγών ροής:
    - Βασικές πηγές: Πηγές απευθείας διαθέσιμες στο API StreamingContext. Παραδείγματα: συστήματα αρχείων και συνδέσεις υποδοχών.
    - Προηγμένες πηγές: Πηγές όπως Kafka, Flume, Kinesis κ.λπ. είναι διαθέσιμες μέσω επιπλέον utility κλάσεων. Αυτά απαιτούν σύνδεση έναντι επιπλέον εξαρτήσεων.

    Τα εισερχόμενα DStreams μπορούν επίσης να δημιουργηθούν από προσαρμοσμένες πηγές δεδομένων.

    Όσον αφορά την αξιοπιστία του παραλήπτη, μπορεί να υπάρχουν δύο είδη πηγών δεδομένων με βάση την αξιοπιστία τους. Πηγές όπως το Kafka και το Flume επιτρέπουν την αναγνώριση των δεδομένων που μεταφέρονται. Εάν το σύστημα που λαμβάνει δεδομένα από αυτές τις αξιόπιστες πηγές αναγνωρίζει σωστά τα ληφθέντα δεδομένα, μπορεί να διασφαλιστεί ότι δεν θα χαθούν δεδομένα λόγω οποιουδήποτε είδους αστοχίας. Αυτό οδηγεί σε δύο είδη αποδέκτες:

- Αξιόπιστος δέκτης: Ένας αξιόπιστος δέκτης στέλνει σωστά την επιβεβαίωση σε μια αξιόπιστη πηγή όταν τα δεδομένα έχουν ληφθεί και αποθηκευτεί στο Spark με αντιγραφή.
- Αναξιόπιστος δέκτης: Ένας αναξιόπιστος δέκτης δεν αποστέλλει επιβεβαίωση σε μια πηγή. Αυτό μπορεί να χρησιμοποιηθεί για πηγές που δεν υποστηρίζουν αναγνώριση, ή ακόμη και για αξιόπιστες πηγές όταν δεν θέλει ή χρειάζεται να μπει στην πολυπλοκότητα της αναγνώρισης.

3. **Μετασχηματισμοί σε DStreams** : Οι μετασχηματισμοί επιτρέπουν την τροποποίηση των δεδομένων από τα εισερχόμενα DStream. Μερικές από τις κοινές λειτουργίες παρατίθενται παρακάτω και τις βλέπουμε και στην εικόνα 12 (Team, 2018):



Εικόνα 12: Λειτουργίες Apache Spark
πηγή: DataFlair , https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2017/03/Apache-Spark-Streaming-Transformation-Operations-01.jpg

Πίνακας 1: Λίστα λειτουργιών Apache Spark
πηγή: DataFlair

Λειτουργίες	Ενέργειες
map(*func*)	Επιστρέφει ένα νέο Dstream προσπελαύνοντας κάθε στοιχείο της πηγής Dstream μέσω της συνάρτησης *func.*
flatMap(*func*)	Όμοιος με τον παραπάνω μετασχηματισμό map, μόνο που κάθε είσοδος μπορεί να γίνει map από 0 μέχρι και περισσότερα εξερχόμενα αντικείμενα.
filter(*func*)	Επιστρέφει ένα νέο Dstream επιλέγοντας μόνο τις εγγραφές της πηγής Dstream στην οποία η *func* επιστρέφει true.
reduceByKey(*func*,*[numTasks]*)	Όταν καλείται σε ζεύγη DStream (K, V), η συνάρτηση ReduceByKey στο Spark επιστρέφει ένα νέο ζεύγος DStream (K, V) όπου οι τιμές για κάθε κλειδί συγκεντρώνονται χρησιμοποιώντας τη συνάρτηση μείωσης που δίνουμε.
countByValue( )	Η συνάρτηση CountByValue καλείται σε DStream στοιχεία τύπου K και επιστρέφει ένα νέο ζεύγος DStream από (K, Long) ζεύγη όπου η τιμή κάθε κλειδιού είναι η συχνότητά του σε κάθε Spark RDD της πηγής DStream.
UpdateStateByKey ( )	Η λειτουργία updateStateByKey μας επιτρέπει να διατηρήσουμε μια κατάσταση, ενώ την ενημερώνουμε συνεχώς με νέες πληροφορίες. Για να τη χρησιμοποιήσουμε, θα πρέπει να κάνουμε τα εξής δύο βήματα: 1. **Να ορίσουμε την κατάσταση** - Η κατάσταση μπορεί να είναι ένας αυθαίρετος τύπος δεδομένων. 2. **Να ορίσουμε τη λειτουργία ενημέρωσης κατάστασης** – Καθορίζουμε με μια συνάρτηση πώς να ενημερώσουμε την κατάσταση χρησιμοποιώντας την προηγούμενη κατάσταση και τις νέες τιμές από μια ροή εισόδου. Σε κάθε δέσμη δεδομένων, το Spark θα εφαρμόσει τη συνάρτηση ενημέρωσης κατάστασης για όλα τα υπάρχοντα κλειδιά, ανεξάρτητα από το εάν έχουν νέα δεδομένα σε κάθε δέσμη δεδομένων ή όχι. Εάν η συνάρτηση ενημέρωσης επιστρέψει *None*, τότε το ζεύγος κλειδιού-τιμής θα εξαλειφθεί.

4. **Δεδομένα εξόδου σε DStreams** : Οι λειτουργίες εξόδου επιτρέπουν την προώθηση των δεδομένων της DStream σε εξωτερικά συστήματα, όπως μια βάση δεδομένων ή συστήματα αρχείων. Δεδομένου ότι οι λειτουργίες εξόδου επιτρέπουν στην πραγματικότητα τα μετασχηματισμένα δεδομένα να καταναλώνονται από εξωτερικά συστήματα, ενεργοποιούν την πραγματική εκτέλεση όλων των μετασχηματισμών DStream. Επί του παρόντος, καθορίζονται οι ακόλουθες λειτουργίες εξόδου, όπως και καταγράφηκαν και στο βιβλίο του Muhammad Asif Abbasi , *Learning Apache Spark 2* (Abbasi, 2017):

Πίνακας 2: Λειτουργίες εξόδου αποτελεσμάτων Apache Spark (Abbasi, 2017)

Λειτουργίες	Ενέργειες
print( )	Τυπώνει τα πρώτα δέκα στοιχεία κάθε δέσμης δεδομένων σε ένα Dstream στον κόμβο που εκτελείται η εφαρμογή ροής δεδομένων. Χρήσιμη λειτουργία για ανάπτυξη κώδικα και εκσφαλμάτωση.
saveAsTextFiles(*prefix,[suffix]***)**	Αποθηκεύει τα περιεχόμενα του Dstream ως αρχείο κειμένου. Το αρχείο κειμένου για κάθε δέσμη δημιουργείτε με βάση το *prefix* και το *suffix*. *"prefix-TIME_IN_MS[.suffix]"*
saveAsObjectFiles(*prefix,[suffix]***)**	Αποθηκεύει το περιεχόμενο του Dstream ως *SourceFiles* σειριοποιημένων αντικειμένων Java.
saveAsHadoopFiles(*prefix,[suffix]***)**	Αποθηκεύει το περιεχόμενο Dstream ως αρχείο Hadoop. Το όνομα αρχείου για κάθε δέσμη δημιουργείτε με βάση το *prefix* και το *suffix*. *"prefix-TIME_IN_MS[.suffix]"*
foreachRDD(*func***)**	Ο πιο γενικός τελεστής εξαγωγής αποτελέσματος ο οποίος εφαρμόζει μια συνάρτηση, *func,* σε κάθε RDD που παράγεται από τη ροή δεδομένων. Η συνάρτηση αυτή πρέπει να ωθήσει τα δεδομένα κάθε RDD σε ένα εξωτερικό σύστημα, όπως η αποθήκευση του RDD σε αρχεία ή εγγραφεί μέσω δικτύου σε μια βάση δεδομένων. Σημειώστε ότι η συνάρτηση func εκτελείται κατά τη διαδικασία εκτέλεσης του βασικού προγράμματος που εκτελεί την εφαρμογή ροής δεδομένων και συνήθως πραγματοποιεί ενέργειες RDD σε αυτό που θα επιβάλλουν τον υπολογισμό των RDD ροής.

5. **DataFrame και Λειτουργίες SQL** : Μπορεί ο οποιοσδήποτε να χρησιμοποιήσει εύκολα πλεόν DataFrames και SQL λειτουργίες για ροή δεδομένων. Κάθε RDD μετατρέπεται σε DataFrame, καταχωρείται ως προσωρινός πίνακας και κατόπιν του υποβάλλονται ερωτήματα SQL. Επίσης, επιτρέπει στο χρήστη να εκτελεί ερωτήματα SQL σε πίνακες που ορίζονται σε ροή δεδομένων από διαφορετικό νήμα.

6. **Λειτουργίες MLlib** : Το Spark προσφέρει επίσης τη δυνατότητα χρήσης αλγορίθμων μηχανικής μάθησης που παρέχονται από το MLlib.

7. **Caching / Προσωρινή αποθήκευση** : Παρόμοια με τα RDD, τα DStreams επιτρέπουν επίσης στους προγραμματιστές να διατηρούν τα δεδομένα της ροής στη μνήμη. Δηλαδή, η χρήση της μεθόδου persist () σε ένα DStream θα διατηρήσει αυτόματα κάθε RDD αυτού του DStream στη μνήμη. Η λειτουργία αυτή είναι χρήσιμη σε περιπτώσεις που τα δεδομένα στο DStream θα υπολογιστούν πολλές φορές (π.χ. πολλαπλές λειτουργίες στα ίδια δεδομένα). Ως εκ τούτου, τα DStreams που δημιουργούνται από λειτουργίες που βασίζονται σε παράθυρο παραμένουν αυτόματα στη μνήμη, χωρίς να χρειάζεται ο προγραμματιστής να καλέσει τη μέθοδο persist().

   Για ροές εισόδου που λαμβάνουν δεδομένα μέσω του δικτύου (όπως, Kafka, Flume, sockets κ.λπ.), το προεπιλεγμένο επίπεδο διάρκειας έχει οριστεί για την αναπαραγωγή των δεδομένων σε δύο κόμβους για ανοχή σφαλμάτων. Σημειώστε ότι, σε αντίθεση με τα RDD, το προεπιλεγμένο επίπεδο επιμονής των DStreams διατηρεί τα δεδομένα σε σειριακή μνήμη.

8. **Σημεία ελέγχου** : Μια εφαρμογή ροής πρέπει να λειτουργεί 24/7 και ως εκ τούτου πρέπει να είναι ανθεκτική σε αστοχίες που δεν σχετίζονται με τη λογική της εφαρμογής, όπως αποτυχίες συστήματος, σφάλματα JVM, κ.λπ. έτσι ώστε να μπορεί να ανακάμψει από αστοχίες. Υπάρχουν δύο τύποι δεδομένων που ελέγχονται:
   - Σημείο ελέγχου metadata
   - Έλεγχος δεδομένων

   Ο έλεγχος των metadata είναι πρωταρχικά απαραίτητος για την ανάκτηση από αστοχίες του προγράμματος οδήγησης που χρησιμοποιεί την εφαρμογή, ενώ τα δεδομένα ή τα σημεία ελέγχου RDD είναι απαραίτητα ακόμη και για βασική λειτουργία, εάν χρησιμοποιούνται μετασχηματισμοί.

Πλεονεκτήματα Apache Spark :

- Υποστηρίζει την αρχιτεκτονική Lambda και παρέχεται δωρεάν με το Apache Spark.
- Δίνει τη δυνατότητα υψηλών αποδόσεων και αποτελεί καλή λύση για πολλές περιπτώσεις χρήσης όπου δεν απαιτείται καθυστέρηση.
- Έχει υψηλή ανοχή σφαλμάτων
- Απλό στη χρήση API υψηλότερου επιπέδου
- Αναφέρεται σε μεγάλο ποσοστό χρηστών και επιδέχεται ραγδαίων βελτιώσεων

Μειονεκτήματα:

- Δεν υποστηρίζει πραγματική ροή δεδομένων, δεν είναι κατάλληλο για απαιτήσεις χαμηλού λανθάνοντος χρόνου.
- Αποτελείται από πάρα πολλές παραμέτρους και είναι δύσκολο στη ρύθμισή του.
- Υστερεί αρκετά συγκριτικά με το Apache Flink το οποίο περιέχει προηγμένα χαρακτηριστικά.

4.2 Apache Flink

Στις μέρες μας, σχεδόν όλα τα δεδομένα δημιουργούνται συνεχώς ως ροές γεγονότων. Αυτό περιλαμβάνει επιχειρηματικές συναλλαγές, αλληλεπιδράσεις με εφαρμογές ιστού ή για κινητές συσκευές, αρχεία καταγραφής αισθητήρων και τροποποιήσεις βάσεων δεδομένων. Όπως αναφέρθηκε και στα πρώτα κεφάλαια, υπάρχουν δύο τρόποι επεξεργασίας δεδομένων που παράγονται συνεχώς, η επεξεργασία παρτίδας και ροής δεδομένων. Στην επεξεργασία ροής, τα δεδομένα απορροφώνται άμεσα και υποβάλλονται σε επεξεργασία από μια εφαρμογή που συνεχώς εκτελείται καθώς φθάνουν σε εκείνη τα δεδομένα. Στη μαζική επεξεργασία, τα δεδομένα καταγράφονται πρώτα και διατηρούνται σε ένα σύστημα αποθήκευσης, όπως ένα σύστημα αρχείων ή ένα σύστημα βάσης δεδομένων, πριν υποβληθούν σε επεξεργασία (περιοδικά) από μια εφαρμογή που επεξεργάζεται ένα οριοθετημένο σύνολο δεδομένων, τις δέσμες. Ενώ η επεξεργασία ροής συνήθως επιτυγχάνει χαμηλότερους χρόνους καθυστέρησης για την παραγωγή αποτελεσμάτων, προκαλεί λειτουργικές προκλήσεις, καθώς οι εφαρμογές ροής εκτελούνται συνεχώς και αδιαλείπτως, απαιτούν υψηλές αποδόσεις ώστε να επιτυγχάνονται η ανάκτηση αποτυχίας και η εγγύηση συνέπειας. Η θεμελιώδης διαφορά μεταξύ εφαρμογών επεξεργασίας παρτίδας και δεδομένων ροής είναι ότι οι εφαρμογές επεξεργασίας ροής επεξεργάζονται συνεχώς ληφθέντα δεδομένα.

Το Apache Flink είναι ένα σύστημα επεξεργασίας δεδομένων δέσμης και ροής. Αντιμετωπίζει πολλές προκλήσεις που σχετίζονται με την επεξεργασία οριοθετημένων και μη οριοθετημένων δεδομένων. Μεταξύ άλλων, το Flink παρέχει ευέλικτη υποστήριξη παραθύρου (windowing), συνεκτικότητα κατάστασης μόλις μία φορά (exactly-once[1]), σημασιολογία χρόνου εκδήλωσης και επεξεργασία ροής κατάστασης. Προσφέρει σύνθετη επεξεργασία συμβάντων και συνεχή εκτέλεση ερωτημάτων.

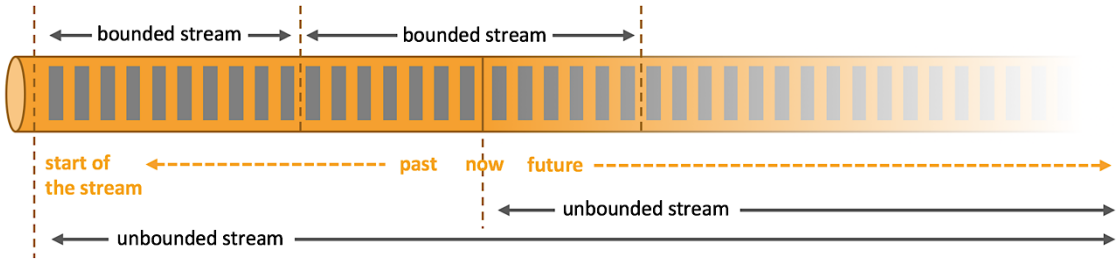
Η αρχιτεκτονική του Apache Flink

Οι μη οριοθετημένες ροές δεδομένων (unbounded streams) έχουν αρχή αλλά όχι καθορισμένο τέλος. Δεν τερματίζουν και παρέχουν δεδομένα καθώς δημιουργούνται. Οι μη περιορισμένες ροές πρέπει να υποβάλλονται σε συνεχή επεξεργασία, δηλαδή, τα συμβάντα πρέπει να αντιμετωπίζονται αμέσως μετά τη λήψη τους. Δεν είναι δυνατόν να

[1] Με τον όρο exactly-once εννοούμε ότι κάθε εισερχόμενο συμβάν επηρεάζει τα τελικά αποτελέσματα ακριβώς μία φορά. Ακόμα και σε περίπτωση βλάβης του μηχανήματος ή του λογισμικού, δεν υπάρχουν διπλά δεδομένα καθώς και δεδομένα που χρησιμοποιούνται χωρίς επεξεργασία.

περιμένετε να φτάσουν όλα τα δεδομένα εισόδου, επειδή η είσοδος δεν είναι περιορισμένη και δεν θα είναι πλήρης οποιαδήποτε στιγμή. Η επεξεργασία μη περιορισμένων δεδομένων απαιτεί συχνά τα συμβάντα να απορροφώνται με μια συγκεκριμένη σειρά, όπως η σειρά με την οποία συνέβησαν τα γεγονότα, για να μπορούν να αιτιολογούν την πληρότητα των αποτελεσμάτων.

Οι οριοθετημένες ροές δεδομένων (bounded streams) έχουν καθορισμένη αρχή και τέλος. Επίσης, μπορούν να υποβληθούν σε επεξεργασία με απορρόφηση όλων των δεδομένων προτού εκτελέσουμε υπολογισμούς. Δεν απαιτείται η επεξεργασία τους βάσει σειράς απορρόφησης στην επεξεργασία οριοθετημένων ροών, επειδή ένα σύνολο οριοθετημένων δεδομένων μπορεί πάντα να ταξινομηθεί. Η επεξεργασία των οριοθετημένων ροών είναι επίσης γνωστή ως επεξεργασία σε παρτίδες. (βλ. Εικόνα 13)



Εικόνα 13: Οριοθετημένες και μη οριοθετημένες ροές δεδομένων στο Apache Flink
πηγή: https://flink.apache.org/flink-architecture.html

Το Apache Flink εξειδικεύεται στην επεξεργασία οριοθετημένων και μη οριοθετημένων συνόλων δεδομένων. Ο ακριβής έλεγχος του χρόνου και της κατάστασης επιτρέπει στον χρόνο εκτέλεσης του Flink να εκτελεί οποιοδήποτε είδος εφαρμογής σε μη περιορισμένες ροές. Οι οριοθετημένες ροές επεξεργάζονται εσωτερικά από αλγόριθμους και δομές δεδομένων που έχουν σχεδιαστεί ειδικά για σύνολα δεδομένων σταθερού μεγέθους, αποδίδοντας εξαιρετική απόδοση.

Αποτελεί ένα κατανεμημένο σύστημα και απαιτεί πόρους υπολογισμού για την εκτέλεση εφαρμογών. Το Flink ενσωματώνεται με όλους τους κοινούς διαχειριστές cluster πόρων, όπως Hadoop YARN[2], Apache Mesos και Kubernetes, αλλά μπορεί επίσης να

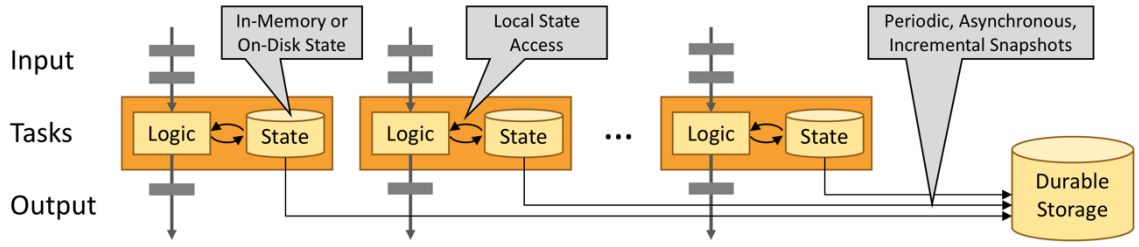
[2] Συλλογή βοηθητικών προγραμμάτων λογισμικού ανοιχτού κώδικα που διευκολύνουν τη χρήση ενός δικτύου πολλών υπολογιστών στην επίλυση προβλημάτων που περιλαμβάνουν τεράστιες ποσότητες δεδομένων και υπολογισμούς. Παρέχει ένα πλαίσιο λογισμικού για κατανεμημένη αποθήκευση και επεξεργασία μεγάλων δεδομένων χρησιμοποιώντας το μοντέλο προγραμματισμού MapReduce.

ρυθμιστεί ώστε να λειτουργεί ως αυτόνομο σύμπλεγμα. (The Apache Software Foundation, 2019)

Το Flink έχει σχεδιαστεί για την εκτέλεση εφαρμογών streaming σε οποιαδήποτε κλίμακα. Οι εφαρμογές παραλληλίζονται σε πιθανώς χιλιάδες διεργασίες που διανέμονται και εκτελούνται ταυτόχρονα σε ένα cluster. Επομένως, μια εφαρμογή μπορεί να αξιοποιήσει σχεδόν απεριόριστα τη CPU, την κύρια μνήμη, το δίσκο και ένα δίκτυο IO[3]. Ο ασύγχρονος και αυξητικός αλγόριθμος σημείων ελέγχου που διαθέτει το Flink, εξασφαλίζει ελάχιστο αντίκτυπο στις καθυστερήσεις επεξεργασίας.

Οι χρήστες του ανέφεραν εντυπωσιακούς αριθμούς επεκτασιμότητας για τις εφαρμογές του Flink που εκτελούνται στους χώρους παραγωγής τους, όπως εφαρμογές που επεξεργάζονται πολλαπλά τρισεκατομμύρια συμβάντα την ημέρα, εφαρμογές που διατηρούν πολλά terabyte δεδομένων, και εφαρμογές που εκτελούνται σε χιλιάδες πυρήνες.

Οι εφαρμογές του Flink είναι βελτιστοποιημένες για τοπική πρόσβαση. Η κατάσταση εργασιών διατηρείται πάντα στη μνήμη ή σε δομές δεδομένων που προσφέρουν αποδοτικό τρόπο πρόσβασης στο δίσκο εφόσον το μέγεθος δεν υπερβαίνει τη διαθέσιμη μνήμη. Ως εκ τούτου, οι διεργασίες εκτελούν όλους τους υπολογισμούς, αποκτώντας τοπική πρόσβαση, συχνά στη μνήμη, αποδίδοντας πολύ χαμηλές καθυστερήσεις επεξεργασίας. Το Flink εγγυάται τη συνοχή της κατάστασης ακριβώς σε μια φορά σε περίπτωση βλάβης, ελέγχοντας περιοδικά και ασύγχρονα την τοπική κατάσταση σε διαρκή αποθήκευση.



Εικόνα 14: Αξιοποίηση απόδοσης μνήμης
πηγή: https://flink.apache.org/flink-architecture.html

[3] Τα δίκτυα IO (Input/Output) είναι ουσιαστικά απλώς χρόνος που αποδίδεται στην αποστολή δεδομένων μεταξύ των ελεγχόμενων διαδικασιών.

Οι εφαρμογές του Apache Flink

Το Apache Flink είναι ένα πλαίσιο για υπολογισμούς σε μη περιορισμένες (unbounded) και οριοθετημένες ροές δεδομένων (bounded). Παρέχει πολλαπλά API σε διαφορετικά επίπεδα αφαίρεσης και προσφέρει ειδικές βιβλιοθήκες για κοινές περιπτώσεις. Παρακάτω, θα παρουσιάσουμε εύχρηστα API και βιβλιοθήκες του Flink.

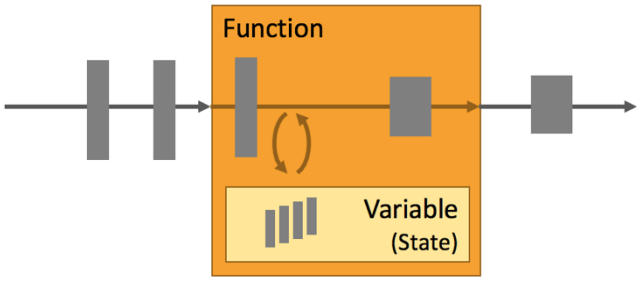
Οι τύποι εφαρμογών που μπορούν να δημιουργηθούν και να εκτελεστούν από ένα πλαίσιο επεξεργασίας ροής καθορίζονται από το πόσο καλά το πλαίσιο ελέγχει τις ροές δεδομένων, την κατάσταση και τον χρόνο. Στη συνέχεια, περιγράφουμε αυτά τα δομικά στοιχεία για εφαρμογές επεξεργασίας ροής δεδομένων και εξηγούμε τις προσεγγίσεις του Flink για τον χειρισμό τους.

Προφανώς, οι ροές είναι μια θεμελιώδης πτυχή της επεξεργασίας ροής. Ωστόσο, οι ροές μπορεί να έχουν διαφορετικά χαρακτηριστικά που επηρεάζουν τον τρόπο με τον οποίο μπορεί και πρέπει να επεξεργάζεται μια ροή. Το Flink είναι ένα ευέλικτο πλαίσιο επεξεργασίας που μπορεί να χειριστεί οποιοδήποτε είδος ροής.

- Οριοθετημένες (bounded) και χωρίς περιορισμούς ροές (unbounded): Οι ροές μπορούν να είναι χωρίς περιορισμούς ή οριοθέτηση, δηλαδή, σύνολα δεδομένων σταθερού μεγέθους. Το Flink διαθέτει εξελιγμένα χαρακτηριστικά για την επεξεργασία μη περιορισμένων ροών, αλλά και τελεστές για την αποτελεσματική επεξεργασία οριοθετημένων ροών.
- Ροές σε πραγματικό χρόνο και καταγεγραμμένες ροές: Όλα τα δεδομένα δημιουργούνται ως ροές. Υπάρχουν δύο τρόποι επεξεργασίας των δεδομένων. Επεξεργασία σε πραγματικό χρόνο καθώς δημιουργείται ή συνεχίζει τη ροή σε ένα σύστημα αποθήκευσης, π.χ. σε ένα σύστημα αρχείων ή ένα αντικείμενο αποθήκευσης και το επεξεργάζεται αργότερα. Οι εφαρμογές Flink μπορούν να επεξεργαστούν εγγεγραμμένες ή σε πραγματικό χρόνο ροές.

Κάθε σημαντική εφαρμογή ροής αποτελεί μια κατάσταση, δηλαδή, μόνο εφαρμογές που εφαρμόζουν μετασχηματισμούς σε μεμονωμένα συμβάντα δεν απαιτούν μια κατάσταση. Κάθε εφαρμογή που εκτελεί μια βασική λογική πρέπει να θυμάται συμβάντα ή ενδιάμεσα αποτελέσματα για να έχει πρόσβαση σε αυτά αργότερα, για

παράδειγμα όταν λαμβάνεται το επόμενο συμβάν ή μετά από συγκεκριμένη χρονική διάρκεια.



Εικόνα 15: Η κατάσταση (state) σε μια ροή δεδομένων του Flink.
πηγή: https://flink.apache.org/flink-applications.html

Η κατάσταση εφαρμογής έχει πρωταρχικό ρόλο στο Flink. Τη σημαντικότητα αυτή θα την ανακαλύψουμε εξετάζοντας όλες τις δυνατότητες που παρέχει το Flink στο πλαίσιο διαχείρισης κατάστασης.

- Multiple State Primitives: Το Flink παρέχει πρωτόκολλα κατάστασης για διαφορετικές δομές δεδομένων, όπως ατομικές τιμές, λίστες ή χάρτες. Οι προγραμματιστές μπορούν να επιλέξουν την αρχική κατάσταση που είναι πιο αποτελεσματική με βάση το μοτίβο πρόσβασης της συνάρτησης.

- State Backends με δυνατότητες σύνδεσης: Η κατάσταση της εφαρμογής διαχειρίζεται και ελέγχεται από ένα backend με δυνατότητες σύνδεσης κατάστασης. Το Flink διαθέτει διαφορετικές καταστάσεις backend που αποθηκεύουν στη μνήμη ή στο RocksDB[4], ένα αποτελεσματικό ενσωματωμένο χώρο αποθήκευσης δεδομένων στο δίσκο. Τα προσαρμοσμένα backends μπορούν επίσης να συνδεθούν.

- Συνάφεια κατάστασης ακριβείας (exactly-once state): Οι αλγόριθμοι ελέγχου και ανάκτησης του Flink εγγυώνται τη συνέπεια της κατάστασης εφαρμογής σε περίπτωση αποτυχίας. Ως εκ τούτου, οι αστοχίες αντιμετωπίζονται με διαφάνεια και δεν επηρεάζουν την ορθότητα μιας εφαρμογής.

[4] Το RocksDB είναι μια ενσωματωμένη βάση δεδομένων υψηλής απόδοσης για δεδομένα κλειδιού-τιμής (key-value). Είναι ένα παρακλάδι του LevelDB της Google που έχει βελτιστοποιηθεί για να εκμεταλλευτεί πολλούς πυρήνες κεντρικής μονάδας επεξεργασίας (CPU) και να κάνει αποτελεσματική χρήση γρήγορης αποθήκευσης, όπως μονάδες στερεάς κατάστασης (SSD), για φόρτους εισόδου / εξόδου (I / O).

- Πολύ μεγάλη κατάσταση: Το Flink είναι σε θέση να διατηρήσει την κατάσταση εφαρμογής σε μέγεθος αρκετών terabyte λόγω του ασύγχρονου και σταδιακού αλγορίθμου σημείου ελέγχου.

- Εξελικτικές Εφαρμογές: Το Flink υποστηρίζει την κλιμάκωση των κρατικών εφαρμογών αναδιανέμοντας την κατάσταση σε περισσότερους ή λιγότερους εργαζόμενους.

Ο χρόνος είναι ένα άλλο σημαντικό συστατικό των εφαρμογών ροής. Οι περισσότερες ροές συμβάντων έχουν εγγενή χρονική σημασιολογία επειδή κάθε συμβάν παράγεται σε μια συγκεκριμένη χρονική στιγμή. Επιπλέον, πολλοί συνηθισμένοι υπολογισμοί ροής βασίζονται στον χρόνο, όπως συγκεντρώσεις παραθύρων, σύνοδος λειτουργίας, ανίχνευση προτύπων και συνδέσεις βάσει χρόνου. Μια σημαντική πτυχή της επεξεργασίας ροής είναι ο τρόπος με τον οποίο μια εφαρμογή μετρά το χρόνο, δηλαδή τη διαφορά του χρόνου συμβάντος και του χρόνου επεξεργασίας.

Το Flink παρέχει ένα πλούσιο σύνολο λειτουργιών που σχετίζονται με το χρόνο.

- Λειτουργία χρόνου συμβάντος: Οι εφαρμογές που επεξεργάζονται ροές με σημασιολογία χρόνου συμβάντος υπολογίζουν τα αποτελέσματα με βάση τις χρονικές σημάνσεις των συμβάντων. Με αυτόν τον τρόπο, η επεξεργασία του συμβάντος επιτρέπει ακριβή και συνεπή αποτελέσματα ανεξάρτητα από το εάν υποβάλλονται σε επεξεργασία τα γεγονότα που έχουν καταγραφεί ή σε πραγματικό χρόνο.

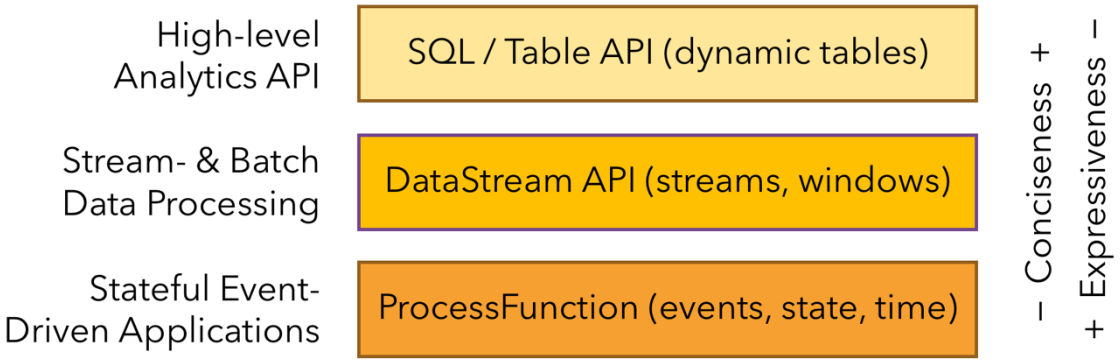
- Υποστήριξη υδατογραφήματος: Το Flink χρησιμοποιεί υδατογραφήματα για να αιτιολογήσει το χρόνο σε εφαρμογές χρόνου εκδήλωσης. Τα υδατογραφήματα είναι επίσης ένας ευέλικτος μηχανισμός για την αντιστάθμιση της καθυστέρησης και της πληρότητας των αποτελεσμάτων.

- Αργός χειρισμός δεδομένων: Κατά την επεξεργασία ροών σε λειτουργία χρόνου-συμβάντος με υδατογραφήματα, μπορεί να συμβεί ότι ο υπολογισμός έχει ολοκληρωθεί πριν από την άφιξη όλων των σχετικών συμβάντων. Τέτοια γεγονότα ονομάζονται καθυστερημένα γεγονότα. Το Flink διαθέτει πολλές επιλογές για τον χειρισμό καθυστερημένων συμβάντων, όπως την αναδρομολόγηση μέσω

πλευρικών εξόδων και την ενημέρωση των αποτελεσμάτων που έχουν ήδη ολοκληρωθεί.

- Λειτουργία χρόνου επεξεργασίας: Εκτός από τη λειτουργία χρόνου εκδήλωσης, το Flink υποστηρίζει επίσης τη σημασιολογία του χρόνου επεξεργασίας που εκτελεί υπολογισμούς όπως ενεργοποιείται από τον χρόνο ρολογιού του μηχανήματος επεξεργασίας. Ο τρόπος επεξεργασίας-χρόνου μπορεί να είναι κατάλληλος για ορισμένες εφαρμογές με αυστηρές απαιτήσεις χαμηλού λανθάνοντος χρόνου που μπορούν να ανεχθούν κατά προσέγγιση αποτελέσματα.

Το Flink παρέχει API τριών επιπέδων. Κάθε API προσφέρει μια διαφορετική αντιστάθμιση μεταξύ συνοπτικότητας και εκφραστικότητας και στοχεύει διαφορετικές περιπτώσεις χρήσης.



Εικόνα 16: Διαστρωματοποιημένο Flink API
πηγή: https://flink.apache.org/flink-applications.html

1. ProcessFunction

Οι ProcessFunctions είναι οι πιο εκφραστικές διεπαφές λειτουργίας που προσφέρει το Flink και παρέχονται για την επεξεργασία μεμονωμένων συμβάντων από μία ή δύο ροές εισόδου ή συμβάντα που ομαδοποιήθηκαν σε ένα παράθυρο. Τα ProcessFunctions παρέχουν λεπτομερή έλεγχο του χρόνου και της κατάστασης. Τα ProcessFunction μπορούν να τροποποιήσουν αυθαίρετα την κατάστασή τους και να καταχωρήσουν χρονοδιακόπτες που θα ενεργοποιήσουν μια λειτουργία επανάκλησης στο μέλλον. Ως εκ τούτου, τα ProcessFunctions μπορούν να εφαρμόσουν μια σύνθετη επιχειρηματική λογική ανά συμβάν, όπως απαιτείται για πολλές εφαρμογές που βασίζονται σε συμβάντα.

Το παρακάτω παράδειγμα (The Apache Software Foundation, 2019) δείχνει ένα KeyedProcessFunction που λειτουργεί σε KeyedStream και συνδυάζει τα γεγονότα START και END. Όταν λαμβάνεται ένα συμβάν START, η συνάρτηση θυμάται τη χρονική σήμανση σε κατάσταση και καταγράφει ένα χρονόμετρο σε τέσσερις ώρες. Εάν ληφθεί ένα συμβάν END πριν ενεργοποιηθεί το χρονόμετρο, η συνάρτηση υπολογίζει τη διάρκεια μεταξύ του συμβάντος END και START, διαγράφει την κατάσταση και επιστρέφει την τιμή. Διαφορετικά, ο χρονοδιακόπτης απλώς ενεργοποιεί και διαγράφει την κατάσταση.

```
/**
 *Matches keyed START and END events and computes the difference
 * between both elements' timestamps. The first String field is the key
 * attribute, the second String attribute marks START and END events.
 */
public static class StartEndDuration
    extends KeyedProcessFunction<String, Tuple2<String, String>, Tuple2
<String, Long>> {
  private ValueState<Long> startTime;
  @Override
  public void open(Configuration conf) {
    // obtain state handle
    startTime = getRuntimeContext()
     .getState(new ValueStateDescriptor<Long>("startTime",Long.class));
  }

  /** Called for each processed event. */
  @Override
  public void processElement(
      Tuple2<String, String> in,
      Context ctx,
      Collector<Tuple2<String, Long>> out) throws Exception {

    switch (in.f1) {
      case "START":
        // set the start time if we receive a start event.
        startTime.update(ctx.timestamp());
        // register a timer in four hours from the start event.
```

```
        ctx.timerService()
          .registerEventTimeTimer(ctx.timestamp() + 4 * 60 * 60 * 1000)
;
        break;
      case "END":
        // emit the duration between start and end event
        Long sTime = startTime.value();
        if (sTime != null) {
          out.collect(Tuple2.of(in.f0, ctx.timestamp() - sTime));
          // clear the state
          startTime.clear();
        }
      default:
        // do nothing
    }
  }

  /** Called when a timer fires. */
  @Override
  public void onTimer(
      long timestamp,
      OnTimerContext ctx,
      Collector<Tuple2<String, Long>> out) {

    // Timeout interval exceeded. Cleaning up the state.
    startTime.clear();
  }
}
```

Πηγή παραδείγματος: https://flink.apache.org/flink-applications.html, (Foundation, 2019)

Το παραπάνω παράδειγμα απεικονίζει την εκφραστική ισχύ του KeyedProcessFunction, αλλά επίσης υπογραμμίζει το γεγονός ότι είναι μια λεκτική διεπαφή.

2. DataStream API

Το DataStream API παρέχει θεμελιώδη στοιχεία για πολλές κοινές λειτουργίες επεξεργασίας ροής, όπως τη λειτουργία παραθύρου, μετασχηματισμούς καταγραφής χρόνου και εμπλουτισμό συμβάντων με ερωτήματα σε εξωτερικούς χώρους αποθήκευσης δεδομένων. Το API DataStream είναι διαθέσιμο για Java και Scala και βασίζεται σε λειτουργίες, όπως map(), reduce() και aggregate(). Οι συναρτήσεις μπορούν να οριστούν επεκτείνοντας διεπαφές ή ως λειτουργίες Java ή Scala lambda.

Το ακόλουθο παράδειγμα (The Apache Software Foundation, 2019) δείχνει πώς μπορείτε να πραγματοποιήσετε περίοδο σύνδεσης σε μια ροή κλικ και να μετρήσετε τον αριθμό των κλικ ανά περίοδο σύνδεσης.

```
// a stream of website clicks
DataStream<Click> clicks = ...

DataStream<Tuple2<String, Long>> result = clicks
  // project clicks to userId and add a 1 for counting
  .map(
    // define function by implementing the MapFunction interface.
    new MapFunction<Click, Tuple2<String, Long>>() {
      @Override
      public Tuple2<String, Long> map(Click click) {
        return Tuple2.of(click.userId, 1L);
      }
    })
  // key by userId (field 0)
  .keyBy(0)
  // define session window with 30 minute gap
  .window(EventTimeSessionWindows.withGap(Time.minutes(30L)))
  // count clicks per session. Define function as lambda function.
  .reduce((a, b) -> Tuple2.of(a.f0, a.f1 + b.f1));
```

3. SQL & Table API

Το Flink διαθέτει δύο σχεσιακά API, το Table API και το SQL. Και τα δύο API είναι ενοποιημένα για επεξεργασία παρτίδας και ροής, δηλαδή, τα ερωτήματα εκτελούνται με την ίδια σημασιολογία σε μη περιορισμένες ροές σε πραγματικό χρόνο ή σε οριοθετημένες, καταγεγραμμένες ροές και παράγουν τα ίδια αποτελέσματα. Το Table API καθώς και το SQL αξιοποιούν το Apache Calcite για ανάλυση, επικύρωση και βελτιστοποίηση ερωτημάτων Μπορούν να ενσωματωθούν απρόσκοπτα με τα DataStream και DataSet APIs και να υποστηρίξουν λειτουργίες κλιμακωτής, συγκεντρωτικής και μορφής πίνακα που καθορίζονται από τον χρήστη.

Τα σχεσιακά API του Flink έχουν σχεδιαστεί για να διευκολύνουν τον ορισμό των αναλυτικών στοιχείων δεδομένων, των δεδομένων pipeline και των εφαρμογών ETL[5].

Το ακόλουθο παράδειγμα δείχνει ένα ερώτημα SQL για να πραγματοποιήσετε περίοδο σύνδεσης σε μια ροή κλικ και να μετρήσετε τον αριθμό των κλικ ανά περίοδο σύνδεσης. Αυτή είναι η ίδια περίπτωση χρήσης όπως στο παράδειγμα του DataStream API.

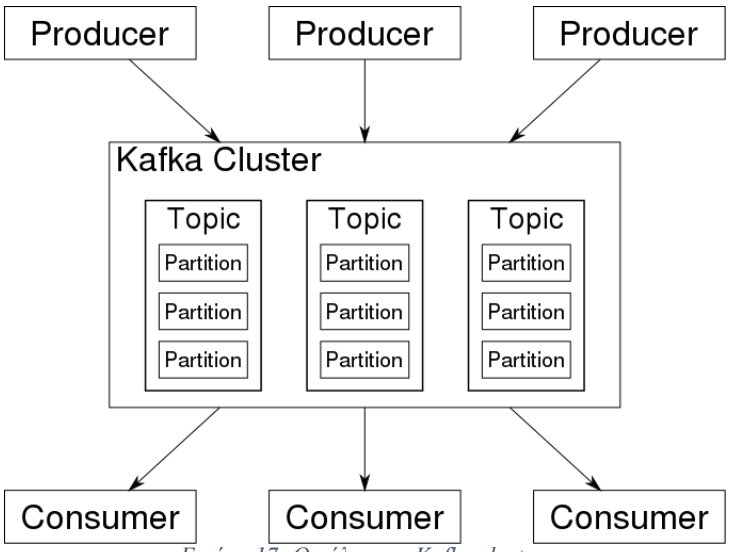
```
SELECT userId, COUNT(*)
FROM clicks
GROUP BY SESSION(clicktime, INTERVAL '30' MINUTE), userId
```

[5] Ένας αγωγός ETL αναφέρεται σε ένα σύνολο διαδικασιών εξαγωγής δεδομένων από μια πηγή εισόδου, μετατροπής των δεδομένων και φόρτωσης σε έναν προορισμό εξόδου, όπως μια βάση δεδομένων, ένα data mart ή μια αποθήκη δεδομένων για αναφορές, ανάλυση και συγχρονισμό δεδομένων. https://databricks.com/glossary/etl-pipeline

4.3 Apache Kafka

Το Apache Kafka, αν και αναπτύχθηκε αρχικά από τους Jay Kreps, Neha Narkhede και Jun Rao για το LinkedIn, είναι μια πλατφόρμα επεξεργασίας ροής ανοιχτού κώδικα που αναπτύχθηκε από το Apache Software Foundation γραμμένο σε Scala και σε Java. Αυτή η πλατφόρμα ροής επιτρέπει να δημοσιεύσουμε και να εγγραφούμε σε ροές εγγραφών (publish and subscribe), να αποθηκεύσουμε ροές εγγραφών με τρόπο ανεκτικό σε σφάλματα και να επεξεργαστούμε ροές εγγραφών κατά τη στιγμή που συμβαίνουν. Αυτό το καθιστά πολύτιμο για εταιρικές λειτουργίες ώστε να επεξεργάζονται ροές δεδομένων εκμεταλλευόμενες το γρήγορο, επεκτάσιμο και ανθεκτικό χαρακτήρα του. Το Kafka εκτελείται ως cluster σε έναν ή περισσότερους διακομιστές που μπορούν να εκτείνονται σε πολλά κέντρα δεδομένων. (Spark, Apache, 2019)

Το Kafka cluster αποθηκεύει ροές αρχείων σε κατηγορίες που ονομάζονται topics (βλ. Εικόνα 17). Κάθε εγγραφή αποτελείται από ένα κλειδί, μια τιμή και μια χρονική σήμανση.



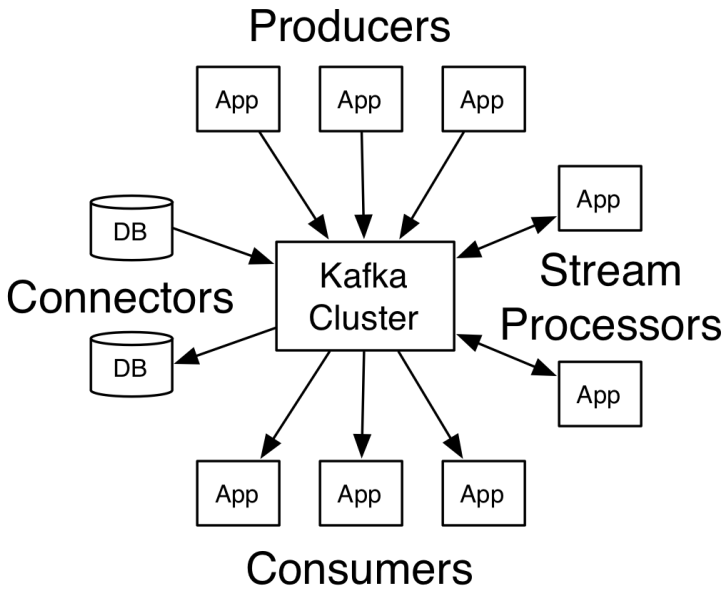
Εικόνα 17: Ο ρόλος του Kafka cluster,
πηγή : Δεδομένα ροής με τη χρήση Kafka , by Ted Struik | Jul 11, 2018 | Consulting, Data & Analytics ,
https://www.itium.nl/data-en-analytics/datastromen-de-baas-met-kafka/

Το Kafka διαθέτει πέντε σημαντικά APIs (Group, 2019):

- Το Producer API επιτρέπει σε μια εφαρμογή να δημοσιεύει μια ροή αρχείων σε ένα ή περισσότερα topics του Kafka.

- Το Consumer API επιτρέπει σε μια εφαρμογή να εγγραφεί σε ένα ή περισσότερα topics και να επεξεργάζεται τη ροή των εγγραφών που παράγονται σε αυτά.
- Το Streams API επιτρέπει σε μια εφαρμογή να ενεργεί ως επεξεργαστής ροής, καταναλώνοντας μια ροή δεδομένων εισόδου από ένα ή περισσότερα topics και παράγοντας μια ροή εξόδου σε ένα ή περισσότερα topics εξόδου, μετατρέποντας αποτελεσματικά τις ροές εισόδου σε ροές εξόδου.
- Το Connector API επιτρέπει την κατασκευή και λειτουργία επαναχρησιμοποιήσιμων παραγωγών ή καταναλωτών που συνδέουν Kafka topics με υπάρχουσες εφαρμογές ή συστήματα δεδομένων. Για παράδειγμα, μια σύνδεση σε μια σχεσιακή βάση δεδομένων μπορεί να καταγράφει κάθε αλλαγή σε έναν πίνακα.
- Το Admin API επιτρέπει τη διαχείριση και τον έλεγχο topic, και άλλων αντικειμένων Kafka.

Στο Kafka η επικοινωνία μεταξύ clients και servers γίνεται με ένα απλό, υψηλής απόδοσης, γλωσσικό πρωτόκολλο, το TCP. Το πρωτόκολλο αυτό διατηρεί συμβατότητα με παλαιότερες εκδόσεις. Δίνεται η δυνατότητα παροχής Java client, αλλά υπάρχει διαθέσιμο και σε πολλές άλλες γλώσσες.

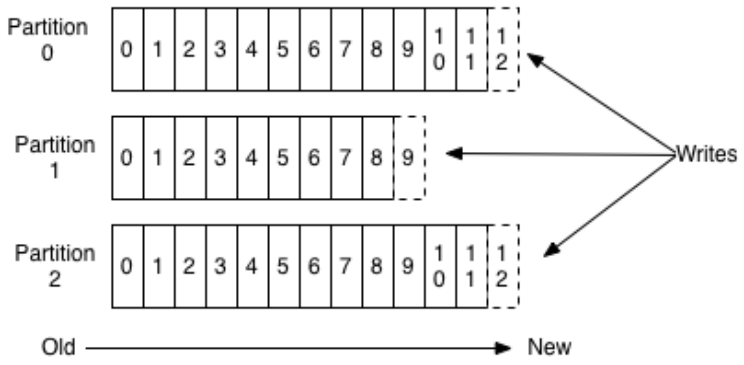


Εικόνα 18: Τα API του Apache Kafka

Ας ρίξουμε μια ματιά στο τι παρέχει το Kafka για τη σχέση ροής αρχείων και topics (Group, 2019).

Ένα topic είναι μια κατηγορία ή ένα όνομα μιας ροής δεδομένων στην οποία καταχωρούνται οι εγγραφές. Τα topics αποτελούνται πάντα από πολλούς συνδρομητές (subscribers). Δηλαδή, ένα topic μπορεί να έχει μηδέν, έναν ή πολλούς καταναλωτές (consumers) που παρακολουθούν τα δεδομένα που γράφονται σε αυτό.

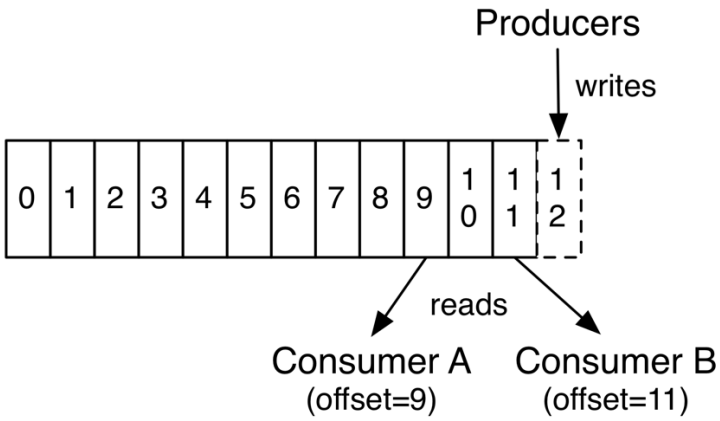
Για κάθε topic, το Kafka cluster διατηρεί ένα κατακερματισμένο αρχείο log που μοιάζει με αυτό:



Εικόνα 19: Ανατομία ενός Kafka topic
πηγή: kafka.apache.org

Κάθε partition (βλ. Εικόνα 19) είναι μια διατεταγμένη, αμετάβλητη ακολουθία εγγραφών που προσαρτάται συνεχώς σε ένα δομημένο αρχείο καταγραφής δεσμεύσεων. Οι εγγραφές στα partition καταχωρούνται με έναν διαδοχικό αριθμό ταυτότητας που ονομάζεται offset, το οποίο προσδιορίζει μοναδικά κάθε εγγραφή εντός του διαμερίσματος.

Το Kafka cluster διατηρεί συνεχώς όλα τα δημοσιευμένα αρχεία - είτε έχουν καταναλωθεί είτε όχι – έχοντας υπόψη μια περίοδο που μπορούν να διατηρηθούν τα αρχεία αυτά. Για παράδειγμα, εάν έχουμε ορίσει την περίοδο διατήρησης των δημοσιευμένων αρχείων σε δύο ημέρες, τότε για αυτές τις δύο ημέρες μετά τη δημοσίευση μιας εγγραφής, τα δεδομένα είναι διαθέσιμα για κατανάλωση, μετά το πέρας όμως των δύο ημερών τα δεδομένα θα απορριφθούν για να ελευθερωθεί χώρος (Group, 2019).



Εικόνα 20: Παραγωγοί και καταναλωτές στο Apache Kafka
πηγή: kafka.apache.org (Group, 2019)

Κάθε διαμέρισμα αποτελεί μια διατεταγμένη, αμετάβλητη ακολουθία εγγραφών που προσαρτάται συνεχώς σε ένα δομημένο log. Οι εγγραφές στα διαμερίσματα καταχωρούνται με έναν διαδοχικό id αριθμό που ονομάζεται offset και προσδιορίζει μοναδικά κάθε εγγραφή εντός του διαμερίσματος.

Κατανομή (Distribution) (Group, 2019)

Τα διαμερίσματα του log κατανέμονται στους διακομιστές του Kafka cluster με κάθε διακομιστή να χειρίζεται δεδομένα και αιτήματα για ένα κομμάτι του partition. Κάθε partition υπάρχει και ως αντίγραφο σε διάφορους διακομιστές για την εύκολη διαχείριση ανοχής σφαλμάτων.

Γεω-αναπαραγωγή (Geo-Replication) (Group, 2019)

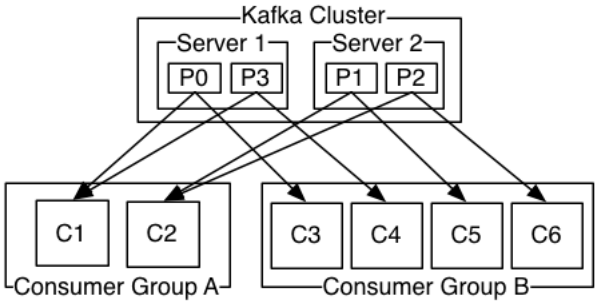
Το Kafka MirrorMaker παρέχει υποστήριξη γεωγραφικής αναπαραγωγής για τις συστάδες μας. Με το MirrorMaker, τα μηνύματα αναπαράγονται σε πολλά κέντρα δεδομένων ή περιοχές cloud. Μπορούμε να το χρησιμοποιήσουμε σε ενεργά ή παθητικά σενάρια για δημιουργία αντιγράφων ασφαλείας και ανάκτηση δεδομένων καθώς και σε σενάρια για εκτέλεση σε τοποθεσίες πιο κοντά στους χρήστες ή να υποστηρίξουμε τις απαιτήσεις δεδομένων τοποθεσίας.

Παραγωγοί (Producers) (Group, 2019)

Οι παραγωγοί δημοσιεύουν δεδομένα στα topic της επιλογής τους. Ο παραγωγός είναι υπεύθυνος για την επιλογή της εγγραφής που θα καταχωρήσει καθώς και σε ποιο διαμέρισμα μέσα στο topic. Αυτό γίνεται με τέτοιο τρόπο για να εξισορροπείται το φορτίο.

Καταναλωτές (Consumers) (Group, 2019)

Οι καταναλωτές διαβάζουν από οποιοδήποτε μεμονωμένο partition, επιτρέποντάς μας να κλιμακώσουμε την απόδοση της κατανάλωσης μηνυμάτων με παρόμοιο τρόπο με την παραγωγή μηνυμάτων. Τα στιγμιότυπα των καταναλωτών μπορούν να υπάρχουν σε ξεχωριστές διαδικασίες ή σε ξεχωριστές μηχανές. Εάν όλα τα στιγμιότυπα των καταναλωτών έχουν την ίδια ομάδα καταναλωτών, τότε οι εγγραφές θα είναι αποτελεσματικά ισορροπημένες με τα στιγμιότυπα των καταναλωτών. Εάν βρίσκονται σε διαφορετικές ομάδες καταναλωτών, τότε κάθε εγγραφή θα μεταδοθεί σε όλους τους καταναλωτές. (βλ. Εικόνα 21)



Εικόνα 21: Οι καταναλωτές (consumers) στο Apache Kafka
πηγή: kafka.apache.org (Group, 2019)

Πολλαπλή μίσθωση (Multi-tenancy) (Group, 2019)

Μπορούμε να αναπτύξουμε με τέτοιο τρόπο το Kafka ως λύση πολλών ενοικιαστών. Το Multi-tenancy ενεργοποιείται ρυθμίζοντας ποια topics μπορούν να παράγουν ή να καταναλώνουν δεδομένα. Υπάρχει επίσης υποστήριξη λειτουργιών για ποσοστώσεις. Οι διαχειριστές μπορούν να ορίσουν και να επιβάλουν ποσοστώσεις σε αιτήματα για τον έλεγχο των πόρων του μεσίτη που χρησιμοποιούνται από τους πελάτες. Για περισσότερες πληροφορίες, ανατρέξτε στο εγχειρίδιο ασφαλείας στο εξής link https://kafka.apache.org/documentation/#security

Εγγυήσεις που παρέχει το Apache Kafka (Group, 2019)

Σε υψηλό επίπεδο η Kafka παρέχει τις ακόλουθες εγγυήσεις:

- Τα μηνύματα που αποστέλλονται από έναν παραγωγό σε ένα συγκεκριμένο topic partition όπου θα προσαρτώνται με τη σειρά που αποστέλλονται. Δηλαδή, εάν μια εγγραφή M1 αποστέλλεται από τον ίδιο παραγωγό ως εγγραφή M2, και η M1 αποστέλλεται πρώτη, τότε το M1 θα έχει χαμηλότερο offset από το M2 και θα εμφανίζεται νωρίτερα στο αρχείο καταγραφής.
- Ένας καταναλωτής βλέπει εγγραφές με τη σειρά που αποθηκεύονται στο αρχείο καταγραφής. (Neha Narkhede, 2017)

Στο Kafka, ένας επεξεργαστής ροής μπορεί να είναι οτιδήποτε λαμβάνει συνεχείς ροές δεδομένων από εισερχόμενα topics, οτιδήποτε εκτελεί κάποια επεξεργασία σε αυτήν την είσοδο δεδομένων και παράγει συνεχείς ροές δεδομένων σε topics εξόδου.

Για παράδειγμα, μια εφαρμογή λιανικής πώλησης μπορεί να λαμβάνει εισερχόμενες ροές πωλήσεων και αποστολές προϊόντων και να εξάγει μια ροή δεδομένων η οποία επανυπολογίζει και προσαρμόζει τις τιμές με βάσει τα δεδομένα που δέχθηκε.

Επίσης, υπάρχει η δυνατότητα απλής επεξεργασίας χρησιμοποιώντας απευθείας τα API παραγωγών και καταναλωτών. Ωστόσο, για πιο σύνθετες λειτουργίες, το Kafka παρέχει ένα πλήρως ενσωματωμένο API ροών. Αυτό επιτρέπει την κατασκευή εφαρμογών που εκτελούν επεξεργασίες υπολογισμού συγκεντρώσεων εκτός ροών ή υπολογισμούς σε ροές που ενώνονται μαζί.

Οι παραπάνω λειτουργίες βοηθάνε στην επίλυση τέτοιων δύσκολων προβλημάτων: όπως τον χειρισμό δεδομένων εκτός παραγγελίας, την επαναληπτική επεξεργασία δεδομένων εισόδου καθώς αλλάζει ο κώδικας κ.λπ.

Όλα τα API ροής δεδομένων βασίζονται στα βασικά χαρακτηριστικά που παρέχει το Kafka: δηλαδή στη χρήση API παραγωγού και καταναλωτή για εισερχόμενα δεδομένα, και για αποθήκευση δεδομένων χρησιμοποιούν τον ίδιο μηχανισμό ομαδοποίησης για ανοχή σφαλμάτων μεταξύ των στιγμιότυπων επεξεργασίας της ροής δεδομένων.

Το Apache Kafka και οι μεγάλες εταιρείες

Το Kafka χρησιμοποιείται σε μεγάλο βαθμό από τον τομέα διαχείρισης δεδομένων μεγάλου όγκου (big-data) ως αξιόπιστος τρόπος για την ταχύτατη απορρόφηση και τη γρήγορη μεταφορά μεγάλων δεδομένων.

Σύμφωνα με το Stackshare[6] υπάρχουν 741 εταιρείες που χρησιμοποιούν το Kafka. Μεταξύ αυτών Uber, Netflix, Activision, Spotify, Slack, Pinterest, Coursera και φυσικά το LinkedIn.

Ας δούμε μερικούς σημαντικούς λόγους όπου μεγάλες επιχειρήσεις και οργανισμοί, όπως το Airbnb και το Netflix , χρησιμοποιούν το Apache Kafka (Wachal, 2020):

1. **Παρακολούθηση δραστηριότητας ιστότοπου**

   Η δραστηριότητα του ιστότοπου (προβολές σελίδων, αναζητήσεις ή άλλες ενέργειες που μπορούν να κάνουν οι χρήστες) δημοσιεύεται σε κεντρικά topics και διατίθεται για επεξεργασία σε πραγματικό χρόνο, διατίθενται πίνακες ελέγχου και αναλυτικά στοιχεία εκτός σύνδεσης σε αποθήκες δεδομένων όπως το BigQuery της Google.

2. **Μετρήσεις**

   Το Kafka χρησιμοποιείται συχνά για αγωγούς δεδομένων παρακολούθησης λειτουργίας και επιτρέπει την ειδοποίηση και την υποβολή εκθέσεων σχετικά με λειτουργικές μετρήσεις. Συγκεντρώνει στατιστικά στοιχεία από κατανεμημένες εφαρμογές και παράγει κεντρικές τροφοδοσίες επιχειρησιακών δεδομένων.

3. **Συγκέντρωση αρχείων καταγραφής**

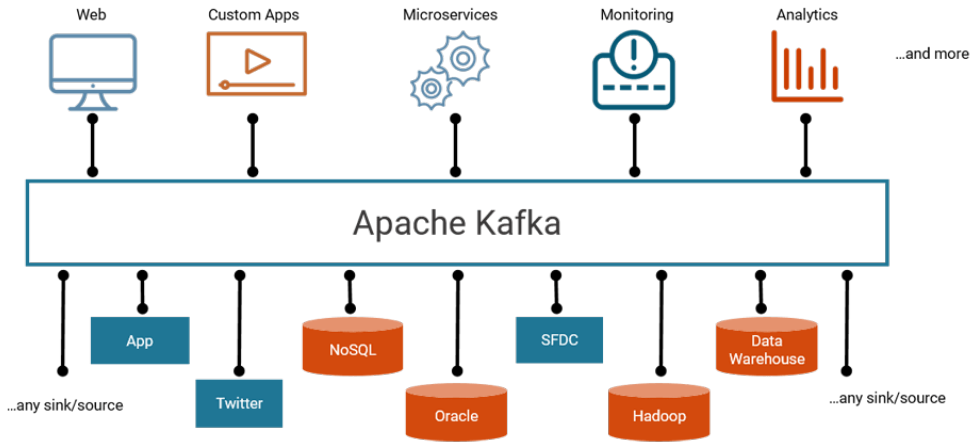
   Το Kafka μπορεί να χρησιμοποιηθεί σε έναν οργανισμό για τη συλλογή αρχείων καταγραφής από πολλές υπηρεσίες και μπορούν να είναι διαθέσιμα σε πολλούς χρήστες και σε απλή μορφή. Παρέχει επεξεργασία χαμηλότερης

[6] Το StackShare είναι μια κοινότητα που βοηθά προγραμματιστές και μηχανικούς να ανακαλύπτουν και να συγκρίνουν εργαλεία λογισμικού.

καθυστέρησης και ευκολότερη υποστήριξη για πολλές πηγές δεδομένων και κατανεμημένη κατανάλωση δεδομένων.

4. **Επεξεργασία ροής**

Ένα οποιοδήποτε framework, όπως το Spark Streaming, διαβάζει δεδομένα από ένα topic, το επεξεργάζεται και γράφει τα επεξεργασμένα δεδομένα σε ένα νέο topic όπου είναι διαθέσιμο για χρήστες και εφαρμογές. Το γεγονός ότι το Kafka παρέχει μεγάλη ανθεκτικότητα και διάρκεια το καθιστά πολύ χρήσιμο στην επεξεργασία δεδομένων ροής. (Group, 2019)



Εικόνα 22: Εφαρμογές Apache Kafka
πηγή : Δεδομένα ροής με τη χρήση Kafka , by Ted Struik | Jul 11, 2018 | Consulting, Data & Analytics , https://www.itium.nl/data-en-analytics/datastromen-de-baas-met-kafka/

Πλεονεκτήματα:

- Πολλαπλοί παραγωγοί (producers). Το Kafka μπορεί να χειριστεί πολλούς παραγωγούς, είτε οι πελάτες χρησιμοποιούν πολλά θέματα είτε το ίδιο θέμα. Αυτό καθιστά το σύστημα ιδανικό για τη συγκέντρωση δεδομένων από πολλά συστήματα frontend. Για παράδειγμα, ένας ιστότοπος που φιλοξενεί περιεχόμενο σε χρήστες μέσω ενός αριθμού μικροϋπηρεσιών μπορεί να έχει ένα μόνο topic για προβολές σελίδας στο οποίο μπορούν να δημιουργούν εγγραφές όλες οι υπηρεσίες χρησιμοποιώντας μια κοινή μορφή. Οι εφαρμογές καταναλωτών μπορούν στη συνέχεια να λάβουν μία ροή προβολών σελίδας για όλες τις εφαρμογές στον ιστότοπο χωρίς να χρειάζεται να συντονίσουν την κατανάλωση από πολλά topics, μία για κάθε εφαρμογή. (Neha Narkhede, 2017, pp. 39-44)

- Πολλαπλοί καταναλωτές (consumers). Εκτός από πολλούς παραγωγούς, το Kafka έχει σχεδιαστεί για πολλούς καταναλωτές οι οποίοι έχουν τη δυνατότητα να διαβάζουν οποιαδήποτε ροή μηνυμάτων χωρίς να παρεμβαίνουν μεταξύ τους. Αυτό έρχεται σε αντίθεση με πολλά συστήματα που χρησιμοποιούν στοίβες δεδομένων όπου όταν ένα μήνυμα καταναλώνεται από έναν πελάτη, τότε δεν είναι διαθέσιμο σε κανέναν άλλο. Πολλοί καταναλωτές του Kafka μπορούν να επιλέξουν να λειτουργήσουν ως μέρος μιας ομάδας και να μοιραστούν μια ροή, διασφαλίζοντας ότι ολόκληρη η ομάδα επεξεργάζεται ένα μήνυμα μόνο μία φορά. (Neha Narkhede, 2017)
- Διατήρηση βάσει δίσκου. Το Kafka όχι μόνο μπορεί να χειριστεί πολλούς καταναλωτές, αλλά η διαρκής διατήρηση μηνυμάτων σημαίνει ότι οι καταναλωτές δεν χρειάζεται πάντα να εργάζονται σε πραγματικό χρόνο. Τα μηνύματα δεσμεύονται σε δίσκο και αποθηκεύονται με ρυθμιζόμενους κανόνες διατήρησης αυτού του περιεχομένου. Αυτές οι επιλογές μπορούν να εφαρμοστούν ανά topic, επιτρέποντας σε διαφορετικές ροές μηνυμάτων να έχουν διαφορετικά ποσά διατήρησης δεδομένων ανάλογα με τις ανάγκες των καταναλωτών. Διαρκή διατήρηση δεδομένων σημαίνει ότι εάν ένας καταναλωτής μείνει πίσω, είτε λόγω αργής επεξεργασίας είτε λόγω υψηλής κυκλοφορίας, δεν υπάρχει κίνδυνος απώλειας δεδομένων. Σημαίνει επίσης, ότι η συντήρηση μπορεί να πραγματοποιηθεί σε καταναλωτές, λαμβάνοντας εφαρμογές εκτός σύνδεσης για σύντομο χρονικό διάστημα, χωρίς να ανησυχείτε για μηνύματα δημιουργίας αντιγράφων ασφαλείας στον παραγωγό ή απώλεια αυτών των δεδομένων. Σε περίπτωση που οι καταναλωτές σταματήσουν τότε τα μηνύματα θα διατηρηθούν στον Apache Kafka. Αυτό επιτρέπει στους καταναλωτές να επανεκκινήσουν και να παραλάβουν την επεξεργασία μηνυμάτων από εκεί που σταμάτησαν χωρίς απώλεια δεδομένων. (Neha Narkhede, 2017)
- Υψηλή απόδοση. Χωρίς να διαθέτει τόσο μεγάλο υλισμικό , το Apache Kafka είναι σε θέση να χειρίζεται δεδομένα υψηλής ταχύτητας και μεγάλου όγκου. Επίσης, μπορεί να υποστηρίξει τη διακίνηση χιλιάδων μηνυμάτων ανά δευτερόλεπτο.
- Χαμηλή καθυστέρηση. Είναι ικανό να χειριστεί χιλιάδες μηνύματα με πολύ χαμηλό λανθάνοντα χρόνο του εύρους των χιλιοστών του δευτερολέπτου, που απαιτείται από την πλειοψηφία των νέων περιπτώσεων χρήσης.

- Ανοχή σε σφάλματα. Ένα από τα μεγαλύτερα πλεονεκτήματα του είναι η ανοχή σφαλμάτων. Υπάρχει μια εγγενής ικανότητα στο Kafka, να είναι ανθεκτικό σε αστοχίες κόμβου / μηχανήματος εντός cluster.
- Διάρκεια. Η διάρκεια αναφέρεται στην επιμονή των δεδομένων / μηνυμάτων στο δίσκο. Επίσης, η αναπαραγωγή μηνυμάτων είναι ένας από τους λόγους ύπαρξης της διάρκειας, έχοντας ως αποτέλεσμα τα μηνύματα να μην χάνονται ποτέ.

Όσον αφορά την αρχιτεκτονική του Apache Kafka τότε τα πλεονεκτήματα γίνονται ακόμα περισσότερα.

- Επεκτασιμότητα. Χωρίς να υπάρξει κάποια καθυστέρηση, σε κατάσταση εκτός λειτουργίας και με την προσθήκη επιπλέον κόμβων, το Kafka μπορεί να κλιμακωθεί. Επιπλέον, μέσα στο Kafka cluster, ο χειρισμός μηνυμάτων είναι απόλυτα διαφανής. Η κατανεμημένη αρχιτεκτονική του Kafka το καθιστά επεκτάσιμο χρησιμοποιώντας δυνατότητες όπως η αναπαραγωγή (replication) και ο διαμερισμός (partitioning).
- Φιλικό προς τον χρήστη. Το Kafka μπορεί να συμπεριφέρεται ή να ενεργεί διαφορετικά ανάλογα με τον χρήστη / καταναλωτή , ακόμα και σε διάφορες γλώσσες προγραμματισμού.
- Το Kafka θα μπορούσε επίσης να χρησιμοποιηθεί για περιπτώσεις χρήσης με δέσμες δεδομένων (batch) και μπορεί επίσης να κάνει τη δουλειά ενός παραδοσιακού εργαλείου ETL [7], λόγω της ικανότητάς του να διατηρεί μηνύματα.

[7] ETL (extract, transform, load) : αποτελεί τη γενική διαδικασία αντιγραφής δεδομένων από μία ή περισσότερες πηγές σε ένα σύστημα προορισμού που αντιπροσωπεύει τα δεδομένα διαφορετικά από τις πηγές ή σε διαφορετικό πλαίσιο από τις πηγές. Η διαδικασία ETL έγινε μια δημοφιλής ιδέα τη δεκαετία του 1970 και χρησιμοποιείται συχνά σε περιπτώσεις αποθήκευσης δεδομένων.

Μειονεκτήματα

- Δε διαθέτει ένα πλήρες πακέτο εργαλείων διαχείρισης και παρακολούθησης, γεγονός που δημιουργεί ανασφάλεια στις επιχειρήσεις που θέλουν να το επιλέξουν.
- Κατά τη μετάδοση ενός μηνύματος ο Apache Kafka χρησιμοποιεί ορισμένες κλήσεις συστήματος για να γίνει η παράδοση του μηνύματος στο καταναλωτή. Σε περίπτωση που το μήνυμα χρειάζεται κάποια τροποποίηση τότε η απόδοση του συστήματος πέφτει και είναι αποδοτικό μόνο σε περιπτώσεις που το μήνυμα παραδίδεται αμετάβλητο.
- Σε περίπτωση αναζήτησης ενός topic θα πρέπει να γίνει πλήρη ταυτοποίηση του ονόματος του topic διαφορετικά δεν μπορεί να εντοπιστεί.

4.4 Apache Samza

Το Apache Samza αποτελεί framework ανοιχτού κώδικα για κατανεμημένη επεξεργασία ροών συμβάντων μεγάλου όγκου. Ο πρωταρχικός σχεδιαστικός του στόχος είναι να υποστηρίξει υψηλή απόδοση για ένα ευρύ φάσμα μοτίβων επεξεργασίας, παρέχοντας παράλληλα λειτουργική ευρωστία σε μαζική κλίμακα που απαιτείται από εταιρείες Διαδικτύου. Το Samza βασίζεται σε τρεις κύριες έννοιες που είναι υπεύθυνες για την υψηλή επεκτασιμότητα και τη λειτουργική ευρωστία του (M.Kleppmann, 2018):

- Επεξεργασία κατατμημένων αρχείων καταγραφής. Οι είσοδοι σε μια εργασία Samza χωρίζονται σε υποδιαιρούμενα υποσύνολα και κάθε διαμέρισμα εισόδου κατανέμεται σε μία μόνο εργασία επεξεργασίας. Ο αριθμός των κατατμήσεων στην είσοδο καθορίζει το βαθμό παραλληλισμού της εργασίας.
- Τοπική κατάσταση ανεκτική σε σφάλματα. Στο Samza η κατάσταση μιας εργασίας καταγράφεται στον τοπικό δίσκο του κόμβου επεξεργασίας. Διατηρεί επίσης, μια μνήμη cache για συχνά χρησιμοποιούμενα αντικείμενα. Σε περίπτωση αστοχίας τοπικού δίσκου, το Samza διατηρεί ένα αναπαραγόμενο αρχείο καταγραφής της κατάστασης.
- Προγραμματισμός εργασιών βάσει cluster. Το Samza δεν διαθέτει ενσωματωμένο μηχανισμό διαχείρισης πόρων και ανάπτυξης. Χρησιμοποιεί το υπάρχον λογισμικό διαχείρισης για το σκοπό αυτό. Υποστηρίζει δυο τρόπους κατανεμημένης λειτουργίας: το Hadoop YARN και την αυτόνομη λειτουργία του.

Η επεξεργασία ροής παίζει όλο και πιο σημαντικό μέρος των αναγκών διαχείρισης δεδομένων πολλών οργανισμών. Οι ροές συμβάντων μπορούν να αντιπροσωπεύουν πολλά είδη δεδομένων, για παράδειγμα, τη δραστηριότητα των χρηστών σε έναν ιστότοπο, την κυκλοφορία αγαθών ή οχημάτων ή τις εγγραφές αρχείων σε μια βάση δεδομένων.

Οι εργασίες επεξεργασίας ροής είναι μακροχρόνιες διεργασίες που καταναλώνουν συνεχώς μία ή περισσότερες ροές συμβάντων, επικαλούνται κάποια λογική εφαρμογών σε κάθε συμβάν, παράγουν ροές εξόδου και πιθανώς εγγράφουν δεδομένα εξόδου σε βάσεις δεδομένων για πιθανά ερωτήματα. Σε αντίθεση, μια διαδικασία δέσμης ή ένα ερώτημα βάσης δεδομένων διαβάζει συνήθως την κατάσταση ενός συνόλου δεδομένων σε μια χρονική στιγμή και στη συνέχεια ολοκληρώνεται, επίσης ένας επεξεργαστής ροής δεν τελειώνει ποτέ, περιμένει συνεχώς την άφιξη νέων συμβάντων και τερματίζεται μόνο από κάποιο διαχειριστή.

Πολλές διεργασίες μπορούν φυσικά να εκφραστούν ως εργασίες επεξεργασίας ροής, για παράδειγμα:

- Συγκεντρώνοντας εμφανίσεις συμβάντων, π.χ., μετρώντας πόσες φορές έχει προβληθεί ένα συγκεκριμένο αντικείμενο.
- Υπολογισμός του ποσοστού ύπαρξης ορισμένων συμβάντων, π.χ. για διαγνωστικά συστήματος, αναφορές και πρόληψη κατάχρησης.
- Εμπλουτισμός συμβάντων με πληροφορίες από μια βάση δεδομένων, π.χ. επέκταση συμβάντων κλικ χρήστη με πληροφορίες σχετικά με τον χρήστη που πραγματοποίησε την ενέργεια.
- Συμμετοχή σε συναφή συμβάντα, π.χ. συμμετοχή σε μια εκδήλωση που περιγράφει ένα μήνυμα ηλεκτρονικού ταχυδρομείου που εστάλη με τυχόν συμβάντα που περιγράφουν τους χρήστες που κάνουν κλικ σε συνδέσμους που περιέχει το μήνυμα ηλεκτρονικού ταχυδρομείου.
- Ενημέρωση κρυφής μνήμης, υλοποιημένες προβολές και ευρετήρια αναζήτησης, π.χ. διατήρηση εξωτερικού ευρετηρίου αναζήτησης πλήρους κειμένου μέσω κειμένου σε βάση δεδομένων.

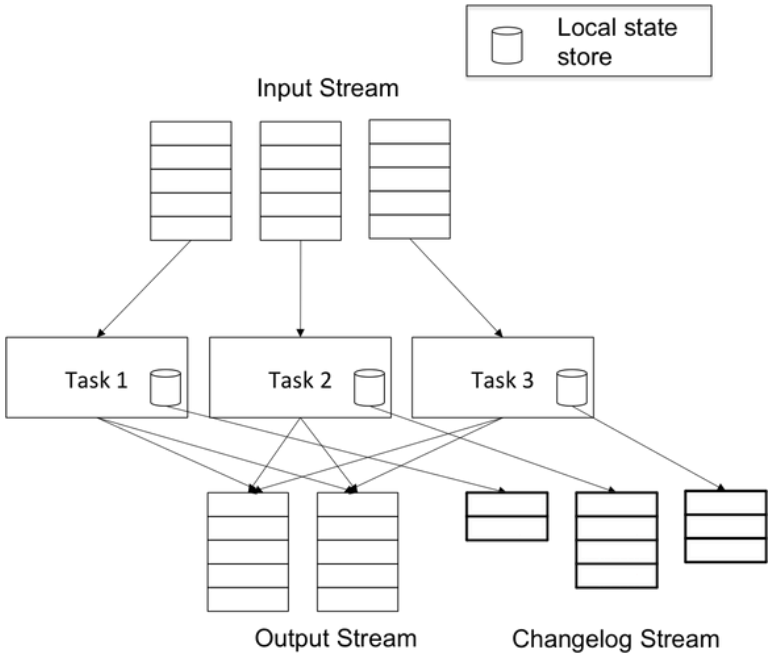
- Χρήση συστημάτων μηχανικής εκμάθησης για την ταξινόμηση συμβάντων, π.χ. για φιλτράρισμα ανεπιθύμητων μηνυμάτων.

Το πλαίσιο επεξεργασίας ροής ανοιχτού κώδικα Apache Samza, μπορεί να χρησιμοποιηθεί για οποιαδήποτε από τις παραπάνω εφαρμογές (Kleppmann M, 2015). Αρχικά αναπτύχθηκε στο LinkedIn, στη συνέχεια δωρίστηκε στο Apache Software Foundation το 2013 και έγινε έργο υψηλού επιπέδου Apache το 2015. Το Apache Samza στις μέρες μας χρησιμοποιείται στον τομέα της παραγωγής από πολλές εταιρείες Διαδικτύου, συμπεριλαμβανομένου του LinkedIn, Netflix, Uber και TripAdvisor.

Το Samza έχει σχεδιαστεί για σενάρια χρήσης που απαιτούν πολύ υψηλή απόδοση όπου σε ορισμένες διαδικασίες παραγωγής, επεξεργάζεται εκατομμύρια μηνύματα ανά δευτερόλεπτο ή τρισεκατομμύρια συμβάντα την ημέρα (Tao Feng, LinkedIn, 2015), (Kartik Paramasivam, LinkedIn, 2016), (Shadi A. Noghabi, 2017). Κατά συνέπεια, ο σχεδιασμός της Samza δίνει προτεραιότητα στην επεκτασιμότητα και τη λειτουργική ευρωστία πάνω από τις περισσότερες άλλες ανησυχίες.

Ο πυρήνας του Samza αποτελείται από αρκετές αφαιρέσεις αρκετά χαμηλού επιπέδου, πάνω από τις οποίες έχουν κατασκευαστεί χειριστές υψηλού επιπέδου (Milinda Pathirage, 2016). Ωστόσο, οι βασικές αφαιρέσεις έχουν σχεδιαστεί προσεκτικά για λειτουργική αντοχή και η επεκτασιμότητα του Samza οφείλεται άμεσα στην επιλογή αυτών των θεμελιωδών αφαιρέσεων. Στη συνέχεια αυτής της ενότητας θα βρείτε περισσότερες λεπτομέρειες σχετικά με αυτές τις αποφάσεις σχεδιασμού και τις πρακτικές τους συνέπειες.

Η προσέγγιση του Apache Samza στη ροή δεδομένων είναι η επεξεργασία μηνυμάτων καθώς λαμβάνονται, ένα κάθε φορά. Το βασικό χαρακτηριστικό της ροής δεδομένων του Apache Samza δεν είναι η χρήση πλειάδων ή το Dstream, αλλά η χρήση μηνυμάτων. Οι ροές δεδομένων χωρίζονται σε διαμερίσματα – κατατμήσεις και κάθε διαμέρισμα αποτελεί μια ταξινομημένη ακολουθία μηνυμάτων μόνο για ανάγνωση με κάθε μήνυμα να έχει ένα μοναδικό αναγνωριστικό (offset). Το σύστημα επίσης υποστηρίζει τη δέσμη δεδομένων, δηλαδή την κατανάλωση πολλών μηνυμάτων από το ίδιο διαμέρισμα ροής διαδοχικά. Οι λειτουργικές μονάδες εκτέλεσης και ροής του Samza είναι και οι δύο με δυνατότητα σύνδεσης (pluggable), αν και ο Samza βασίζεται συνήθως στο YARN του Hadoop (ακόμα ένας διαπραγματευτής πόρων) και στον Apache Kafka.



Εικόνα 23: Απεικόνιση λειτουργίας Apache Samza
πηγή: https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node

Samza SQL

Το Samza SQL μας επιτρέπει να ορίσουμε τη λογική επεξεργασίας ροής δηλωτικά ως ερώτημα SQL. Αυτό μας δίνει τη δυνατότητα να δημιουργήσουμε αγωγούς ροής χωρίς κώδικα Java ή διαμόρφωση, εκτός εάν χρειάζεστε λειτουργίες που καθορίζονται από το χρήστη (UDF).

Παρακάτω θα δούμε ένα παράδειγμα Samza SQL (samza.apache.org, 2019) το οποίο μας επιτρέπει να δούμε τη χρήση του Samza SQL API.

- Το filter μας παρουσιάζει τη διαδικασία φιλτραρίσματος και την εισαγωγή μιας Samza SQL διεργασίας.

```
-- Filter Profile change-capture stream by 'Product Manager'
-- title and project basic profile data to a kafka topic.
           INSERT INTO kafka.ProductManagerProfiles
           SELECT memberId, firstName, lastName, company
           FROM kafka.ProfileChanges
           WHERE standardize(title) = 'Product Manager'

           Πηγή: (Samza Apache Org, 2019)
```

- Παρακάτω βλέπουμε την έκφραση CASE-WHEN σε συνδυασμό με το UDF.

```
-- For each profile in Kafka Profile change capture stream, identify whether the
-- profile is a quality profile or not and insert the result into QualityProfi
-- kafka topic. Please note the usage of GetSqlField UDF to extract the
company
-- name field from nested record.

           INSERT INTO kafka.QualityProfile
           SELECT id, status, case when (profilePicture <> null and industryName <>
           null and
           GetSqlField(positions, 'Position.companyName') <> null)
           then 1 else 0 end as quality
           FROM kafka.ProfileChanges

           Πηγή: (Samza Apache Org, 2019)
```

- Η εντολή join δείχνει πώς να εκτελέσουμε μια ένωση ροής-πίνακα. Λάβετε υπόψη ότι η λειτουργία join δεν είναι πλήρως ολοκληρωμένη και προσπαθούν να τη σταθεροποιήσουν.

```
-- NOTE: Join Operator is currently not fully stable,
            --       we are actively working on stabilizing it.
            -- Enrich PageViewEvent with member profile data
            INSERT INTO kafka.tracking.EnrichedPageVIewEvent
            SELECT *
            FROM Kafka.PageViewEvent as pv
              JOIN Kafka.ProfileChanges.`$table` as p
                ON pv.memberid = p.memberid

             Πηγή: (Samza Apache Org, 2019)
```

- Η εντολή group by δείχνει πως να ομαδοποιούμε δεδομένα.

```
-- NOTE: Groupby Operator is currently not fully stable,
    --       we are actively working on stabilizing it.
-- Emit Page view counts collected grouped by page key in the last
-- 5 minutes at 5 minute interval and send the result to a kafka topic.
-- Using GetSqlField UDF to extract page key from the requestHeader.
insert into kafka.groupbyTopic
  select GetSqlField(pv.requestHeader) as __key__,
GetPageKey(pv.requestHeader) as pageKey, count(*) as Views
  from kafka.`PageViewEvent` as pv
  group by GetSqlField(pv.requestHeader)
```

Πηγή: (Samza Apache Org, 2019)

Κάθε εργασία Samza SQL αποτελείται από μία ή περισσότερες δηλώσεις Samza SQL. Κάθε δήλωση αντιπροσωπεύει έναν αγωγό συνεχούς ροής.

Η υποστήριξη της SQL χρησιμοποιεί εσωτερικά το Apache Calcite[8], το οποίο παρέχει ανάλυση SQL και σχεδιασμό ερωτημάτων. Το ερώτημα μεταφράζεται αυτόματα στο API υψηλού επιπέδου της Samza και εκτελείται στη μηχανή εκτέλεσης του Samza.

Η αντιστοίχιση από το SQL στο API υψηλού επιπέδου του Samza είναι μια απλή ντετερμινιστική αντιστοίχιση.

Στον πίνακα 3 βλέπουμε τις υποστηριζόμενες λειτουργίες SQL.

Πίνακας 3: Υποστηριζόμενες λειτουργίες SQL

Λειτουργία	Εντολή
PROJECTION	SELECT / INSERT / UPSET
FILTERING	WHERE έκφραση
UDFs	Udf_name(ορίσματα)
JOIN	[LEFT/RIGHT] JOIN .. ON ..
AGGREGATION	COUNT (…) .. GROUP BY

[8] Το Apache Calcite είναι ένα πλαίσιο ανοιχτού κώδικα για την κατασκευή βάσεων δεδομένων και συστημάτων διαχείρισης δεδομένων. Περιλαμβάνει ένα πρόγραμμα ανάλυσης SQL, ένα API για τη δημιουργία εκφράσεων σε σχεσιακή άλγεβρα και μια μηχανή σχεδιασμού ερωτημάτων. Ως πλαίσιο, το Calcite δεν αποθηκεύει τα δικά του δεδομένα ή μεταδεδομένα, αλλά επιτρέπει την πρόσβαση σε εξωτερικά δεδομένα και μεταδεδομένα μέσω προσθηκών.

Η γραμματική του Samza SQL είναι ένα υποσύνολο δυνατοτήτων που υποστηρίζονται από το πρόγραμμα ανάλυσης SQL του Calcite. Από το παρακάτω παράδειγμα παρατηρούμε πως χρησιμοποιείται έμπρακτα.

```
statement:
  |   insert
  |   query

query:
  values
  | {
      select
    }

insert:
      ( INSERT | UPSERT ) INTO tablePrimary
      [ '(' column [, column ]* ')' ]
      query

select:
  SELECT
  { * | projectItem [, projectItem ]* }
  FROM tableExpression
  [ WHERE booleanExpression ]
  [ GROUP BY { groupItem [, groupItem ]* } ]

projectItem:
  expression [ [ AS ] columnAlias ]
  | tableAlias . *

tableExpression:
  tableReference [, tableReference ]*
  | tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ]
JOIN tableExpression [ joinCondition ]

joinCondition:
  ON booleanExpression
  | USING '(' column [, column ]* ')'

tableReference:
  tablePrimary
  [ [ AS ] alias [ '(' columnAlias [, columnAlias ]* ')' ]
]

tablePrimary:
  [ TABLE ] [ [ catalogName . ] schemaName . ] tableName

values:
  VALUES expression [, expression ]*
```

Εκτός από τους υπάρχοντες λογικούς τελεστές της SQL, το Samza SQL επιτρέπει στο χρήστη να επεκτείνει τη λειτουργικότητά της εκτελώντας κώδικα του χρήστη μέσω καθορισμένων λειτουργιών από το χρήστη (UDFs) ως μέρος του αγωγού επεξεργασίας ροής που αντιστοιχεί στην SQL.

Δεδομένου ότι η μέθοδος εκτέλεσης του UDF παίρνει μια σειρά από αντικείμενα γενικού τύπου ως παράμετρο, το UDF του Samza SQL είναι αρκετά ευέλικτο ώστε να υποστηρίζει πολυμορφικές λειτουργίες UDF με διάφορα σύνολα ορισμάτων, εφόσον τα υποστηρίζουν οι υλοποιήσεις του UDF.

Για παράδειγμα, στην παρακάτω δήλωση SQL, το UDF θα περάσει μια συστοιχία δυο αντικειμένων με το πρώτο στοιχείο να περιέχει το id του τύπου "LONG" και το δεύτερο το όνομα του στοιχείου τύπου "String". Ο τύπος των αντικειμένων που περνούν εξαρτάται από τον τύπο αυτών των πεδίων σε μορφή μηνύματος Samza SQL.

```
select myudf(id, name) from identity.profile
```

Η Samza SQL υποστηρίζει απλά ερωτήματα, συμπεριλαμβανομένων επιλογών και προβολών.

Επεξεργασία κατατμημένων αρχείων καταγραφής σε συνεργασία με το Apache Kafka

Μια εργασία στο Apache Samza αποτελείται από ένα σύνολο στιγμιότυπων Java Virtual Machine (JVM), που ονομάζονται διεργασίες, οι οποίες επεξεργάζονται κάθε υποσύνολο των δεδομένων εισόδου. Ο κώδικας που εκτελείται σε κάθε JVM περιλαμβάνει το framework Samza και τον κώδικα του χρήστη που εφαρμόζει την απαιτούμενη λειτουργικότητα για μια συγκεκριμένη εφαρμογή. Το κύριο API για τον κώδικα του χρήστη είναι η διεπαφή Java StreamTask, η οποία ορίζει μια μέθοδο με όνομα process(). Στην εικόνα 24 βλέπουμε δύο παραδείγματα κλάσεων χρήστη που εφαρμόζουν τη διεπαφή StreamTask.

```
class SplitWords implements StreamTask {

  static final SystemStream WORD_STREAM =
    new SystemStream("kafka", "words");

  public void process(
        IncomingMessageEnvelope in,
        MessageCollector out,
        TaskCoordinator _) {

    String str = (String) in.getMessage();

    for (String word : str.split(" ")) {
      out.send(
        new OutgoingMessageEnvelope(
          WORD_STREAM, word, 1));
    }
  }
}
```

```
class CountWords implements StreamTask,
                            InitableTask {

  private KeyValueStore<String, Integer> store;

  public void init(Config config,
                   TaskContext context) {
    store = (KeyValueStore<String, Integer>)
      context.getStore("word-counts");
  }

  public void process(
        IncomingMessageEnvelope in,
        MessageCollector out,
        TaskCoordinator _) {

    String word = (String) in.getKey();
    Integer inc = (Integer) in.getMessage();

    Integer count = store.get(word);
    if (count == null) count = 0;
    store.put(word, count + inc);
  }
}
```

Εικόνα 24: Κλάσεις χρηστών για την υλοποίηση μετρητή συχνότητας λέξεων με τη χρήση του StreamTask API του Apache Samza.
πηγή: Kleppmann and Kreps 2015, © 2015 IEEE, (Kreps, 2015)

Μόλις μια εργασία Samza αναπτυχθεί και αρχικοποιηθεί, το framework καλεί τη μέθοδο process() μία φορά για κάθε μήνυμα σε οποιαδήποτε από τις ροές εισόδου. Η εκτέλεση αυτής της μεθόδου μπορεί να έχει διάφορα αποτελέσματα, όπως τη δημιουργία ερωτήματος ή την ενημέρωση τοπικής κατάστασης και την αποστολή μηνυμάτων σε ροές εξόδου. Αυτό το μοντέλο υπολογισμού είναι ανάλογο με μια εργασία mapping στο γνωστό μοντέλο προγραμματισμού MapReduce (Jeffrey Dean and Sanjay Ghemawat, Google Inc., n.d.), με τη διαφορά ότι η εισαγωγή μιας εργασίας Samza είναι συνήθως ατέρμονη.

Ομοίως με το MapReduce, κάθε εργασία Samza είναι μια διαδικασία με ένα νήμα που επαναλαμβάνει μια ακολουθία εγγραφών εισόδου. Οι είσοδοι σε μια εργασία Samza χωρίζονται σε διαχωριστικά υποσύνολα και κάθε διαμέρισμα εισόδου αντιστοιχεί σε μία μόνο εργασία επεξεργασίας. Περισσότερα από ένα διαμερίσματα μπορούν να αντιστοιχιστούν στην ίδια εργασία επεξεργασίας, οπότε η επεξεργασία αυτών των διαμερισμάτων παρεμβάλλεται στο νήμα εργασιών. Ωστόσο, ο αριθμός των κατατμήσεων στην είσοδο καθορίζει τον μέγιστο βαθμό παραλληλισμού της εργασίας.

Το log interface υποθέτει ότι κάθε διαμέρισμα της εισόδου είναι μια εντελώς ταξινομημένη ακολουθία εγγραφών και ότι κάθε εγγραφή σχετίζεται με έναν μονοτονικά αυξανόμενο αριθμό ακολουθίας ή αναγνωριστικό (γνωστό ως offset). Δεδομένου ότι οι εγγραφές σε κάθε διαμέρισμα διαβάζονται διαδοχικά, μια εργασία μπορεί να

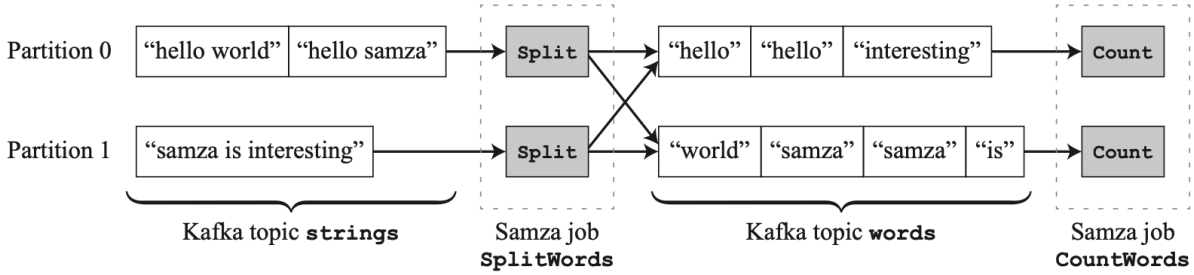
παρακολουθεί την πρόοδό της γράφοντας περιοδικά το offset της τελευταίας read εγγραφής σε διαρκή αποθήκευση. Εάν επανεκκινηθεί μια εργασία επεξεργασίας ροής, συνεχίζει να καταναλώνει την είσοδο από το τελευταίο offset.

Συνήθως, το Samza χρησιμοποιείται σε συνδυασμό με το Apache Kafka. Το Kafka, όπως είδαμε σε παραπάνω ενότητα, παρέχει ένα κατακερματισμένο, ανεκτικό σε σφάλματα που επιτρέπει στους εκδότες να προσαρτούν μηνύματα σε ένα διαμέρισμα καταγραφής και οι καταναλωτές (συνδρομητές) να διαβάζουν διαδοχικά τα μηνύματα σε ένα διαμέρισμα καταγραφής. Το Kafka επιτρέπει επίσης σε εργασίες επεξεργασίας ροής, να επανεπεξεργάζονται αρχεία που έχουν ήδη δει, επαναφέροντας το offset του καταναλωτή σε προηγούμενη θέση, γεγονός που είναι χρήσιμο κατά την ανάκαμψη από αστοχίες.

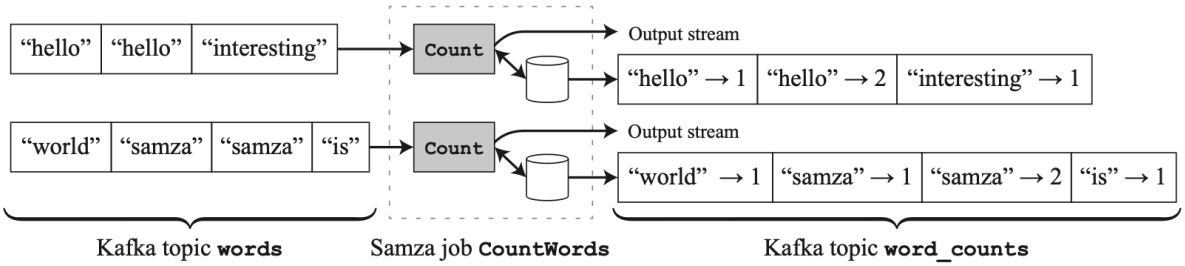
Ωστόσο, η διασύνδεση ροής του Samza είναι με δυνατότητες σύνδεσης: εκτός από το Kafka, μπορεί να χρησιμοποιήσει οποιοδήποτε σύστημα αποθήκευσης ή ανταλλαγής μηνυμάτων ως είσοδο, υπό την προϋπόθεση ότι το σύστημα μπορεί να συμμορφωθεί με το αντίστοιχο log interface. Από προεπιλογή, το Samza μπορεί επίσης να διαβάσει αρχεία από το Hadoop Distributed Filesystem (HDFS) ως είσοδο, με τρόπο που να μοιάζει με τις εργασίες του MapReduce, σε ανταγωνιστική απόδοση (Shadi A. Noghabi, 2017). Στο LinkedIn, το Samza αναπτύσσεται συνήθως με εισόδους βάσης δεδομένων Databus. Το Databus είναι μια τεχνολογία συλλογής δεδομένων για τις αλλαγές που γίνονται και τις οποίες καταγράφει το αρχείο καταγραφής εγγραφών σε μια βάση δεδομένων . Το αρχείο αυτό καθίσταται διαθέσιμο από εφαρμογές για κατανάλωση (Shirshanka Das, 2012). Η επεξεργασία της ροής εγγραφών σε μια βάση δεδομένων επιτρέπει στις εργασίες να διατηρούν εξωτερικούς δείκτες ή να υλοποιούν προβολές σε δεδομένα βάσης δεδομένων και είναι ιδιαίτερα σχετικές σε συνδυασμό με την υποστήριξη του Apache Samza.

Ενώ κάθε διαμέρισμα μιας ροής εισόδου αντιστοιχεί σε μία συγκεκριμένη διεργασία μιας εργασίας Samza, τα διαμερίσματα εξόδου δεν δεσμεύονται σε διεργασίες. Δηλαδή, όταν ένα StreamTask εκπέμπει μηνύματα εξόδου, μπορεί να τα εκχωρήσει σε οποιοδήποτε διαμέρισμα της ροής εξόδου. Το γεγονός αυτό μπορεί να χρησιμοποιηθεί για την ομαδοποίηση σχετικών στοιχείων δεδομένων στο ίδιο διαμέρισμα. Για παράδειγμα, στην εφαρμογή καταμέτρησης λέξεων που απεικονίζεται στην εικόνα 24, η εργασία SplitWords επιλέγει το διαμέρισμα εξόδου για κάθε λέξη βάσει ενός κατακερματισμού της

λέξης. Αυτό διασφαλίζει ότι όταν διαφορετικές εργασίες συναντούν εμφανίσεις της ίδιας λέξης, όλες είναι γραμμένες στο ίδιο διαμέρισμα εξόδου, από όπου μια εργασία κατευθύνεται προς τη ροή των δεδομένων και μπορεί να διαβάσει και να συγκεντρώσει τα συμβάντα.



Εικόνα 25: Στιγμιότυπο μιας διεργασίας Apache Samza η οποία καταναλώνει δεδομένα εισόδου ενός διαμερίσματος αλλά αποστέλλει δεδομένα εξόδου σε οποιοδήποτε διαμέρισμα.



Εικόνα 26: Διεργασία τοπικού διαμερίσματος η οποία καθίσταται διαρκής εκπέμποντας ένα αρχείο καταγραφής αλλαγών στο Kafka.

Όταν οι εργασίες ροής συντίθενται σε αγωγούς επεξεργασίας πολλών φάσεων, η έξοδος μιας εργασίας γίνεται είσοδος σε μια άλλη εργασία. Σε αντίθεση με πολλά άλλα framework επεξεργασίας ροής, το Samza δεν εφαρμόζει το δικό του επίπεδο μεταφοράς μηνυμάτων για την παράδοση μηνυμάτων μεταξύ των χειριστών ροής. Αντ 'αυτού, χρησιμοποιείται το Kafka για αυτόν τον σκοπό, δεδομένου ότι γράφει όλα τα μηνύματα στο δίσκο, παρέχει ένα μεγάλο buffer μεταξύ των σταδίων του αγωγού επεξεργασίας, που περιορίζεται μόνο από τον διαθέσιμο χώρο στο δίσκο στους Kafka brokers.

Συνήθως, το Kafka έχει ρυθμιστεί ώστε να διατηρεί μηνύματα αρκετές ημέρες ή και εβδομάδες σε κάθε θέμα. Έτσι, εάν ένα στάδιο του αγωγού επεξεργασίας αποτύχει ή αρχίσει να λειτουργεί αργά, το Kafka μπορεί απλώς να ρυθμίσει την είσοδο σε αυτό το στάδιο, αφήνοντας άφθονο χρόνο για την επίλυση του προβλήματος. Σε αντίθεση με τα συστήματα τα οποία απαιτούν από έναν παραγωγό να επιβραδύνει εάν ο καταναλωτής δεν

μπορεί να συνεχίσει, η αποτυχία μιας εργασίας Samza δεν επηρεάζει οποιεσδήποτε εργασίες που παράγουν τις εισροές της. Το γεγονός αυτό είναι ζωτικής σημασίας για την εύρωστη λειτουργία συστημάτων μεγάλης κλίμακας, καθώς παρέχει περιορισμό σφαλμάτων. Έτσι ένα σφάλμα σε ένα μέρος του συστήματος δεν επηρεάζει αρνητικά άλλα μέρη του συστήματος.

Τα μηνύματα απορρίπτονται μόνο εάν το αποτυχημένο ή αργό στάδιο επεξεργασίας δεν επιδιορθωθεί εντός της περιόδου διατήρησης του Kafka topic. Σε αυτήν την περίπτωση, η απόρριψη μηνυμάτων είναι επιθυμητή, επειδή απομονώνει το σφάλμα. Η εναλλακτική λύση που μπορεί να εφαρμοστεί είναι η διατήρηση των μηνυμάτων επ 'αόριστον έως ότου επιδιορθωθεί η εργασία. Αυτό όμως θα οδηγούσε σε εξάντληση πόρων (εξάντληση μνήμης ή χώρο στο δίσκο), η οποία θα προκαλούσε διαδοχική αστοχία που επηρεάζει άσχετα μέρη του συστήματος.

Συμβαίνει συχνά σε μεγάλους οργανισμούς - εταιρείες, μια ροή συμβάντων που παράγεται από μια ομάδα να καταναλώνεται από μία ή περισσότερες θέσεις εργασίας ή ακόμα και από διαφορετικές ομάδες. Τη διεργασία αυτή μπορεί να την υποστηρίξει το Apache Samza σε συνδυασμό με το Apache Kafka χάρη στα χαρακτηριστικά του σχεδιασμού του. Οι εργασίες μπορούν και λειτουργούν σε διαφορετικά επίπεδα. Για παράδειγμα, μια ροή που παράγεται από μια σημαντική εργασία παραγωγής μπορεί να καταναλωθεί από πολλές αναξιόπιστες πειραματικές εργασίες. Χρησιμοποιώντας το Apache Kafka για προσωρινή αποθήκευση των δεδομένων μεταξύ των θέσεων εργασίας διασφαλίζεται ότι η προσθήκη ενός αναξιόπιστου καταναλωτή δεν επηρεάζει αρνητικά τις πιο σημαντικές θέσεις εργασίας στο σύστημα.

Τέλος, ένα επιπλέον πλεονέκτημα της χρήσης του Kafka για μεταφορά μηνυμάτων είναι ότι κάθε ροή μηνυμάτων στο σύστημα είναι προσβάσιμη για εντοπισμό σφαλμάτων και παρακολούθηση σε οποιοδήποτε σημείο, δίνοντας τη δυνατότητα να συνδεθεί ένας καταναλωτής για να ελέγξει τη ροή μηνυμάτων.

Αναλύοντας τη λειτουργία του Apache Samza συμπεραίνουμε ότι χρησιμοποιεί εργασίες δημοσίευσης / εγγραφής, οι οποίες παρατηρούν τη ροή δεδομένων, επεξεργάζονται μηνύματα και εξάγουν τα αποτελέσματά τους σε άλλες ροές. Το Samza μπορεί να χωρίσει μια ροή σε πολλά διαμερίσματα και να δημιουργήσει ένα αντίγραφο της

εργασίας για κάθε διαμέρισμα. Επίσης, χρησιμοποιεί το σύστημα ανταλλαγής μηνυμάτων του Apache Kafka, την αρχιτεκτονική και τις εγγυήσεις του, για να προσφέρει buffering, ανοχή σφαλμάτων και κατάσταση αποθήκευσης.

Το Apache Samza διαθέτει API ανάκλησης μηνυμάτων των διεργασιών. Λειτουργεί με παροχή ανοχής σφαλμάτων και μετεγκαθιστά τις εργασίες των χρηστών σε άλλο μηχάνημα εάν αποτύχει ένα μηχάνημα στο σύμπλεγμα. Επεξεργάζεται μηνύματα με τη σειρά που γράφτηκαν και διασφαλίζει ότι δεν θα χαθεί κανένα μήνυμα. Επίσης, είναι επεκτάσιμο καθώς χωρίζεται και διανέμεται σε όλα τα επίπεδα.

Πλεονεκτήματα :

- Πολύ καλό στη διατήρηση μεγάλων ποσοτήτων πληροφορίας (ιδανικό για χρήση σε περιπτώσεις σύνδεσης ροών – joining streams) χρησιμοποιώντας το αρχείο καταγραφής RockDb και Kafka.
- Είναι ανεκτικό σε σφάλματα και παρέχει υψηλές αποδόσεις χρησιμοποιώντας τις ιδιότητες του Apache Kafka.
- Χαμηλός λανθάνων χρόνος.

Μειονεκτήματα:

- Στενά συνδεδεμένο με το Kafka.
- Υποστηρίζει μόνο γλώσσες Java Virual Machine (JVM).
- Δεν υποστηρίζει πολύ χαμηλό λανθάνοντα χρόνο.
- Δεν παρέχει “exactly-once semantics”. Με τον όρο αυτό εννοούμε ότι κάθε εισερχόμενο συμβάν επηρεάζει τα τελικά αποτελέσματα ακριβώς μία φορά. Ακόμα και σε περίπτωση βλάβης του μηχανήματος ή του λογισμικού, δεν υπάρχουν διπλά δεδομένα καθώς και δεδομένα που χρησιμοποιούνται χωρίς επεξεργασία.

5 Αρχιτεκτονική Lambda

Ο Nathan Marz [9] (Warren, 2015, pp. 284 - 299) αναφέρθηκε κάποια στιγμή εκτενώς στην ιδέα αυτή την οποία και ονόμασε Lambda Architecture, η οποία εφαρμόζεται στο Backtype [10] και στο Twitter. Η αρχιτεκτονική Lambda αποτελεί μια αρχιτεκτονική επεξεργασίας δεδομένων που έχει σχεδιαστεί για να χειρίζεται τεράστιες ποσότητες δεδομένων, εκμεταλλευόμενη τις μεθόδους επεξεργασίας δεδομένων δέσμης (batch data) και δεδομένων ροής (stream data). Αυτή η προσέγγιση στην αρχιτεκτονική προσπαθεί να εξισορροπήσει τον λανθάνοντα χρόνο, την απόδοση και την ανοχή σφαλμάτων χρησιμοποιώντας την επεξεργασία δεδομένων δέσμης για να παρέχει ολοκληρωμένες και ακριβείς παρουσιάσεις δεδομένων δέσμης, ενώ ταυτόχρονα χρησιμοποιεί επεξεργασία δεδομένων ροής σε πραγματικό χρόνο για την παροχή παρουσίασης δεδομένων στο διαδίκτυο. Οι δύο έξοδοι προβολής μπορούν και ενώνονται πριν από την παρουσίαση τους. Η άνοδος της αρχιτεκτονικής Lambda συσχετίζεται με την ανάπτυξη μεγάλων δεδομένων, αναλυτικών στοιχείων σε πραγματικό χρόνο και την προσπάθεια μετριασμού των καθυστερήσεων του MapReduce[11].

Η αρχιτεκτονική Lambda περιγράφει ένα σύστημα η οποία αποτελείται από τρία επίπεδα: της επεξεργασίας δεδομένων δέσμης, την γρήγορη ή αλλιώς την επεξεργασία δεδομένων πραγματικού χρόνου και του επιπέδου της εξυπηρέτησης ερωτημάτων.

Ας ρίξουμε μια γρήγορη ματιά σε αυτά τα τρία επίπεδα.

- **Επίπεδο Δέσμης**

  Το επίπεδο δέσμης προ-επεξεργάζεται τα αποτελέσματα χρησιμοποιώντας ένα κατανεμημένο σύστημα επεξεργασίας που μπορεί να χειριστεί πολύ μεγάλες ποσότητες δεδομένων. Στοχεύει στην απόλυτη ακρίβεια με δυνατότητα επεξεργασίας όλων των διαθέσιμων ειδών δεδομένων κατά τη δημιουργία

[9] Nathan Marz : δημιουργός του open-source project Apache Storm το 2011 και ιδρυτής της Red Planet Labs.

[10] Backtype : υπηρεσία ανάλυσης κοινωνικών μέσων SaaS που ξεκίνησε τον Αύγουστο του 2008. Αρχικά, επικεντρώθηκε συγκεντρώνοντας μόνο σχόλια ιστολογίου από διάφορα ιστολόγια επιτρέποντας έτσι στους χρήστες να τα αναζητήσουν ανά θέμα ή συντάκτη. Στη συνέχεια, κυκλοφόρησε το BackTweets, αφιερωμένο στην παρακολούθηση συνομιλιών micro-blogging site, με δυνατότητα αναζήτησης βάσει keywords ή διευθύνσεων URL.

[11] Το MapReduce είναι ένα framework για την επεξεργασία παράλληλων προβλημάτων σε μεγάλα σύνολα δεδομένων χρησιμοποιώντας μεγάλο αριθμό υπολογιστών και αποτελεί μέρος του Apache Hadoop.

παρουσίασης τους . Αυτό σημαίνει ότι μπορεί να διορθώσει τυχόν σφάλματα, υπολογίζοντας εκ νέου με βάση το πλήρες σύνολο δεδομένων και, στη συνέχεια, ενημερώνοντας τις υπάρχουσες προβολές. Η έξοδος συνήθως αποθηκεύεται σε μια βάση δεδομένων μόνο για ανάγνωση, με ενημερώσεις που αντικαθιστούν πλήρως τις υπάρχουσες προ-επεξεργασμένες παρουσιάσεις δεδομένων.

Το Apache Hadoop είναι το κυρίαρχο σύστημα επεξεργασίας δεδομένων δέσμης που χρησιμοποιείται στις περισσότερες αρχιτεκτονικές υψηλής απόδοσης. Προσφέρει παράλληλη επεξεργασία και ελαστικότητα σε σχεσιακές βάσεις δεδομένων και τα Snowflake, Redshift, Azure Cosmos DB και Big Query είναι μερικά παραδείγματα αποθήκευσης δεδομένων νέφους που χρησιμοποιούνται επίσης για αυτό τον σκοπό με το Apache Hadoop.

- **Επίπεδο Γρήγορης επεξεργασίας**

Το επίπεδο αυτό θυσιάζει την απόδοση γιατί στοχεύει στην ελαχιστοποίηση της καθυστέρησης παρέχοντας παρουσιάσεις δεδομένων σε πραγματικό χρόνο στα πιο πρόσφατα δεδομένα. Ουσιαστικά, το επίπεδο αυτό είναι υπεύθυνο για την πλήρωση του "κενού" που προκαλείται από την καθυστέρηση του επιπέδου δέσμης στην παροχή προβολών με βάση τα πιο πρόσφατα δεδομένα. Οι προβολές αυτού του επιπέδου ενδέχεται να μην είναι τόσο ακριβείς ή πλήρεις όσο αυτές που τελικά παράγονται από το επίπεδο δέσμης, αλλά είναι διαθέσιμες σχεδόν αμέσως μετά τη λήψη δεδομένων και μπορούν να αντικατασταθούν όταν γίνουν διαθέσιμες οι προβολές του επιπέδου δέσμης για τα ίδια δεδομένα.

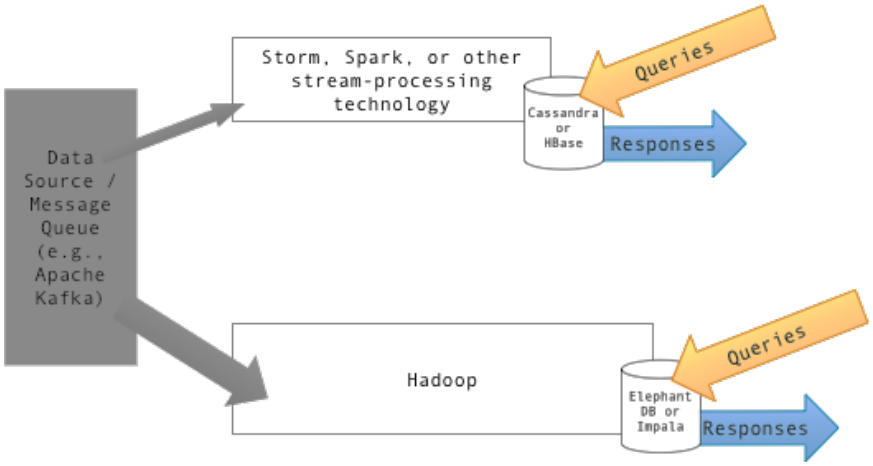
Οι τεχνολογίες επεξεργασίας ροής που χρησιμοποιούνται συνήθως σε αυτό το επίπεδο περιλαμβάνουν τα Apache Storm, SQLstream, Apache Spark και Azure Cosmos DB. Τα αποτελέσματα τους συνήθως αποθηκεύονται σε γρήγορες βάσεις δεδομένων NoSQL.

- **Επίπεδο εξυπηρέτησης**

Η έξοδος από τα δύο παραπάνω επίπεδα αποθηκεύεται στο επίπεδο εξυπηρέτησης, το οποίο ανταποκρίνεται σε ad-hoc ερωτήματα επιστρέφοντας

προ-επεξεργασμένες προβολές ή δημιουργώντας παρουσιάσεις από τα επεξεργασμένα δεδομένα.

Παραδείγματα τεχνολογιών που χρησιμοποιούνται στο επίπεδο εξυπηρέτησης περιλαμβάνουν το Druid, το οποίο παρέχει ένα μόνο cluster για τον χειρισμό της παραγωγής και από τα δύο προηγούμενα επίπεδα. Το επίπεδο εξυπηρέτησης περιλαμβάνει τα Apache Cassandra, Apache HBase, Azure Cosmos DB, MongoDB, VoltDB ή Elasticsearch για έξοδο αποτελεσμάτων από γρήγορο επίπεδο και τα Elephant DB, Apache Impala, SAP HANA ή Apache Hive για έξοδο αποτελεσμάτων από το επίπεδο δέσμης δεδομένων.

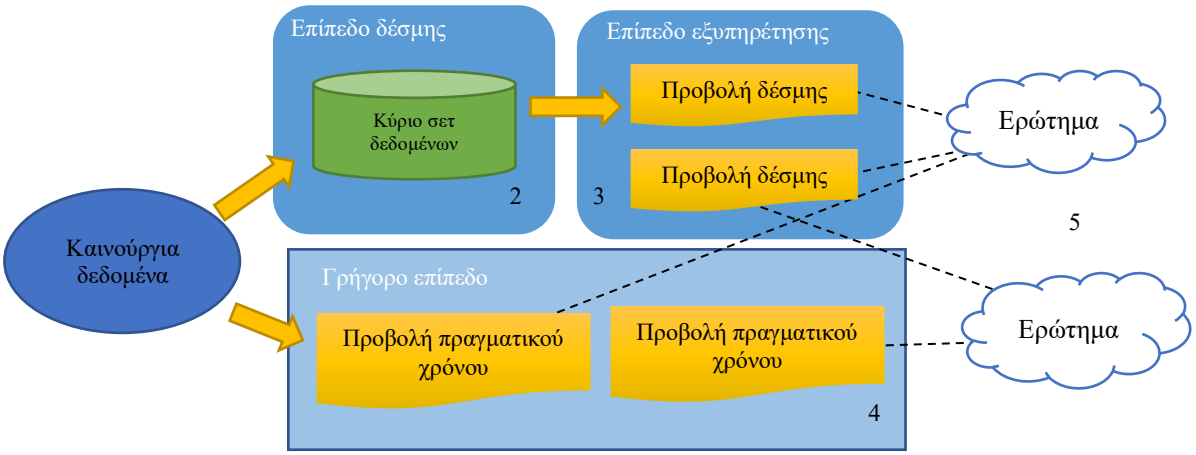


Εικόνα 27: Ροή δεδομένων κατά την επεξεργασία και το πέρασμα από τα επίπεδα εξυπηρέτησης στην αρχιτεκτονική Lambda

Πηγή: Textractor - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=34963986

Όπως αναφέρθηκε και παραπάνω, η αρχιτεκτονική Lambda (LA) στοχεύει στην ικανοποίηση αναγκών για τη δημιουργίας στιβαρού συστήματος το οποίο θα είναι ανεκτικό στα σφάλματα, τόσο απέναντι στις βλάβες υλικού όσο και στα ανθρώπινα λάθη και το οποίο θα είναι σε θέση να εξυπηρετήσει ένα ευρύ φάσμα φορτίου εργασιών στις οποίες απαιτείται γρήγορη προσπέλαση και ενημέρωση δεδομένων με μικρό λανθάνοντα χρόνο. Επίσης, το σύστημα το οποίο θα προκύψει θα πρέπει να είναι γραμμικά επεκτάσιμο.

Παρατηρώντας το σε high-level προοπτική έχουμε την εικόνα 28 όπως αποτυπώθηκε από τους Michael Hausenblas & Nathan Bijnens στην ιστοσελίδα τους για την αρχιτεκτονική Lambda (Bijnens, 2017).



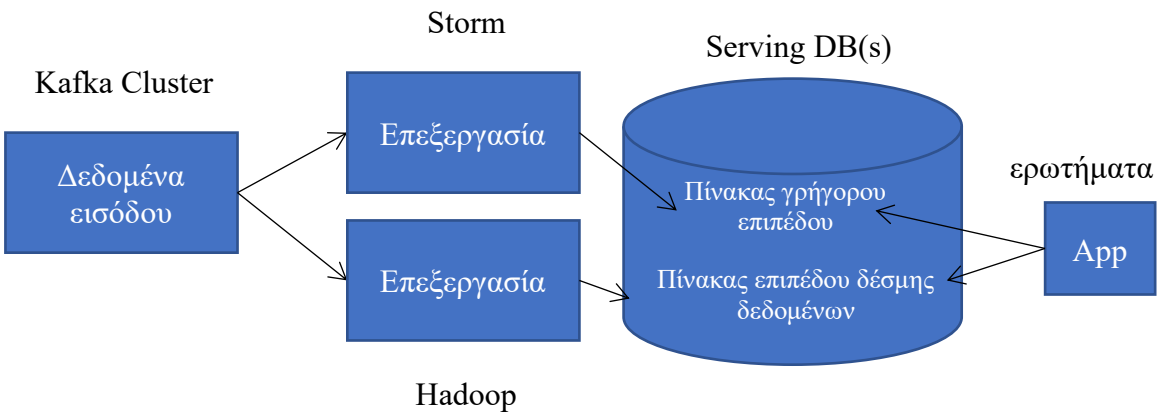
Εικόνα 28: Επίπεδα αρχιτεκτονικής Lambda
πηγή: http://lambda-architecture.net

Οι ακολουθία της διαδικασίας έχει ως εξής:

1. Όλα τα δεδομένα τα οποία εισέρχονται στο σύστημα διαχωρίζονται στα δυο επίπεδα δέσμης και γρήγορης επεξεργασίας.
2. Το επίπεδο δέσμης έχει δυο λειτουργίες: να διαχειριστεί το κύριο σετ δεδομένων και να προ-επεξεργαστεί τις προβολές δεδομένων.
3. Το επίπεδο εξυπηρέτησης τοποθετεί σε ευρετήριο τις προβολές των δεδομένων δέσμης ώστε να εφαρμοστούν τα ερωτήματα σε αυτά, χωρίς καθυστερήσεις.
4. Το γρήγορο επίπεδο έχει να ασχοληθεί με τα πρόσφατα δεδομένα και με αυτό τον τρόπο μας αποζημιώνει για την υψηλή καθυστέρηση της ενημέρωσης των δεδομένων στο επίπεδο εξυπηρέτησης.
5. Οποιοδήποτε εισερχόμενο ερώτημα μπορεί να απαντηθεί ενοποιώντας τα αποτελέσματα που προκύπτουν από τις προβολές δέσμης και τις προβολές πραγματικού χρόνου.

Όπως αναφέρθηκε και παραπάνω, η αρχιτεκτονική Lambda αποτελείται από μια αμετάβλητη ακολουθία δεδομένων η οποία χρησιμοποιείται για να δημιουργήσει ένα σύστημα παράλληλης επεξεργασίας δεδομένων. Υπάρχουν όμως διαφορετικές μορφές της παραπάνω λειτουργίας ανάλογα με το σύστημα στο οποίο θα εφαρμοστεί. Για παράδειγμα στα framework όπως το Kafka, Storm καθώς και στο Hadoop υπάρχουν διαφορετικοί τρόποι εφαρμογής αυτής της αρχιτεκτονικής καθώς χρησιμοποιούμε συνήθως δυο διαφορετικές βάσεις δεδομένων για την αποθήκευση των πινάκων που λαμβάνουμε σαν

επιστρεφόμενα αποτελέσματα, η πρώτη για τα δεδομένα πραγματικού χρόνου και την άλλη για τις ενημερώσεις των δεδομένων δέσμης.



Εικόνα 29: Αρχιτεκτονική Lambda με Kafka Cluster, Storm & Hadoop
πηγή: https://www.oreilly.com/radar/questioning-the-lambda-architecture/

5.1 Πλεονεκτήματα αρχιτεκτονικής Lambda

Ένα από τα πλεονεκτήματα αυτής της αρχιτεκτονικής είναι ότι διατηρεί τα δεδομένα εισόδου αμετάβλητα. Το γεγονός επίσης, ότι εφαρμόζεται πειθαρχημένη μοντελοποίηση στο μετασχηματισμό δεδομένων μέσα από μια σειρά σταδίων είναι ένα έξτρα πλεονέκτημα το οποίο καθιστά ανιχνεύσιμες τις μεγάλες MapReduce ροές εργασίας και μας επιτρέπει τον εντοπισμό σφαλμάτων σε κάθε στάδιο ανεξάρτητα.

Άλλο ένα πλεονέκτημα είναι ότι στην αρχιτεκτονική αυτή γίνεται αντιληπτό το πρόβλημα της επανεπεξεργασίας των δεδομένων ροής, ζήτημα το οποίο αποτελεί βασικό κλειδί για την επεξεργασία των δεδομένων εισόδου ώστε να παράγουμε αποτέλεσμα. Στην περίπτωση αυτή ο κώδικας που έχουμε συντάξει θα αλλάζει πάντα διότι αντλούμε δεδομένα εξόδου μέσα από δεδομένα τα οποία εισάγονται στο σύστημα μας και κάθε φορά υπολογίζεται η έξοδος τους. Για ποιο λόγο όμως αλλάζει ο κώδικας; Ο λόγος είναι ότι είτε κάθε φορά πρέπει να υπολογίζουμε νέα πεδία εξόδου τα οποία δεν χρειαζόμασταν προηγουμένως, είτε επειδή εντοπίσαμε κάποιο λάθος και θα πρέπει να το διορθώσουμε. Στην περίπτωση αυτή θα πρέπει να ξαναδημιουργήσουμε τα αποτελέσματα εξόδου.

Πολλοί προγραμματιστές που ασχολούνται με την επεξεργασία δεδομένων ροής δεν το λαμβάνουν αυτό υπόψη και καταλήγουν σε ένα σύστημα το οποίο δεν μπορεί να εξελιχθεί γρήγορα και άμεσα λόγω του δύσκολου τρόπου χειρισμού της λειτουργίας επανεπεξεργασίας των δεδομένων εξόδου.

Η προσθήκη ενός άλλου στρώματος σε μια αρχιτεκτονική έχει σημαντικά πλεονεκτήματα. Πρώτον, τα δεδομένα μπορούν να υποβληθούν σε επεξεργασία υψηλής ακρίβειας εμπλέκοντας διάφορους αλγόριθμους χωρίς να χάσουν βραχυπρόθεσμες πληροφορίες, ειδοποιήσεις και πληροφορίες που παρέχονται από το επίπεδο πραγματικού χρόνου. Επίσης, η προσθήκη ενός στρώματος αντισταθμίζεται μειώνοντας δραματικά τις απαιτήσεις αποθήκευσης τυχαίων εγγραφών. Η αποθήκευση δεδομένων δέσμης παρέχει επίσης την επιλογή εναλλαγής δεδομένων σε προκαθορισμένους χρόνους και δεδομένα διαφόρων εκδόσεων.

Τελευταίο και σημαντικό, η προσθήκη της δεξαμενής δεδομένων των ανεπεξέργαστων δεδομένων προσφέρει την επιλογή ανάκτησης από ανθρώπινα λάθη, γεγονός που άλλες αρχιτεκτονικές δεν μπορούν να το πράξουν. Μια άλλη επιλογή είναι η αναδρομική ενίσχυση της παραγωγής δεδομένων ή των αλγορίθμων μηχανικής μάθησης και η εφαρμογή τους σε όλα τα δεδομένα ιστορικού.

5.2 Μειονεκτήματα αρχιτεκτονικής Lambda

Ένα από τα βασικά μειονεκτήματα αυτής της αρχιτεκτονικής, όπως κατεγράφει από τον Jay Kreps στο ηλεκτρονικό του άρθρο για την ιστοσελίδα O'Reilly (Kreps, 2014), είναι η συντήρηση του κώδικα. Εξαιρετικά επίπονο γεγονός για τον προγραμματιστή αποτελεί η περίπτωση που πρέπει να συντηρήσουμε κώδικα ο οποίος πρέπει να εφαρμοστεί σε δυο διαφορετικά και περίπλοκα σύστημα. Ο προγραμματισμός σε κατανεμημένα frameworks όπως το Storm και το Hadoop είναι περίπλοκος. Αναπόφευκτα, ο κώδικας καταλήγει να κατασκευάζεται ειδικά για το framework στο οποίο λειτουργεί. Η προκύπτουσα λειτουργική πολυπλοκότητα των συστημάτων που εφαρμόζουν την Αρχιτεκτονική Lambda είναι το μόνο πράγμα που φαίνεται να έχει συμφωνηθεί παγκοσμίως από όλους τους προγραμματιστές που την εφαρμόζουν.

Υπάρχει όμως και μια άλλη δυσκολία η οποία εντοπίζεται στο γεγονός ότι το σύστημα επεξεργασίας δεδομένων δεν μπορεί να εξελιχθεί και να χειρίζεται εξ ολοκλήρου το πρόβλημα στο τερματικό σημείο που πρέπει να καταλήξει. Μια προσέγγιση που προτάθηκε για την επίλυση αυτού είναι να έχουμε μια γλώσσα ή ένα αφηρημένο framework τόσο σε πραγματικό χρόνο όσο και σε δέσμες δεδομένων. Γράφοντας έτσι τον κώδικα μας και χρησιμοποιώντας framework υψηλότερου επιπέδου, μεταγλωττίζοντας το στη συνέχεια για επεξεργασία δεδομένων ροής ή για εφαρμογή MapReduce έχουμε τη δυνατότητα να επεξεργαστούμε πιο γρήγορα τα δεδομένα μας χωρίς να γίνεται αντιληπτή η διεργασία αυτή. Το Summingbird είναι ένα framework που το κάνει αυτό. Αυτό σίγουρα κάνει τα πράγματα λίγο καλύτερα, αλλά δεν έχει αποδειχθεί ότι λύνει το πρόβλημα.

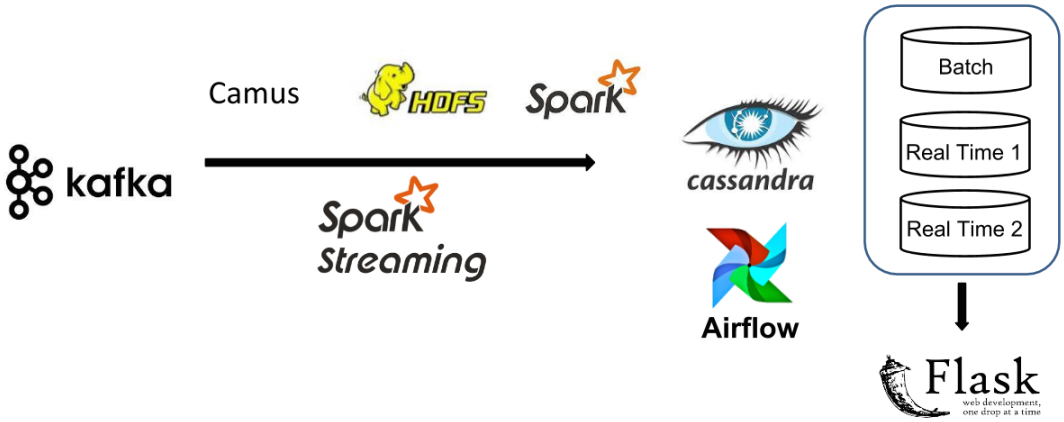
Ακόμη και αν μπορέσουμε να αποφύγουμε την διπλή κωδικοποίηση της εφαρμογής μας δύο φορές, το λειτουργικό βάρος της εκτέλεσης και του εντοπισμού σφαλμάτων δύο συστημάτων θα είναι πολύ υψηλό. Κάθε νέα abstract μέθοδος μπορεί να παρέχει μόνο τα χαρακτηριστικά που υποστηρίζονται από τη διασταύρωση των δύο συστημάτων.

5.3 Εξέλιξη αρχιτεκτονικής Lambda

Έχοντας στο μυαλό όλα τα παραπάνω πλεονεκτήματα και μειονεκτήματα της αρχιτεκτονικής Lambda και αναλογιζόμενοι τις σημερινές ανάγκες της αγοράς δημιουργήθηκε η ανάγκη εξέλιξης της επεξεργασίας τεράστιων όγκων δεδομένων και το πέρασμα σε μια πιο γρήγορη προσέγγιση των πραγμάτων ώστε οι χρήστες να μην περιμένουν σχεδόν ένα 24ωρο για να λάβουν στα χέρια τους τα αποτελέσματα μιας ανάλυσης δεδομένων. Με τον τρόπο αυτό, περάσαμε στην αξιοποίηση της γρήγορης πρόσβασης σε δεδομένα ιστορικού με δεδομένα ροής σε πραγματικό χρόνο χρησιμοποιώντας το Apache Spark (Core, SQL, Streaming), Apache Parquet, Twitter Stream κ.λπ.

Η επιχειρηματική πραγματικότητα έχει αλλάξει, οπότε τώρα η λήψη ταχύτερων αποφάσεων είναι πιο πολύτιμη. Επιπλέον, οι τεχνολογίες έχουν εξελιχθεί. Οι πάροχοι Kafka, Storm, Trident, Samza, Spark, Flink, Parquet, Avro, Cloud κ.λπ. είναι πλέον γνωστοί και υιοθετούνται ευρέως τόσο από μηχανικούς όσο και από επιχειρήσεις.

Για να παρατηρήσουμε καλύτερα το πως εφαρμόζεται η αρχιτεκτονική Lambda ας δούμε με ποιον τρόπο αναλύονται τα tweets με ειδήσεις οικονομικού περιεχομένου στο Twitter, με τον τρόπο που αποτυπώθηκαν από την ομάδα του Medium Insight (Insight Team, 2016) σε ανάρτηση τους. (samza.apache.org, 2019)



Εικόνα 30: Διεργασία επεξεργασίας tweets
πηγή: Insight

Τα δεδομένα εισόδου έρχονται ως μηνύματα JSON, με δεδομένα που συντίθενται από κανονική διανομή και ειδήσεις που προέρχονται από το API του Twitter. Αυτά τα μηνύματα JSON ωθούνται στο Kafka και χρησιμοποιούνται τόσο από το επίπεδο δέσμης όσο και από το επίπεδο πραγματικού χρόνου.

Χρησιμοποιώντας το Kafka στην αρχή της διεργασίας για αποδοχή των δεδομένων εισόδου, διασφαλίζεται ότι όσο τα μηνύματα εισέρχονται στο σύστημα θα παραδίδονται, ανεξάρτητα από διάφορες αστοχίες υλικού ή δικτύου.

Στο επίπεδο παρτίδας, το Camus[12] χρησιμοποιείται για την κατανάλωση όλων των μηνυμάτων από το Kafka και αποθηκεύονται σε HDFS[13] (Hadoop Distributed File System), στη συνέχεια το Spark αθροίζει το ιστορικό συναλλαγών για να πάρει έναν

[12] Το Camus εφαρμόζει MapReduce που αναπτύχθηκε από το LinkedIn για τη φόρτωση δεδομένων από την Kafka σε HDFS. Είναι σε θέση να αντιγράφει σταδιακά δεδομένα από την Kafka σε HDFS έτσι ώστε κάθε εκτέλεση της εργασίας του MapReduce να παίρνει εκεί που έπαυσε η προηγούμενη εκτέλεση.

[13] Το HDFS είναι το κύριο σύστημα αποθήκευσης δεδομένων που χρησιμοποιείται από εφαρμογές Hadoop. Χρησιμοποιεί μια αρχιτεκτονική NameNode και DataNode για την υλοποίηση ενός κατανεμημένου συστήματος αρχείων που παρέχει πρόσβαση υψηλής απόδοσης σε δεδομένα σε Hadoop clusters με εξαιρετικά επεκτάσιμη κλίμακα.

ακριβή αριθμό αποθεμάτων που κατέχει κάθε χρήστης. Τα συγκεντρωτικά αποτελέσματα στη συνέχεια εγγράφονται σε έναν πίνακα βάσης δεδομένων Cassandra[14].

Στο επίπεδο ροής, τα μηνύματα Kafka καταναλώνονται σε πραγματικό χρόνο χρησιμοποιώντας το Spark Streaming. Το Spark Streaming δεν είναι εξ'ολοκλήρου σε πραγματικό χρόνο όπως το Storm, καθώς οι μικρές δέσμες δεδομένων μεταδίδουν δεδομένα σε RDD [15]με ανάλυση έως 500ms. Ωστόσο, επιτρέπει την επαναχρησιμοποίηση του κώδικα Spark στο επίπεδο δέσμης δεδομένων και η καθυστέρηση δημιουργίας μικρών δεσμών δεδομένων είναι ελάχιστη , σχεδόν ασήμαντη, για την ενημέρωση νέων tweet.

Τα αποτελέσματα από τα επίπεδα δέσμης δεδομένων καθώς και από το επίπεδο πραγματικού χρόνου καταγράφονται στο Cassandra και τα δεδομένα παρέχονται μέσω μιας διεπαφής ιστού του Flask. Λόγω του υψηλού αριθμού ανταλλαγής δεδομένων που καταγράφεται στο σύστημα, η ικανότητα γρήγορης εγγραφής του Cassandra είναι η πλέον κατάλληλη.

[14] Apache Cassandra : παρέχει υπηρεσίες αποθήκευσης βάσεων δεδομένων με χαμηλό χρόνο καθυστέρησης για τους χρήστες με επεκτασιμότητα και υψηλή διαθεσιμότητα χωρίς να διακυβεύεται η απόδοση.

[15] RDD : Ένα ανθεκτικό κατανεμημένο σύνολο δεδομένων. Αποτελεί τη βασική αφηρημένη κλάση στο Spark. Αντιπροσωπεύει μια αμετάβλητη, κατατμημένη συλλογή στοιχείων που μπορούν να λειτουργήσουν παράλληλα. Περιέχει τις βασικές λειτουργίες που είναι διαθέσιμες σε όλα τα RDD, όπως map, filter και persist.

6 Συγκριτική ανάλυση των frameworks ροής δεδομένων

Σε αυτό το κεφάλαιο, θα πραγματοποιήσουμε μια σύγκριση των frameworks που παρουσιάστηκαν στα προηγούμενα κεφάλαια - ενότητες. Τα κριτήρια σύγκρισης προέκυψαν με διάφορα χαρακτηριστικά που είναι απαραίτητα για την επεξεργασία δεδομένων ροής όπως ο μηχανισμός window, τα αποδεικτικά παράδοσης των μηνυμάτων και η επεξεργασία μηνυμάτων, η διαχείριση κατάστασης και ο μηχανισμός ανοχής σφαλμάτων. Επίσης, πραγματοποιείται σύγκριση των frameworks με βάσει κάποια επιπλέον χαρακτηριστικά, όπως οι πηγές δεδομένων, οι γλώσσες προγραμματισμού που υποστηρίζουν, η δυνατότητα χρήσης SQL κ.λπ.

6.1 Μηχανισμός Παραθύρου - Windowing

Το Samza υποστηρίζει χρονικά παράθυρα (time windows), παράθυρα συνεδρίας (session windows) και καθολικά παράθυρα (global windows). Ένα καθολικό παράθυρο είναι ένα απλό ατέρμων παράθυρο σε ολόκληρη τη ροή δεδομένων - μηνυμάτων. Το Samza υποστηρίζει ταυτόχρονα σταθερά και συρόμενα χρονικά παράθυρα. Επίσης, υποστηρίζει μόνο το χρονικό παράθυρο επεξεργασίας, αλλά με την ενσωμάτωση του Apache Beam, υποστηρίζει και το παράθυρο event-time. (Samza Apache Org, 2019)

Το Flink υποστηρίζει παράθυρα με βάση το χρόνο (time-based windows), παράθυρα συνεδρίας (session windows) και τα καθολικά παράθυρα (global windows). Δίνει τη δυνατότητα για σταθερά και συρόμενα χρονικά παράθυρα και διαθέτει υποστήριξη τόσο για το χρόνο επεξεργασίας όσο και για το χρόνο εκδήλωσης. (Flink Apache Org, 2019).

Το Storm έχει επίσης υποστήριξη για διαφορετικά παράθυρα. Υποστηρίζει τόσο σταθερό όσο και κυλιόμενο χρόνο. Δεν διαθέτει υποστήριξη για παράθυρα συνεδρίας (session windows), διαθέτει όμως καθολικά παράθυρα (global windows). Τέλος, το Storm παρέχει υποστήριξη τόσο για παράθυρα χρόνου επεξεργασίας (processing time windows) όσο και για παράθυρα χρόνου γεγονότος (event-time windows) (Software Foundation, Apache, 2019).

Το Spark Streaming υποστηρίζει σταθερό και συρόμενο παράθυρο, δεν διαθέτει υποστήριξη για περιόδους συνεδρίας και παγκόσμιο παράθυρο, παρέχει υποστήριξη τόσο για επεξεργασία όσο και για επεξεργασίες χρονικού διαστήματος. (Spark, Apache, 2019)

Πίνακας 4: Συγκριτικός πίνακας λειτουργίας παραθύρου Apache Frameworks, (Mian, February 23, 2020)

Λειτουργία Παραθύρου (Windowing)	Apache Samza	Apache Flink	Apache Storm	Apache Spark
Fixed Windows	NAI	NAI	NAI	NAI
Sliding Windows	NAI	NAI	NAI	NAI
Session Windows	NAI	NAI	OXI	OXI
Global Windows	NAI	NAI	NAI	OXI

6.2 Διαχείριση κατάστασης δεδομένων - State management

Η διαχείριση και ο έλεγχος της κατάστασης των δεδομένων ροής είναι ένα σημαντικό χαρακτηριστικό των frameworks που εφαρμόζονται σε ροές δεδομένων, λόγω του ότι πρέπει να γίνονται συνεχείς και επαναλαμβανόμενοι έλεγχοι σε αυτά. Παραδείγματα υπολογισμού και ελέγχου της κατάστασης αποτελούν οι λειτουργίες count και join. Αυτά τα είδη λειτουργιών απαιτούν έλεγχο της κατάστασης και δεν αποτελούν ασήμαντες λειτουργίες. Διαφορετικές μηχανές ροής παρέχουν διαφορετικούς μηχανισμούς διαχείρισης κατάστασης.

Το Samza εφαρμόζει τη διαχείριση της κατάστασης με μια τεχνική που ονομάζεται state-store όπου κάθε εργασία του Samza έχει τη δική της βάση δεδομένων κατάστασης. Το state-store σχετίζεται με μια συγκεκριμένη εργασία του Samza η οποία αποθηκεύει δεδομένα που αντιστοιχούν σε αυτήν τη συγκεκριμένη εργασία. Επίσης, το Samza αποθηκεύει την κατάσταση τοπικά στον ίδιο υπολογιστή στον οποίο εκτελείται η εργασία. Αυτό οφείλεται στο γεγονός ότι επιτυγχάνουμε απόδοση χωρίς να εκτελούμε απομακρυσμένα ερωτήματα μέσω δικτύου. Επίσης, η κατάσταση (state) των δεδομένων αποθηκεύεται σε δίσκο έτσι ώστε να διατηρούμε περισσότερη πληροφορία απ' ότι αν αποθηκεύαμε στη μνήμη. Οποιαδήποτε μηχανή αποθήκευσης - LevelDB, LMDB - μπορεί να χρησιμοποιηθεί για αποθήκευση καταστάσεων. Το Samza για την αποθήκευση της

κατάστασης αποστέλλει έναν μηχανισμό κλειδιού-τιμής (key-value) που βασίζεται στο RocksDB. Το γεγονός ότι η κατάσταση αποθηκεύεται τοπικά σε συνδυασμό με το ότι μια Samza διεργασία εκτελείται στον ίδιο υπολογιστή αποτελεί μεγάλο πλεονέκτημα για το Samza, ειδικά για μια εργασία όπου υπάρχει μεγάλος αποθηκευτικός χώρος.

Το Spark διατηρεί καταστάσεις τόσο σε συνεχές μοντέλο όσο και σε μικρές παρτίδες. Η κατάσταση διατηρείται χρησιμοποιώντας δύο εξωτερικά συστήματα αποθήκευσης: ένα ημερολόγιο εγγραφής προς τα εμπρός που υποστηρίζει ανθεκτικές, ατομικές εγγραφές και ένα σημείο αποθήκευσης που μπορεί να αποθηκεύσει μεγάλη ποσότητα δεδομένων με διάρκεια επιτρέποντας παράλληλη επεξεργασία (S3, HDFS)

Το Storm υποστηρίζει τόσο αποθήκευση στη μνήμη όσο και στο δίσκο για διατήρηση της κατάστασης. Επίσης, διαθέτει προεπιλεγμένη εφαρμογή κατάστασης στη μνήμη.

Το Flink υποστηρίζει τοπικό και εξωτερικό χώρο αποθήκευσης για τη διαχείριση της κατάστασης. Η τοπική αποθήκευση για διαχείριση κατάστασης ορίζεται είτε στην τοπική μνήμη είτε σε δίσκο με μια βάση δεδομένων κλειδιού-τιμής (key-value), όπως το RocksDB. Επίσης, ο εξωτερικός χώρος αποθήκευσης χρησιμοποιείται για τη διατήρηση της κατάστασης.

Πίνακας 5: Μηχανισμοί κατάστασης στα διάφορα frameworks

Μηχανές streaming	Τρόποι διαχείρισης κατάστασης
Apache Samza	Key-value , αποθήκευση τοπικά, RocksDB
Apache Flink	Key-value , τοπική καθώς και εξωτερική αποθήκευση, RocksDB
Apache Storm	Key-value, χρήση μνήμης, τοπική αποθήκευση,
Apache Spark	Εξωτερική αποθήκευση , state-store

6.3 Εγγύηση επεξεργασίας δεδομένων

Το Samza εγγυάται την επεξεργασία τουλάχιστον μία φορά (at-least-once) (samza.apache.org, 2019). Διασφαλίζει ότι ένα μήνυμα θα υποβληθεί σε επεξεργασία τουλάχιστον μία φορά, αλλά ενδέχεται να το επανεπεξεργαστεί περισσότερες από μία φορές, για παράδειγμα να πραγματοποιήσει υποβολή επανεπεξεργασίας των ροών σε περίπτωση αποτυχίας. Διασφαλίζει, επίσης, ότι δεν θα χαθούν δεδομένα, διατηρώντας πιθανά αντίγραφα. Τα παραπάνω επιτυγχάνονται χρησιμοποιώντας τη λειτουργία checkpoint.

Για να χρησιμοποιηθεί από ένα σύστημα εισαγωγής θα πρέπει να διασφαλίζει τις ακόλουθες διεργασίες :

1) Μια εισερχόμενη ροή χωρίζεται σε ένα ή περισσότερα διαμερίσματα. Κάθε διαμέρισμα είναι ανεξάρτητο το ένα από το άλλο. Κάθε διαμέρισμα αναπαράγεται σε πολλά διαμερίσματα για να συνεχιστεί η διαθεσιμότητα ροής σε περίπτωση βλάβης του μηχανήματος.

2) Τα μηνύματα σε κάθε διαμέρισμα είναι διαδοχικά και έχουν σταθερή σειρά. Κάθε μήνυμα έχει μια μετατόπιση που προσδιορίζει μοναδικά τη θέση του μηνύματος στο διαμέρισμα. Παρομοίως, τα μηνύματα καταναλώνονται διαδοχικά σε κάθε διαμέρισμα.

3) Μια διεργασία Samza μπορεί να αρχίσει να καταναλώνει την ακολουθία των μηνυμάτων από οποιαδήποτε αρχική μετατόπιση. Συστήματα όπως το Kafka διασφαλίζουν τις απαιτήσεις αυτές.

Το Spark διασφαλίζει και αυτό την επεξεργασία μιας τουλάχιστον φοράς μέσω καταγραφής checkpoint και με τη δημιουργία log. Τα offset των δεδομένων ροής που υποβάλλονται σε επεξεργασία αποθηκεύονται σε σημεία ελέγχου checkpoints καθώς και σε ένα log ημερολόγιο εγγραφής. Σε περίπτωση αποτυχίας η επανεπεξεργασία μπορεί να ξεκινήσει από την αρχή αλλά δεν θα επεξεργαστεί τα αναγνωρισμένα δεδομένα.

Και το Flink υποστηρίζει επεξεργασία τουλάχιστον μίας φοράς χρησιμοποιώντας αλγόριθμο σημείων ελέγχου checkpoint και ανάκτησης των σημείων αυτών. Σε περίπτωση

αποτυχίας, το Apache Flink επαναφέρει το πιο πρόσφατο στιγμιότυπο από το χώρο αποθήκευσης. Ο αλγόριθμος του Flink βασίζεται σε μια τεχνική που εισήχθη το 1985 από τους Chandy και Lamport.

Τέλος, το Storm διασφαλίζει εγγύηση επεξεργασίας τουλάχιστον μίας φοράς αν και διαθέτει το Trident το οποίο μπορεί να υποστηρίξει επεξεργασία ακριβώς μία φορά. (_Apache Software Foundation, 2019)

Πίνακας 6: Εγγυήσεις επεξεργασίας στα διάφορα frameworks

Μηχανές streaming	Εγγυήσεις Επεξεργασίας
Apache Samza	Ακριβώς μια (Exactly-Once)
Apache Flink	Ακριβώς μια (Exactly-Once)
Apache Storm	Τουλάχιστον μια και Ακριβώς μια (Exactly-Once) με τη χρήση Trident
Apache Spark	Ακριβώς μια (Exactly-Once)

6.4 Ανεκτικότητα σε σφάλματα - Fault tolerance

Οι μηχανισμοί ροής πρέπει να έχουν τη δυνατότητα γρήγορης επανάκαμψης από τις αποτυχίες που μπορεί να προκύψουν για να συνεχίσουν την επεξεργασία των ροών. Αυτή η δυνατότητα ονομάζεται ανοχή σφαλμάτων. Το Samza επιτυγχάνει ανοχή σφαλμάτων αποθηκεύοντας κάθε αλλαγή σε μια ξεχωριστή ροή που ονομάζεται changelog. Σε περίπτωση αποτυχίας, η ανάκτηση της επεξεργασίας μπορεί να γίνει διαβάζοντας δεδομένα από το changelog. Συνήθως, ένα Kafka topic που είναι συμπιεσμένο σε log χρησιμοποιείται ως changelog. Διατηρεί την πιο πρόσφατη τιμή κάθε κλειδιού. Επίσης, σε περίπτωση αποτυχίας, το Samza προσπαθεί να επαναπρογραμματίσει τις εργασίες στους ίδιους κεντρικούς υπολογιστές για να επαναχρησιμοποιήσει το στιγμιότυπο της τοπικής κατάστασης που είναι αποθηκευμένο στον ίδιο κεντρικό υπολογιστή. Η δυνατότητα αυτή ονομάζεται host-affinity. Με τη δυνατότητα αυτή, το Samza δεν χρειάζεται να αλλάξει την τοπική κατάσταση από το changelog. Εάν η κατάσταση είναι μεγάλη σε ποσότητα μπορεί να χρειαστεί πολύς χρόνος για επεξεργασία σε πραγματικό χρόνο, και έτσι δεν γίνεται αποδοτική.

Το Flink εφαρμόζει μηχανισμό ανοχής σφαλμάτων με συνδυασμό checkpoint και επανάληψης ροής. Ένα σημείο ελέγχου αποτελεί στιγμιότυπο ροής εισόδου μαζί με την κατάσταση σε ένα συγκεκριμένο σημείο. Σε περίπτωση αποτυχίας, η ροή δεδομένων ροής μπορεί να επανεπεξεργαστεί από το σημείο ελέγχου επαναφέροντας την κατάσταση των λειτουργιών και επαναλαμβάνοντας τα γεγονότα από το σημείο του σημείου ελέγχου.

Το Spark εφαρμόζει μηχανισμούς ανοχής σφαλμάτων μέσω σημείων ελέγχου και καταγραφής εγγράφων. Το αρχείο καταγραφής παρακολουθεί ποια δεδομένα έχουν υποβληθεί σε επεξεργασία και εγγράφεται στη δεξαμενή εξόδου με αξιοπιστία, ενώ το σημείο ελέγχου κρατά το στιγμιότυπο της κατάστασης των λειτουργιών. Σε περίπτωση αποτυχίας, παρακολουθεί ποια κατάσταση έχει ενημερώσει τελευταία το αρχείο καταγραφής και υπολογίζει την κατάσταση από τα δεδομένα που ξεκινούν από αυτό το μέρος. Μπορούν να χρησιμοποιηθούν τόσο για έλεγχο σημείων ελέγχου όσο και για εξωτερική αποθήκευση εγγραφής, όπως τα HDFS, S3.

Το Storm χρησιμοποιεί μηχανισμό χειρισμού για την ανοχή σφαλμάτων (acking mechanisim). Το acking mechanisim παρακολουθεί την ολοκλήρωση κάθε δέντρου tuple με κατακερματισμό. Η τιμή του checksum γίνεται μηδέν εάν όλες οι πλειάδες αναγνωριστούν επιτυχώς. Στο Storm, η τοπολογία σχετίζεται με ένα "χρονικό όριο μηνυμάτων". Αν κάθε μήνυμα δεν επεξεργαστεί σε συγκεκριμένα χρονικά όρια, το Storm θεωρεί ότι η ροή έχει αποτύχει και το επαναλαμβάνει. Ομοίως, ο μηχανισμός ελέγχει περιοδικά την κατάσταση της ροής για να διασφαλίσει ότι μπορεί να γίνει επανεκκίνηση από την αποθηκευμένη κατάσταση εφόσον χρειαστεί.

Πίνακας 7: Ανεκτικότητα σε σφάλματα στα διάφορα frameworks

Μηχανές streaming	Ανεκτικότητα σε σφάλματα
Apache Samza	Change log , Host-affinity
Apache Flink	Checkpoint , Επανεκτέλεση της ροής
Apache Storm	Checkpoint , Επανεκτέλεση της ροής
Apache Spark	Checkpoint , Write-ahead-log

6.5 Λοιπές Συγκρίσεις

Οι εφαρμογές Samza μπορούν να γραφτούν μόνο σε Java τη δεδομένη στιγμή. Προσφέρει ενσωματωμένη υποστήριξη για διαφορετικές πηγές δεδομένων όπως Apache Kafka, AWS Kinesis, Azure Event Hubs, Elastic Search και Apache Hadoop. Επίσης, είναι πολύ εύκολο να ενσωματωθεί με άλλες πηγές. Το Samza μπορεί και διαβάζει ροές από το AWS Kinesis, αλλά δεν μπορεί να εγγράψει σε αυτό. Έχει επίσης περιορισμένη υποστήριξη SQL.

Οι εφαρμογές Flink μπορούν να γραφτούν σε Java, Scala και Python. Προσφέρει ενσωματωμένη υποστήριξη για διαφορετικές πηγές δεδομένων, όπως Apache Kafka, AWS Kinesis, Apache Cassandra, RabbitMQ, Apache Nifi, Twitter Streaming API, Google Pub / Sub, Elastic Search και Apache Hadoop. Το Flink διαθέτει επίσης υποστήριξη SQL.

Οι εφαρμογές Spark μπορούν να γραφτούν σε Java, Scala, Python και R. Προσφέρει ενσωματωμένη υποστήριξη για διαφορετικές πηγές δεδομένων όπως Apache Kafka, AWS Kinesis, AWS S3, Azure Event Hubs και Delta Lake. Η δομημένη ροή έχει επίσης υποστήριξη για SQL.

Οι εφαρμογές Storm μπορούν να γραφτούν σε Java. Προσφέρει ενσωματωμένη υποστήριξη για Apache Kafka, HDFS, Hive, Solr, Cassandra, RocketMQ, JMS, JDBC, MQTT, Redis, Events Hubs, Kinesis, Kestrel, MongoDB, OpenTSDB, PMML και Elasticsearch Storm έχει επίσης υποστήριξη για SQL.

Πίνακας 8: Frameworks και οι γλώσσες που υποστηρίζουν

Λοιπά Χαρακτηριστικά	Samza	Flink	Storm	Spark
Γλώσσες Προγραμματισμού	Java	Java, Scala, Python	Java	Java, Scala, Python, R
Πηγές Δεδομένων	Kafka, Eventhubs, AWS Kinesis κτλ.	Kafka, Eventhubs, AWS kinesis κτλ.	Kafka, Rab-bitMq / AMPQ, AWS kinesis κτλ.	Kafka, AWS S3, AWS kinesis, Even Hubs κτλ.
Υποστήριξη SQL	Samza SQL	Flink SQL	Storm SQL	Spark SQL
Περιβάλλον Ανάπτυξης	Standalone, Hadoop YARN	Standalone, Hadoop, YARN, Apache Mesos, AWS κτλ.	Apache Zookeeper	Standalone, Hadoop YARN, Apache Mesos, kubernetes

Από τη σύγκριση, δεν μπορούμε να πούμε ποια είναι η καλύτερη μηχανή ροής, επειδή κάθε μηχανή ροής έχει κάποια πλεονεκτήματα και κάποιους περιορισμούς. Πώς επιλέγω τελικά το κατάλληλο framework για δεδομένα ροής; Η απάντηση βρίσκεται στο ότι εξαρτάται από τις περιπτώσεις χρήσης. Ένα framework ροής μπορεί να ταιριάζει περισσότερο από κάποιο άλλο για μια συγκεκριμένη περίπτωση χρήσης. Για παράδειγμα εάν μια εταιρεία διαθέτει προσωπικό που εξειδικεύεται στη γλώσσα R, τότε το Spark framework μπορεί να είναι μια καλή επιλογή επειδή διαθέτη τέτοια υποστήριξη. Στην περίπτωση που κάποιοι χρησιμοποιούν ήδη το Kafka ως μεσίτη μηνυμάτων τότε μπορεί να αποτελεί μια καλή επιλογή σε αντίθεση με το Samza. Ομοίως, εάν κάποιος πρέπει να δουλέψει με το Twitter streaming API, το Storm είναι μια καλή επιλογή, καθώς διαθέτει ενσωματωμένη υποστήριξη για το Twitter-streaming API.

Ένα άλλο σενάριο είναι εάν κάποιος χρειάζεται επεξεργασία παρτίδας και επεξεργασία ροής, τότε το Flink ή το Spark αποτελούν καλές επιλογές επειδή και τα δύο υποστηρίζουν επεξεργασία παρτίδας. Ομοίως, για μηχανική εκμάθηση, το Analytic Flink και το Spark αποτελούν καλές επιλογές λόγω των ενσωματωμένων βιβλιοθηκών

μηχανικής μάθησης. Τέλος, εάν χρειαζόμαστε επεξεργασία δεδομένων ακριβώς για μια φορά, τότε το Spark και το Flink είναι οι σωστές επιλογές, καθώς και τα δύο προσφέρουν επεξεργασία exact-once.

6.6 Ομοιότητες και διαφορές στους Apache μηχανισμούς ροής δεδομένων

Στη συνέχεια παρατίθεται ο πίνακας 9 στον οποίο παρουσιάζονται συγκεντρωτικά τις ομοιότητες και τις διαφορές των Apache frameworks ροής δεδομένων.

Πίνακας 9: Συγκεντρωτικός πίνακας Apache frameworks

	Storm	Spark Streaming	Flink	Kafka	Samza
Category	ESP/CEP	ESP	ESP/CEP	ESP	ESP
Streaming API	Spout, Tuple	HDFS, DStream	DataStream	KafkaStream	Message
Cluster Management	Mesos, Yarn	Mesos, Yarn, Standalone	Yarn, Tez, Standalone	Any	Yarn
Process Model	Event	Micro-batch Batch	Event Micro-Batch Batch	Event	Event
Latency	Very low	Medium	Low	Low	Low
Throughput	Low	High	High	Medium	High
Fault-Tolerance	Yes	Yes	Yes	Yes	Yes
Language Support	Any JVM	Scala/Java/Python	Scala/Java/Python	Java	Scala/Java
Delivery Guarantee	At Least Once	Exactly Once	Exactly Once	At Least Once	At Least Once
Windowing	Time-based Count-based	Time-based	Time-based Count-based	Time-based	Time-based

7. Πειραματική μελέτη

Σε την ενότητα αυτή, θα παρουσιάσουμε το πειραματικό περιβάλλον και το πρωτόκολλό που ακολουθήθηκε. Στη συνέχεια, θα σχολιαστούν τα ληφθέντα αποτελέσματα.

Όλοι οι πειραματισμοί πραγματοποιήθηκαν σε ηλεκτρονικό υπολογιστή εξοπλισμένο με 4 CPU, 8 GB κύρια μνήμη και 512 GB τοπική αποθήκευση. Για τις δοκιμές χρησιμοποιήθηκε το Kafka 2.6.0, το Flink 1.11.2, το Spark 2.4.7, το Samza 1.5.1 και το Storm 1.2.3. Για το πειραματικό μας πρωτόκολλο, χρησιμοποιήθηκε το Twitter4J API2 για τη ροή των tweets που περιέχουν τη λέξη "COVID-19" σε πραγματικό χρόνο (Huang, 2020). Κάθε tweet αποτελείται από ένα αρχείο JSON με ένα σύνολο χαρακτηριστικών όπως το tweet id και το user id του χρήστη, την ημερομηνία δημιουργίας του tweet, και τα keywords του tweet. Το πειραματικό πρωτόκολλο συνίσταται στην εκτέλεση μιας ρουτίνας Extract, Transform και Load (ETL) που εξάγει tweets χρησιμοποιώντας Apache Kafka προκειμένου να εξασφαλίσει τον ίδιο ρυθμό ροής κατά την αξιολόγηση των frameworks που μελετώνται , μετατρέπει τα tweets διατηρώντας μόνο χαρακτηριστικά όπως το id tweet, το περιεχόμενο του tweet, την ημερομηνία και πληροφορίες χρήστη, και φορτώνει τα μετασχηματισμένα tweets στην ElasticSearch[16].

Σε αυτή τη διαδικασία μελετήθηκαν:

(1) ο αριθμός των μηνυμάτων που υποβάλλονται σε επεξεργασία από κάθε πλαίσιο σε μια δεδομένη περίοδο,

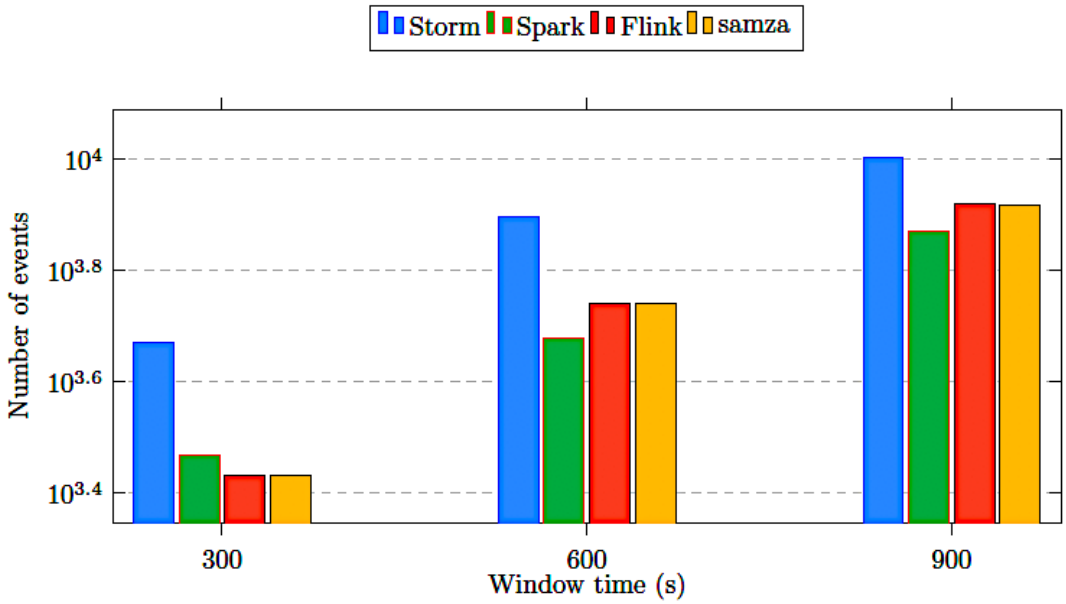
(2) το αντίκτυπο του μεγέθους του μηνύματος στον αριθμό των επεξεργασμένων μηνυμάτων και

(3) η κατανάλωση πόρων από τα διάφορα frameworks που εφαρμότηκαν.

[16] Η Elasticsearch, είναι μια μηχανή αναζήτησης πραγματικού χρόνου και ανάλυσης δεδομένων. Βασίζεται στο apache lucene, υψηλών επιδόσεων βιβλιοθήκη για μηχανές αναζήτησης.

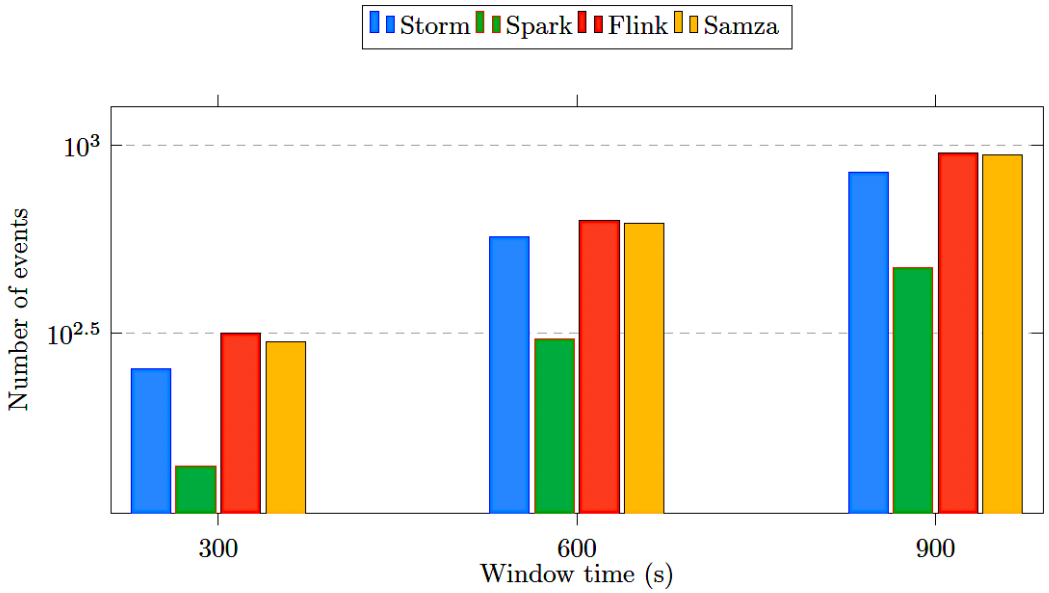
7.1 Αποτελέσματα

Το Γράφημα 1 δείχνει ότι τα Flink, Samza και Storm έχουν καλύτερους ρυθμούς επεξεργασίας σε σύγκριση με το Spark. Αυτό μπορεί να εξηγηθεί από το λανθάνοντα χρόνο. Στην πραγματικότητα, ο λανθάνων χρόνος του Spark κρατάει για δευτερόλεπτα ενώ στην περίπτωση των Flink, Samza και Storm διαρκεί κλάσματα δευτερολέπτων.



Γράφημα 1:Αντίκτυπο χρόνου παραθύρου στο χρόνο συμβάντων προς επεξεργασία (100 KB ανά μήνυμα)

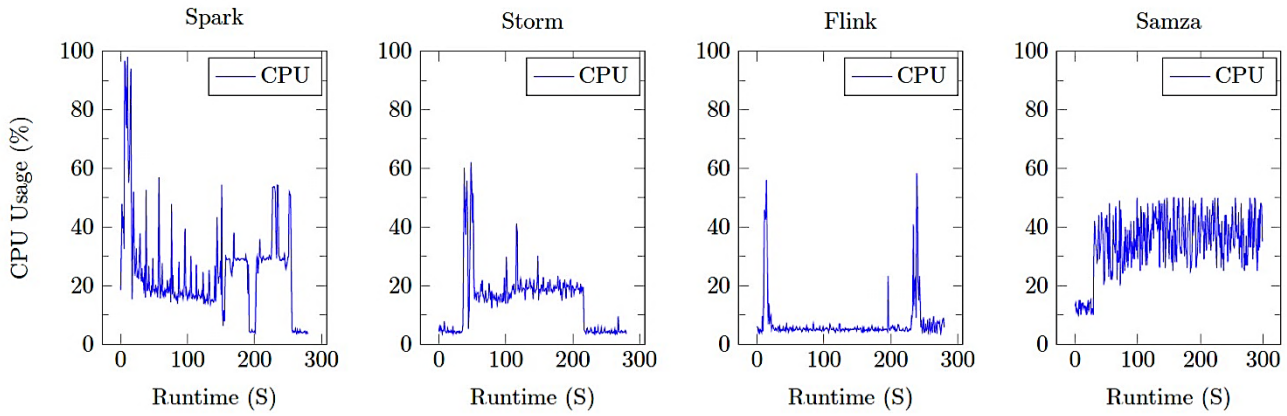
Στο επόμενο πείραμα, άλλαξε το μέγεθος των μηνυμάτων προς επεξεργασία. Χρησιμοποιήθηκαν 5 tweets ανά μήνυμα (περίπου 500 KB ανά μήνυμα). Τα αποτελέσματα παρουσιάζονται στο Γράφημα 2 και δείχνουν ότι τα Samza και Flink είναι πολύ αποδοτικά σε σύγκριση με το Spark, ειδικά για μεγάλα μηνύματα.



Γράφημα 2: Αντίκτυπο χρόνου παραθύρου στο χρόνο συμβάντων προς επεξεργασία (500 KB ανά μήνυμα)

Κατανάλωση CPU

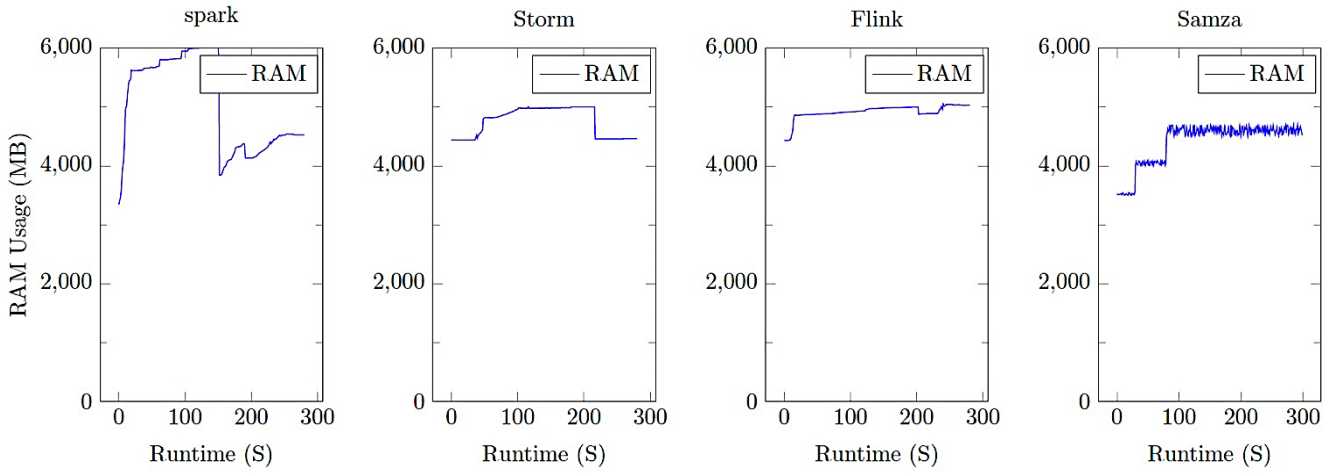
Όπως φαίνεται στο Γράφημα 3, η κατανάλωση CPU με το Flink είναι χαμηλή σε σύγκριση με τα Spark, Samza και Storm. Το Flink εκμεταλλεύεται περίπου το 10% της διαθέσιμης CPU, ενώ η χρήση της CPU με το Storm ποικίλλει μεταξύ 15% και 18%. Ωστόσο, το Flink μπορεί να προσφέρει καλύτερα αποτελέσματα από το Storm όταν οι πόροι της CPU αξιοποιούνται περισσότερο. Στη βιβλιογραφία αναφέρεται ότι το Flink έχει σχεδιαστεί για να επεξεργάζεται μεγάλα μηνύματα, σε αντίθεση με το Storm που μπορεί να αντιμετωπίσει μόνο μικρά μηνύματα (π.χ. μηνύματα που προέρχονται από αισθητήρες). Σε αντίθεση με τα Flink, Samza και Storm, το Spark συλλέγει δεύτερο τα δεδομένα των event και εκτελεί εργασίες επεξεργασίας μετά από αυτό. Ως εκ τούτου, τίθενται σε επεξεργασία περισσότερα από ένα μηνύματα, γεγονός που εξηγεί την υψηλή χρήση της CPU του Spark. Λόγω της φύσης του Flink, κάθε μήνυμα σχετίζεται με ένα νήμα και καταναλώνεται σε κάθε παράθυρο. Συνεπώς, ο χαμηλός όγκος δεδομένων που τίθενται σε επεξεργασία δεν επηρεάζει τη χρήση των πόρων της CPU. Το Samza από την άλλη εκμεταλλεύεται περίπου το 55% της διαθέσιμης CPU επειδή βασίζεται στην έννοια των εικονικών πυρήνων και κάθε εργασία ή partition αντιστοιχίζεται σε έναν αριθμό εικονικών πυρήνων. Στην πραγματικότητα, αναπτύσσει πολλά νήματα (ένα για κάθε διαμέρισμα), το οποίο εξηγεί την έντονη χρήση της CPU από το Samza σε σύγκριση με τα άλλα frameworks.



Γράφημα 3: Κατανάλωση CPU

Κατανάλωση μνήμης RAM

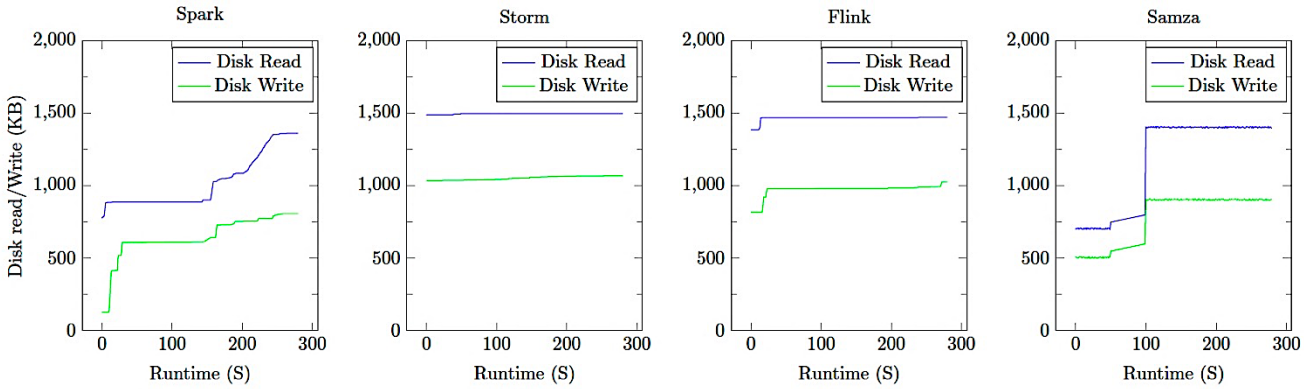
Το Γράφημα 4 δείχνει το κόστος της επεξεργασίας ροής συμβάντων όσον αφορά την κατανάλωση RAM. Το Spark έφτασε τα 6 GB (75% των διαθέσιμων πόρων) λόγω της ικανότητα εκτέλεσης σε μικρο-παρτίδες (επεξεργασία ενός γκρουπ μηνυμάτων κάθε φορά). Τα Flink, Samza και Storm δεν ξεπέρασαν τα 5 GB (περίπου το 61% της διαθέσιμης μνήμης RAM) καθώς η συμπεριφορά τους συνίσταται στην επεξεργασία μόνο μεμονωμένων μηνυμάτων. Το Spark από την άλλη διατήρησε μικρό αριθμό μηνυμάτων που επεξεργάστηκε.



Γράφημα 4: Κατανάλωση μνήμης RAM

Κατανάλωση δίσκου για εργασίες Read/Write

Το Γράφημα 5 απεικονίζει το ποσοστό χρήσης δίσκου από τα frameworks που μελετήθηκαν. Οι καμπύλες υποδηλώνουν το ποσό των εργασιών ανάγνωσης / εγγραφής. Η ποσότητα των εργασιών εγγραφής στο Flink και στο Storm είναι σχεδόν κοντά. Τα Flink, Samza και Storm έχουν συχνή πρόσβαση στο δίσκο και είναι πιο γρήγορα από το Spark όσον αφορά τον αριθμό των επεξεργασμένων μηνυμάτων. Όπως αναφέρθηκε και σε προηγούμενες ενότητες, το Spark είναι ένα framework μνήμης γεγονός που εξηγεί τη χαμηλότερη χρήση του δίσκου.



Γράφημα 5: Χρήση δίσκου για εργασίες Read / Write

Χρήση εύρους ζώνης (Bandwidth)

Όπως φαίνεται στο Γράφημα 6, η ποσότητα των δεδομένων που ανταλλάσσονται ανά δευτερόλεπτο κυμαίνεται μεταξύ 375 KB / s και 385 KB / s στην περίπτωση του Flink, για το Storm κυμαίνεται μεταξύ 387 KB / s και 390 KB / s. Στην περίπτωση του Samza είναι περίπου 400 Mb / s . Ο αριθμός αυτός είναι υψηλός σε σύγκριση με το Spark όπου η χρήση του εύρους ζώνης δεν ξεπέρασε τα 220 KB / s. Αυτό οφείλεται στη μείωση της συχνότητας της διαδικασίας σειριοποίησης και μετεγκατάστασης μεταξύ των cluster nodes, καθώς το Spark επεξεργάζεται μια ομάδα μηνυμάτων σε κάθε λειτουργία. Κατά συνέπεια, ο αριθμός των δεδομένων μειώνεται. Εδώ να υπενθυμίσουμε ότι τα Storm, Samza και Flink έχουν σχεδιαστεί για την επεξεργασία ροής.

7.2 . Συμπεράσματα

Με την αυξανόμενη ποσότητα δεδομένων που δημιουργούνται από δισεκατομμύρια συσκευές σε όλο τον κόσμο, η επεξεργασία ροής δεδομένων γίνεται βασική απαίτηση για frameworks μεγάλων ποσοτήτων δεδομένων. Ο κύριος στόχος της παρούσας διπλωματικής εργασία είναι να μελετήσει θεωρητικά και να αξιολογήσει πειραματικά τα πιο δημοφιλή frameworks για επεξεργασία ροής δεδομένων μεγάλης κλίμακας. Παρουσιάστηκαν τα Apache Spark, Kafka, Flink, Samza, Storm και η αρχιτεκτονική Lambda και κατηγοριοποιήθηκαν σύμφωνα με τα κύρια χαρακτηριστικά τους. Αξιολογήθηκε επίσης η απόδοση του κάθε framework, όσον αφορά την κατανάλωση πόρων. Η μελέτη των framework βασίστηκε σε συγκεκριμένα χαρακτηριστικά των συστημάτων επεξεργασίας ροής. Η εργασία μπορεί να επεκταθεί πραγματοποιώντας περισσότερα πειράματα σχετικά με τη συχνότητα και το μέγεθος των δεδομένων εισερχόμενων συμβάντων.

Βιβλιογραφία

- Apache Software Foundation, 2019. *Trident Tutorial.* [Ηλεκτρονικό] Available at: https://storm.apache.org/releases/current/Trident-tutorial.html [Πρόσβαση 27 9 2020].
- Abbasi, M. A., 2017. *Learning Apache Spark 2.* Birmingham: Packt Publishing.
- Altti Ilari Maarala, M. R. M. S. S. P. a. J. R., 2015. *Low latency analytics for streaming traffic data with Apache Spark.* s.l., s.n., pp. 2855-2858.
- Apache, 2020. *Apache Storm version 2.4.5 documentation.* [Ηλεκτρονικό] Available at: https://spark.apache.org/docs/latest/streaming-programming-guide.html#deploying-applications [Πρόσβαση 21 5 2020].
- Bersimis, M., 2020. *Master Of Thesis A Framework for Public Health Monitoring based on Statistical Process Monitoring Techniques.* Piraeus, Athens: University Of Piraeus.
- Bijnens, H. M. a. N., 2017. *Lambda Architecture.* [Ηλεκτρονικό] Available at: http://lambda-architecture.net/ [Πρόσβαση 22 5 2020].
- Doblander C, R. T. J. H., 2014. *Processing big events with showers and streams.Specifying big data benchmarks.* Berlin/Heidelberg: Springer.
- Flink Apache Org, 2019. *Flink Windows.* [Ηλεκτρονικό] Available at: https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/operators/windows.html [Πρόσβαση 26 9 2020].
- Foundation, T. A. S., 2019. *What is Apache Flink? — Applications.* [Ηλεκτρονικό] Available at: https://flink.apache.org/flink-applications.html [Πρόσβαση 12 7 2020].
- Group, A. T., 2019. *Apache Kafka.* [Ηλεκτρονικό] Available at: https://kafka.apache.org/intro [Πρόσβαση 21 3 2020].
- Hausenblas, M. a. N. B., 2017. *Lambda Architecture.* [Ηλεκτρονικό] Available at: lambda-architecture.net [Πρόσβαση 21 6 2020].
- Huang, X. J. A. B. D. Q. S. &. D. M., 2020. *Coronavirus Twitter Data: A collection of COVID-19 tweets with automated annotations.* [Online] Available at: https://zenodo.org/record/4136656#.X5k-oi81u_E [Accessed 13 9 2020].
- Insight Team, 2016. *Medium.* [Ηλεκτρονικό] Available at: https://blog.insightdatascience.com/implementing-lambda-architecture-to-track-real-time-updates-f99f03e0c53 [Πρόσβαση 21 5 2020].

- Jafarpour, H. R. D. a. D. G., 2019. *KSQL: Streaming SQL Engine for Apache Kafka.* s.l., EDBT.
- Jeffrey Dean and Sanjay Ghemawat, Google Inc., χ.χ. *MapReduce: Simplified Data Processing on Large Clusters.* s.l., USENIX Association.
- Kartik Paramasivam, LinkedIn, 2016. *Stream processing with Apache Samza - Current and Future,* s.l.: s.n.
- Kato, K. e. a., 2019. *"Construction Scheme of a Scalable Distributed Stream Processing Infrastructure Using Ray and Apache Kafka.".* s.l., s.n., pp. 368-377.
- Kleppmann M, K. J., 2015. *Kafka, Samza and the Unix philosophy of distributed data.* s.l., IEEE Data Eng Bull 38(4):4–14.
- Kreps, J., 2014. *O'REILLY.* [Ηλεκτρονικό]
  Available at: https://www.oreilly.com/radar/questioning-the-lambda-architecture/
  [Πρόσβαση 22 5 2020].
- Kreps, M. K. a. J., 2015. *Kafka , Samza and the Unix Philosophy of Distributed Data.* s.l., s.n.
- Leang, B. e. a., 2019. *"Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment.".* s.l.:s.n.
- M.Kleppmann, 2018. *Apache Samza, Encycl. Big Data Technol..* s.l.:Springer, Cham.
- Maarala, A. I. R. M. S. M. P. S. &. R. J., 2015. *Low Latency Analytics for Streaming Traffic Data with Apache Spark, IEEE International Conference on Big Data.* s.l., s.n., pp. 2855-2858.
- McDonald, C., 2015. *Spark Streaming with HBase.* [Ηλεκτρονικό]
  Available at: https://mapr.com/blog/spark-streaming-hbase/
  [Πρόσβαση 31 5 2020].
- Mian, S. Z., February 23, 2020. *Comparative Analysis of Data Stream Processing Systems - Master's Thesis in Information Technology.* Jyväskylä: University of Jyväskylä.
- Milinda Pathirage, J. H. Y. P. B. P., 2016. *SamzaSQL: Scalable Fast Data Management with Streaming SQL.* Chicago, IL, USA , IEEE.
- Narkhede, N. a. S. G. a. P. T., 2017. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale.* s.l.:O'Reilly Media, Inc.
- Neha Narkhede, G. S. a. T. P., 2017. *Kafka: The Definitive Guide by Neha Narkhede, Gwen Shapira, and Todd Palino (O'Reilly).* s.l.:O'Reilly.
- Neha Narkhede, G. S. a. T. P., 2017. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale.* 1η έκδοση επιμ. s.l.:O'Reilly Media.
- Perera, S., 2018. . *"13 Stream Processing Patterns for Building Streaming and Realtime Applications."My Views of the World and Systems.* [Online]
  Available at: iwringer.wordpress.com/2015/08/03/patterns-for-streaming-realtime-analytics/
  [Accessed 4 5 2020].

- Perera, S., 2018. *Medium.* [Ηλεκτρονικό]
  Available at: https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97
  [Πρόσβαση 5 4 2020].
- RabindraLamsal - School of Computer and Systems Sciences, J., 2020. *CORONAVIRUS (COVID-19) TWEETS DATASET.* [Online]
  Available at: http://ieee-dataport.org/open-access/coronavirus-covid-19-tweets-dataset
  [Accessed 7 Σεπτέμβριος 2020].
- Riccomini, C., 2013. *Apache Samza: LinkedIn's Real-time Stream Processing Framework.* [Ηλεκτρονικό]
  Available at: https://engineering.linkedin.com/datastreams/apache-samza-linkedins-real-time-stream-processing-framework
  [Πρόσβαση 12 7 2020].
- Samza Apache Org, 2019. *Samza - Core Concepts.* [Ηλεκτρονικό]
  Available at: https://samza.apache.org/learn/documentation/latest/core-concepts/core-concepts.html
  [Πρόσβαση 26 9 2020].
- samza.apache.org, 2019. *Samza SQL.* [Ηλεκτρονικό]
  Available at: https://samza.apache.org/learn/documentation/latest/api/samza-sql.html
  [Πρόσβαση 26 09 2020].
- samza.apache.org, 2019. *Samza Windowing.* [Ηλεκτρονικό]
  Available at: https://samza.apache.org/learn/documentation/0.7.0/container/windowing.html
  [Πρόσβαση 26 9 2020].
- Shadi A. Noghabi, S. A. N. K. P. K. P. Y. P. N. R. N. R. J. B. J. B. I. S. G., 2017. Samza: stateful scalable stream processing at LinkedIn.
- Sherif Sakr, A. Y. Z., 2019. *Alignment Creation - Encyclopedia of Big Data Technologies,* s.l.: Springer, Cham..
- Shirshanka Das, C. B. ,. K. S. B. G. ,. B. V. ,. S. N. D. Z. L. G. J. W. P. G. B. S. S. T. A. P., 2012. *All aboard the Databus!: Linkedin's scalable consistent change data capture platform.* s.l., SoCC '12: Proceedings of the Third ACM Symposium on Cloud Computing.
- Software Foundation, Apache, 2019. *Windowing Support in Core Storm.* [Online]
  Available at: http://storm.apache.org/releases/2.1.0/Windowing.html
  [Accessed 24 9 2020].
- Spark, Apache, 2019. *Spark SQL Guide.* [Online]
  Available at: https://spark.apache.org/docs/latest/sql-programming-guide.html
  [Accessed 28 5 2020].
- Tao Feng, LinkedIn, 2015. *Benchmarking Apache Samza: 1.2 million messages per second on a single node,* s.l.: s.n.
- Team, D., 2018. *Data-Flair.* [Ηλεκτρονικό]
  Available at: https://data-flair.training/blogs/apache-spark-streaming-

transformation-operations/
[Πρόσβαση 10 5 2020].

- The Apache Software Foundation, 2019. *Apache Flink.* [Ηλεκτρονικό]
  Available at: https://flink.apache.org/flink-applications.html
  [Πρόσβαση 11 9 2020].
- The Apache Software Foundation, 2019. *What is Apache Flink? — Architecture.* [Ηλεκτρονικό]
  Available at: https://flink.apache.org/flink-architecture.html
  [Πρόσβαση 31 8 2020].
- Wachal, M., 2020. *What is Apache Kafka and what are Kafka use cases?.* [Ηλεκτρονικό]
  Available at: https://blog.softwaremill.com/what-is-apache-kafka-and-what-are-kafka-use-cases-871666dd4eed
  [Πρόσβαση 4 5 2020].
- Warren, N. M. w. J., 2015. *Big Data : Principles and Best Practices of Scalable Real-Time Data Systems.* s.l.:Manning Publications.
- Wissem Inoubli, S. A. H. M. M. M. E. N., 2018. *A Comparative Study on Streaming Frameworks for Big Data. VLDB 2018 - 44th International Conference on Very Large Data Bases : Workshop LADaS - Latin American Data Science.* Rio de Janeiro, Brazil, s.n.
- WSO2, 2019. *"Stream Processing 101: From SQL to Streaming SQL in 10 Minutes.".* [Ηλεκτρονικό]
  Available at: wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-tenminutes/.
  [Πρόσβαση 6 7 2020].
- Xuechun Ji, M. Z. ,. M. Z. a. Q. W., 2020. Query Execution Optimization in Spark SQL. *Hindawi,* 7 2.p. 12.