Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Μοντελοποίηση επίδοσης συστημάτων κατανεμημένης επεξεργασίας ροών δεδομένων

Διπλωματική εργασία

Γεώργιος Κ. Παπαμαργαρίτης

Επιβλέπων καθηγητής: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2019

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Μοντελοποίηση επίδοσης συστημάτων κατανεμημένης επεξεργασίας ροών δεδομένων

Διπλωματική εργασία

Γεώργιος Κ. Παπαμαργαρίτης

Επιβλέπων καθηγητής: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Νοεμβρίου 2019.

------------------------	------------------------	------------------------
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Επίκουρος Καθηγητής	Δημήτριος Τσουμάκος Αναπληρωτής Καθηγητής Ιονίου Πανεπιστημίου

Αθήνα, Νοέμβριος 2019

....
Γεώργιος Κ. Παπαμαργαρίτης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Γεώργιος Κ. Παπαμαργαρίτης

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

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

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

// Περίληψη

Στην παρούσα διπλωματική εργασία μελετάται η λειτουργία και η επιδόση συστημάτων κατανεμημένης επεξεργασίας ροών δεδομένων και γίνεται μια πρώτη προσπάθεια για τη μοντελοποίηση αυτών. Συγκεκριμένα, σχεδιάζεται και υλοποιείται μέσω του Flink framework ένα cluster στο οποίο εισέρχεται μεγάλος όγκος μηνυμάτων σε πραγματικό χρόνο από ένα κατανεμημένο σύστημα μεταφοράς μηνυματών, το Kafka cluster. Τα μηνύματα αυτά δημιουργούνται με τυχαίο τρόπο από γεννήτριες, ο ρυθμός παραγωγής των οποίων είναι ελεγχόμενος. Για να μπορέσουμε να προσομοιώσουμε υπολογισμούς σε ροές δεδομένων που μπορούν να ταυτιστούν με περιπτώσεις του πραγματικού κόσμου επιλέχθηκαν 3 είδη εργασιών προς εξέταση. Αυτές είναι οι Filtering, Aggregation on Windows και Window Joins εργασίες.

Έχοντας υλοποιήσει τα παραπάνω, στη συνέχεια εκτελείται μια σειρά πειραμάτων διαφοροποιώντας κάθε φορά τις συνθήκες και τις παραμέτρους της κάθε εκτέλεσης. Οι παράμετροι αυτές αφορούν τόσο την εσωτερική λειτουργία του ίδιου του Flink και τους πόρους του συστήματος όσο και το είδος των δεδομένων εισόδου αλλά και της επιλεγμένης εργασίας. Με τον τρόπο αυτό λαμβάνονται υπόψη όλοι οι παράγοντες που μπορούν να επηρεάσουν την επίδοση του συστήματος κατανεμημένης επεξεργασίας ροής δεδομένων.

Με τα σετ δεδομένων που δημιουργούνται από την πειραματική διαδικασία επιδιώκουμε τη δημιουργία μοντέλων για την πρόβλεψη τόσο του μέγιστου ρυθμού επεξεργασίας δεδομένων του συστήματος όσο και των καθυστερήσεων που προκύπτουν με το ρυθμό αυτό. Ο στόχος της έρευνας μας επιτυγχάνεται με την αξιολόγηση και την επιλογή των καλύτερων μοντέλων για κάθε μία από τις διαφορετικές εργασίες προς εκτέλεση.

Λέξεις κλειδιά

Σύστημα κατανεμημένης επεξεργασίας ροών δεδομένων, κατανεμημένο σύστημα μεταφοράς μηνυμάτων, Flink cluster, Kafka cluster, Producers, Filtering, Aggregation on Windows, Window Joins, Maximum Sustainable Throughput, Event time Latency, Processing time Latency, μοντέλα μηχανικής μάθησης, παλινδρόμηση

Abstract

This diploma thesis examines the operation of distributed stream processing systems and attempts to understand their functionality and create a model that describes their performance. Specifically, Flink framework is used to design and implement a cluster that receives a large amount of data in real time coming from a distributed message system, a Kafka cluster. Those data volumes are generated randomly by machines acting as producers whose production rate can be adjusted per case. In order to simulate data stream calculations in a distributed environment that match real world scenarios, 3 types of data processing jobs were selected as the most common for the experiments. These are Filtering jobs, Aggregation on Windows jobs and Window Joins jobs.

Having implemented all of the above, a series of experiments are then performed each time choosing different parameters and conditions for every job execution. These parameters have to do with the internal Flink operation and the system resources as well as the type of data input and executed job. This way all factors that may affect the performance of the distributed stream processing system are taken into account.

With the created data sets from the experimental procedure we aim to use machine learning algorithms to create models that are able to predict the maximum sustainable throughput of the system as well as the latencies observed at this processing rate. The goal of this thesis is achieved by evaluating and selecting the best models for each of the different jobs that are executed in the distributed stream processing system.

Keywords

Distirbuted stream processing systems, distributed message system, Flink cluster, Kafka cluster, Producers, Filtering, Aggregation on Windows, Window Joins, Maximum Sustainable Throughput, Event time Latency, Processing time Latency, machine learning models, regression

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας, αρχικά θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου Νεκτάριο Κοζύρη για τη δυνατότητα που μου έδωσε με αυτό το θέμα να ασχοληθώ σε βάθος με τον τομέα των κατανεμημένων συστημάτων.

Ιδιαίτερα θα ήθελα να ευχαριστήσω την Δρ. Κατερίνα Δόκα για την βοήθεια και την καθοδήγηση της κατά την εκπόνηση της παρούσας εργασίας.

Ακόμα θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου για την αμέριστη στήριξη και βοήθεια όλα αυτά τα χρόνια.

Τέλος ευχαριστώ τους φίλους μου εντός και εκτός σχολής για τα όμορφα αυτά φοιτητικά χρόνια.

Περιεχόμενα

Περίληψη(σελ 6)

Abstract(σελ 8)

Ευχαριστίες(σελ 10)

Περιεχόμενα(σελ 12)

1 Πρόλογος
- 1.1 Δομή εργασίας(σελ 17)

2 Θεωρητικό Υπόβαθρο
- 2.1 Συστήματα κατανεμημένης επεξεργασίας ροών δεδομένων .
  - 2.1.1 Ανάγκη και εφαρμογή στις μέρες μας(σελ 19)
  - 2.1.2 Στόχοι(σελ 20)
- 2.2 Stream-First Αρχιτεκτονική
  - 2.2.1 Παραδοσιακή Αρχιτεκτονική vs Streaming Αρχιτεκτονική(σελ 21)
  - 2.2.2 Επίπεδο μεταφοράς μηνύματος(σελ 22)
  - 2.2.3 Επίπεδο επεξεργασίας μηνύματος(σελ 24)
- 2.3 Χειρισμός του χρόνου
  - 2.3.1 Έννοιες του χρόνου(σελ 25)
  - 2.3.2 Windows(σελ 25)
  - 2.3.3 Watermarks(σελ 27)
- 2.4 Stateful υπολογισμοί
  - 2.4.1 Η έννοια του Consistency(σελ 28)
  - 2.4.2 End-to-end Consistency(σελ 29)

3 Τεχνικό Υπόβαθρο
- 3.1 Flink
  - 3.1.1 Κατανεμημένο Περιβάλλον Εκτέλεσης Υπολογισμών(σελ 30)
  - 3.1.2 Δρομολόγηση εργασιών(σελ 32)
  - 3.1.3 Fault Tolerance (Checkpointing)(σελ 33)

3.2 Kafka
3.2.1 Kafka Brokers και Zookeeper(σελ 36)
3.2.2 Kafka Producer(σελ 37)
3.2.3 Kafka Consumer(σελ 39)
3.2.4 Αξιολόγηση του Apache Kafka(σελ 40)

4 Σχεδίαση του συστήματος
4.1 Εισαγωγή(σελ 45)
4.2 Δομικά στοιχεία
4.2.1 Στήσιμο των Producers(σελ 45)
4.2.2 Στήσιμο του Kafka cluster(σελ 47)
4.2.3 Στήσιμο του Flink cluster(σελ 48)
4.3 Flink Jobs
4.3.1 Filtering(σελ 48)
4.3.2 Aggregations on Windows(σελ 49)
4.3.3 Window joins(σελ 50)
4.4 Μετρικές
4.4.1 Υπολογισμός Maximum Sustainable Throughput(MST)(σελ 51)
4.4.2 Υπολογισμός Event-time Latencies και Processing-time Latencies(σελ 54)

5 Πειραματική Μελέτη
5.1 Επεξήγηση της πειραματικής αξιολόγησης(σελ 57)
5.2 Επιλογή των παραμέτρων
5.2.1 Παράμετροι σχετικές με τη λειτουργία του Flink Cluster (σελ 59)
5.2.2 Παράμετροι σχετικές με το είδος των δεδομένων εισόδου(σελ 60)
5.2.3 Παράμετροι σχετικές με την υποβληθείσα εργασία(σελ 60)
5.3 Πειραματικά αποτελέσματα
5.3.1 MST-Filtering(σελ 61)
5.3.2 Latencies-Filtering(σελ 62)
5.3.3 MST-Aggregation on Windows(σελ 63)
5.3.4 Latencies-Aggregation on Windows(σελ 65)
5.3.5 MST-Window Joins(σελ 67)
5.3.6 Latencies-Window Joins(σελ 68)
5.4 Χρήσιμα συμπεράσματα από την πειραματική διαδικασία(σελ 70)

6 Μοντελοποίηση
6.1 Μετρικές αξιολόγησης
6.1.1 Μέσο Απόλυτο Σφάλμα (MAE)(σελ 74)
6.1.2 Μέσο Απόλυτο Ποσοστιαίο Σφάλμα (MAPE) (σελ 75)
6.1.3 Μέσο Τετραγωνικό Σφάλμα (MSE)(σελ 75)

6.2 Χρήση αλγορίθμων παλινδρόμησης για την εκτίμηση του MST και των Latencies
6.2.1 Πολλαπλή γραμμική παλινδρόμηση (Multiple linear regression)(σελ 76)
6.2.2 Πολλαπλή πολυωνυμική παλινδρόμηση δευτέρου βαθμού (Polynomial regression)(σελ 77)
6.2.3 Παλινδρόμηση με Δέντρα Απόφασης (Decision Tree Regression)(σελ 78)
6.2.4 Παλινδρόμηση με Τυχαία Δάση (Random Forest Regression)(σελ 79)
6.2.5 Παλινδρόμηση με Xgboost(σελ 80)
6.3 Αξιολόγηση Αλγορίθμων(σελ 82)

7 Επίλογος
7.1 Σύνοψη και Συμπεράσματα(σελ 84)
7.2 Μελλοντικές επεκτάσεις(σελ 85)

Βιβλιογραφία(σελ 86)

Κεφάλαιο 1

Πρόλογος

Τα τελευταία χρόνια παρατηρείται μια συνεχόμενη αύξηση του πλήθους των δεδομένων τα οποία παράγονται και συγκεντρώνονται από πολλών ειδών συσκευές, όπως υπολογιστές, κινητά, κάμερες ασφαλείας, μικρόφωνα, αισθητήρες κλπ. Είναι ενδεικτικό το γεγονός ότι μόνο τα τελευταία δύο χρόνια έχει παραχθεί το 90% του συνολικού όγκου των δεδομένων της γης, ενώ σύμφωνα με μια αναφορά της IDC (International Data Corporation) προβλέπεται ότι το 2025 ο συνολικός όγκος των δεδομένων θα φτάσει τα 175 ZettaBytes από 33 ZettaBytes το 2018 [1]. Αυτό οφείλεται σε μεγάλο ποσοστό στην επιθυμία του ανθρώπου να ψηφιοποιήσει τον κόσμο. Ένας άνθρωπος που πλέον έχει στόχο να παρατηρεί, να βλέπει, να ακούει, να καταγράφει όσα συμβαίνουν γύρω του, να τα επεξεργάζεται και να τα χρησιμοποιεί για να βελτιώσει πτυχές τόσο της επαγγελματικής όσο και της προσωπικής του ζωής. Χαρακτηριστικό παράδειγμα αποτελούν επιχειρήσεις που με την διαρκή λήψη δεδομένων (αγορές, πωλήσεις κλπ) και καταγραφή των προτιμήσεων των πελατών τους μπορούν πλέον να προσφέρουν εξατομικευμένες υπηρεσίες και προϊόντα, ενώ είναι σε θέση να προβλέψουν μελλοντικές καταστάσεις και να οργανώσουν καλύτερα το στρατηγικό τους πλάνο. Επίσης, με την εξέλιξη της ιατρικής δίνεται η δυνατότητα για την καλύτερη παρακολούθηση του ανθρώπινου οργανισμού και την αποκρυπτογράφηση του ανθρώπινου εγκεφάλου με αποτέλεσμα να δημιουργούνται νέες προοπτικές για την αντιμετώπιση σπάνιων παθήσεων που παλαιότερα φάνταζαν αδύνατο να θεραπευτούν.

Για να γίνει επομένως εφικτή η λήψη και καταγραφή δεδομένων συγκεκριμένου σκοπού δημιουργήθηκε πληθώρα ετερογενών συσκευών έχοντας ως αποτέλεσμα την μετάβαση στην εποχή του Internet of Things και των Big Data. Με τον όρο Big Data αναφερόμαστε σε μη δομημένα σετ δεδομένων που ο όγκος τους και η πολυπλοκότητα τους καθιστούν δύσκολη τη διαχείριση και την επεξεργασία τους με τα παραδοσιακά εργαλεία λογισμικού που χρησιμοποιούνταν τα προηγούμενα χρόνια. Δημιουργήθηκε λοιπόν η ανάγκη για την ανάπτυξη νέων πακέτων λογισμικού και συστημάτων που θα μπορούν να χειριστούν αυτό το μέγεθος των δεδομένων, να τα διαμοιράσουν στα διαθέσιμα resources (δηλαδή σε κατανεμημένα περιβάλλοντα), να εκτελέσουν υπολογισμούς πάνω σε αυτά και να φροντίσουν για την αποτελεσματική αποθήκευση και συντήρηση τους. Πέρα από τα εργαλεία αυτά, προέκυψε και η ανάγκη για τη δημιουργία μαθηματικών μοντέλων που θα μπορούν να εξάγουν από τον όγκο των δεδομένων αυτών την χρήσιμη πληροφορία και να την εκμεταλλευτούν για να εντοπίσουν μοτίβα και να κάνουν προβλέψεις με αποδεκτά ποσοστά σφάλματος. Τα μοντέλα αυτά άνοιξαν τον δρόμο για την εισαγωγή και εφαρμογή αλγορίθμων μηχανικής μάθησης από όλα τα μέλη του επιχειρηματικού κόσμου που διαχειρίζονται δεδομένα και δημιούργησαν αμέτρητες νέες προοπτικές για την εξέλιξη τους και την αντιμετώπιση δυσκολιών.

Λαμβάνοντας υπόψη όλα τα παραπάνω, στόχος της διπλωματικής αυτής είναι να εξεταστεί σε βάθος η λειτουργία συστημάτων που δίνουν τη δυνατότητα για τη

διενέργεια υπολογισμών πάνω σε ροές μεγάλου όγκου δεδομένων, ένα από τα σημαντικότερα ζητούμενα της ψηφιακής εποχής. Συγκεκριμένα, θα μελετηθεί το Apache Flink framework που υπόσχεται κατανεμημένη επεξεργασία ροών δεδομένων με χαμηλό latency και υψηλό throughput, προσφέροντας παράλληλα μηχανισμούς ανάνηψης από τυχόν σφάλματα και εξασφαλίζοντας ότι οι υπολογισμοί πάνω στα δεδομένα θα οδηγήσουν σε σωστά αποτελέσματα. Έγινε προσπάθεια να δοκιμαστεί το σύστημα σε συνθήκες που μπορούν να ταυτιστούν με πραγματικές περιπτώσεις του έξω κόσμου, δηλαδή ο όγκος των δεδομένων που δέχεται ως είσοδο το σύστημα να είναι αρκετά μεγάλος και να προέρχεται από διαφορετικές πηγές, ενώ για τη σωστή αξιολόγηση δόθηκε έμφαση στην αποτροπή παραγόντων που μπορούν να επηρεάσουν την απόδοση του ή να το μπλοκάρουν, όπως η χρήση εξωτερικών βάσεων δεδομένων για την αποθήκευση των αποτελεσμάτων. Πέρα από την παραπάνω μελέτη, απώτερος σκοπός ήταν η εξαγωγή συμπερασμάτων για τις επιπτώσεις εσωτερικών παραμέτρων του Apache Flink, του μεγέθους του cluster, του είδους της εργασίας που εκτελείται και των δεδομένων εισόδου στην απόδοση (throughput) και στις καθυστερήσεις των υπολογισμών (latencies) του συστήματος. Με βάση τα συμπεράσματα αυτά, θα μπορέσουν να δημιουργηθούν μοντέλα μηχανικής μάθησης που θα έχουν την ικανότητα να κάνουν προβλέψεις για εργασίες πάνω σε ροές δεδομένων που εκτελούνται σε Flink clusters. Οι προβλέψεις αυτές, αν είναι ακριβείς, καταλαβαίνει κανείς ότι μπορούν να βοηθήσουν σημαντικά στο σχεδιασμό εφαρμογών, με την επιλογή των κατάλληλων παραμέτρων και τη διανομή των resources, ώστε να επιτευχθούν τα επιθυμητά αποτελέσματα απόδοσης και ταχύτητας των υπολογισμών.

1.1 Δομή Εργασίας

Το υπόλοιπο της παρούσας διπλωματικής είναι οργανωμένο ως εξής:

Στο κεφάλαιο 2 παρέχεται το θεωρητικό υπόβαθρο που είναι απαραίτητο ώστε ο αναγνώστης να εξοικειωθεί με όρους και λειτουργικότητες που χρησιμοποιούνται στα επόμενα κεφάλαια. Συγκεκριμένα, γίνεται μια εισαγωγή στα συστήματα κατανεμημένης επεξεργασίας ροών δεδομένων και επεξηγούνται αναλυτικά η αρχιτεκτονική τους και οι αρχές που τα διέπουν.

Στο κεφάλαιο 3 γίνεται επαφή με το τεχνικό υπόβαθρο της εργασίας, δηλαδή με τα frameworks που χρησιμοποιήθηκαν για τη διεξαγωγή των πειραμάτων. Εδώ περιγράφονται λεπτομερώς τα βασικά χαρακτηριστικά των Apache Flink και Apache Kafka και αναλύονται εκτενώς ο τρόπος λειτουργίας τους και οι δυνατότητες που προσφέρουν στο χρήστη.

Στο κεφάλαιο 4 γίνεται η σχεδίαση των υπό εξέταση συστημάτων και η προετοιμασία για τη διεξαγωγή των πειραμάτων. Πιο συγκεκριμένα, το κεφάλαιο αυτό αναλώνεται στο στήσιμο των Producers (δηλαδή των μηχανημάτων που θα παράγουν μεγάλο όγκο τυχαίων δεδομένων σε μορφή μηνυμάτων), του Kafka Cluster (για την αποθήκευση των μηνυμάτων σε ουρές και το διαμοιρασμό τους) και του Flink Cluster (για την επεξεργασία των μηνυμάτων) ενώ παρουσιάζονται και οι εργασίες (jobs) πάνω στις οποίες θα εξεταστούν τα συστήματα. Τέλος, γίνεται αναφορά στη μεθοδολογία που

εφαρμόστηκε για την καταγραφή των μετρήσεων και στο monitoring σύστημα που χρησιμοποιήθηκε.

Στο κεφάλαιο 5 παρουσιάζονται τα αποτελέσματα των πειραμάτων. Κύριος στόχος του κεφαλαίου αυτού πάντως είναι να δημιουργηθούν τα σετ δεδομένων που θα χρησιμοποιηθούν στη συνέχεια για την εφαρμογή μοντέλων μηχανικής μάθησης.

Στο κεφάλαιο 6 γίνεται αναφορά και επεξήγηση των μοντέλων μηχανικής μάθησης που θα εφαρμοστούν πάνω στα πειραματικά αποτελέσματα. Εδώ επιτυγχάνεται και ο βασικός στόχος της διπλωματικής αυτής με την επιλογή του κατάλληλου μοντέλου που μπορεί να δώσει ακριβείς προβλέψεις για τις καθυστερήσεις (Latencies) και το μέγιστο throughput (Maximum Sustainable Throughput) που μπορεί να αντέξει ένα Flink Cluster, το οποίο εκτελεί υπολογισμούς πάνω σε κάποια ροή δεδομένων, σε συγκεκριμένες συνθήκες και με συγκεκριμένες παραμέτρους.

Στο κεφάλαιο 7, παρουσιάζονται τα τελικά συμπεράσματα που προκύπτουν από την διπλωματική εργασία και προτείνονται μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Συστήματα κατανεμημένης επεξεργασίας ροών δεδομένων

Όπως έχει αναφερθεί ήδη στην εισαγωγή, στην εποχή μας υπάρχει ένα αυξανόμενο πλήθος εφαρμογών από τις οποίες τεράστιοι όγκοι δεδομένων προωθούνται σε servers για επεξεργασία σε πραγματικό χρόνο. Οι εφαρμογές αυτές περιλαμβάνουν τη λήψη μετρήσεων από πολλών ειδών αισθητήρες, συναλλαγές στο χρηματιστήριο, παρακολούθηση της κίνησης στους δρόμους αλλά και της διαδικτυακής κίνησης. Γίνεται εύκολα αντιληπτό, από την φύση των παραπάνω εφαρμογών, ότι τα δεδομένα που προκύπτουν από αυτές έχουν τη μορφή μιας ροής από γεγονότα που μπορούν να επεξεργαστούν σαν tuples. Το μέγεθος ωστόσο των δεδομένων αυτών δεν επιτρέπει την επεξεργασία σε πραγματικό χρόνο από τα παραδοσιακά κεντρικοποιημένα συστήματα. Δημιουργήθηκε λοιπόν μια νέα κατηγορία συστημάτων γνωστά ως συστήματα κατανεμημένης επεξεργασίας δεδομένων ή αλλιώς distributed stream processing systems (DSPS).

2.1.1 Ανάγκη και εφαρμογή στις μέρες μας

Η κατανεμημένη επεξεργασία ροών δεδομένων βρίσκει εφαρμογή σε όλο και μεγαλύτερο φάσμα της επιχειρηματικής ζωής αφού προσφέρει νέες δυνατότητες και οδηγεί στην ευκολότερη επίτευξη επιχειρηματικών στόχων. Οι βασικότεροι τομείς πάνω στους οποίους είναι πλέον αναγκαία είναι οι παρακάτω:

➔ **Εμπόριο και Μάρκετινγκ:**

Στη σημερινή εποχή το μεγαλύτερο ποσοστό των αγορών δεν γίνεται πλέον με τον παραδοσιακό τρόπο αλλά με κλικς σε διαδικτυακές πλατφόρμες των εκάστοτε επιχειρήσεων. Αυτό σημαίνει ότι δεδομένα που αφορούν αγορές προϊόντων και υπηρεσιών μπορεί να αρχίσουν να καταφθάνουν με μεγάλους ρυθμούς και να απαιτούν ιδιαίτερο χειρισμό για την εξαγωγή συμπερασμάτων σχετικά με τις πωλήσεις της επιχείρησης. Μάλιστα τα συμπεράσματα αυτά στη συνέχεια θα μπορούσαν να χρησιμοποιηθούν για την χρέωση των διαφημίσεων στην πλατφόρμα, αλλά και την προσφορά εξατομικευμένων πακέτων στους πελάτες. Στο παρελθόν κάτι τέτοιο θα απαιτούσε την αποθήκευση του συνόλου των δεδομένων και έπειτα θα γινόταν η εξαγωγή της χρήσιμης πληροφορίας. Αυτό ωστόσο μπορεί να αποδειχθεί δαπανηρό σε hardware (τεράστιες βάσεις δεδομένων) , είναι αδύνατο να εφαρμοστεί σε περιπτώσεις που θέλουν αποτελέσματα σε σχεδόν πραγματικό χρόνο, ενώ αυξάνεται και ο κίνδυνος

να χαθούν χρήσιμα δεδομένα. Όλα τα παραπάνω μπορούν πλέον να αντιμετωπιστούν από διαφορετική σκοπιά με τη χρήση συστημάτων που αντιμετωπίζουν την πληροφορία σαν ροή δεδομένων και προσπαθούν να εξάγουν το χρήσιμο κομμάτι αυτής αποδοτικά και σε πραγματικό χρόνο.

➔ **<u>The Internet of Things</u>:**

Το IoT είναι ίσως ο πιο κρίσιμος τομέας που απαιτεί την επεξεργασία ροών δεδομένων. Εδώ ο χρόνος της εκτέλεσης των υπολογισμών αλλά και η εγκυρότητα αυτών αποτελούν τους σημαντικότερους παράγοντες για την επιτυχή λειτουργία των εφαρμογών. Οι πληροφορίες που καταγράφουν αισθητήρες ποικίλων ειδών είναι απαραίτητο τις περισσότερες φορές να αναλυθούν άμεσα για τη λήψη κάποιας απόφασης, την ανανέωση πινάκων ελέγχου και την ειδοποίηση σε περίπτωση κινδύνων. Χαρακτηριστικό παράδειγμα αποτελούν συστήματα για την παρακολούθηση δρομολογίων σε σταθμούς τρένων που καταγράφουν πληροφορίες σχετικά με την ταχύτητα των τρένων, την τοποθεσία τους και τις συνθήκες του καιρού και προειδοποιούν σε περίπτωση πιθανών κινδύνων. Ακόμη ένα παράδειγμα αποτελούν τα έξυπνα αυτοκίνητα με ένα τεράστιο αριθμό αισθητήρων συνδεδεμένων πάνω τους, αλλά και η παρακολούθηση σε πραγματικό χρόνο της κίνησης στους δρόμους.

➔ **<u>Τραπεζικά συστήματα</u>:**

Άλλο ένα πεδίο που εξελίσσεται όλο και περισσότερο τα τελευταία χρόνια αποτελούν οι τραπεζικές συναλλαγές. Με τις διαδικτυακές αγορές που πραγματοποιούνται πλέον οποιαδήποτε στιγμή της ημέρας οι τράπεζες θα πρέπει να είναι σε θέση να λαμβάνουν πληροφορίες για τις συναλλαγές αυτές, να φροντίζουν για την πραγματοποίηση αυτών και να ανιχνεύουν πιθανές προσπάθειες υποκλοπής από τρίτα πρόσωπα. Όπως γίνεται εύκολα αντιληπτό, απαιτείται τα συστήματα που εξεργάζονται τα δεδομένα αυτά να είναι ακριβή στους υπολογισμούς τους και να έχουν άμεση απόκριση.

➔ **<u>Τηλεπικοινωνίες</u>**:

Τέλος, στον τομέα των τηλεπικοινωνιών η καταγραφή και επεξεργασία ροών δεδομένων είναι απαραίτητη για τη δυναμική δρομολόγηση των κλήσεων, τη χρήση διαφορετικών πύργων κινητής τηλεφωνίας με βάση την τοποθεσία και την αναγνώριση ανωμαλιών και πιθανών βλαβών.

2.1.2 Στόχοι

Η ικανότητα επεξεργασίας ροών δεδομένων σε σχεδόν πραγματικό χρόνο δεν είναι το μόνο πλεονέκτημα των DSPS (distributed stream processing systems). Εξίσου σημαντικοί στόχοι αποτελούν η υψηλή απόδοση (Throughput) αλλά και η ικανότητα του συστήματος να διαχειριστεί τις διακοπές. Ένα καλό σύστημα θα πρέπει να είναι σε θέση να επανέλθει από κάποιο σφάλμα που μπορεί να προκύψει σε κάποιο κόμβο, αλλά και να συνεχίσει να υπολογίζει σωστά και ακριβή αποτελέσματα μετά από αυτό. Τίθεται λοιπόν το ζήτημα του fault-tolerance με exactly-once guarantees (ανοχή σε σφάλματα και υπολογισμός του σωστού αποτελέσματος μόνο μια φορά και όχι παραπάνω). Παράλληλα είναι σημαντικό η μέθοδος που χρησιμοποιείται για την εξασφάλιση των παραπάνω να μην προσθέτει μεγάλο overhead που μπορεί να

επηρεάσει αισθητά την απόδοση του συστήματος ή να προκαλέσει καθυστερήσεις.

Άλλος ένας σημαντικός στόχος των DSPS είναι η επεξεργασία της ροής των γεγονότων με βάση τον χρόνο που συνέβησαν και όχι τον χρόνο που εισήλθαν στο σύστημα. Οι δύο αυτές έννοιες του χρόνου είναι διαφορετικές και οδηγούν σε διαφορετικούς υπολογισμούς, αφού είναι συχνό φαινόμενο τα γεγονότα να φτάνουν out-of-order και να απαιτούνται μηχανισμοί για την αναδιάταξη τους. Τέλος είναι εξίσου κρίσιμο τα συστήματα να παρέχουν φιλικά APIs για το γράψιμο κώδικα και εργαλεία για debugging ζητήματα.

2.2. Stream-First Αρχιτεκτονική

Τα τελευταία χρόνια η επεξεργασία μεγάλου όγκου δεδομένων πραγματοποιούνταν με την οργάνωση και τον χωρισμό αυτών σε batches και έπειτα οι υπολογισμοί λάμβαναν τόπο σε μεγάλης κλίμακας clusters. Το βασικότερο προγραμματιστικό μοντέλο που εδραιώθηκε για την επίτευξη των παραπάνω είναι το MapReduce. Παρά τις προσπάθειες ωστόσο να εφαρμοστεί η ίδια μεθοδολογία σε ροές δεδομένων οι απαιτήσεις για μικρές καθυστερήσεις (Latencies) στους υπολογισμούς και μεγάλη απόδοση είναι αποτρεπτικοί παράγοντες. Παράλληλα, δεν υπάρχουν τόσο ξεκάθαρα μοντέλα όπως το MapReduce για τον χειρισμό τέτοιου είδους δεδομένων. ****

2.2.1 Παραδοσιακή Αρχιτεκτονική vs Streaming Αρχιτεκτονική

Παραδοσιακά, η τυπική αρχιτεκτονική ενός backend συστήματος αποτελούνταν από μια κεντρική βάση δεδομένων που κρατούσε όλα τα σημαντικά δεδομένα μιας επιχείρησης. Με άλλα λόγια, η βάση αυτή (είτε SQL είτε NoSQL) διατηρούσε όλα τα “φρέσκα” δεδομένα που αντιπροσώπευαν την κατάσταση της επιχείρησης εκείνη τη στιγμή. Αυτά θα μπορούσαν να εκφράζουν πόσοι και ποιοι χρήστες έχουν συνδεθεί στην πλατφόρμα της, ποιοι από αυτούς είναι ενεργοί και ποια είναι η τρέχουσα κατάσταση του λογαριασμού τους. Κάθε φορά λοιπόν που μια εφαρμογή απαιτούσε δεδομένα θα έπρεπε να απευθύνεται στη βάση αυτή για να τα πάρει. Παράλληλα, η χρήση κατανεμημένων συστημάτων αρχείων για αποθήκευση παρουσιάζει αρκετές δυσκολίες όσον αφορά το συγχρονισμό και την ανανέωση των δεδομένων και καθυστερήσεις στους σύνθετους batch υπολογισμούς. Τα κυριότερα προβλήματα επομένως που παρατηρούνται στις επιχειρήσεις είναι τα παρακάτω:

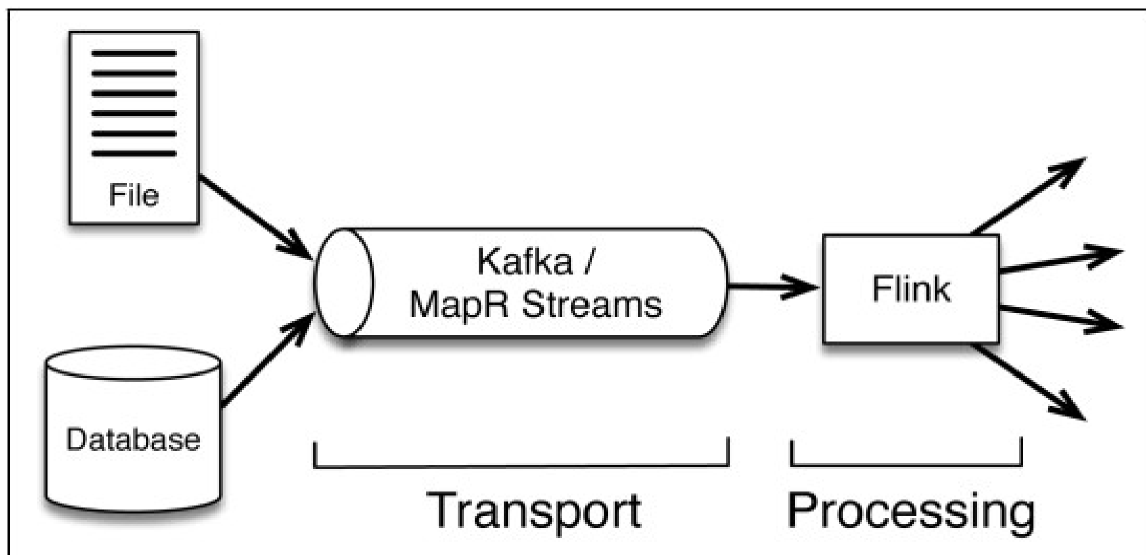
- Ο ρυθμός της απορρόφησης κρίσιμων δεδομένων από εφαρμογές που τα χρειάζονται άμεσα είναι μικρός με αποτέλεσμα να προκαλούνται σημαντικές καθυστερήσεις.
- Η παραδοσιακή αρχιτεκτονική είναι πολύ μονολιθική με τη βάση δεδομένων να λειτουργεί ως μοναδική πηγή της “αλήθειας” για το ποια είναι η κατάσταση του συστήματος μια δεδομένη χρονική στιγμή.
- Τα συστήματα αυτά δεν είναι ανθεκτικά σε σφάλματα, ενώ η προσπάθεια για την εφαρμογή τεχνολογιών ανάνηψης μπορεί να αποδειχθεί δαπανηρή σε hardware και

αρκετά πολύπλοκη.

Η σύγχρονη και εναλλακτική προσέγγιση που υιοθετεί η streaming αρχιτεκτονική λύνει πολλά από τα παραπάνω προβλήματα που αντιμετωπίζουν οι επιχειρήσεις όταν εργάζονται με μεγάλης κλίμακας συστήματα. Ο σχεδιασμός αυτός των συστημάτων βασίζεται στην ελεύθερη και συνεχόμενη ροή των δεδομένων από τις πηγές στις εφαρμογές αλλά και μεταξύ των ίδιων των εφαρμογών. Στην περίπτωση αυτή δεν υπάρχει κάποια κεντρική βάση να κρατάει την κατάσταση του συστήματος, αλλά αυτή μοιράζεται ανάμεσα σε όλα τα στοιχεία που συνθέτουν το σύστημα. Έτσι, οι ίδιες οι εφαρμογές σχηματίζουν την δική τους άποψη για τον κόσμο γύρω τους καταγράφοντας πληροφορίες, αποθηκεύοντας δεδομένα σε τοπικές βάσεις και χρησιμοποιώντας δικά τους συστήματα κατανεμημένων αρχείων.

2.2.2 Επίπεδο μεταφοράς μηνύματος

Μια Streaming αρχιτεκτονική υλοποιείται χρησιμοποιώντας αποδοτικά δύο βασικά συστατικά, ένα σύστημα για την μεταφορά των μηνυμάτων (message transport) και ένα για την επεξεργασία της ροής των δεδομένων (stream processing system). Σκοπός του πρώτου επιπέδου είναι να συλλέξει γεγονότα (events) από ποικίλες πηγές (producers) και να τα κάνει διαθέσιμα σε εφαρμογές και services που τα ζητάνε (subscribers). Το δεύτερο επίπεδο είναι υπεύθυνο για την εκτέλεση των υπολογισμών, τη διατήρηση της κατάστασης της εφαρμογής (state) σε τοπικό επίπεδο αλλά και την μεταφορά δεδομένων σε άλλες εφαρμογές και συστήματα που τα χρειάζονται. Η αναπαράσταση μιας τέτοια αρχιτεκτονικής φαίνεται στην παρακάτω εικόνα (Εικόνα 2-1)



Εικόνα 2-1. Εδώ το Apache Kafka/MapR Streams μπορεί να αντικατασταθεί με οποιαδήποτε άλλο σύστημα μεταφοράς μηνυμάτων και το Flink με οποιαδήποτε άλλο σύστημα επεξεργασίας.

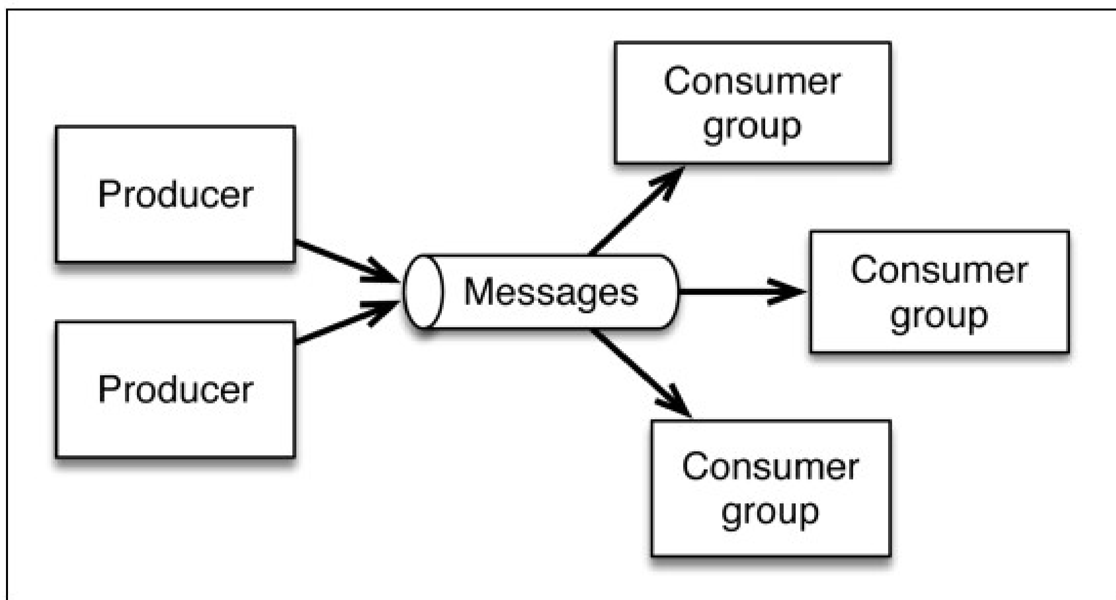
Στο συγκεκριμένο υποκεφάλαιο θα αναλυθούν οι βασικές αρχές που πρέπει να διέπουν και οι απαιτήσεις που πρέπει να ικανοποιούν τα συστήματα για τη μεταφορά μηνυμάτων, ένα από τα οποία είναι και το Apache Kafka που θα μας απασχολήσει στη συνέχεια. Αυτές είναι οι παρακάτω:

➔ **Performance with Persistence**

Ένας βασικός ρόλος του επιπέδου μεταφοράς είναι να λειτουργεί σαν buffer (η αλλιώς σαν μια ουρά) που διατηρεί τα μηνύματα που συγκεντρώνει από τις πηγές για κάποιο χρονικό διάστημα, ώστε να υπάρχει ασφάλεια και ικανότητα επαναφοράς ολόκληρου του συστήματος σε περιπτώσεις διακοπών του. Έτσι, πέρα από τη διανομή των μηνυμάτων αποδοτικά στο επόμενο επίπεδο, είναι πολύ σημαντικό για τη σύγχρονη αρχιτεκτονική τα συστήματα μεταφοράς μηνυμάτων να μπορούν να επαναλάβουν το μοίρασμα των δεδομένων για να ξαναγίνουν υπολογισμοί που λόγω κάποιου σφάλματος δεν μπόρεσαν να πραγματοποιηθούν την πρώτη φορά.

➔ **Εξυπηρέτηση πολλαπλών και διαφορετικού τύπου δεδομένων Producers και Consumers**

Μία σύγχρονη τεχνολογία μεταφοράς μηνυμάτων θα πρέπει να επιτρέπει τη συλλογή δεδομένων από πολλαπλούς Producers και τη διανομή αυτών σε πολλαπλούς Consumers. Αυτό, όπως θα δούμε και παρακάτω, επιτυγχάνεται με τη δημιουργία topics στα οποία εγγράφονται (subscribe) οι Consumers και περιμένουν την προώθηση των μηνυμάτων στην ουρά από τους Producers. Τα δεδομένα ενός topic δεν προωθούνται σε όλους τους Consumers ενος Consumer Group (μια ομάδα από Consumers που έχουν κάνει subscribe στο ίδιο topic) αλλά μοιράζονται σε αυτούς. Ένα παράδειγμα μιας τέτοιας αρχιτεκτονικής παρουσιάζεται στην Εικόνα 2-2.



Εικόνα 2-2.

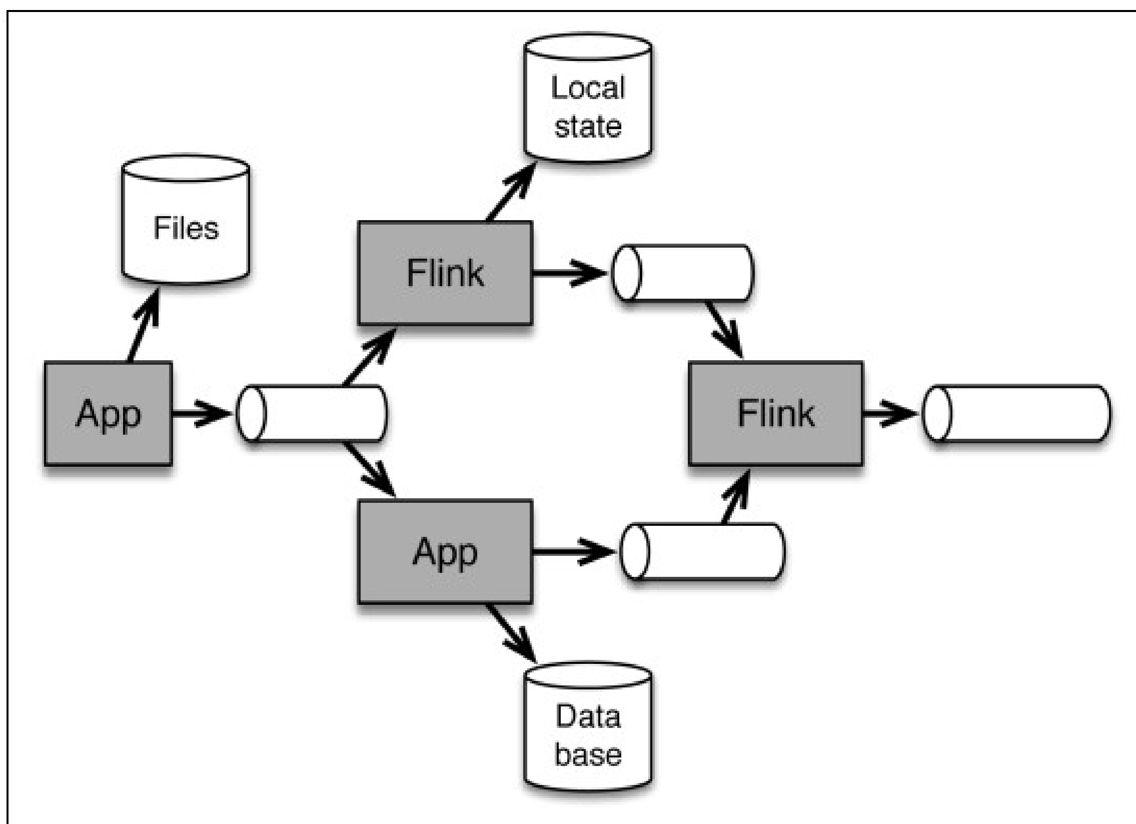
➔ **Replication of Streams:**

Είδαμε ότι οι περισσότερες εφαρμογές που απαιτούν αυτού του είδους την αρχιτεκτονική έχουν πολύ σημαντικές αποστολές και λανθασμένα αποτελέσματα στους υπολογισμούς μπορεί να αποβούν μοιραία (εφαρμογές στον ιατρικό, τραπεζικό τομέα,

τηλεπικοινωνίες κλπ). Η διατήρηση επομένως αντιγράφων (replication) των δεδομένων σε διαφορετικές τοποθεσίες αποκτά ιδιαίτερη σημασία ώστε αυτά να είναι διαθέσιμα σε περίπτωση που κάποιο στοιχείο του συστήματος αποτύχει. Το παραπάνω σχέδιο ωστόσο απαιτεί επικοινωνία και συγχρονισμό μεταξύ των κόμβων που χειρίζονται τα ίδια δεδομένα, ώστε να διαθέτουν πάντα την νεότερη έκδοση αυτών. Η υλοποίηση γίνεται συνήθως με μια αρχιτεκτονική master-slave και κυκλικό replication. Ο master κόμβος είναι υπεύθυνος για να ενημερώσει τους υπόλοιπους για τυχόν αλλαγές και να τους αποστείλει τα σωστά offsets των μηνυμάτων (δηλαδή να τους ενημερώσει σε ποιο ακριβώς σημείο της ροής των δεδομένων βρίσκεται ο ίδιος) ώστε να μπορούν αυτοί με τη σειρά τους να επανεκκινήσουν το σύστημα από το ίδιο σημείο σε περίπτωση σφάλματος.

2.2.3 Επίπεδο επεξεργασίας μηνύματος

Τα συστήματα κατανεμημένης επεξεργασίας ροών δεδομένων (DSPS) ανήκουν σε αυτό το επίπεδο και όπως έχει ήδη αναφερθεί, σε αυτό το στάδιο γίνονται όλοι οι απαραίτητοι υπολογισμοί για να εξαχθεί η χρήσιμη πληροφορία από τον όγκο των δεδομένων και να προωθηθεί σε κάποια εφαρμογή. Καταλήγουμε επομένως σε μια αρχιτεκτονική σαν αυτή που φαίνεται στην Εικόνα 2-3.



Εικόνα 2-3. Παρουσιάζεται μια streaming αρχιτεκτονική στο σύνολο της. Η ροή των δεδομένων απεικονίζεται με ένα μαύρο βέλος και συνδέει συστήματα και εφαρμογές μεταξύ τους. Παρατηρούμε ότι οι ουρές μεταφοράς μηνυμάτων (απεικονίζονται με έναν κύλινδρο) εξυπηρετούν πολλαπλούς Consumers ενώ τα συστήματα κρατούν πλέον τοπικά αντίγραφα της κατάστασης τους καταργώντας ουσιαστικά την ανάγκη για μια κεντρική βάση δεδομένων.

Τα βασικά πλεονεκτήματα των DSPS όπως το Flink πέρα από την απόδοση και τους μικρούς χρόνους απόκρισης είναι:

- Υπολογισμοί με βάση το **Event-time** των δεδομένων
- **Stateful** υπολογισμοί με εξασφάλιση του **Consistency**

Η επεξήγηση των όρων αυτών και η σημασία τους παρουσιάζεται στα δύο επόμενα υποκεφάλαια.

2.3 Χειρισμός του χρόνου

2.3.1 Έννοιες του χρόνου

Στην επεξεργασία ροής δεδομένων συνήθως συναντάμε 3 διαφορετικές έννοιες του χρόνου:

➔ **<u>Event-time</u>**

Πρόκειται για το χρόνο που το γεγονός συμβαίνει στον πραγματικό κόσμο. Συγκεκριμένα μια εγγραφή δεδομένων, την ώρα που δημιουργείται της αποδίδεται ένα timestamp (από κάποιον server συνήθως) που αποτελεί μέρος της εγγραφής.

➔ **<u>Ingestion-time</u>**

Πρόκειται για τον χρόνο που η εγγραφή εισέρχεται στο DSPS από κάποιο σύστημα μεταφοράς μηνυμάτων. Και εκεί συνήθως υπάρχει ένας μηχανισμός που αναθέτει timestamps στις εγγραφές για τη λήψη μετρικών του συστήματος.

➔ **<u>Processing-time</u>**

Πρόκειται για τον χρόνο της επεξεργασίας της εγγραφής από το DSPS. Αυτός μετριέται από το ρολόι του εκάστοτε μηχανήματος που είναι υπεύθυνο για την εκτέλεση των υπολογισμών.

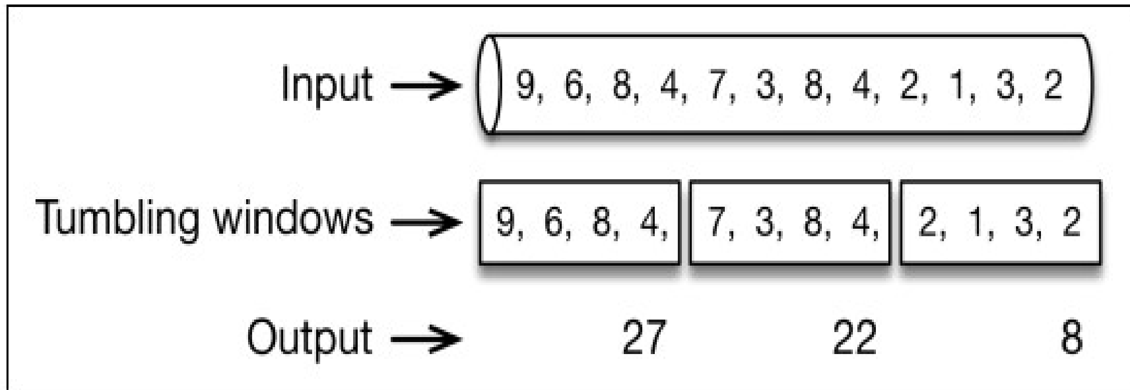
2.3.2 Windows

Τα Windows είναι ένας μηχανισμός για την ομαδοποίηση και τη συγκέντρωση δεδομένων συνήθως με βάση το χρόνο με στόχο την πραγματοποίηση υπολογισμών πάνω στο σύνολο αυτών. Στο κεφάλαιο αυτό θα αναλυθούν οι σημαντικότερες κατηγορίες των Time Windows που χρησιμοποιήθηκαν και αργότερα για την εκτέλεση των πειραμάτων.

➔ **Tumbling Windows**

Στην εικόνα 2-4 παρουσιάζεται η λειτουργία ενός τέτοιου τύπου παραθύρου. Ανάλογα με το μέγεθος (χρονικό) που θα επιλεγεί για το παράθυρο, θα εκτελεστεί ο

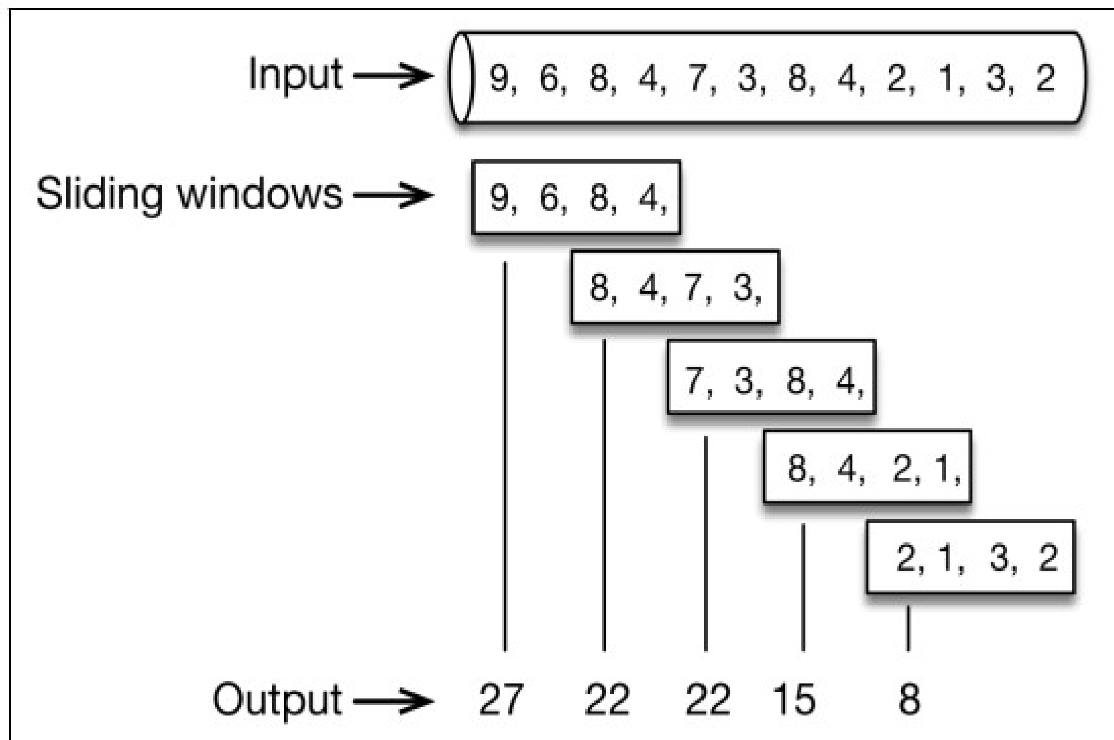
επιθυμητός υπολογισμός πάνω στο σύνολο των δεδομένων που έχουν συγκεντρωθεί στα χρονικά αυτά όρια που έχουν οριστεί (στην περίπτωση της εικόνας υπολογίζεται το άθροισμα των στοιχείων της εισόδου).



Εικόνα 2-4.

Sliding Windows

Σε αυτού του είδους τα παράθυρα πέρα από το μέγεθος ορίζεται και μια παράμετρος που αφορά την ολίσθηση του παραθύρου. Όπως γίνεται αντιληπτό και από την εικόνα αν ορίσουμε το μέγεθος του παραθύρου στα 10 και την ολίσθηση στα 5 λεπτά, τότε στην έξοδο θα παίρνουμε κάθε 5 λεπτά το άθροισμα του συνόλου των στοιχείων των τελευταίων 10 λεπτών.



Εικόνα 2-5.

2.3.3 Watermarks

Μετά την εξήγηση των διαφορετικών εννοιών του χρόνου και των διαφόρων ειδών χρονικών παραθύρων είναι σημαντικό να καταλάβει κανείς πώς αντιλαμβάνονται τα παράθυρα το πέρασμα του χρόνου ώστε να προωθήσουν το αποτέλεσμα του υπολογισμού στο επόμενο επίπεδο. Για να γίνει αυτό ωστόσο είναι σημαντικό να διευκρινιστεί ποια έννοια του χρόνου θα χρησιμοποιηθεί.

Στην περίπτωση που επιθυμούμε να δουλέψουμε με βάση το Processing Time γίνεται εύκολα αντιληπτό ότι δεν απαιτείται κάποιος ιδιαίτερος μηχανισμός αφού το μηχάνημα στο οποίο γίνονται οι υπολογισμοί μετράει το χρόνο χρησιμοποιώντας το δικό του ρολόι. Έτσι, αν το μέγεθος του παραθύρου οριστεί στα 10 λεπτά, οι πράξεις θα εκτελεστούν στο σύνολο των δεδομένων που έχουν μαζευτεί στο μηχάνημα μέσα στο χρόνο αυτό.

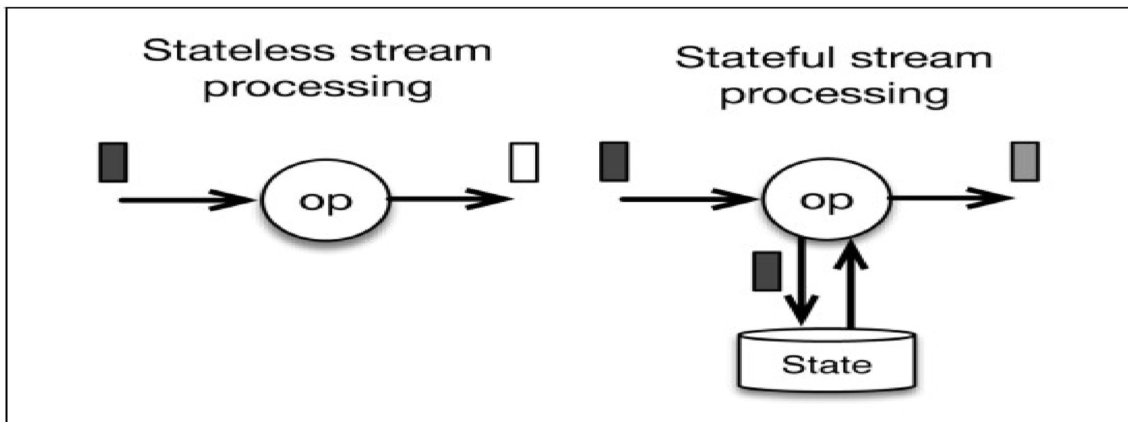
Όταν ωστόσο επιθυμούμε να δουλέψουμε με βάση το Event Time (κάτι που ισχύει για την πλειοψηφία των περιπτώσεων) και θέλουμε για παράδειγμα να υπολογίσουμε κάποιο άθροισμα στοιχείων από εγγραφές που έχουν **δημιουργηθεί** την τελευταία ώρα είναι αδύνατο να γνωρίζουμε εάν αυτά θα έχουν φτάσει στο μηχάνημα μέσα στο χρόνο που έχει οριστεί (αν μετράμε με το ρολόι του μηχανήματος). Είναι πιθανό λοιπόν μετά το πέρας της ώρας να μην έχουμε όλες τις εγγραφές, με αποτέλεσμα ο υπολογισμός που θα προωθήσουμε στο επόμενο επίπεδο να είναι λανθασμένος. Αυτό συμβαίνει διότι το Event με το Processing Time διαφέρουν σημαντικά ενώ υπάρχουν και περιπτώσεις που οι εγγραφές μπορεί να φτάσουν εκτός σειράς (out-of-order).

Καθίσταται απαραίτητη επομένως για τη δεύτερη περίπτωση η χρήση Watermarks. Τα Watermarks αντιμετωπίζονται σαν κανονικές εγγραφές, ρέουν ελεύθερα μέσα στο σύστημα και έχουν τη μορφή ενός timestamp. Το timestamp αυτό συνήθως εξάγεται μέσα από την εγγραφή και δηλώνει το χρόνο που αυτή δημιουργήθηκε. Όταν λοιπόν ένα Watermark φτάσει σε ένα μηχάνημα που εκτελεί κάποιον υπολογισμό σε παράθυρο, του επιτρέπει να προωθήσει το χρόνο τόσο όσο ορίζει το timestamp του. Στόχος του τέλειου Watermark είναι να διασφαλίσει ότι μετά το πέρας του δεν μπορεί να υπάρξει εγγραφή που να έχει δημιουργηθεί πριν από το χρόνο που δηλώνεται από αυτό. Έτσι για παράδειγμα αν θέλουμε το άθροισμα των τιμών στοιχείων που έχουν δημιουργηθεί από τις 13:00-14:00, τότε μόνο όταν έρθει ένα Watermark με timestamp μεγαλύτερο του 14:00 μπορεί το παράθυρο να κλείσει και να κάνει διαθέσιμο το αποτέλεσμα του. Αυτό βέβαια μπορεί να μην καλύπτει κάποιες περιπτώσεις που τα δεδομένα φτάνουν εκτός σειράς, αλλά αυτό εξαρτάται και από το πόσο συχνά παράγουμε τα Watermarks και με ποιο κριτήριο ορίζουμε το timestamp τους. Σε κάθε περίπτωση πάντως είναι σημαντικό να βρεθεί μια ισορροπία που θα διασφαλίζει σωστά αποτελέσματα χωρίς να προκαλεί μεγάλες καθυστερήσεις στο κλείσιμο των παραθύρων.

2.4 Stateful υπολογισμοί

Ένας υπολογισμός πάνω σε ροή δεδομένων μπορεί να είναι είτε Stateless είτε

Stateful. Ένα Stateless πρόγραμμα λαμβάνει στην είσοδο κάθε εγγραφή ξεχωριστά και δίνει στην έξοδο αποτελέσματα που προκύπτουν αποκλειστικά μόνο από την τελευταία εγγραφή. Αντίθετα, τα Stateful προγράμματα ανανεώνουν συνεχώς την κατάσταση τους με την εμφάνιση μιας νέας εγγραφής με στόχο να κάνουν υπολογισμούς πάνω σε σύνολα δεδομένων. Όπως γίνεται αντιληπτό η τελευταία κατηγορία έχει μεγαλύτερες απαιτήσεις για τη σωστή διατήρηση του state και την εγκυρότητα των αποτελεσμάτων σε περιπτώσεις σφαλμάτων. Οι περιπτώσεις του stateless και του stateful stream processing παρουσιάζονται στην εικόνα 2-6.



Εικόνα 2-6.

2.4.1 Η έννοια του Consistency

Όταν εκτελούμε Stateful υπολογισμούς είναι πολύ σημαντικό να διασφαλίζεται το Consistency. Ουσιαστικά με τον όρο αυτό εννοούμε την ικανότητα του συστήματος να συνεχίσει να παράγει ορθά αποτελέσματα σε περίπτωση που το σύστημα ανακάμψει έπειτα από κάποιο σφάλμα. Ιδανικά, τα αποτελέσματα των υπολογισμών μετά από κάποιο χρόνο λειτουργίας του συστήματος χωρίς σφάλματα θα πρέπει να είναι ίδια με αυτά της περίπτωσης που συμβούν σφάλματα. Στον κλάδο της επεξεργασίας ροής δεδομένων διακρίνονται 3 διαφορετικά επίπεδα Consistency.

➔ **At most once**

Στο επίπεδο αυτό δεν υπάρχει καμία διασφάλιση ότι θα έχουμε το σωστό αποτέλεσμα ενός υπολογισμού μετά την ανάκαμψη του συστήματος από σφάλμα.

➔ **At least once**

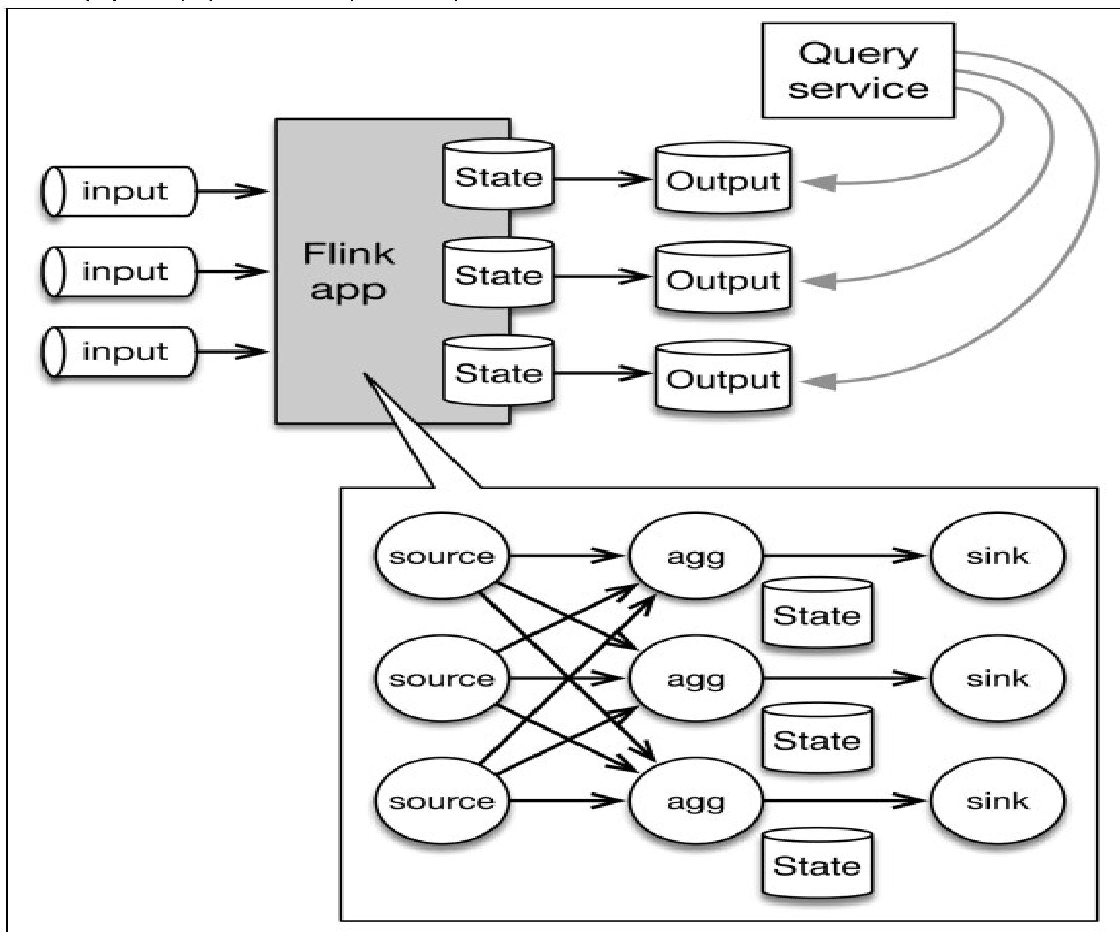
Εδώ, γνωρίζουμε ότι το σωστό αποτέλεσμα θα υπάρχει στο τέλος αλλά μπορεί να υπάρχουν διπλά στοιχεία σε αυτό (duplicates) λόγω της επανεκτέλεσης υπολογισμών μετά την ανάκαμψη. Για παράδειγμα στην περίπτωση υπολογισμού ενός αθροίσματος, το σύστημα θα βγάζει στην έξοδο πάντα αποτέλεσμα μεγαλύτερο ή ίσο του πραγματικού αθροίσματος, αλλά ποτέ μικρότερο. Διασφαλίζεται επομένως ότι θα πραγματοποιηθούν όλοι οι επιθυμητοί υπολογισμοί τουλάχιστον μία φορά.

➔ **Exactly once**

Το επίπεδο αυτό του Consistency ικανοποιεί τις πιο απαιτητικές εφαρμογές όπου είναι κρίσιμο να έχουμε πάντοτε ένα και μοναδικό αποτέλεσμα στην έξοδο και αυτό να είναι ορθό.

2.4.2 End-to-end Consistency

Συνοψίζοντας το κεφάλαιο αυτό, στην Εικόνα 2-7 παρουσιάζεται η αρχιτεκτονική μιας ολοκληρωμένης εφαρμογής όπου υλοποιείται κατανεμημένη επεξεργασία ροής δεδομένων με Stateful υπολογισμούς. Σαν είσοδος μπορεί να χρησιμοποιηθεί ένα κατανεμημένο σύστημα μεταφοράς μηνυμάτων (όπως το Kafka) και σαν έξοδος (output) ένα οποιοδήποτε κατανεμημένο σύστημα αρχείων. Παρατηρεί κανείς ότι η τοπολογία του Flink αποτελείται από 3 operators, το διάβασμα της εισόδου από τις πηγές, την εκτέλεση των Stateful υπολογισμών (ενδεικτικά φαίνεται ένα aggregation) και την προώθηση των αποτελεσμάτων σε ένα sink που με τη σειρά του θα τα κάνει διαθέσιμα στην έξοδο. Κάτι που αξίζει να προσέξει κανείς είναι η αποθήκευση και διατήρηση του state σε κάθε βήμα των υπολογισμών όχι μόνο μέσα στο κατανεμημένο σύστημα επεξεργασίας δεδομένων (Flink) , αλλά και έξω από αυτό. Έτσι σε περίπτωση που κάποιος κόμβος του συνολικού συστήματος αποτύχει, υπάρχουν οι αναγκαίες πληροφορίες για την επανεκκίνηση αυτού από το ίδιο ακριβώς σημείο. Επίσης, ανάλογα με την κρισιμότητα της εφαρμογής μπορούν να τεθούν οι κατάλληλοι περιορισμοί ώστε να εξασφαλίζεται το Exactly once επίπεδο του Consistency στα αποτελέσματα. Αυτό στο Flink επιτυγχάνεται μέσω των Checkpoints στα οποία θα γίνει ιδιαίτερη αναφορά στο επόμενο κεφάλαιο.



Εικόνα 2-7.

Κεφάλαιο 3

Τεχνικό Υπόβαθρο

3.1 Flink

Το Apache Flink είναι μία open source πλατφόρμα που προσφέρει κατανεμημένη επεξεργασία ροών αλλά και batches δεδομένων. Αναπτύχθηκε μεταξύ 2010-2014 από 3 πανεπιστήμια του Βερολίνου σε συνεργασία με άλλα ευρωπαϊκά εκπαιδευτικά ιδρύματα. Το Δεκέμβριο του 2014 συμπεριλήφθηκε στα top-level projects του Apache Software Foundation. Το Flink μπορεί να τρέξει σε εκατοντάδες ή και χιλιάδες μηχανήματα που εκτελούν παράλληλους υπολογισμούς εξασφαλίζοντας την ανοχή του συστήματος σε σφάλματα και Exactly once Consistency στα αποτελέσματα.

3.1.1 Κατανεμημένο Περιβάλλον Εκτέλεσης Υπολογισμών

Ένα Flink cluster αποτελείται από 2 ειδών μηχανήματα:

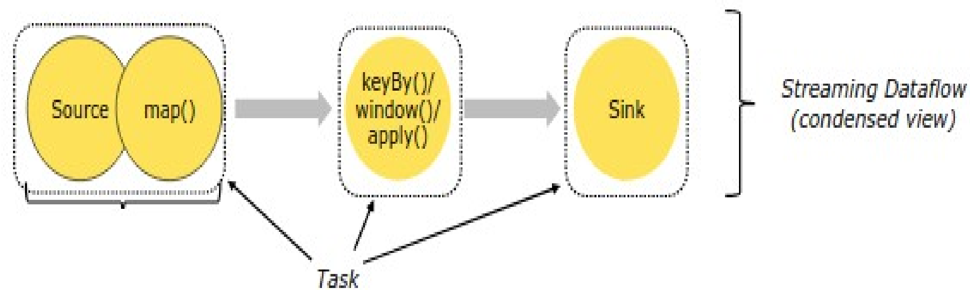
➔ **Job Manager**

Οι κόμβοι αυτοί (αποκαλούνται και masters) συντονίζουν την κατανεμημένη εκτέλεση των υπολογισμών. Είναι υπεύθυνοι για τη δρομολόγηση των εργασιών στους Task Managers, τη διατήρηση των checkpoints και την ανάνηψη του συστήματος σε περίπτωση αποτυχιών. Πάντα θα πρέπει να υπάρχει τουλάχιστον 1 Job Manager ο οποίος θα είναι ο leader και οι υπόλοιποι θα είναι standby για λόγους υψηλής διαθεσιμότητας.

➔ **Task Manager**

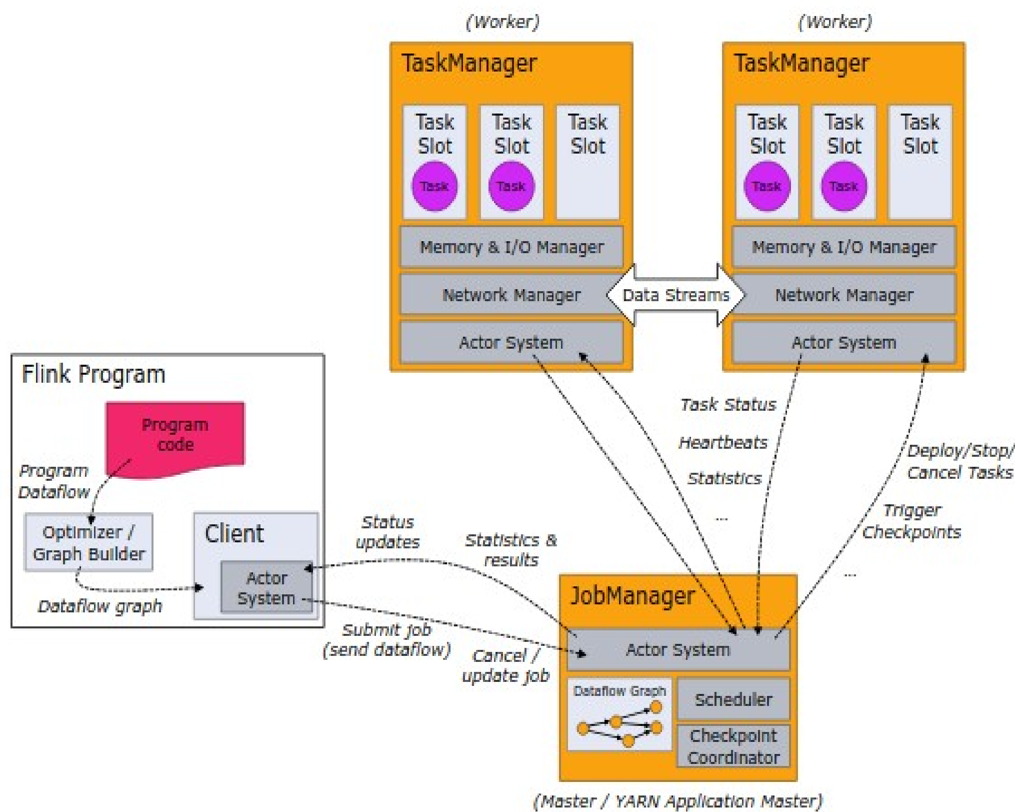
Οι κόμβοι αυτοί (αποκαλούνται και workers) είναι καθαρά υπεύθυνοι για την εκτέλεση των εργασιών (subtasks) που τους ανατίθενται από τον Job Manager. Κάθε ένας από αυτούς χωρίζεται σε έναν αριθμό από task slots. Κάθε task slot αντιπροσωπεύει ένα υποσύνολο των διαθέσιμων πόρων του μηχανήματος. Για παράδειγμα, ένας κόμβος με 3 task slots θα διαθέσει στο καθένα από αυτά το 1/3 της διαθέσιμης μνήμης του. Συνήθως επιλέγεται ο αριθμός των task slots να είναι ίσος με αυτόν των πυρήνων του επεξεργαστή του μηχανήματος.

Πέρα από τα 2 αυτά στοιχεία υπάρχει και ο **Client** (υπολογιστής-πελάτης) που δεν αποτελεί μέρος του περιβάλλοντος εκτέλεσης αλλά χρησιμοποιείται για τη δημιουργία του dataflow graph (γράφος ροής των δεδομένων) από τον κώδικα του προγραμματιστή και την υποβολή του στο Job Manager (Εικόνα 3-4). Ο Client μπορεί στη συνέχεια να παραμείνει συνδεδεμένος με το σύστημα για να λαμβάνει ενημερώσεις για την πρόοδο, καθώς και στατιστικά για το job που υπέβαλλε.



Εικόνα 3-4. Φαίνεται ένα dataflow graph που υποβάλλεται στον Job Manager από τον Client. Απεικονίζονται οι 4 operators που θα δράσουν πάνω στη ροή των δεδομένων.

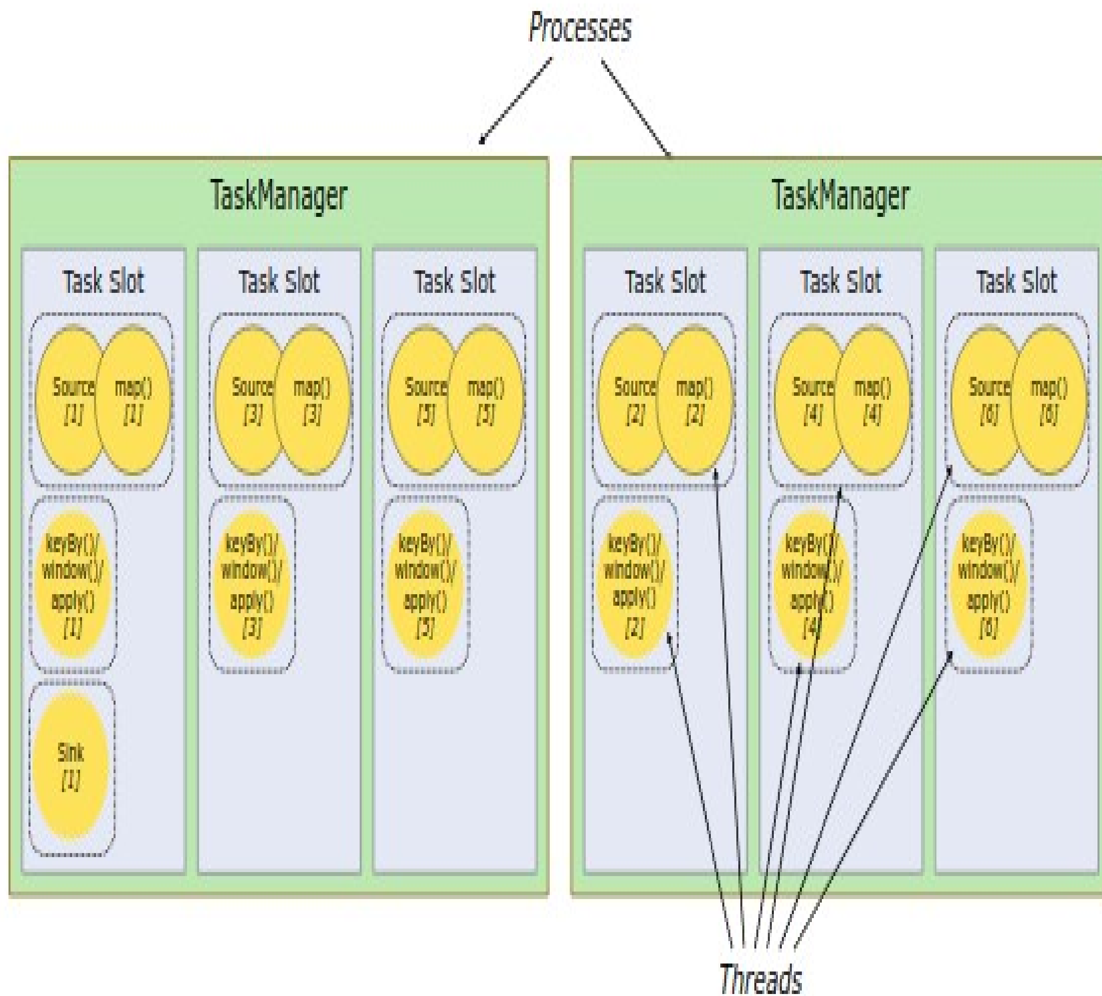
Σε συνέχεια των παραπάνω ο Job Manager διαθέτοντας το dataflow graph, χωρίζει τους operators σε tasks τοποθετώντας συχνά περισσότερους από 1 operator σε 1 task. Αυτό συμβαίνει διότι κάθε task εκτελείται από ένα μόνο νήμα, οπότε ομαδοποιώντας τους operators μειώνεται το κόστος της επικοινωνίας μεταξύ νημάτων και αυξάνεται η συνολική απόδοση του συστήματος. Η λογική πίσω από αυτό το μηχανισμό ομαδοποίησης μπορεί να ρυθμιστεί για να ικανοποιεί τις ανάγκες της εκάστοτε επεξεργασίας δεδομένων. Το επόμενο βήμα είναι ο διαμοιρασμός των tasks απο τον Job Manager στα διαθέσιμα task slots όπου πραγματοποιούνται οι επιθυμητοί υπολογισμοί και τα αποτελέσματα καταλήγουν τελικώς από τους Task Managers σε κάποιο κατανεμημένο σύστημα αρχείων. Μια σύνοψη όλων των παραπάνω παρουσιάζεται στην Εικόνα 3-5.



Εικόνα 3-5.

3.1.2 Δρομολόγηση Εργασιών

Για κάθε task ενός job στο Flink, όπως αυτά που είδαμε προηγουμένως, επιλέγεται από τον προγραμματιστή ένα επίπεδο παραλληλισμού, δηλαδή πόσα task slots θα πραγματοποιούν παράλληλα υπολογισμούς για την ολοκλήρωση του. Όπως φαίνεται στην Εικόνα 3-6 για το task Source-map και για το keyBy-window-apply έχει επιλεγεί επίπεδο παραλληλισμού ίσο με 6 ενώ για το task Sink επίπεδο παραλληλισμού ίσο με 1, για τη συγκέντρωση προφανώς των δεδομένων σε ένα μόνο μηχάνημα. Ο Job Manager επομένως ανάλογα με το επίπεδο αυτό απλώς μοιράζει τα tasks στα διαθέσιμα slots και ελέγχει τακτικά για τυχόν σφάλματα και αποτυχίες. Κάτι ακόμα που γίνεται αντιληπτό από την εικόνα είναι ότι διαφορετικά tasks μπορούν να μοιράζονται το ίδιο slot, δημιουργώντας ουσιαστικά ένα pipeline από εργασίες. Αυτό μπορεί να οδηγήσει σε σημαντικά καλύτερη απόδοση, δεδομένου ότι κάθε task slot (ταυτίζεται συνήθως με έναν πυρήνα του επεξεργαστή του μηχανήματος) έχει 2 ή παραπάνω διαθέσιμα νήματα (hyper-threading) για ταυτόχρονη εκτέλεση εργασιών.



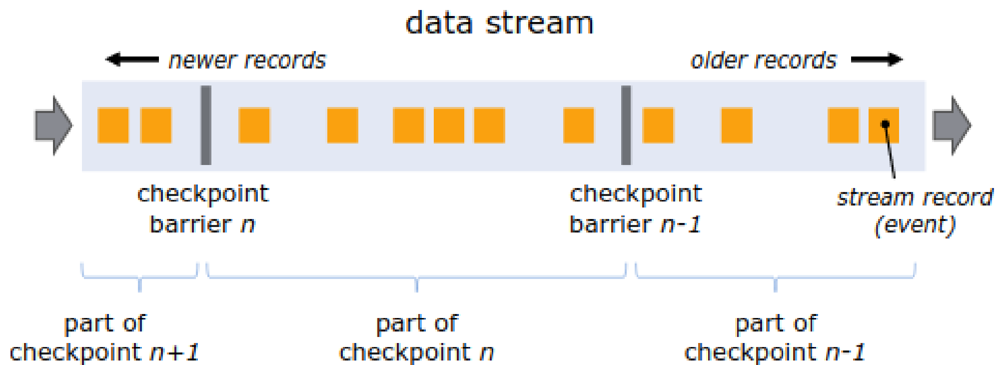
Εικόνα 3-6.

3.1.3 Fault Tolerance (Checkpointing)

Το Apache Flink προσφέρει ένα μηχανισμό αντοχής σφαλμάτων για την ανάκτηση με συνέπεια (Consistency) της κατάστασης (state) των εφαρμογών ροής δεδομένων. Ο μηχανισμός αυτός εξασφαλίζει ότι ακόμη και σε περίπτωση βλαβών το σύστημα θα έχει εκτελέσει όλους τους απαραίτητους υπολογισμούς ακριβώς μία φορά και θα βρίσκονται στην έξοδο του τα σωστά αποτελέσματα.

Για να επιτευχθεί κάτι τέτοιο τραβιούνται συνεχώς στιγμιότυπα (snapshots) από την κατανεμημένη ροή των δεδομένων, που απεικονίζουν την κατάσταση του συστήματος τη δεδομένη στιγμή. Η κατάσταση αυτή αποθηκεύεται σε κάποιο επιλεγμένο σημείο (αποκαλείται state backend) το οποίο συνήθως είναι ο Job Manager ή ένα HDFS (κατανεμημένο σύστημα αρχείων). Για εφαρμογές με μικρή κατάσταση, τα στιγμιότυπα αυτά είναι πολύ ελαφριά και μπορούν να σχεδιάζονται συχνά χωρίς να έχουν κάποιο αντίκτυπο στη συνολική απόδοση της εφαρμογής.

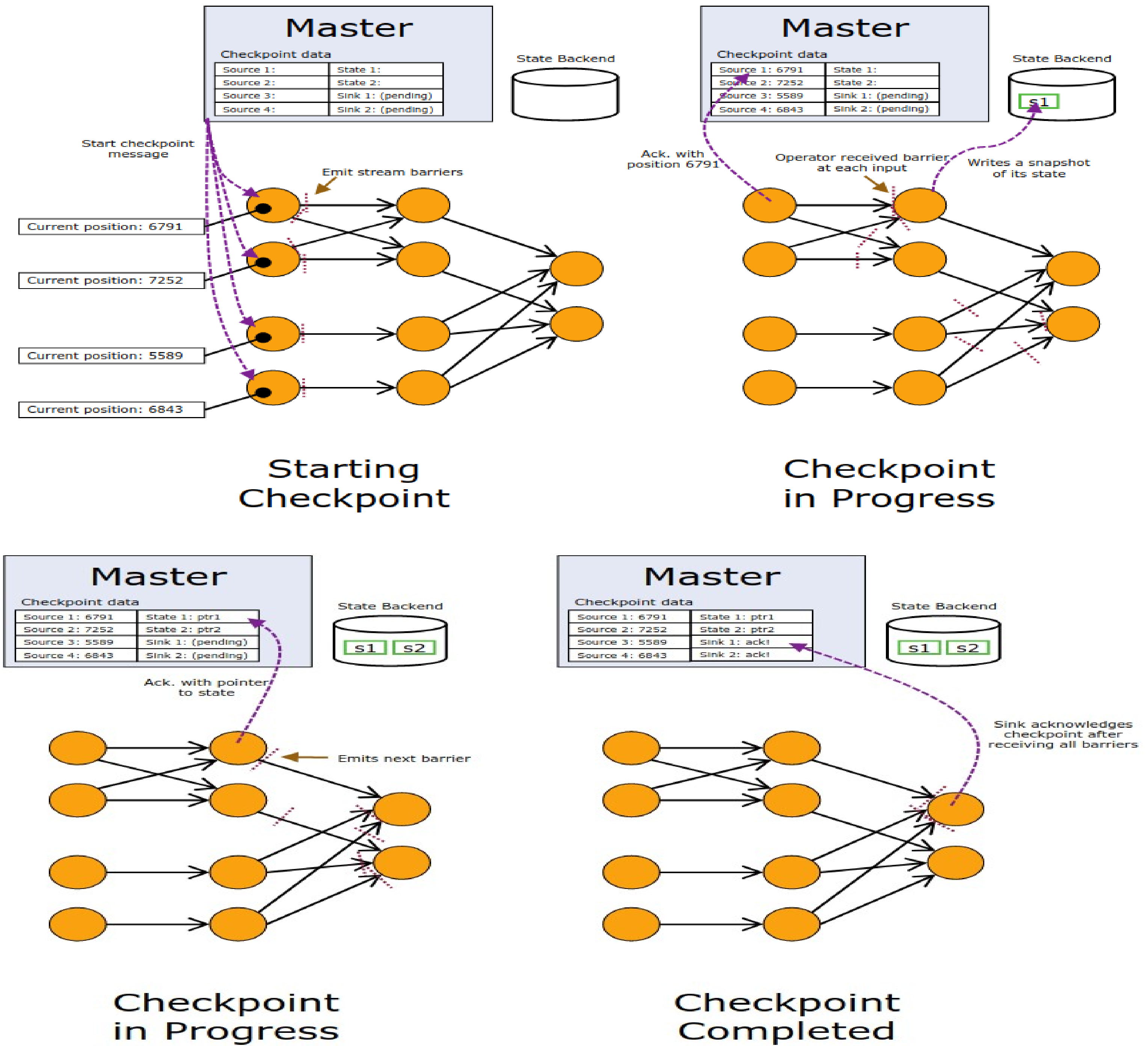
Ο μηχανισμός αυτός, που περιγράφηκε σε συντομία παραπάνω για τη δημιουργία Checkpoints, είναι γνωστός ως “Lightweight Asynchronous Snapshots for Distributed Dataflows”. Το βασικό του στοιχείο είναι τα stream barriers. Αυτά εισάγονται στη ροή των δεδομένων και ακολουθούν ολόκληρη την πορεία των εγγραφών μέσα στους διάφορους operators, χωρίς ποτέ να τις ξεπερνούν. Κάθε barrier ουσιαστικά χωρίζει τις εγγραφές σε αυτές που θα περιλαμβάνονται στο παρόν στιγμιότυπο και σε εκείνες που θα βρίσκονται στο επόμενο. Χαρακτηρίζονται πάντα από το id του στιγμιότυπου, το οποίο βοηθούν να σχεδιαστεί (Εικόνα 3-7).



Εικόνα 3-7.

Τα stream barriers εισάγονται στις πηγές σε μια θέση Sn, η οποία δηλώνει την θέση μέχρι την οποία τα δεδομένα θα καλύπτονται από το snapshot n. Η θέση αυτή για κάθε snapshot αποθηκεύεται στον Job Manager. Όταν ένας operator με πολλαπλές πηγές εισόδου λάβει ένα barrier που αφορά το snapshot n από μια πηγή, τότε σταματάει να προωθεί στην έξοδο τα αποτελέσματα των υπολογισμών πάνω στα δεδομένα της πηγής αυτής (αλλά αντίθετα τα κρατάει σε buffers) μέχρι να λάβει όλα τα barriers από όλες τις πηγές που αφορούν το snapshot αυτό. Όταν γίνει αυτό, αποθηκεύεται η κατάσταση του στο state backend (δηλαδή το αποτέλεσμα των υπολογισμών μέχρι τη στιγμή αυτή) και φυλάσσεται στον Job Manager ένας δείκτης που δείχνει στην

κατάσταση αυτή. Επίσης, τα barriers προωθούνται στο επόμενο επίπεδο μαζί και με τα δεδομένα που μέχρι πρότινος ήταν αποθηκευμένα σε προσωρινούς buffers. Όταν όλοι οι operators εξόδου (sinks) λάβουν όλα τα barrier του snapshot n, τότε αυτό αναγνωρίζεται ως έγκυρο και ο Job Manager μπορεί να ανατρέξει σε αυτό οποιαδήποτε στιγμή για να επανεκκινήσει το σύστημα με συνέπεια από τη θέση Sn. Τα παραπάνω αποτυπώνονται λεπτομερώς και στην Εικόνα 3-8.



Εικόνα 3-8.

3.2 Kafka

Το Apache Kafka είναι ένα κατανεμημένο σύστημα μεταφοράς μηνυμάτων που χρησιμοποιείται για τη συλλογή και την παράδοση μεγάλου όγκου δεδομένων με μικρές καθυστερήσεις (low latency). Το Kafka προσδίδει ανθεκτικότητα (persistence) στη ροή των μηνυμάτων, διατηρώντας τα σε δίσκους με τη μορφή logs και επιτρέπει σε πολλαπλούς καταναλωτές να διαβάζουν από τη ροή αυτή σε διαφορετικές θέσεις και με διαφορετικές ταχύτητες. Αναπτύχθηκε αρχικά από το LinkedIn και έγινε open source το 2011. Στη συνέχεια, το 2012 συμπεριλήφθηκε στα Apache Top-Level Projects.

Οι βασικοί όροι που χρησιμοποιούνται για την ανάπτυξη ενός τέτοιου συστήματος είναι οι παρακάτω:

➔ **Topic**

Για να διαχειριστεί το πλήθος των διαφορετικών ροών δεδομένων το Kafka λειτουργεί με topics. Ομαδοποιεί δηλαδή τα δεδομένα με βάση το περιεχόμενο τους και το σκοπό που θα εξυπηρετήσουν.

➔ **Partition**

Κάθε topic αποτελείται από πολλά διαμερίσματα (partitions) που συνήθως βρίσκονται σε διαφορετικούς φυσικούς κόμβους για την αποδοτικότερη διαχείριση των δεδομένων.

➔ **Record**

Οι πηγές στέλνουν τα μηνύματα στο Kafka με τη μορφή records (εγγραφών). Μια εγγραφή ουσιαστικά είναι ένα ζεύγος κλειδιού-τιμής, ενώ περιέχεται σε αυτή και το όνομα του topic στο οποίο θα συμπεριληφθεί. Το κλειδί χρησιμοποιείται για να διανεμηθούν ομοιόμορφα (ή με οποιαδήποτε λογική επιλεγεί) οι τιμές στα partitions του topic.

➔ **Broker**

Πρόκειται για τους φυσικούς κόμβους που συνθέτουν το Kafka cluster και χρησιμοποιούνται για τη συλλογή και αποθήκευση των δεδομένων.

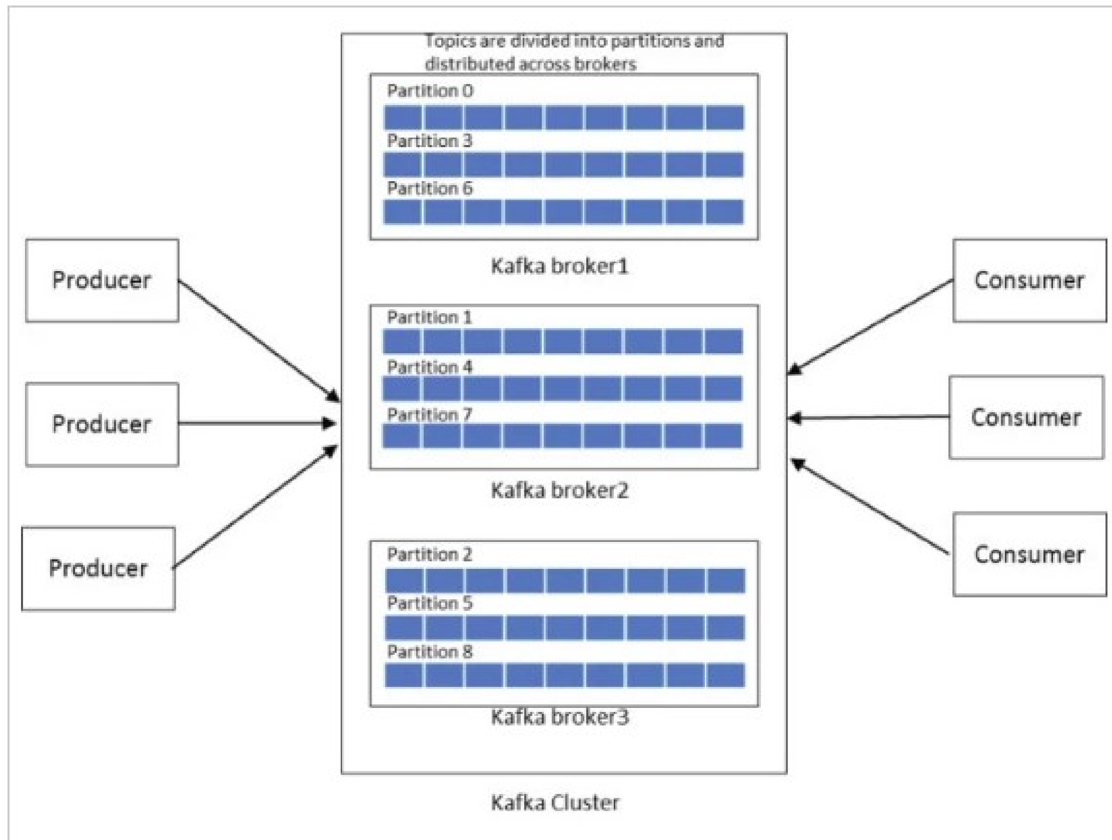
➔ **Producer**

Οι κόμβοι που παράγουν τα δεδομένα και τα στέλνουν με τη μορφή εγγραφών στους brokers στο επιλεγμένο topic.

➔ **Consumer**

Οι κόμβοι που εγγράφονται σε κάποιο Kafka topic και περιμένουν να γίνουν διαθέσιμα τα δεδομένα για την κατανάλωση τους.

Μια παραστατική παρουσίαση των παραπάνω όρων και της Kafka αρχιτεκτονικής φαίνεται στην Εικόνα 3-1.



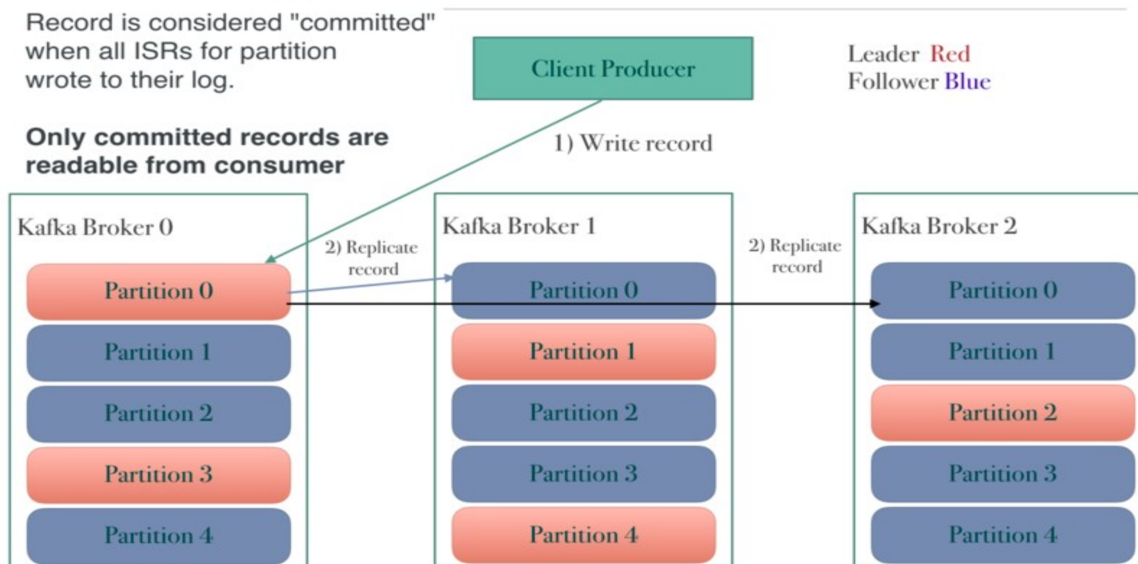
Εικόνα 3-1.

3.2.1 Kafka Brokers και Zookeeper

Όπως εξηγήθηκε και στην εισαγωγή, ένα Kafka cluster αποτελείται από κόμβους οι οποίοι αποκαλούνται brokers. Αυτοί διατηρούν στο δίσκο τους τα partitions από όλα τα δημιουργημένα topics. Έτσι όταν μια εγγραφή φτάσει στο σύστημα, ανάλογα με το topic name της και το κλειδί της, θα αποθηκευτεί στο κατάλληλο partition και θα λάβει ένα μοναδικό id που θα απεικονίζει την ακριβή θέση της μέσα στο partition. Το id αυτό αποκαλείται offset. Η λογική αυτή με τη χρήση διαμερισμάτων σε διαφορετικούς φυσικούς κόμβους για τη διαχείριση της ίδιας ροής δεδομένων επιτρέπει στο Kafka να κλιμακώνεται εύκολα, αφού δεν έχει πλέον περιορισμούς στο υλικό από έναν μόνο server, ενώ μπορεί να εξυπηρετεί ταυτόχρονα και πολλαπλούς consumers αφού αυτοί διάβάζουν εγγραφές από partitions που βρίσκονται σε διαφορετικά μηχανήματα.

Επίσης μιας και ένα από τα σημαντικότερα χαρακτηριστικά του Kafka είναι η ανθεκτικότητα, θα πρέπει να υπάρχει κάποια ασφάλεια σε περίπτωση που κάποιος κόμβος αποτύχει. Για το λόγο αυτό όταν δημιουργείται ένα topic, πέρα από τον αριθμό των partitions, δηλώνεται και ο αριθμός των κόμβων που θα κρατούν αντίγραφα αυτών (replication-factor). Έτσι για κάθε διαμέρισμα υπάρχει ένας leader server που εξυπηρετεί τις ανάγκες για γραψίματα και διαβάσματα και κάποιοι followers οι οποίοι αποθηκεύουν επίσης τις εγγραφές και ενημερώνονται τακτικά από τον leader, ώστε να

έχουν πάντα τη νεότερη έκδοση αυτών. Οι followers αυτοί είναι γνωστοί σαν ISRs (In Sync Replicas), δηλαδή κόμβοι που βρίσκονται σε συγχρονισμό όσον αφορά τα δεδομένα με τον leader. Τα παραπάνω φαίνονται στην Εικόνα 3-2.



Εικόνα 3-2.

Για να επιτευχθούν όλες οι απαιτήσεις που περιγράφηκαν παραπάνω είναι αναγκαίο να υπάρχει ένας ακόμα server που θα κρατάει όλες αυτές τις πληροφορίες για το Kafka cluster και θα προσφέρει λύσεις σε περιπτώσεις που έχουμε σφάλματα. Τον ρόλο αυτό αναλαμβάνει ο Zookeeper. Αναλυτικά, ο κόμβος αυτός καταγράφει όλα τα topics που έχουν δημιουργηθεί, τον αριθμό των partitions τους και τους brokers που είναι υπεύθυνοι για κάθε partition. Επίσης, αναπροσαρμόζει δυναμικά την τοπολογία του Kafka cluster οταν προστεθεί νέος broker, ενώ όταν ένας κόμβος αποτύχει φροντίζει να εκλέξει έναν νέο leader ανάμεσα στους ISRs και να επιτρέψει την ομαλή συνέχεια στη λειτουργία του συστήματος.

3.2.2 Kafka Producer

Για τη δημιουργία ενός Kafka Producer (του προγράμματος δηλαδή που θα τρέχει στον κόμβο) που θα προωθεί τα μηνύματα της εφαρμογής στο Kafka cluster, ένας προγραμματιστής θα πρέπει να λάβει υπόψη του αρκετές παραμέτρους για τη βέλτιστη λειτουργία και απόδοση. Οι σημαντικότερες από αυτές είναι οι παρακάτω:

Send Method

Υπάρχουν 3 διαφορετικές μέθοδοι για την επικοινωνία και τη μεταφορά των εγγραφών από τον Producer στον broker:

➔ **Synchronous send**

Κάθε φορά που αποστέλλεται μια εγγραφή το πρόγραμμα μπλοκάρει και περιμένει να πάρει επιβεβαίωση (acknowledgement) για την επιτυχή αποστολή καθώς και πληροφορίες για το partition και το offset που αυτή γράφτηκε. Κάτι τέτοιο ωστόσο μπορεί να αποδειχθεί χρονοβόρο και να μειώσει σημαντικά την απόδοση.

➔ **Asynchronous send**

Σε αυτή την περίπτωση οι εγγραφές απλώς τοποθετούνται στους buffers για αποστολή και το πρόγραμμα δεν μπλοκάρει καθόλου. Οι επιβεβαιώσεις συγκεντρώνονται ασύγχρονα (και στο background) χωρίς να επηρεάζεται σημαντικά η απόδοση.

➔ **Fire and Forget**

Όταν οι απώλειες δεδομένων δεν θεωρούνται σημαντικές μπορεί να χρησιμοποιηθεί η τακτική Fire and Forget όπου οι εγγραφές τοποθετούνται στους buffers για αποστολή και από κει και πέρα δεν λαμβάνουμε καμία πληροφορία για την πορεία τους.

Batch size

Όπως αναφέρθηκε, οι εγγραφές τοποθετούνται σε buffers ανά partition για μαζική αποστολή. Μόνο όταν κάποιος από αυτούς γεμίσει πραγματοποιείται και η αποστολή των δεδομένων. Η επιλογή λοιπόν του μεγέθους των buffers, γνωστό και ως Batch size, μπορεί να επηρεάσει σημαντικά την απόδοση ενός Kafka Producer. Ένα μεγάλο Batch size με μικρό ρυθμό παραγωγής δεδομένων θα οδηγήσει σε καθυστερήσεις, ενώ μικρό Batch size με μεγάλο ρυθμό παραγωγής θα έχει ως αποτέλεσμα μειωμένη απόδοση. Ανάλογα με τις ανάγκες της εφαρμογής είναι σημαντικό να βρεθεί το κατάλληλο μέγεθος για τη βελτιστοποίηση της λειτουργίας της.

Acks

Σε έναν Kafka Producer μπορούν να επιλεγούν 3 ειδών Acknowledgements για την αποστολή των εγγραφών:

➔ **Acks 1 (Leader)**

Με την επιλογή αυτή η επιβεβαίωση έρχεται μόλις η εγγραφή αποθηκευτεί επιτυχώς στον Leader κόμβο χωρίς να γνωρίζουμε αν ενημερώθηκαν και οι followers κόμβοι. Κάτι τέτοιο μπορεί να βοηθήσει στην ταχύτητα της εφαρμογής αλλά σε περίπτωση που ο Leader αποτύχει πριν γίνει το replication θα υπάρξει απώλεια δεδομένων.

➔ **Acks -1 (All)**

Θέτοντας το Acks=-1 παρέχεται η μεγαλύτερη δυνατή ασφάλεια ώστε να μην χαθούν καθόλου δεδομένα. Στην περίπτωση αυτή ο leader στέλνει το acknowledgment πίσω στον Producer μόνο όταν σιγουρευτεί ότι οι εγγραφές έχουν αντιγραφεί επιτυχώς σε όλους τους followers.

➔ **Acks 0 (None)**

Η επιλογή αυτή είναι χρήσιμη μόνο όταν ενδιαφέρει η μεγιστοποίηση της

απόδοσης του συστήματος και οι απώλειες δεδομένων δεν είναι κρίσιμες για τη λειτουργία της εφαρμογής. Δεν παρέχεται καμία επιβεβαίωση για την επιτυχή αποστολή και αποθήκευση των εγγραφών.

Retries

Στην περίπτωση που ο Producer δεν λάβει επιβεβαίωση για την αποστολή κάποιας εγγραφής, θα προσπαθήσει να ξαναστείλει την ίδια εγγραφή μόνο όταν η παράμετρος Retries>0.

Max In Flight Requests Per Connection

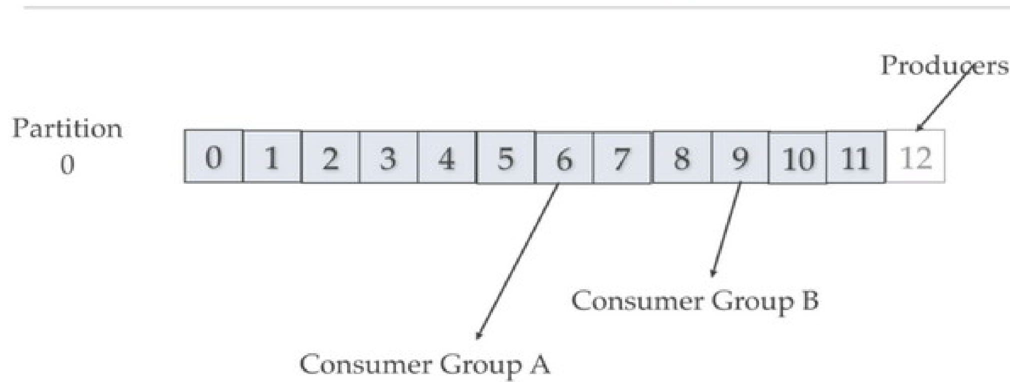
Πρόκειται για το μέγιστο αριθμό των αιτημάτων για αποστολή που μπορεί να εξυπηρετήσει ο Producer ανά partition χωρίς να έχει λάβει acknowledgments. Συνήθως επιλέγεται η τιμή 1 ώστε σε περίπτωση που η αποστολή μιας εγγραφής αποτύχει, να εξασφαλίζεται ότι δεν θα γραφτεί στο ίδιο partition πρώτα κάποια άλλη πριν γίνει προσπάθεια επαναποστολής της προβληματικής εγγραφής (αν Retries>0). Αν επιλεγεί μεγαλύτερη τιμή για την παράμετρο αυτή, η απόδοση του συστήματος αυξάνεται σημαντικά λόγω του pipelining στην εξυπηρέτηση των αιτημάτων αλλά είναι πιθανό να μην τηρείται η σωστή σειρά των εγγραφών (με βάση το χρόνο δημιουργίας) μέσα στα partitions. Ανάλογα με την κρισιμότητα και τους σκοπούς της εφαρμογής επιλέγεται η κατάλληλη τιμή.

Είναι σημαντικό επίσης να τονιστεί ότι για την επιλογή του partition που θα καταλήξει κάθε εγγραφή χρησιμοποιείται το κλειδί που συνοδεύει την τιμή της. Από προεπιλογή, οι εγγραφές που έχουν το ίδιο κλειδί οδηγούνται στο ίδιο partition, αλλά μπορεί να κατασκευαστεί και να χρησιμοποιηθεί κάποιος custom partitioner για τη διανομή των τιμών με διαφορετικό τρόπο. Σε περίπτωση που οι εγγραφές δεν εμπεριέχουν κλειδιά, μοιράζονται στα διαθέσιμα partitions με λογική round-robin.

3.2.3 Kafka Consumer

Οι κόμβοι αυτοί, όπως έχει αναφερθεί, κάνουν εγγραφή σε κάποιο topic και περιμένουν να διαβάσουν δεδομένα μόλις αυτά γίνουν διαθέσιμα στα partitions. Συνήθως οργανώνονται σε Consumer Groups ανάλογα με το σκοπό που εξυπηρετούν, τις πράξεις δηλαδή που εκτελούν πάνω στις εγγραφές. Τα partitions ενός topic μοιράζονται κυκλικά στους διαθέσιμους κόμβους του Group, έτσι ώστε ο καθένας να διαβάζει διαφορετικό μέρος των συνολικών δεδομένων. Το γεγονός αυτό επιτρέπει στο Kafka να διαχειρίζεται αποδοτικά τον όγκο των πληροφοριών και να επιτρέπει την κλιμάκωση των συστημάτων κατανεμημένης επεξεργασίας ροής δεδομένων. Στην περίπτωση που ο αριθμός των Consumers ξεπερνά αυτόν των partitions τότε αναγκαστικά κάποιοι θα μείνουν αδρανείς. Ένα σημαντικό χαρακτηριστικό του Apache Kafka είναι ότι τα δεδομένα διατηρούνται στους δίσκους των brokers για κάποιο χρονικό διάστημα (το οποίο μπορεί να ρυθμιστεί ανάλογα με τις ανάγκες του συστήματος) με αποτέλεσμα οι Consumers να έχουν τη δυνατότητα να διαβάζουν

γραμμικά από οποιοδήποτε offset επιθυμούν. Το offset αυτό είναι και το μοναδικό metadata που κρατούν οι brokers για κάθε Consumer, ώστε να επιτρέπεται και η ανάνηψη του συστήματος στις περιπτώσεις που έχουμε σφάλματα. Στην Εικόνα 3-3 παρουσιάζονται 2 Consumer Groups που διαβάζουν εγγραφές από το ίδιο partition αλλά σε διαφορετικά σημεία, ενώ την ίδια στιγμή ένας Producer γράφει νέα δεδομένα στο partition. Πρόκειται για ένα πολύ χαρακτηριστικό στιγμιότυπο της λειτουργίας του Kafka και της παραλληλοποίησης των εργασιών που προσφέρει.



Εικόνα 3-3.

3.2.4 Αξιολόγηση του Apache Kafka

Στην ενότητα αυτή θα γίνει μια προσπάθεια αξιολόγησης του Apache Kafka που θα βοηθήσει αρκετά στο σχεδιασμό του συστήματος στο επόμενο κεφάλαιο. Συγκεκριμένα θα προσδιοριστεί η επίδραση του αριθμού των brokers, του αριθμού των Producers, του μεγέθους των μηνυμάτων και του replication-factor στην απόδοση του συστήματος.

- Για το Kafka cluster έχουμε στη διάθεση μας 3 VMs του Openstack με 4VCPUs, 8GB RAM και 160GB αποθηκευτικού χώρου το καθένα.
- Για τους Producers έχουμε 2 μηχανήματα με 2VCPUs και 4GB RAM το καθένα.
- Ο Zookeeper τρέχει σε 1 VM με 1VCPU, 1GB RAM και 100GB αποθηκευτικού χώρου.

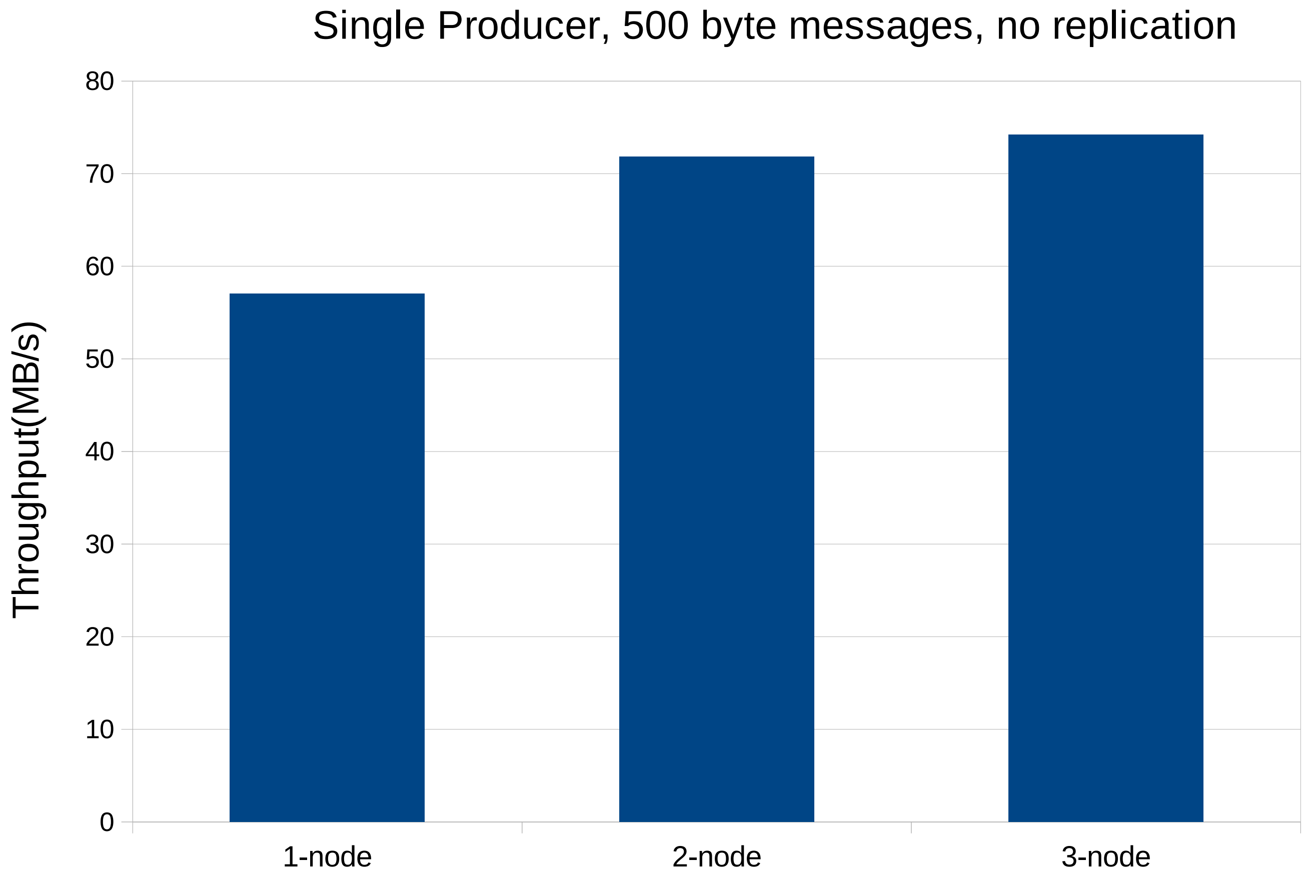
Πριν τη διεξαγωγή των πειραμάτων δημιουργείται κάθε φορά ένα topic με όνομα ¨data¨ και με 24 partitions στο Kafka cluster, όπως φαίνεται από την παρακάτω εντολή:

./bin/kafka-topics.sh --create --topic data --zookeeper localhost:2181 --partitions 24 --replication-factor <επιθυμητός αριθμός>

Στη θέση <επιθυμητός αριθμός> τοποθετείται κάθε φορά το κατάλληλο replication-

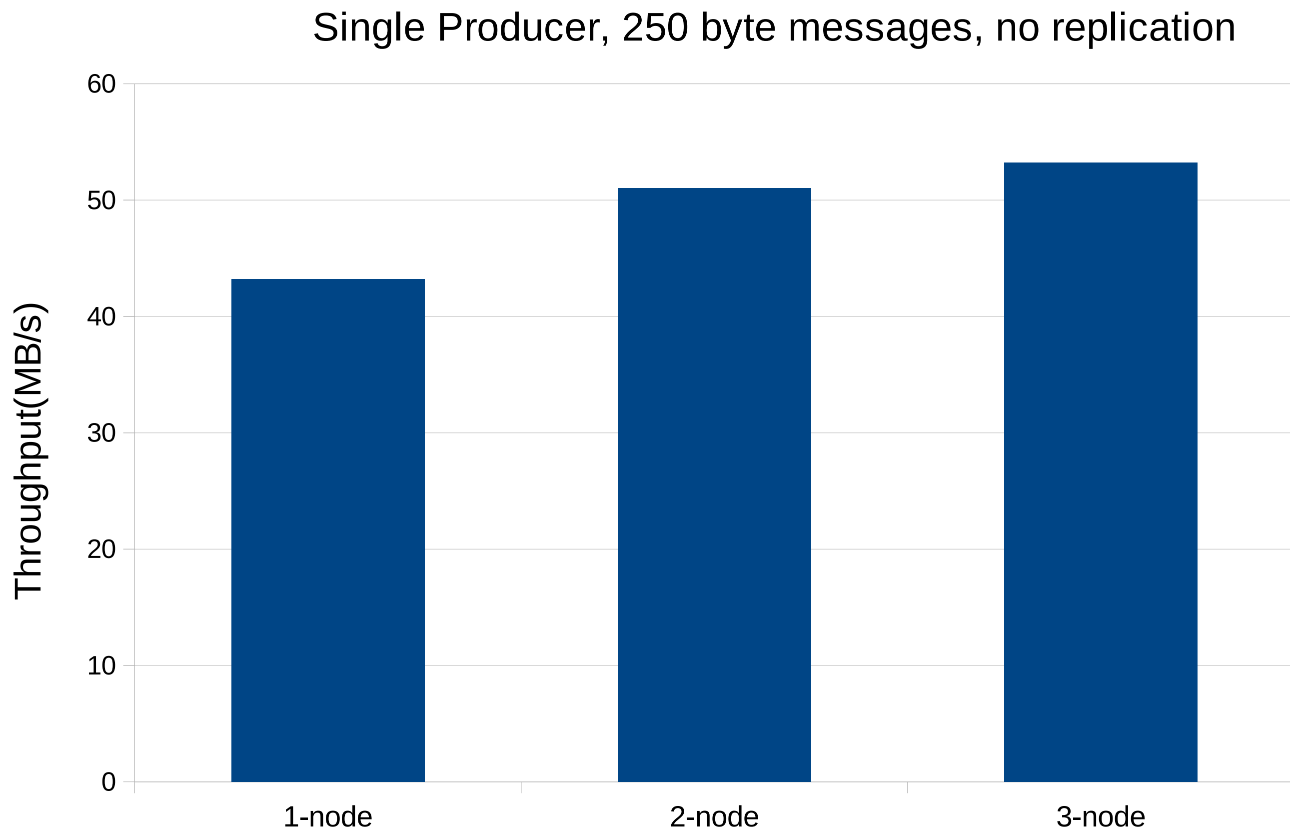
factor. Είναι επίσης σημαντικό να επισημανθεί ότι οι Producers χρησιμοποιούν **asynchronous send** και το **batch size** επιλέχθηκε ίσο με **9KB** ανα partition, η οποία παρατηρήθηκε ότι ήταν η βέλτιστη τιμή. Τέλος τα retries τέθηκαν ίσα με 3, ενώ τα acks και τα max in flight requests per connection αφέθηκαν στις default τιμές τους (1 και 5 αντίστοιχα).

➔ **1 Producer, μηνύματα μεγέθους 500 Bytes, replication-factor=1**



Διάγραμμα 3-1.

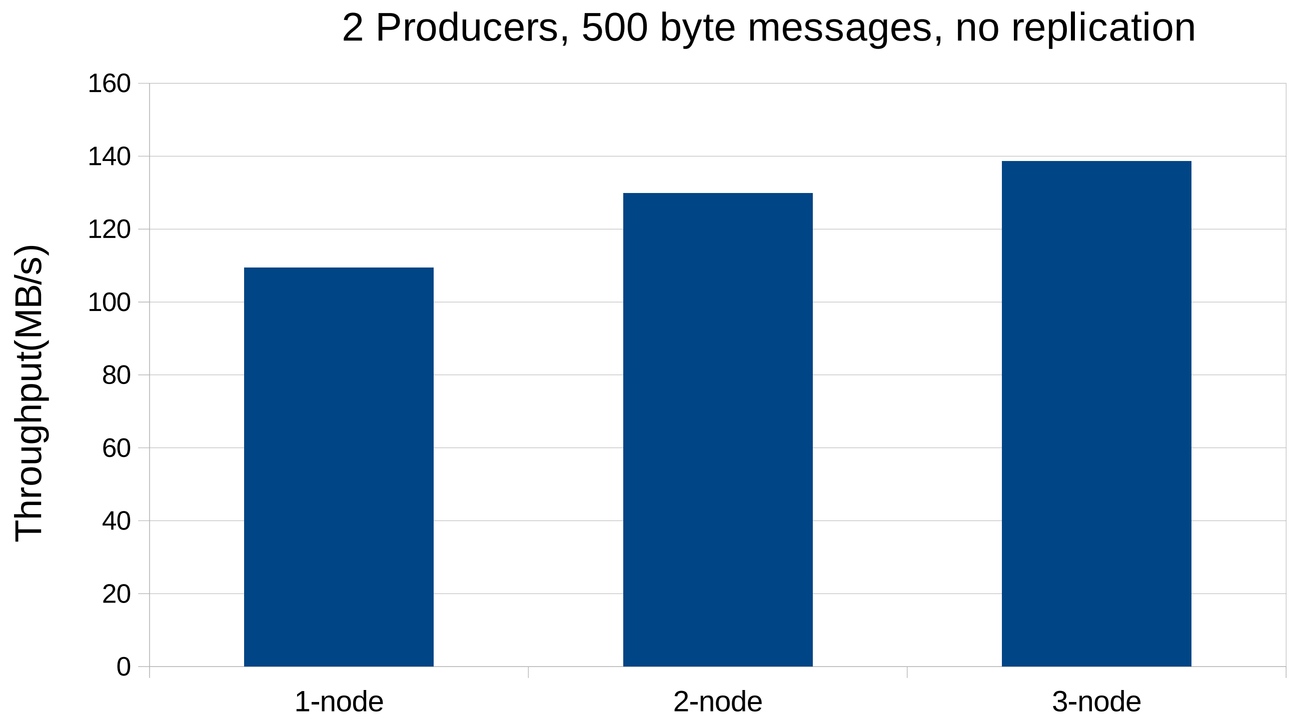
➔ **<u>1 Producer, μηνύματα μεγέθους 250 Bytes, replication-factor=1</u>**



Διάγραμμα 3-2.

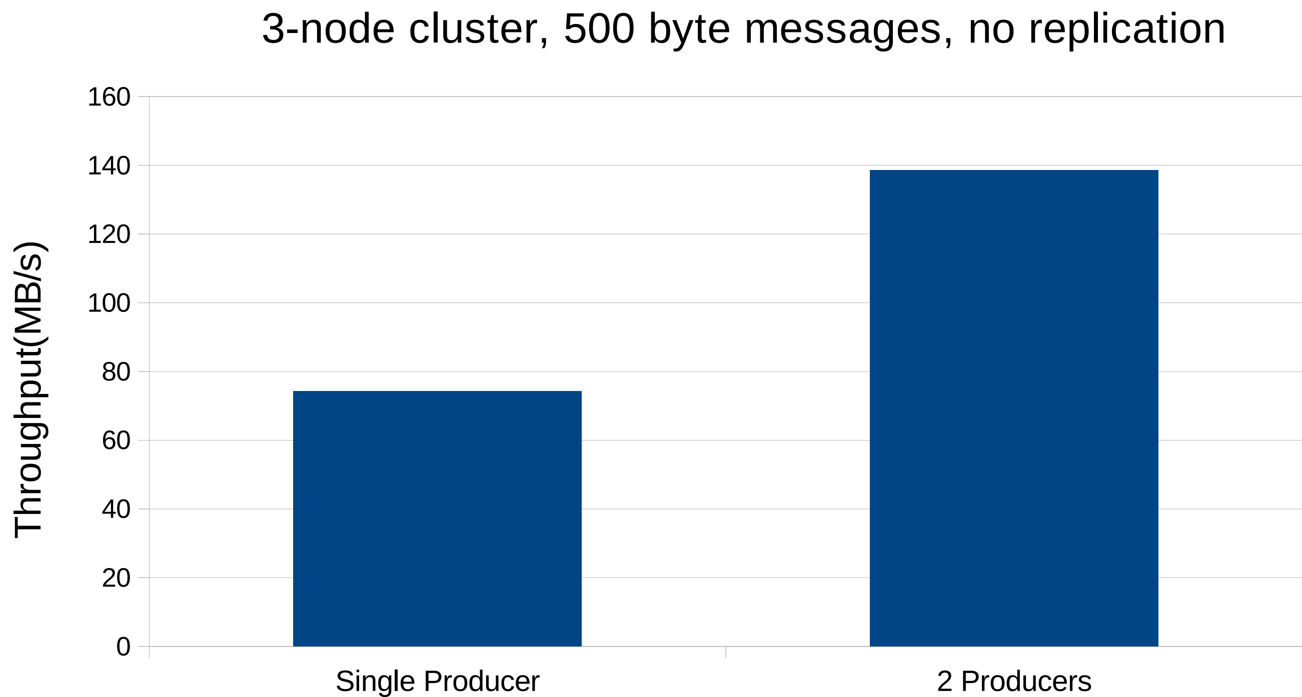
Από τα πρώτα 2 διαγράμματα φαίνεται πώς κλιμακώνεται το Apache Kafka με την προσθήκη παραπάνω brokers και το διαμοιρασμό των παραγόμενων μηνυμάτων σε μεγαλύτερο αριθμό μηχανημάτων. Με μηνύματα μεγέθους 500 Bytes το Throughput έφτασε τα 74MB/s σε cluster με 3 κόμβους, ενώ με μηνύματα 250 Bytes άγγιξε τα 54MB/s. Αυτό οδηγεί και σε ένα ακόμα συμπέρασμα για την επίδραση του μεγέθους της εγγραφής στη συνολική απόδοση. Είναι λογικό μηνύματα με μεγαλύτερο μέγεθος να έχουν ως αποτέλεσμα μεγαλύτερο συνολικό Throughput, αφού τα μηνύματα που θα προωθούνται στο cluster ανά μονάδα χρόνου θα είναι λιγότερα και κατά συνέπεια μικρότερο θα είναι και το overhead της επεξεργασίας τους.

➔ **<u>2 Producers, μηνύματα μεγέθους 500 Bytes, replication-factor=1</u>**



Διάγραμμα 3-3.

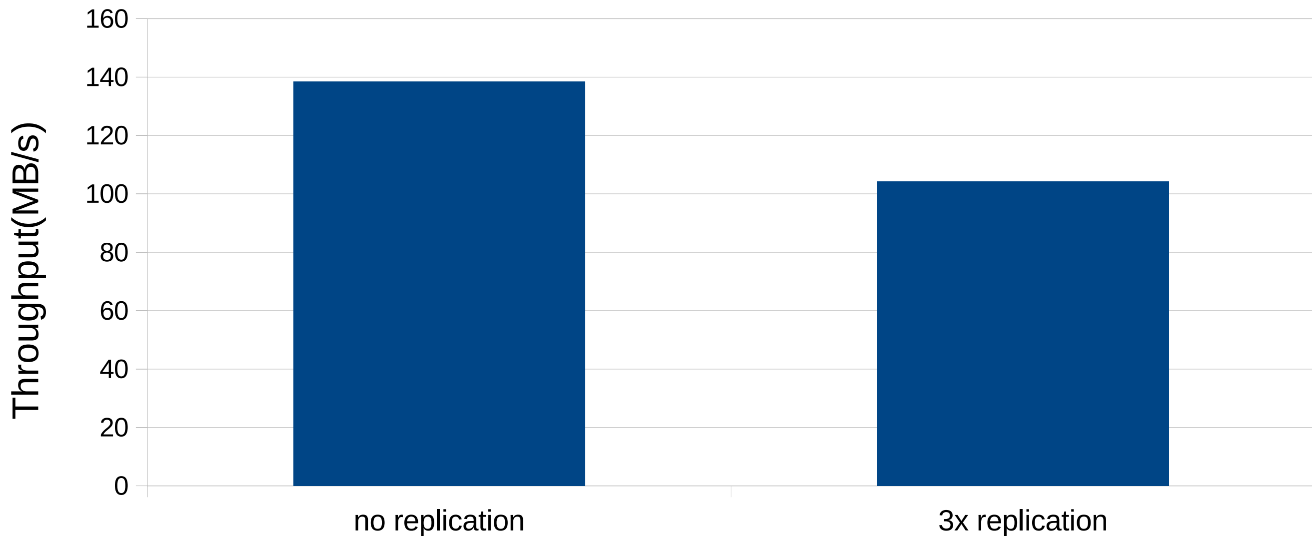
➔ **<u>3 Brokers, μηνύματα μεγέθους 500 Bytes, replication-factor=1</u>**



Διάγραμμα 3-4.

Με την προσθήκη ενός ακόμη Producer και τη διατήρηση του μέγεθους των μηνυμάτων στα 500 Bytes, το Throughput σχεδόν διπλασιάζεται σε κάθε περίπτωση φτάνοντας τα 139MB/s για 3-node cluster. Είναι σημαντικό εδώ να διευκρινιστεί ότι το network bandwidth για τον κάθε Producer στα πειράματα μας ήταν 1Gbps, κάτι το οποίο μπορεί να αποτελέσει bottleneck στην αύξηση της απόδοσης. Με την αύξηση επομένως των μηχανημάτων που παράγουν δεδομένα έχουμε σημαντική κλιμάκωση και στο σύστημα.

➔ **2 Producers, μηνύματα μεγέθους 500 Bytes, 3 brokers**



Διάγραμμα 3-5.

Το τελευταίο πείραμα είχε ως στόχο να δείξει το κόστος της χρήσης replicas στην απόδοση του Apache Kafka. Από το διάγραμμα είναι προφανές ότι άμα επιθυμούμε τα δεδομένα του κάθε broker να αντιγράφονται στους άλλους 2 για ασφάλεια, τότε η μείωση στο συνολικό Throughput θα είναι αισθητή. Συγκεκριμένα, το κόστος για το replication-factor=3 υπολογίστηκε στο 25%.

Κεφάλαιο 4

Σχεδίαση του συστήματος

4.1 Εισαγωγή

Σκοπός της διπλωματικής αυτής είναι να κατασκευαστεί ένα μοντέλο που θα μπορεί να προβλέψει για οποιαδήποτε εργασία σε ένα Flink cluster το μέγιστο throughput που μπορεί να αντέξει το σύστημα (δηλαδή τα MB/s των δεδομένων που εισέρχονται σε αυτό) χωρίς να αποτύχει, καθώς και τις καθυστερήσεις στους υπολογισμούς (latencies) όταν το σύστημα έχει τη μέγιστη απόδοση. Για να επιτευχθεί κάτι τέτοιο είναι απαραίτητο να εκτελεστεί ένας μεγάλος αριθμός πειραμάτων για διαφορετικές εργασίες με διαφορετικές ροές δεδομένων στην είσοδο και με διαφορετικές εσωτερικές παραμέτρους του Flink, ώστε να δημιουργηθεί ένα σετ δεδομένων ικανό να δώσει ακριβείς προβλέψεις με τη χρήση αλγορίθμων μηχανικής μάθησης. Θα πρέπει επομένως να σχεδιαστεί ένα σύστημα που θα απομονώνει το Flink από εξωτερικούς παράγοντες που μπορούν να επηρεάσουν τη λειτουργία του και να μειώσουν την αποδοτικότητα του και να επιλεχθούν οι κατάλληλες παράμετροι προς εξέταση σε κάθε περίπτωση.

4.2 Δομικά Στοιχεία

Το ζητούμενο σύστημα θα αποτελείται από 3 δομικά στοιχεία που περιγράφηκαν αναλυτικά σε προηγούμενα κεφάλαια, τους Producers, το Kafka cluster και το Flink cluster. Είχαμε στη διάθεση μας 30 VCPUs, 50GB RAM και 1TB αποθηκευτικού χώρου να τα διαμοιράσουμε σε 10 virtual machines του Openstack.

4.2.1 Στήσιμο των Producers

- **2VMs με 2VCPUs και 4GB RAM το καθένα**

Σκοπός των Producers στην περίπτωση μας ήταν να παράγουν όσο το δυνατόν περισσότερα δεδομένα ανά μονάδα χρόνου ώστε να μπορούν να στρεσάρουν κάθε διαφορετική εργασία και με διαφορετικές παραμέτρους που εκτελούνταν στο Flink cluster. Για τη βελτιστοποίηση της παραγωγής επιλέχθηκαν οι παρακάτω παράμετροι που χρησιμοποιήθηκαν και στην **Ενότητα 3.2.4** για την αξιολόγηση του Apache Kafka:

➔ **<u>Asynchronous send</u>**

Προτιμήθηκε από το Synchronous send για τη μείωση των καθυστερήσεων και

από το Fire and Forget για την ειδοποίηση σε περίπτωση αποτυχίας προώθησης των δεδομένων.

- ➔ **Batch size = 9KB**
  Προσαρμόστηκε ανά partition (επιλέχθηκαν 24) ώστε να βελτιστοποιείται η προώθηση των δεδομένων στους brokers του Kafka cluster.

- ➔ **Acks = 1**
  Τα δεδομένα μας δεν είναι τόσο κρίσιμα ώστε να απαιτείται να υπάρχουν διαθέσιμα αντίγραφα αυτών, οπότε η επιβεβαίωση της εγγραφής τους στον Leader κόμβο κρίθηκε αρκετή.

- ➔ **Retries = 3**
  Ένας ικανοποιητικός αριθμός προσπαθειών για επαναποστολή εγγραφών σε περίπτωση αποτυχίας.

- ➔ **Max in Flight Requests Per Connection = 5**
  Επιλέχθηκε με στόχο να αυξηθεί ο ρυθμός αποστολής των εγγραφών ανά μονάδα χρόνου. Όπως εξηγήθηκε στο προηγούμενο κεφάλαιο, Max in Flight Requests Per Connection > 0 σημαίνει ότι μπορεί να επηρεαστεί το ordering των εγγραφών ανα partition, αλλά στη συγκεκριμένη περίπτωση αυτό δεν επηρεάζει την εγκυρότητα των πειραματικών αποτελεσμάτων.

Για τη δημιουργία των Producers χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java. Παρακάτω παρουσιάζεται και ένα στιγμιότυπο από τη συνάρτηση που ήταν υπεύθυνη για τον ορισμό των παραμέτρων κατά τη δημιουργία τους.

```java
private static Producer<Long,FinalProducerRecord> createProducer() {
/* a producer needs the addresses of brokers,a client id,the name of the topic, key-value
and their serialization shchema
 */
    Properties props = new Properties();

    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, BOOTSTRAP_SERVERS);
    //id to pass to the server when making requests so the server can track the source of requests
    // beyond just IP/port by passing a producer name for things like server-side request logging.
    props.put(ProducerConfig.CLIENT_ID_CONFIG, v: "KafkaProducer");

    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, LongSerializer.class.getName());

    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, FinalProducerSerializer.class.getName());
    //Batch up to 9K buffer sizes.
    props.put(ProducerConfig.BATCH_SIZE_CONFIG,  v: 9000);
    //Set the number of retries - retries
    props.put(ProducerConfig.RETRIES_CONFIG, v: 3);
    // - max.in.flight.requests.per.connection (default 5)
    props.put(ProducerConfig.MAX_IN_FLIGHT_REQUESTS_PER_CONNECTION,
            v: 5);
    //Only retry after one second.
    props.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, v: 1_000);

    return new KafkaProducer<>(props);
}
```
Εικόνα 4-1.

Στην περίπτωση μας οι εγγραφές που προωθούνται από τους Producers δημιουργούνται και από αυτούς. Συγκεκριμένα, αποφασίστηκε αυτές να έχουν μέγεθος 500 Bytes ώστε να ταυτίζονται όσο είναι δυνατό με αληθινές περιπτώσεις του έξω κόσμου. Η μορφή τους επιλέχθηκε να είναι αυτή που παρουσιάζεται στον Πίνακα 4-1.

Το πεδίο **nameofsomething** αποτελεί ουσιαστικά το αναγνωριστικό της εγγραφής και παράγεται τυχαία από γεννήτρια, το **numberofsomething** (ακέραιος από 1-1000) αναπαριστά έναν αριθμό που σχετίζεται με αυτή και θα χρησιμοποιηθεί στους υπολογισμούς, ενώ τα **2 timestamps** εισήχθησαν για την λήψη μετρικών του συστήματος. Το πρώτο δηλώνει το χρόνο που δημιουργήθηκε η εγγραφή (Event-time), ενώ το δεύτερο το χρόνο που αυτή έφτασε στον κόμβο του Flink cluster για την επεξεργασίας της (Ingestion-time). Τέλος, το πεδίο **text** περιέχει ένα κείμενο σταθερού μεγέθους Bytes που χρησιμοποιήθηκε μονάχα για την αύξηση του συνολικού μεγέθους της εγγραφής, ώστε να φτάσει τα 500 Bytes.

String nameofsomething	Ανάλογα με το πόσες διαφορετικές εγγραφές επιθυμούμε να υπάρχουν σε κάθε πείραμα μας παράγονται ομοιόμορφα και τα αντίστοιχα μοναδικά αναγνωριστικά (keys)
int numberofsomething	1-1000
long timestamp1	Event-time
long timestamp2	Ingestion-time
String text	Τυχαίοι χαρακτήρες για να συμπληρωθεί το μέγεθος των 500Bytes

Πίνακας 4-1.

Την κάθε εγγραφή (value) συνοδεύει το topic στο οποίο θα συμπεριληφθεί και ένα κλειδί (key) σύμφωνα με το οποίο θα επιλεγεί το αντίστοιχο partition. Για να μοιραστούν ισόποσα οι εγγραφές μας στους brokers αποφασίστηκε το κλειδί αυτό να

4.2.2 Στήσιμο του Kafka cluster

- **Brokers: 3VMs με 4VCPUs, 8GB RAM και 180GB αποθηκευτικού χώρου**
- **Zookeeper: 1VM με 1VCPU, 1GB RAM και 100GB αποθηκευτικού χώρου**

Στα 4 μηχανήματα αυτά εγκαταστάθηκε το Kafka framework και έγιναν τα κατάλληλα configurations για να εξασφαλιστεί η ομαλή επικοινωνία του Kafka cluster με τα υπόλοιπα στοιχεία του συστήματος. Η σημαντικότερη ρύθμιση αφορούσε το retention policy των Kafka Brokers. Πρόκειται για την πολιτική βάσει της οποίας οι Brokers αποφασίζουν για πόσο χρονικό διάστημα ή πόσο όγκο δεδομένων θα κρατούν στο δίσκο τους για λόγους ανοχής σε σφάλματα αλλά και ικανότητας επανακκίνησης του συστήματος από παρελθοντικά σημεία. Ο δίσκος των κόμβων του Kafka cluster ορίστηκε να έχει μέγεθος 180GB, οπότε επιλέχθηκε τα log files που περιέχουν όλες τις εγγραφές που καταφθάνουν από τους Producers να ανανεώνονται όταν αυτός φτάσει σε πληρότητα 50%. Όταν λοιπόν ο δίσκος κάποιου Broker φτάσει τα 90GB δεδομένων, αρχίζουν να διαγράφονται αρχεία με εγγραφές προκειμένου να ελευθερωθεί χώρος και

να εξασφαλιστεί η ομαλή λειτουργία του συστήματος.

4.2.3 Στήσιμο του Flink cluster

- **Job Manager: 1VM με 2VCPUs, 4GB RAM και 40GB αποθηκευτικού χώρου**
- **Task Managers: 3VMs με 2VCPUs, 4GB RAM και 40GB αποθηκευτικού χώρου το καθένα**

Στα μηχανήματα αυτά εγκαταστάθηκε το Flink framework και επίσης έγιναν οι κατάλληλες ρυθμίσεις για να επιτρέπεται η επικοινωνία των Task Managers με τον Job Manager (passwordless ssh). Η σημαντικότερη σχεδιαστική απόφαση αφορούσε την επιλογή του αριθμού των task slots ανά Task Manager. Εφόσον τα μηχανήματα μας διαθέτουν 2VCPUs το καθένα, αυτός ήταν και ο αριθμός που αποφασίστηκε να εφαρμοστεί για τα πειράματα μας.

4.3 Flink Jobs

Το επόμενο βήμα μετά την ολοκλήρωση του στησίματος είναι να συνταχθούν τα προγράμματα επεξεργασίας δεδομένων που θα τρέχουν στα task slots των Flink Task Managers. Όπως τονίστηκε και στην αρχή αυτού του κεφαλαίου, απώτερος στόχος είναι η πραγματοποίηση προβλέψεων για το MST και τα Latencies κάθε εργασίας που εκτελείται σε Flink cluster. Για το λόγο αυτό, επιλέχτηκαν οι 3 πιο συνηθισμένες διεργασίες που μπορούν να εφαρμοστούν πάνω σε δεδομένα, η καθεμία από τις οποίες έχει διαφορετικές απαιτήσεις σε υλικό (hardware) για να πραγματοποιηθεί. Με τις μετρήσεις που θα ληφθούν από τα πειράματα του επόμενου κεφαλαίου, πάνω στις εργασίες αυτές με διαφορετικές παραμετροποιήσεις για το Flink cluster και το είδος των δεδομένων, θα επιδιώξουμε στη συνέχεια με τη βοήθεια μοντέλων μηχανικής μάθησης να εκπληρώσουμε το σκοπό της παρούσας διπλωματικής κάνοντας εκτιμήσεις για οποιοδήποτε πλέον Flink σύστημα που εκτελεί κάποια από τις παρακάτω επεξεργασίες δεδομένων.

4.3.1 Filtering

Πρόκειται για ένα απλό φιλτράρισμα στις εγγραφές ανάλογα με το **nameofsomething** που παριστάνει το κλειδί τους. Στο παράδειγμα της Εικόνας 4-2 το πεδίο αυτό, έχοντας οριστεί στους Producers να παίρνει τιμές από ‘a-z’, συγκρίνεται με το ‘m’ και επιτρέπεται η προώθηση μόνο των μηνυμάτων με κλειδί από ‘n-z’.

```
//OPERATORS--->FILTER
DataStream<FinalProducerRecord> filteredstream=environment
        .addSource(flinkKafkaConsumer)
        .filter(new FilterFunction<FinalProducerRecord>() {
            @Override
            public boolean filter(FinalProducerRecord value) throws Exception {
                return (value.getNameofsomething().compareTo("m")<0);
            }
        });


//SINK
DataStreamSink<FinalProducerRecord> outputstream= filteredstream
        .map(new MetricsExposingMapFunction_filter())
        .addSink(new DiscardingSink<>());

environment.execute( jobName: "Filtering");
```

Εικόνα 4-3.

4.3.2 Aggregation on Windows

Το συγκεκριμένο job έχει ουσιαστικά ως στόχο να υλοποιήσει ένα συγκεκριμένο υπολογισμό πάνω σε ένα σύνολο εγγραφών που έχουν συγκεντρωθεί σε ένα παράθυρο με χρονικά όρια. Στην περίπτωση μας, όπως φαίνεται και στην Εικόνα 4-3, οι εγγραφές με βάση το κλειδί τους (**nameofsomething**) τοποθετούνται σε αντίστοιχα Tumbling Windows με μέγεθος 60s, όπου υπολογίζεται ο μέσος όρος των **numberofsomething** τους. Είναι σημαντικό εδώ να επισημανθεί ότι οι υπολογισμοί αυτοί γίνονται με βάση το Event-Time των μηνυμάτων, που σημαίνει ότι τα παράθυρα αντιλαμβάνονται το πέρασμα του χρόνου μέσω Watermarks (όπως εξηγήθηκε στο κεφάλαιο 2.3.3) που ρέουν στο σύστημα και περιέχουν τα timestamps δημιουργίας των εγγραφών.

```
//OPERATORS
DataStream<Tuple4<String,Double, Long, Long>> aggregatestream=environment
        .addSource(flinkKafkaConsumer) DataStreamSource<FinalProducerRecord>
        .keyBy(FinalProducerRecord::getNameofsomething) KeyedStream<FinalProducerRecord,String>
        .window(TumblingEventTimeWindows.of(Time.seconds(60))) WindowedStream<FinalProducerRecord,String,TimeWindow>
        .aggregate(new Average());

//SINK
DataStreamSink<Tuple4<String, Double, Long, Long>> outputstream= aggregatestream
        .map(new MetricsExposingMapFunction_aggregate())
        .addSink(new DiscardingSink<>());

environment.execute( jobName: "Aggregation");
```

Εικόνα 4-4.

4.3.3 Window joins

Η τελευταία περίπτωση και πιο απαιτητική για ένα κατανεμημένο σύστημα επεξεργασίας δεδομένων είναι τα Window joins. Η προσέγγιση που ακολουθήθηκε για να προσομοιωθεί μια τέτοια εργασία ήταν η είσοδος στο Flink cluster 2 ροών εγγραφών με την ίδια μορφή που προέρχονται όμως από διαφορετικές πηγές. Οι εγγραφές που έχουν κοινό **nameofsomething** κατευθύνονται στο ίδιο Tumbling Window και εφαρμόζεται για κάθε ζεύγος που ισχύει **RecordOfInputStream1.nameofsomething = RecordOfInputStream2.nameofsomething** ένας υπολογισμός. Στην δικιά μας περίπτωση εκτελείται μια απλή σύγκριση (Εικόνα 4-4). Η μεγάλη διαφορά της εργασίας αυτής με τα Window aggregations είναι ότι τα διάφορα παράθυρα που δημιουργούνται λόγω των πολλαπλών κλειδιών διατηρούνται στη μνήμη μέχρι να κλείσουν και να εκτελεστούν οι επιθυμητές πράξεις μεταξύ όλων των πιθανών συνδυασμών (Εικόνα 4-4). Αντίθετα, στην προηγούμενη περίπτωση με τον υπολογισμό του μέσου όρου, οι εγγραφές με το που εισέλθουν στο παράθυρο προστίθεται η τιμή του **numberofsomething** τους στο συνολικό άθροισμα και στη συνέχεια δεν είναι αναγκαίες για τη λήψη του τελικού αποτελέσματος αφού με το κλείσιμο του παραθύρου απαιτείται απλώς μια διαίρεση με το συνολικό αριθμό τους. Έτσι στη μνήμη διατηρούνται μόνο τα προσωρινά συνολικά αθροίσματα. Αυτό το στοιχείο είναι που καθιστά τα Window joins memory intensive εφαρμογές. Να τονιστεί ότι και στην εργασία αυτή τα παράθυρα αντιλαμβάνονται το πέρασμα του χρόνου με βάση το Event-time, ένα πολύ σημαντικό χαρακτηριστικό για τη λήψη εγκυρότερων αποτελεσμάτων.

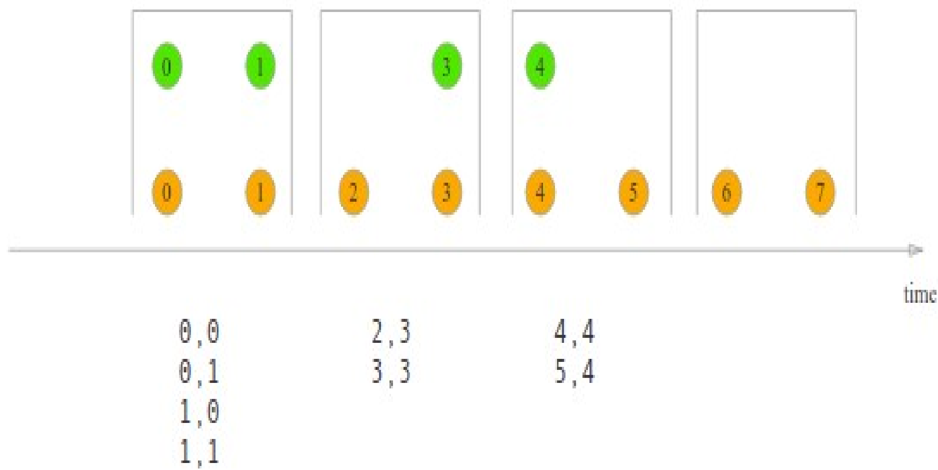
```
//OPERATORS
DataStream<FinalProducerRecord> inputStream1 = environment.addSource(flinkKafkaConsumer1);
DataStream<FinalProducerRecord> inputStream2 = environment.addSource(flinkKafkaConsumer2);

DataStream<FinalProducerRecord> joinedstream=inputStream1
        .join(inputStream2) JoinedStreams<FinalProducerRecord,FinalProducerRecord>
        .where(FinalProducerRecord::getNameofsomething).equalTo(FinalProducerRecord::getNameofsomething) Joined
        .window(TumblingEventTimeWindows.of(Time.seconds(10))) WithWindow<FinalProducerRecord,FinalProducerRecord,String,T
        .apply(new JoinFunction <FinalProducerRecord,FinalProducerRecord,FinalProducerRecord>() {

                @Override
                public FinalProducerRecord join(FinalProducerRecord first,FinalProducerRecord second) {
                    if (first.getTimestamp2() <= second.getTimestamp2())
                        return first;
                    else
                        return second;
                }
            });
//SINK
DataStreamSink<FinalProducerRecord> outputstream= joinedstream
        .map(new MetricsExposingMapFunction_join())
        .addSink(new DiscardingSink<>());

environment.execute( jobName: "Join");
```

Εικόνα 4-5.



Εικόνα 4-6.

4.4 Μετρικές

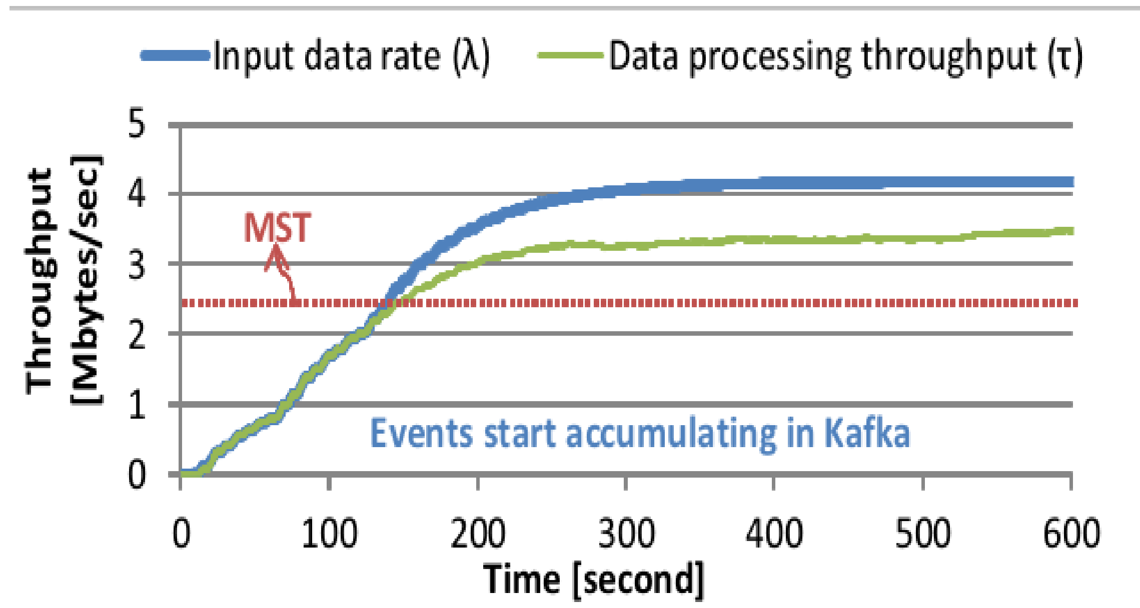
4.4.1 Υπολογισμός Maximum Sustainable Throughput(MST)

Όπως έχει ήδη τονιστεί, στόχος είναι να εκτελεστεί μια σειρά πειραμάτων ώστε να υπολογιστεί το MST για διαφορετικές εργασίες σε Flink Cluster και με διαφορετικές παραμέτρους. Με τον τρόπο αυτό θα δημιουργηθεί ενα σετ δεδομένων ικανό να λειτουργήσει σαν είσοδος σε μοντέλα μηχανικής μάθησης για την παραγωγή προβλέψεων. Με τον όρο Maximum Sustainable Throughput (MST) εννοούμε το μέγιστο φόρτο δεδομένων ανά μονάδα χρόνου που μπορεί να χειριστεί το σύστημα χωρίς να γεμίσουν οι ουρές του και να εμφανιστούν καθυστερήσεις στην είσοδο νέων δεδομένων για την επεξεργασία των παλιών. Όταν σε ένα σύστημα κατανεμημένης επεξεργασίας δεδομένων, όπως το Flink, ο όγκος των δεδομένων που εισέρχεται στο σύστημα είναι μεγαλύτερος από αυτόν που μπορεί να επεξεργαστεί, εμφανίζεται ένας μηχανισμός γνωστός ως backpressure. Αυτός ο μηχανισμός ουσιαστικά, ελαττώνει το ρυθμό εισαγωγής εγγραφών στο σύστημα μέχρι να προχωρήσουν οι υπολογισμοί και να δημιουργηθεί χώρος στις ουρές. Κάτι τέτοιο ωστόσο μπορεί να προκαλέσει σοβαρές καθυστερήσεις και δεν είναι επιθυμητό.

Για τον υπολογισμό του MST σε κάθε περίπτωση εφαρμόστηκαν οι παρακάτω μεθοδολογίες ανάλογα με το είδος της εργασίας προς εκτέλεση στο Flink Cluster.

➔ **Network intensive job**

Η λογική που ακολουθήθηκε για τέτοιου είδους εργασίες ήταν η σταδιακή αύξηση του ρυθμού παραγωγής των δεδομένων με την προσθήκη Producers, μέχρι να αρχίσουν να συνωστίζονται εγγραφές στις ουρές του Kafka και ο ρυθμός παραγωγής να ξεπεράσει το ρυθμό κατανάλωσης. Στο σημείο ακριβώς που συμβαίνει αυτό εντοπίζεται και το MST του συγκεκριμένου συστήματος. Τιμές για ρυθμό εισόδου εγγραφών μεγαλύτερο από το MST θα οδηγήσουν τελικά στην εξάντληση του αποθηκευτικού χώρου του Kafka cluster, αφού ο όγκος των δεδομένων που θα αναγκάζεται να διατηρεί για επεξεργασία συνεχώς θα μεγαλώνει.

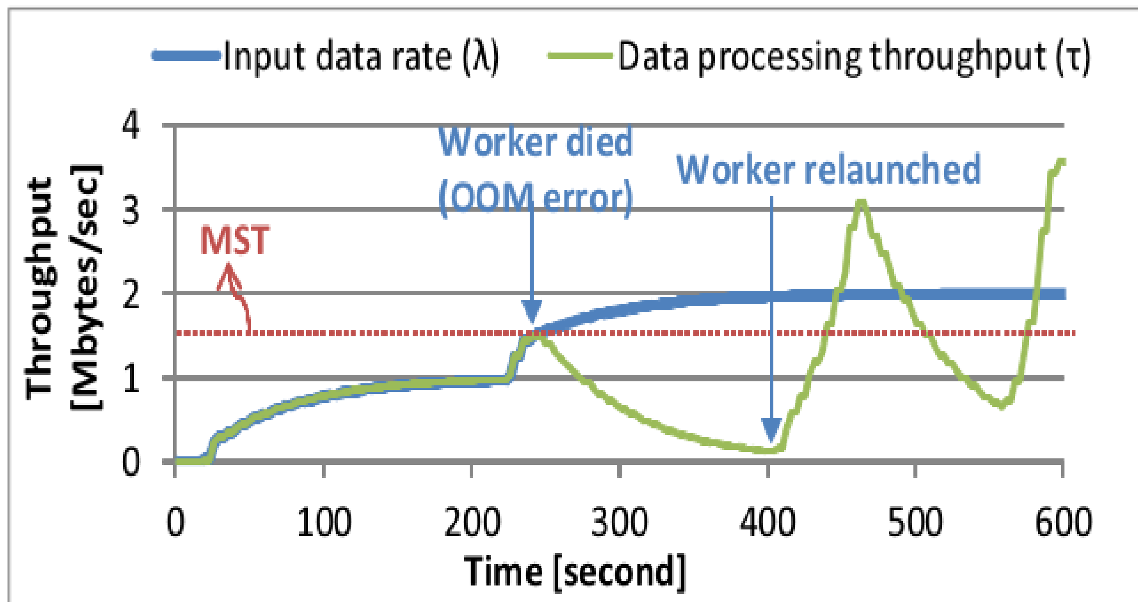


Εικόνα 4-7. Στην εικόνα φαίνεται ένα απλό παράδειγμα εφαρμογής της παραπάνω μεθοδολογίας. Στα πρώτα 130 δευτερόλεπτα και ενώ ο ρυθμός εισόδου δεδομένων στο σύστημα αυξάνεται, το σύστημα είναι ικανό να απορροφήσει το φόρτο αυτό και να εκτελέσει τους υπολογισμούς χωρίς ιδιαίτερες καθυστερήσεις. Από εκεί και έπειτα ωστόσο, οι 2 ρυθμοί ακολουθούν διαφορετικές πορείες και το Flink Cluster δεν μπορεί να διαχειριστεί τον όγκο των δεδομένων με αποτέλεσμα να συνωστίζεται στις ουρές.

➔ **Memory intensive job**

Σε εφαρμογές που η επεξεργασία των δεδομένων έχει υψηλές απαιτήσεις σε μνήμη, για να εντοπίσει κανείς το MST θα πρέπει να αυξάνει σταδιακά το ρυθμό παραγωγής μέχρι κάποιος κόμβος του Flink Cluster ή και παραπάνω να αποτύχουν λόγω μη διαθέσιμης επιπλέον μνήμης. Αυτό που θα ακολουθήσει στη συνέχεια είναι μια προσπάθεια του συστήματος να επανεκκινήσει τους αποτυχόντες κόμβους, ώστε να συνεχιστεί ομαλά η εκτέλεση των υπολογισμών.

Οι task managers θα επιδιώξουν να απορροφήσουν όλο τον όγκο των εγγραφών που έχασαν ώστε να επανέλθει η ισορροπία και τελικά θα οδηγηθούν ξανά σε αποτυχία από τους περιορισμούς της μνήμης. Το MST εντοπίζεται λίγο πριν το πρώτο σφάλμα μνήμης.



Εικόνα 4-8. Εντοπίζεται η πρώτη αποτυχία του worker κόμβου στα 250 δευτερόλεπτα (Out of Memory) , έπειτα επανεκκινείται στα 400 δευτερόλεπτα με σκοπό να απορροφήσει όλα τα επιπλέον δεδομένα και στα 480 δευτερόλεπτα οδηγείται και σε δεύτερο σφάλμα. Ανάλογα με τις φορές που έχουμε ορίσει να επανεκκινείται το Flink cluster σε περιπτώσεις σφαλμάτων, η εικόνα αυτή θα συνεχιστεί μέχρι τον ολικό τερματισμό του συστήματος.

Για τη λήψη των παραπάνω μετρικών, δηλαδή του ρυθμού εισόδου των δεδομένων στο Kafka και του ρυθμού επεξεργασίας αυτών από το Flink (ρυθμός εξόδου από το Kafka), σε πραγματικό σχεδόν χρόνο ήταν απαραίτητη η χρήση ενός monitoring εργαλείου που θα προσφέρει τη δυνατότητα οπτικοποίησης των μετρήσεων για την εξαγωγή έγκυρων συμπερασμάτων. Επιλέχθηκε λοιπόν για το σκοπό αυτό το **Datadog monitoring service**. Η λειτουργία του βασίζεται στην εγκατάσταση agents στα μηχανήματα που δομούν το Kafka cluster (brokers), οι οποίοι παρακολουθούν και αντλούν συνεχώς μετρήσεις από αυτά. Είναι σημαντικό εδώ να τονιστεί ότι τα Kafka metrics είναι από προεπιλογή διαθέσιμα μέσω του Java Management Extensions (JMX) interface σε πόρτα του μηχανήματος η οποία πρέπει να οριστεί από το χρήστη. Ο datadog agent επομένως είναι λογισμικό υπεύθυνο να πάρει τις μετρήσεις αυτές από το JMX και να τις προωθήσει στο Datadog, όπου πραγματοποιείται το visualization αυτών. Ένα στιγμιότυπο από το περιβάλλον που δημιουργήθηκε στο Datadog για το monitoring του Kafka παρουσιάζεται παρακάτω.

Εικόνα 4-9.

4.4.2 Υπολογισμός Event-time Latencies και Processing-time Latencies

Πέρα από την ικανότητα πρόβλεψης του MST για εργασίες σαν αυτές που περιγράφηκαν παραπάνω, θεωρήθηκε σημαντικό να γίνει και μια προσπάθεια εκτίμησης των καθυστερήσεων του Flink cluster όταν αυτό δέχεται στην είσοδο εγγραφές με ρυθμό κοντά σε αυτόν του Maximum Sustainable Throughput (90% του MST). Οι εκτιμήσεις αυτές θα δώσουν μια ακόμα πιο ξεκάθαρη εικόνα των δυνατοτήτων του Apache Flink σαν κατανεμημένο σύστημα επεξεργασίας δεδομένων και κυρίως σε εφαρμογές με απαιτήσεις απόκρισης σχεδόν πραγματικού χρόνου, όπου η ελαχιστοποίηση των καθυστερήσεων αποτελεί το σημαντικότερο παράγοντα επιτυχίας τους.

Στα πειράματα μας εξετάστηκαν 2 ειδών Latencies, οι Event-Time και οι Processing-time Latencies.

- Με τον όρο Event-time Latency μια εγγραφής, εννοείται η διαφορά ανάμεσα στο χρόνο που η εγγραφή εξήλθε από το Flink cluster και στο χρόνο που αυτή δημιουργήθηκε στην πηγή (Event-time).
- Με τον όρο Processing-time Latency μιας εγγραφής, εννοείται η διαφορά ανάμεσα στο χρόνο που η εγγραφή εξήλθε από το Flink cluster και στο χρόνο που αυτή εισήλθε σε αυτό (Ingestion-time).

Για τη λήψη των παραπάνω μετρικών εφαρμόστηκε διαφορετική μεθοδολογία για κάθε εργασία. Σε κάθε περίπτωση ωστόσο, ρυθμίστηκε η πηγή (Producer) να δίνει

τιμή στο πεδίο **timestamp1** κάθε εγγραφής ορίζοντας το ίσο με το χρόνο δημιουργίας της. Αντίστοιχα, το πεδίο **timestamp2** επιλέχθηκε να παίρνει τιμή με την είσοδο του κάθε μηνύματος στον Flink κόμβο.

➔ **Filtering**

Στα filtering jobs, η λήψη των Event και Processing time Latencies ήταν μια απλή διαδικασία. Κάθε εγγραφή έχει συμπληρωμένα τα πεδία timestamp1 και timestamp2 όταν φτάσει στον Filter operator και αν ικανοποιεί τα κριτήρια του τότε προωθείται στο επόμενο επίπεδο. Εκεί, για την εξυπηρέτηση των αναγκών μας και πριν την έξοδο από το Flink cluster, δημιουργήθηκε ένας Map operator μέσα από τον οποίο περνούν οι εγγραφές και εκτελούνται πάνω σε αυτές οι παρακάτω υπολογισμοί για την αποτίμηση των Latencies:

Event-time Latency (per record) = currentTime – timestamp1 (of record)

Processing-time Latency (per record) = currentTime – timestamp2 (of record)

Με τον τρόπο αυτό λαμβάνουμε για κάθε εγγραφή τις ζητούμενες μετρικές. Είναι σημαντικό να τονιστεί ότι για να εξασφαλιστεί η εγκυρότητα των μετρήσεων με την παραπάνω μέθοδο είναι απαραίτητο να υπάρχει συγχρονισμός στα ρολόγια όλων των μηχανημάτων που συνθέτουν το συνολικό σύστημα (Producers + Kafka cluster + Flink cluster).

➔ **Aggregation on Windows**

Όταν έχουμε να κάνουμε με stateful υπολογισμούς η διαδικασίας λήψης των μετρικών αυτών περιπλέκεται. Εφόσον η επεξεργασία γίνεται πάνω σε σύνολα δεδομένων που συγκεντρώνονται σε ένα παράθυρο με χρονικά όρια, δεν είναι δυνατό να υπολογιστεί για κάθε εγγραφή το Processing και Event time Latency της, αφού μέσα στις τιμές των αποτελεσμάτων θα συμπεριλαμβάνεται και ο χρόνος αναμονής της μέχρι να κλείσει το παράθυρο. Για το λόγο αυτό, ορίστηκε μια διαφορετική μεθοδολογία για τη λήψη των μετρήσεων αυτών. Αποφασίστηκε τα Latencies να μην υπολογίζονται πλέον για κάθε εγγραφή, αλλά για κάθε Aggregation operator, δηλαδή για κάθε παράθυρο που σχηματίζεται από την ομαδοποίηση των εγγραφών με βάση το κλειδί τους και έχει σαν έξοδο το μέσο όρο των τιμών των numberofsomething πεδίων τους.

Event-time Latency (per Window Aggregation operator) = currentTime – maxtimestamp1

Processing-time Latency (per Window Aggregation operator) = currentTime – maxtimestamp2

Όπως φαίνεται από τους παραπάνω τύπους για να λάβουμε τα Latencies για κάθε παράθυρο, θα πρέπει να υπολογιστεί το maxtimestamp1 και το maxtimestamp2, δηλαδή οι μέγιστες τιμές από τα timestamp1 και timestamp2 όλων των εγγραφών του παραθύρου. Με τον τρόπο αυτό, χρησιμοποιούνται ουσιαστικά τα timestamps της τελευταίας εγγραφής που εισήλθε στο παράθυρο πριν αυτό κλείσει ώστε να μην περιλαμβάνεται ο χρόνος αναμονής στα

αποτελέσματα των μετρικών καθυστέρησης. Οι τιμές που λαμβάνουμε θεωρείται ότι είναι αντιπροσωπευτικές για κάθε εγγραφή του παραθύρου.

➔ **Window Joins**

Με την ίδια λογική μιας και πρόκειται για stateful υπολογισμό, χρησιμοποιήθηκαν οι παρακάτω τύποι για τον υπολογισμό των Latencies.

Event-time Latency (per Window Join operator) = currentTime – maxtimestamp1

Processing-time Latency (per Window Join operator) = currentTime – maxtimestamp2

Κεφάλαιο 5

Πειραματική Μελέτη

5.1 Επεξήγηση της πειραματικής διαδικασίας

Στο προηγούμενο κεφάλαιο έγιναν όλες οι απαιτούμενες διαδικασίες προετοιμασίας του συστήματος για τη διεξαγωγή των πειραμάτων. Έχοντας πλέον σε ετοιμότητα όλα τα επιμέρους δομικά στοιχεία του συνολικού συστήματος (Producers, Kafka cluser, Flink cluster) καθώς και στη διάθεση μας μεθόδους για την παρακολούθηση αυτών, επόμενο βήμα αποτελεί ο καθορισμός της μεθοδολογίας για τη λήψη των επιθυμητών μετρήσεων. Επομένως, για κάθε διαφορετικά παραμετροποιημένη περίπτωση εκτέλεσης μιας εργασίας στο Flink cluster ακολουθήθηκαν τα βήματα που περιγράφονται παρακάτω για την εύρεση του MST και τον υπολογισμό των Latencies (Event time, Processing time). Να σημειωθεί ότι για τον καλύτερο έλεγχο της διαδικασίας χρησιμοποιήθηκαν **bash scripts** για την παράλληλη εκκίνηση εκτέλεσης των προγραμμάτων και τον τερματισμό αυτών.

➔ **Βήμα 1**
Εκκίνηση των στοιχείων του Kafka cluster, δηλαδή των Brokers και του Zookeepers, ώστε να είναι έτοιμα να υποδεχθούν τις εγγραφές που θα παραχθούν στη συνέχεια.

➔ **Βήμα 2**
Δημιουργία του Kafka topic μέσω του Zookeeper.

./bin/kafka-topics.sh --create --topic data --zookeeper localhost:2181 --partitions 24 --replication-factor 1

Για όλες τις περιπτώσεις εκτέλεσης αποφασίστηκε να υπάρχουν 24 partitions για την ομοιόμορφη διανομή των εγγραφών στη συνέχεια στον επιλεγμένο αριθμό των Task Managers (2, 4, 6). Επίσης ορίστηκε το replication-factor=1, αφού οι απώλειες δεδομένων σε περίπτωση αποτυχιών δεν αποτελεί κρίσιμο ζήτημα στα πλαίσια της πειραματικής διαδικασίας.

➔ **Βήμα 3**
Επιλογή της εργασίας προς εκτέλεση (Filtering, Aggregation on Windows, Window join) και υποβολή της στο Job Manager. Εκείνος με τη σειρά του χωρίζει την εργασία σε tasks, εκκινεί τους Task Managers και τη διανέμει σε αυτούς.

➔ **Βήμα 4**
Εκκίνηση των Producers και παρακολούθηση του ρυθμού εισόδου των

εγγραφών και του ρυθμού απορρόφησης και επεξεργασίας αυτών από το Flink cluster μέσω του Datadog monitoring service. Μέσα από τη διαδικασία αυτή, στόχος μας είναι να λάβουμε το MST της συγκεκριμένης εργασίας. Επομένως, για να εφαρμόσουμε τη μεθοδολογία που περιγράφηκε στο υποκεφάλαιο 4.4.1 και να εντοπίσουμε το σημείο όπου ο πρώτος ρυθμός ξεπερνάει τον δεύτερο θα πρέπει να μπορούμε να ελέγξουμε το ρυθμό με τον οποίο οι Producers παράγουν τα δεδομένα. Για το λόγο αυτό, εισάγουμε καθυστερήσεις στα προγράμματα που τρέχουν στα Producers μηχανήματα, ώστε να λαμβάνουμε κάθε φορά στην έξοδο εγγραφές στον ρυθμό τον οποίο επιθυμούμε. Η διαδικασία επομένως για την εύρεση του MST ανάγεται στα παρακάτω απλά βήματα:

1)Εκκίνηση των Producers με μεγάλη καθυστέρηση ώστε ο ρυθμός εισόδου των εγγραφών να είναι ίσος με αυτόν της επεξεργασίας τους.

2)Τερματισμός των Producers και επανεκκίνηση τους με μικρότερη καθυστέρηση, ώστε να αυξηθεί σε μικρό βαθμό ο ρυθμός εισόδου.

3)Αν ο ρυθμός εισόδου ξεπεράσει αυτή τη φορά τον ρυθμό επεξεργασίας (network intensive job) ή αν το σύστημα αποτύχει (memory intesive job) έχουμε καταλήξει στο ζητούμενο MST, σύμφωνα με τη μεθοδολογία μας.

4)Αν όχι, τότε επιστρέφουμε ξανά στο βήμα 2.

➔ **<u>Βημα 5</u>**

Σειρά έχει ο υπολογισμός των Latencies της συγκεκριμένης εργασίας όταν το σύστημα δουλεύει στα όρια του, δηλαδή όταν ο ρυθμός εισόδου εγγραφών σε αυτό είναι κοντά στο MST. Στα πειράματα μας ορίστηκε ο ρυθμός εισόδου για το συγκεκριμένο βήμα να είναι ίσος με το 90% του MST, ώστε να μην διατρέχει ο κίνδυνος να οδηγηθεί κάποιος κόμβος σε αποτυχία. Αφήνοντας επομένως το Flink cluster να εκτελέσει τους υπολογισμούς του για κάποια ώρα, λαμβάνουμε μέσω των μετρητών που έχουμε κατασκευάσει τις ζητούμενες τιμές για τα Event και Processing time Latencies. Οι τιμές αυτές γίνονται ορατές μέσω του Flink dashboard όπως φαίνεται στην Εικόνα 5-1.

➔ **<u>Βήμα 6</u>**

Παίρνοντας τα ζητούμενα αποτελέσματα, γίνεται η καταγραφή των μετρήσεων, τερματίζεται το σύστημα και στη συνέχεια επιστρέφουμε στο Βήμα 1 με διαφορετικές παραμέτρους αυτή τη φορά.

5.2 Επιλογή των παραμέτρων

Για να αποκτήσει νόημα η δημιουργία ενός μοντέλου που θα πραγματοποιεί προβλέψεις για διαφορετικά παραμετροποιημένες εργασίες που εκτελούνται σε ένα Flink cluster, είναι πολύ σημαντικό να επιλεχθούν προσεκτικά οι παράμετροι αυτοί. Συγκεκριμένα, για τις διαφορετικές τιμές των MST και Latencies που εντοπίζονται για

κάθε εργασία, οι επιλεγμένες παράμετροι θα πρέπει να είναι και αυτές που καθορίζουν τις διαφορές αυτές. Διαφορετικά το μοντέλο θα βασίζεται σε λανθασμένα στοιχεία και δεν θα θεωρείται έγκυρο. Επιλέχθηκαν λοιπόν για καθεμία από τις 3 υπό μελέτη εργασίες οι παράμετροι που θεωρήθηκε ότι επηρεάζουν κατά σε σημαντικό βαθμό την απόδοση και την ταχύτητα τους. Αυτές ανάλογα με το είδος τους παρουσιάζονται αναλυτικά στη συνέχεια.

5.2.1 Παράμετροι σχετικές με τη λειτουργία του Flink Cluster

- **Task slots**

  Πρόκειται ουσιαστικά για τη σημαντικότερη παράμετρο που λαμβάνει κάποιος υπόψη όταν πρόκειται να εκτελέσει κάποια εργασία σε Flink cluster. Όσο περισσότερα task slots είναι διαθέσιμα, τόσο μεγαλύτερος είναι και ο παραλληλισμός της εργασίας με αποτέλεσμα να αυξάνεται και η συνολική απόδοση του συστήματος. Είναι σημαντικό να γίνει φανερό κατά την εκπαίδευση του μοντέλου πόσο μεγάλη είναι η αύξηση αυτή για κάθε εργασία αλλά και πως επηρεάζει τις καθυστερήσεις σε κάθε περίπτωση η κατανομή των tasks σε περισσότερους κόμβους.

- **Checkpoint Mode**

  Στα υποκεφάλαια 2.4.1 και 3.1.4 έγινε εκτενής ανάλυση των διαφορετικών επιπέδων του Consistency καθώς και του τρόπου με τον οποίο το Flink διαχειρίζεται τα σφάλματα για να εξασφαλίσει την ομαλή συνέχιση της λειτουργίας του συστήματος. Θεωρήθηκε κρίσιμο να διερευνηθεί η επιρροή, στην απόδοση και την ταχύτητα, των διαφορετικών μεθόδων ανοχής σε σφάλματα σε κάθε περίπτωση εκτέλεσης εργασιών.

- **Buffer Timeout**

  Το Flink χρησιμοποιεί buffers για να κρατά προσωρινά τις εγγραφές πριν τις προωθήσει στο επόμενο επίπεδο επεξεργασίας τους. Με την παράμετρο Buffer Timeout εννοείται ο χρόνος παραμονής στους buffers αυτούς. Όσο μεγαλύτερος είναι αυτός τόσο περισσότερες εγγραφές θα συγκεντρωθούν, με αποτέλεσμα να αυξηθεί και ο ρυθμός επεξεργασίας των δεδομένων στο cluster. Ωστόσο κάτι τέτοιο μπορεί να έχει σοβαρό αντίκτυπο στις καθυστερήσεις. Η παράμετρος αυτή μπορεί να πάρει και την **τιμη -1**, κάτι το οποίο ορίζει ότι η προώθηση στο επόμενο επίπεδο θα πραγματοποιηθεί μόλις γεμίσουν πλήρως οι buffers.

- **Watermark Interval (Μόνο για εργασίες με Windows)**

  Τονίστηκε στο υποκεφάλαιο 2.3.3 η σημασία των watermarks στη λειτουργία ενός κατανεμημένου συστήματος επεξεργασίας δεδομένων, όπως το Flink, και στην ανάγκη εκτέλεσης υπολογισμών σε δεδομένα με βάση το Event time τους. Ορίζεται επομένως για κάθε εργασία η παράμετρος Watermark Interval που καθορίζει το πόσο συχνά θα εισάγονται watermarks στη ροή των εγγραφών και άρα το πόσο γρήγορα θα αντιλαμβάνονται τα παράθυρα το πέρασμα του χρόνου. Μικρές τιμές ωστόσο μπορεί να προσθέσουν σημαντικό overhead στο συνολικό σύστημα.

➢ **Allowed Lateness (Μόνο για εργασίες με Windows)**

Πρόκειται για τον εξτρά χρόνο που θα χρειαστεί να περιμένει ένα παράθυρο αφότου έχει αντιληφθεί ότι έφτασε η ώρα για το κλείσιμο του. Αυτός ο χρόνος είναι αρκετά σημαντικός σε πολλές περιπτώσεις που τα δεδομένα φτάνουν εκτός σειράς και υπάρχει ο κίνδυνος κάποια από αυτά να μην καταφτάσουν στην ώρα τους και να χάσουν το παράθυρο των υπολογισμών για το οποίο προορίζονταν.

5.2.2 Παράμετροι σχετικές με το είδος των δεδομένων εισόδου

➢ **Number of Keys (Μόνο για εργασίες με Windows)**

Πέρα από τις παραμέτρους που αφορούν το περιβάλλον του Flink, είναι σημαντικό να εξεταστούν και κάποιες που αφορούν τα δεδομένα πάνω στα οποία γίνονται οι υπολογισμοί. Αν και το μέγεθος των μηνυμάτων για το σύνολο των πειραμάτων μας αποφασίστηκε να παραμείνει σταθερό, λήφθηκαν ωστόσο μετρήσεις για δεδομένα με διαφορετικό αριθμό μοναδικών αναγνωριστικών (ή αλλιώς κλειδιών). Ο αριθμός αυτός είναι σημαντικός αφού καθορίζει και τον αριθμό των παραθύρων που θα δημιουργηθούν σε εργασίες όπως τα Aggregation on Windows και Window joins. Τα παράθυρα αυτά διαμοιράζονται στα διάθεσιμα task slots και συνεπώς το πλήθος τους επηρεάζει σημαντικά το φόρτο εργασίας των Task Managers.

5.2.3 Παράμετροι σχετικές με την υποβληθείσα εργασία

➢ **Filter Selectivity (Μόνο για εργασίες Filtering)**

Στις εργασίες που πραγματοποιούν κάποιο φιλτράρισμα στα δεδομένα της εισόδου αποφασίστηκε να διερευνηθεί σαν παράμετρος και το selectivity του φίλτρου, δηλαδή το ποσοστό του συνόλου των εγγραφών στις οποίες επιτρέπεται η διέλευση μέσα από αυτό. Όσο περισσότερες εγγραφές προωθούνται στο επόμενο επίπεδο τόσο μεγαλύτερη θα είναι και η συμφόρηση που θα προκληθεί.

➢ **Window Size (Μόνο για εργασίες με Windows)**

Όσον αφορά τις εργασίες με παράθυρα δεν θα μπορούσε να παραληφθεί η παράμετρος Window Size. Μεγαλύτερα παράθυρα σημαίνει μεγαλύτερος όγκος δεδομένων και περισσότεροι υπολογισμοί πάνω στο σύνολο αυτών. Ειδικότερα, στην περίπτωση των Window joins που αποτελούν εργασίες πολύ απαιτητικές σε μνήμη, το μέγεθος του παραθύρου μπορεί να επηρεάσει σημαντικά την ομαλή λειτουργία του συστήματος. Από την άλλη, η εφαρμογή παραθύρων μικρών σε μέγεθος σημαίνει αυτόματα ότι θα είναι συχνότερη και η δημιουργία αυτών.

5.3 Πειραματικά Αποτελέσματα

5.3.1 MST-Filtering

Task Slots	Checkpoint Mode	Buffer Timeout (ms)	Filter Selectivity		**MST (MB/s)**
1	Exactly-once	100	50%		29.54
2	Exactly-once	100	50%		35.79
4	Exactly-once	100	50%		58.77
1	Exactly-once	-1	50%		27.14
2	Exactly-once	-1	50%		37.00
4	Exactly-once	-1	50%		60.12
1	Exactly-once	10	50%		26.06
2	Exactly-once	10	50%		38.39
4	Exactly-once	10	50%		54.72
1	No	100	50%		31.35
2	No	10	50%		40.12
4	No	-1	50%		63.87
1	Exactly-once	100	96%		20.61
2	Exactly-once	100	96%		28.15

4	Exactly-once	100	96%		52.31
1	No	-1	4%		28.62
2	No	-1	4%		38.51
4	Exactly-once	-1	4%		59.06

Πίνακας 5-1.

5.3.2 Latencies-Filtering

Task Slots	Checkpoint Mode	Buffer Timeout (ms)	Filter Selectivity		**EtL avg (s)**	**PtL avg (s)**
1	Exactly-once	100	50%		0.12	0
2	Exactly-once	100	50%		0.27	0
4	Exactly-once	100	50%		0.35	0
1	Exactly-once	-1	50%		0.18	0
2	Exactly-once	-1	50%		0.32	0
4	Exactly-once	-1	50%		0.38	0
1	Exactly-once	10	50%		0.13	0
2	Exactly-once	10	50%		0.19	0
4	Exactly-once	10	50%		0.31	0
1	No	100	50%		0.08	0
2	No	10	50%		0.15	0
4	No	-1	50%		0.06	0
1	Exactly-once	100	96%		0.17	0

2	Exactly-once	100	96%		0.11	0
4	Exactly-once	100	96%		0.21	0
1	No	-1	4%		0.13	0
2	No	-1	4%		0.15	0
4	Exactly-once	-1	4%		0.16	0

Πίνακας 5-2.

5.3.3 MST-Aggregation on Windows

Task Slots	Watermark Interval (ms)	Allowed Lateness (s)	Buffer Timeout (ms)	Window Size (s)	Number of Keys		**MST (MB/s)**
1	100	0	100	10	26		19.58
2	100	0	100	10	26		34.62
4	100	0	100	10	26		43.32
1	100	0	-1	10	26		16.66
2	100	0	-1	10	26		27.45
4	100	0	-1	10	26		42.10
1	100	0	10	10	26		19.71
2	100	0	10	10	26		28.52
4	100	0	10	10	26		41.64
1	100	0	-1	60	26		21.94
2	100	0	-1	60	26		27.65
4	100	0	-1	60	26		45.02
1	1	0	-1	60	26		19.79
2	1	0	100	60	26		27.47
4	1	0	-1	60	26		40.67
1	1	0	10	10	26		20.00
2	1	0	-1	10	26		28.27
4	1	0	100	10	26		41.37
1	50	0	-1	60	26		20.77
2	100	0	-1	10	512		31.57

4	1	0	-1	60	512		37.92
1	1	0	-1	60	512		17.41
2	50	0	-1	60	1		26.61
4	100	0	-1	60	1		38.04
1	50	0	100	10	1		16.69
2	50	0	10	10	1000		30.14
4	50	0	100	30	1		37.72
1	100	0	10	10	1000		18.88
2	1	0	10	60	1000		25.00
2	100	0	-1	30	1		25.09
1	100	0	10	30	1000		20.47
2	100	0	-1	10	1000		34.17
4	100	0	-1	60	1000		44.96
4	100	0	-1	10	1000		47.82
4	100	0	-1	10	1		41.96
1	100	0	-1	60	1		21.76
1	100	0	10	10	1		18.60
2	50	0	-1	60	1000		38.27
1	100	1	-1	10	1000		18.04
1	100	5	-1	60	1000		15.04
1	100	1	-1	10	1		18.27
1	100	5	-1	60	1		19.21
2	100	1	100	10	1000		29.28
2	100	5	100	60	1000		31.06
2	100	1	100	10	1		31.04
2	100	10	100	60	1		25.89
4	100	1	10	10	1000		43.06
4	100	10	10	60	1000		41.50
4	100	1	10	10	1		45.21
4	100	5	10	60	1		37.69

Πίνακας 5-3.

5.3.4 Latencies-Aggregation on Windows

Task Slots	Watermark Interval (ms)	Allowed Lateness (s)	Buffer Timeout (ms)	Window Size (s)	Number of Keys		**EtL avg (s)**	**PtL avg (s)**
1	100	0	100	10	26		0.11	0.05
2	100	0	100	10	26		0.13	0.07
4	100	0	100	10	26		0.08	0.05
1	100	0	-1	10	26		0.11	0.05
2	100	0	-1	10	26		0.09	0.06
4	100	0	-1	10	26		0.10	0.07
1	100	0	10	10	26		0.13	0.07
2	100	0	10	10	26		0.09	0.05
4	100	0	10	10	26		0.10	0.07
1	100	0	-1	60	26		0.12	0.06
2	100	0	-1	60	26		0.12	0.05
4	100	0	-1	60	26		0.09	0.05
1	1	0	-1	60	26		0.07	0.01
2	1	0	100	60	26		0.05	0.01
4	1	0	-1	60	26		0.03	0.01
1	1	0	10	10	26		0.09	0.01
2	1	0	-1	10	26		0.05	0.01
4	1	0	100	10	26		0.03	0.01
1	50	0	-1	60	26		0.09	0.02
2	100	0	-1	10	512		0.22	0.11
4	1	0	-1	60	512		0.07	0.05
1	1	0	-1	60	512		0.10	0.05
2	50	0	-1	60	1		0.06	0.04
4	100	0	-1	60	1		0.10	0.07
1	50	0	100	10	1		0.08	0.03
2	50	0	10	10	1000		0.20	0.13
4	50	0	100	30	1		0.08	0.03

1	100	0	10	10	1000		0.23	0.15
2	1	0	10	60	1000		0.15	0.11
2	100	0	-1	30	1		0.07	0.03
1	100	0	10	30	1000		0.24	0.16
2	100	0	-1	10	1000		0.29	0.16
4	100	0	-1	60	1000		0.24	0.22
4	100	0	-1	10	1000		0.18	0.16
4	100	0	-1	10	1		0.08	0.06
1	100	0	-1	60	1		0.10	0.05
1	100	0	10	10	1		0.14	0.06
2	50	0	-1	60	1000		0.14	0.07
1	100	1	-1	10	1000		1.18	1.15
1	100	5	-1	60	1000		5.19	5.17
1	100	1	-1	10	1		1.09	1.07
1	100	5	-1	60	1		5.20	5.13
2	100	1	100	10	1000		1.26	1.24
2	100	5	100	60	1000		5.24	5.22
2	100	1	100	10	1		1.05	1.01
2	100	10	100	60	1		10.13	10.13
4	100	1	10	10	1000		1.21	1.19
4	100	10	10	60	1000		10.09	10.09
4	100	1	10	10	1		1.16	1.14
4	100	5	10	60	1		5.04	5.00

Πίνακας 5-4.

5.3.5 MST-Window Joins

Task Slots	Checkpoint mode	Buffer Timeout (ms)	Window Size (s)	Number of Keys		**MST (MB/s)**
4	Exactly-once	-1	10	26		1.64
4	Exactly-once	10	10	26		1.50
4	Exactly-once	100	10	26		1.68
4	No	100	10	100		2.96
4	No	-1	1	100		10.70
4	No	100	10	4		0.53
4	Exactly-once	100	20	26		1.04
4	Exactly-once	-1	1	26		4.60
4	No	-1	5	26		2.38
4	Exactly-once	-1	1	100		10.27
4	Exactly-once	10	1	10		3.50
4	Exactly-once	100	6	10		1.47
4	No	100	15	49		1.60
4	No	10	1	49		7.11
2	Exactly-once	-1	1	49		7.01
2	Exactly-once	-1	10	49		1.07
2	Exactly-once	100	6	49		2.21
2	No	100	4	10		1.24
2	No	10	1	10		2.62
2	No	-1	15	10		0.53
2	Exactly-once	10	6	100		3.01
2	Exactly-once	100	1	4		1.94
2	No	10	10	100		1.68
2	Exactly-once	-1	5	1000		7.83
2	No	-1	5	1000		8.44
6	Exactly-once	-1	10	26		2.07
6	Exactly-once	100	1	26		4.80
6	Exactly-once	100	5	26		2.24
6	No	10	15	26		1.49

6	No	-1	5	100		5.53
6	Exactly-once	-1	10	100		3.40
6	No	-1	10	100		3.64
6	No	10	15	100		3.02
6	Exactly-once	10	1	4		2.53
6	Exactly-once	10	10	4		0.66
6	No	100	5	4		0.79
6	No	100	10	4		0.74
6	Exactly-once	-1	15	4		0.51
6	No	10	10	1000		8.76
6	No	100	20	1000		5.63

Πίνακας 5-5.

5.3.6 Latencies-Window Joins

Task Slots	Checkpoint mode	Buffer Timeout (ms)	Window Size (s)	Number of Keys		**Etl avg (s)**	**Ptl avg (s)**
4	Exactly-once	-1	10	26		3.99	3.99
4	Exactly-once	10	10	26		3.47	3.36
4	Exactly-once	100	10	26		3.77	3.76
4	No	100	10	100		3.83	3.81
4	No	-1	1	100		0.84	0.80
4	No	100	10	4		3.24	3.24
4	Exactly-once	100	20	26		7.22	7.20
4	Exactly-once	-1	1	26		0.76	0.75
4	No	-1	5	26		2.76	2.74
4	Exactly-once	-1	1	100		1.04	0.96
4	Exactly-once	10	1	10		0.54	0.53
4	Exactly-once	100	6	10		2.21	2.21
4	No	100	15	49		4.25	4.25

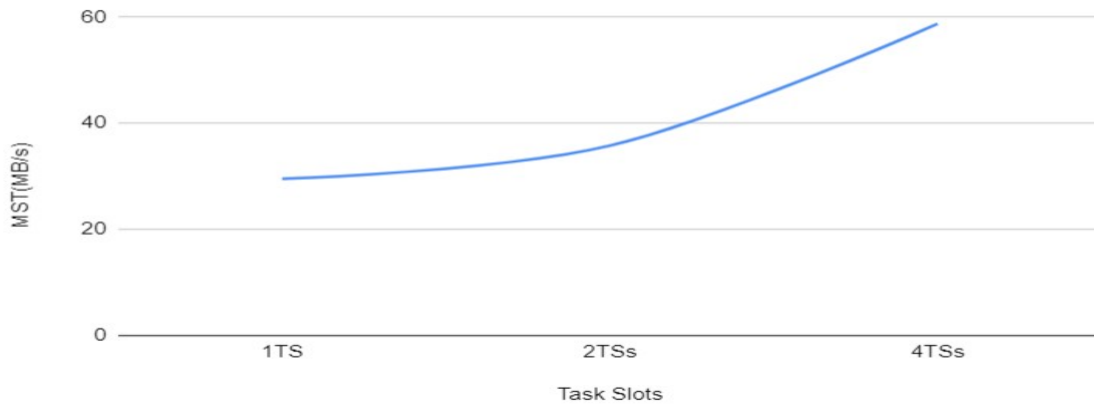
4	No	10	1	49		0.62	0.61
2	Exactly-once	-1	1	49		1.12	1.05
2	Exactly-once	-1	10	49		3.13	3.13
2	Exactly-once	100	6	49		2.54	2.18
2	No	100	4	10		1.66	1.65
2	No	10	1	10		0.58	0.58
2	No	-1	15	10		5.03	5.03
2	Exactly-once	10	6	100		2.37	1.93
2	Exactly-once	100	1	4		1.74	1.70
2	No	10	10	100		3.26	3.16
2	Exactly-once	-1	5	1000		1.60	1.52
2	No	-1	5	1000		1.71	1.50
6	Exactly-once	-1	10	26		4.48	4.47
6	Exactly-once	100	1	26		0.48	0.48
6	Exactly-once	100	5	26		1.73	1.73
6	No	10	15	26		4.28	4.28
6	No	-1	5	100		2.41	2.40
6	Exactly-once	-1	10	100		4.38	4.37
6	No	-1	10	100		4.63	4.58
6	No	10	15	100		4.68	4.67
6	Exactly-once	10	1	4		0.56	0.55
6	Exactly-once	10	10	4		3.25	3.25
6	No	100	5	4		1.80	1.80
6	No	100	10	4		3.76	3.76
6	Exactly-once	-1	15	4		6.34	6.34
6	No	10	10	1000		3.11	2.91
6	No	100	20	1000		5.59	5.58

Πίνακας 5-6.

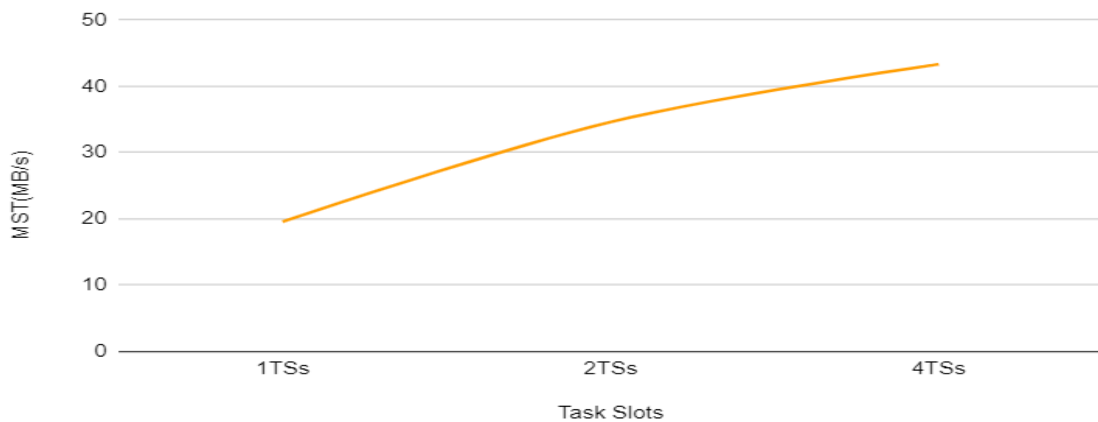
5.4 Χρήσιμα συμπεράσματα από την πειραματική διαδικασία

- **<u>Επίδραση του αριθμού των Task Slots στο MST</u>**

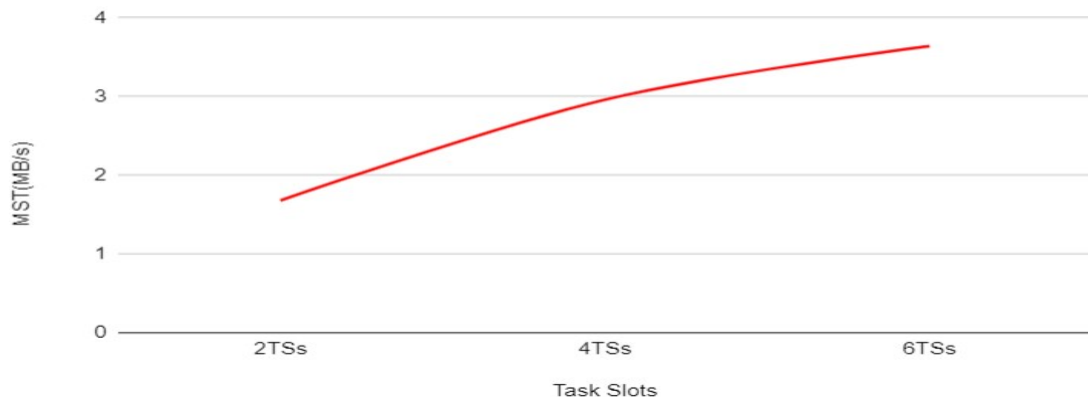
<u>Filtering</u>



<u>Aggregation on Windows</u>

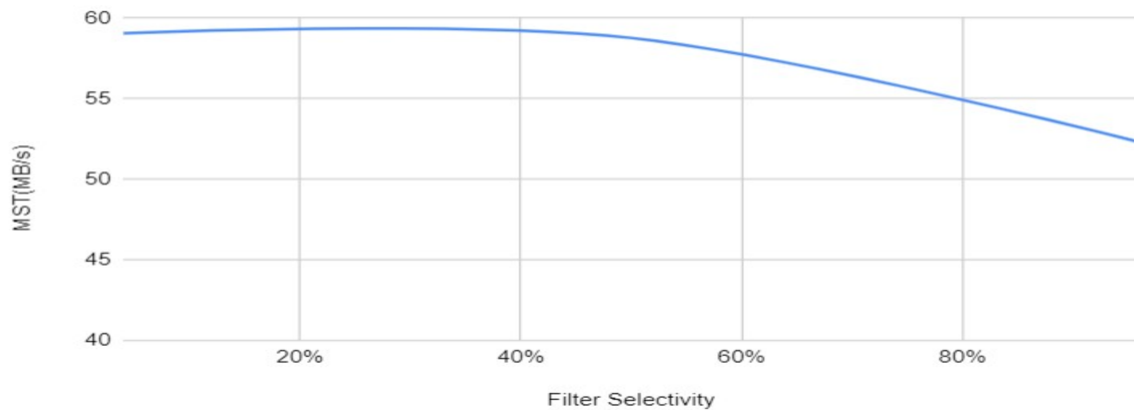


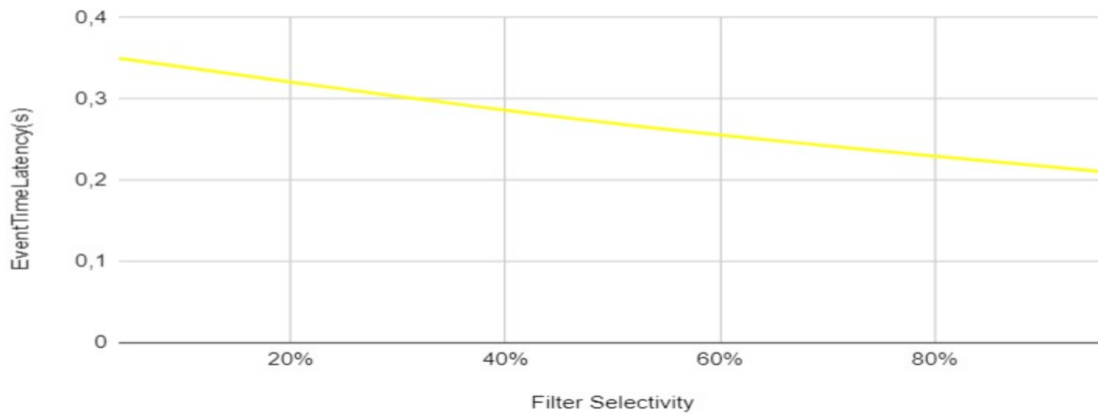
<u>Window Joins</u>



- **Επίδραση του Filter Selectivity στο MST και στα Latencies**

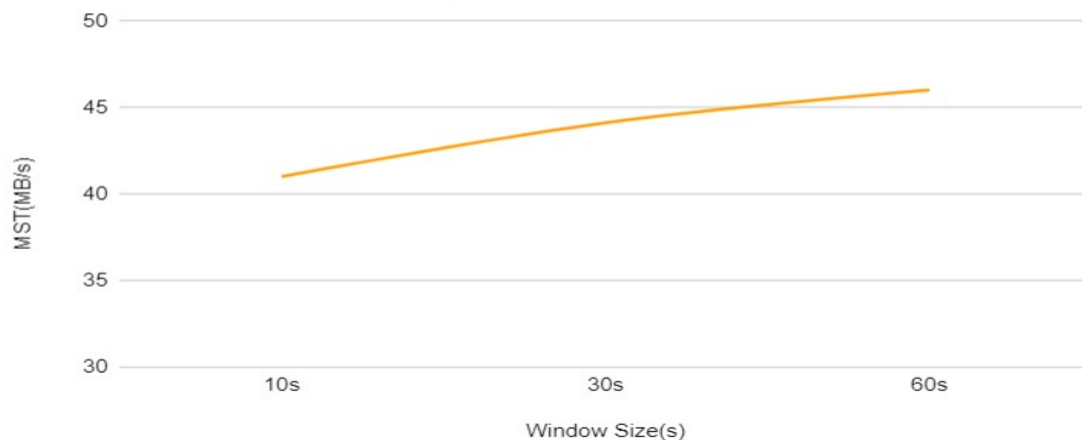
Filtering

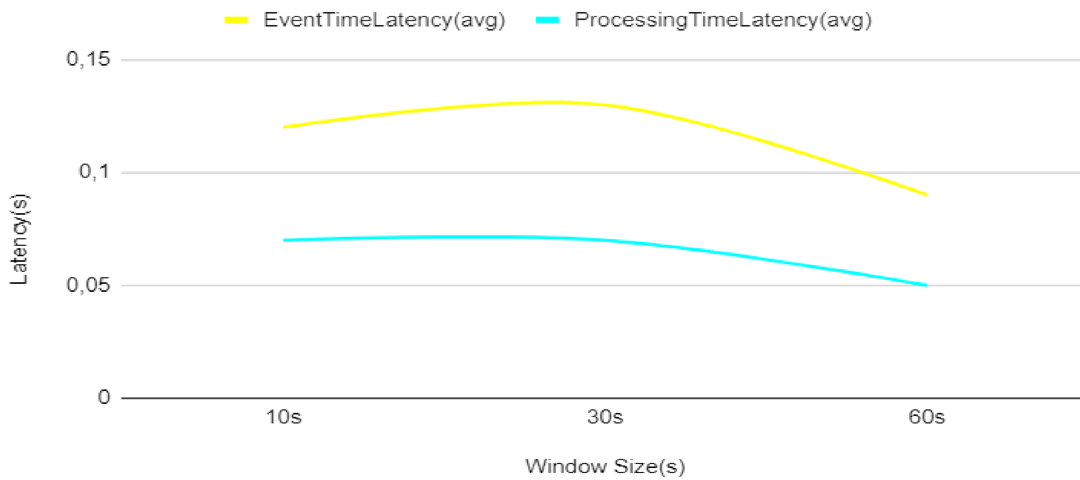




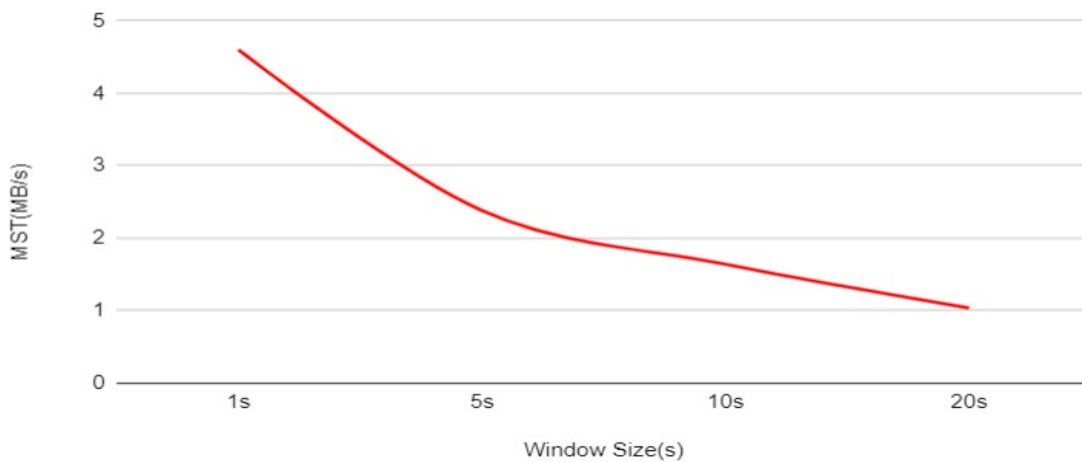
- **Επίδραση του Window Size στο MST και στα Latencies**

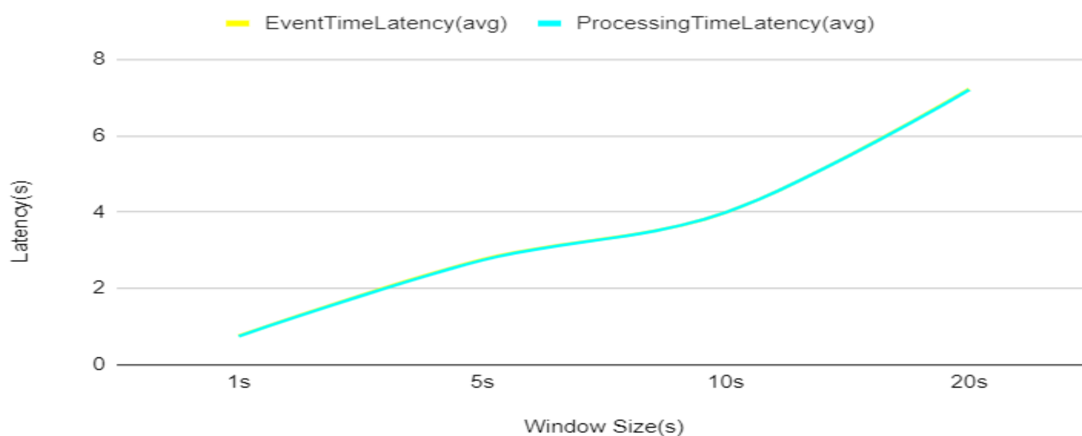
Aggregation on Windows





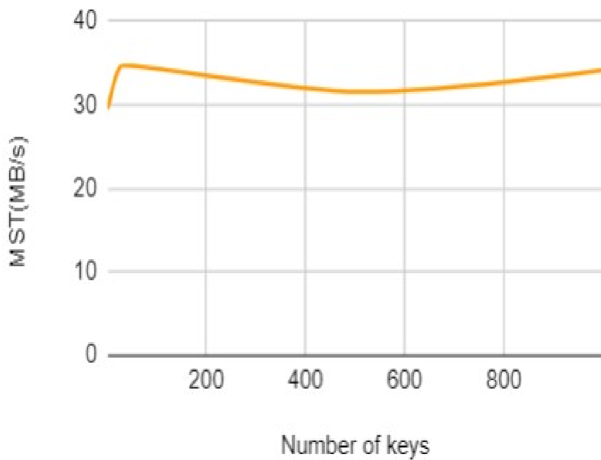
Window Joins

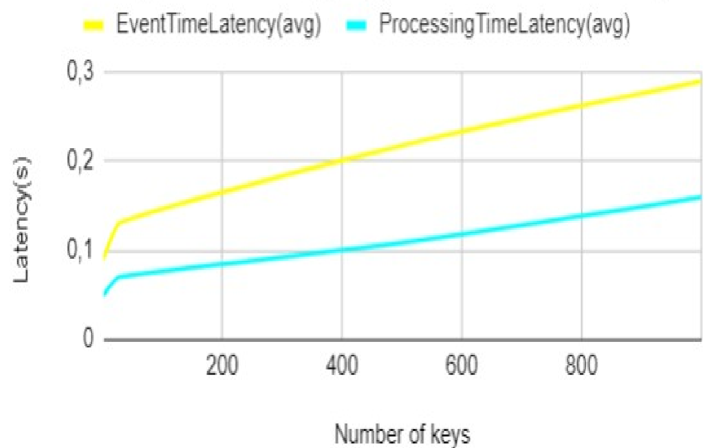




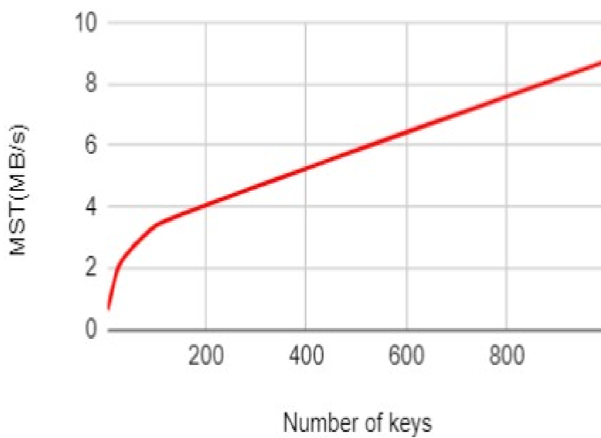
- **Επίδραση του Number of keys στο MST και στα Latencies (κρατώντας σταθερές όλες τις υπόλοιπες μεταβλητές)**

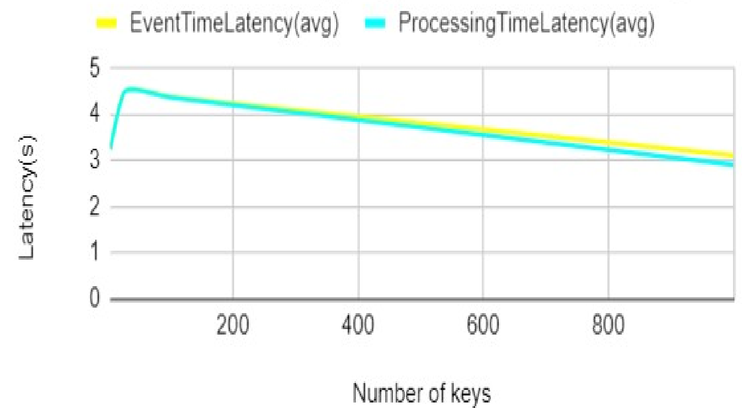
Aggregation on Windows





Window Joins





Κεφάλαιο 6

Μοντελοποίηση

Με τη διεξαγωγή των πειραμάτων και τη λήψη των μετρήσεων του MST και των Latencies για τις 3 υπό μελέτη εργασίες, έχουμε πλέον στη διάθεση μας τα σετ δεδομένων για να τροφοδοτήσουμε αλγόριθμους μηχανικής μάθησης. Μέσω των αλγορίθμων αυτών θα δημιουργηθούν μοντέλα ικανά να πραγματοποιήσουν προβλέψεις για το MST και τα Latencies διαφορετικά παραμετροποιημένων εργασιών που εκτελούνται σε Flink cluster. Θα μπορούν δηλαδή να δώσουν ακριβείς εκτιμήσεις για το μέγιστο ρυθμό επεξεργασίας και τις καθυστερήσεις που προκύπτουν με το ρυθμό αυτό, μιας εργασίας που ανήκει σε μια από τις 3 κατηγορίες που ορίσαμε και εκτελείται σε ένα επιλεγμένο Flink περιβάλλον. Για να γίνει αυτό ωστόσο, θα πρέπει πρώτα τα μοντέλα αυτά να εκπαιδευτούν πάνω στα σετ δεδομένων που κατασκευάσαμε.

Στόχος επομένως του τελικού αυτού κεφαλαίου είναι να παρουσιάσει τη διαδικασία δημιουργίας των μοντέλων αυτών με την επιλογή διαφόρων αλγορίθμων μηχανικής μάθησης και έπειτα να καταλήξει στην επιλογή του καλύτερου από αυτά μέσα από κάποια διαδικασία αξιολόγησης. Τα κριτήρια της αξιολόγησης αυτής παρουσιάζονται στο αμέσως επόμενο υποκεφάλαιο.

6.1 Μετρικές αξιολόγησης

Για την εύρεση του βέλτιστου αλγορίθμου και κατά συνέπεια του βέλτιστου μοντέλου πρέπει να εξεταστεί η καταλληλότητα του. Δηλαδή, πρέπει να βρεθεί το κατά πόσο ο εκάστοτε αλγόριθμος προσφέρει ικανοποιητικά αποτελέσματα. Για την αξιολόγηση των μοντέλων δεν υπάρχει ένα προφανές μέτρο εύρεσης της ακρίβειας τους. Αντιθέτως, υπάρχουν διάφορες μετρικές για την μέτρηση των σφαλμάτων μεταξύ πραγματικών και προβλεπόμενων τιμών όπως το μέσο απόλυτο σφάλμα, το μέσο απόλυτο ποσοστιαίο σφάλμα και το μέσο τετραγωνικό σφάλμα.

6.1.1 Μέσο απόλυτο σφάλμα (MAE)

Το μέσο απόλυτο σφάλμα (Mean Absolute Error), ή σε συντομογραφία MAE, εκφράζει ένα μέτρο της ακρίβειας της πρόβλεψης έναντι των πραγματικών τιμών. Συγκεκριμένα, πρόκειται για το άθροισμα της απόλυτης τιμής των διαφορών μεταξύ πραγματικής και προβλεπόμενης τιμής διαιρεμένο με το πλήθος των παρατηρήσεων-προβλέψεων. Όσο μεγαλύτερη είναι η τιμή του δείκτη τόσο μικρότερη προκύπτει η ακρίβεια της μεθόδου που εφαρμόστηκε. Υπολογίζεται όπως παρακάτω:

$$MAE = \frac{1}{n}\sum_{j=1}^{n} |y_{j-} \widehat{y_j}|$$

6.1.2 Μέσο απόλυτο ποσοστιαίο σφάλμα (MAPE)

Ορισμένες φορές είναι πιο χρήσιμος ο υπολογισμός των σφαλμάτων πρόβλεψης σε καθαρά ποσοστιαία μορφή. Το μέσο απόλυτο ποσοστιαίο σφάλμα είναι εκφρασμένο επί τις εκατό και λαμβάνει τιμές μεγαλύτερες ή ίσες του μηδενός με τις μικρότερες τιμές να υποδηλώνουν και καλύτερη απόδοση του αλγορίθμου. Το μέσο απόλυτο ποσοστιαίο σφάλμα δίνεται ως εξής:

$$MAPE = \frac{100\%}{n}\sum_{j=1}^{n} \frac{|y_j - \widehat{y_j}|}{y_j}$$

6.1.3 Μέσο τετραγωνικό σφάλμα (MSE)

Το μέσο απόλυτο σφάλμα είναι ακόμα ένα μέτρο της ακρίβειας της πρόβλεψης το οποίο όμως δίνει πολύ μεγαλύτερο βάρος στα μεγάλα σφάλματα (αν αναλογιστούμε πως τα σφάλματα τετραγωνίζονται) και μικρότερο βάρος στα μικρά σφάλματα. Υπολογίζεται από τον παρακάτω τύπο:

$$MSE = \frac{1}{n}\sum_{j=1}^{n} \left(y_j - \widehat{y_j}\right)^2$$

6.2 Χρήση αλγορίθμων παλινδρόμησης για την εκτίμηση του MST και των Latencies

6.2.1 Πολλαπλή γραμμική παλινδρόμηση (Multiple linear regression)

Στο μοντέλο της πολλαπλής γραμμικής παλινδρόμησης τα δεδομένα μοντελοποιούνται χρησιμοποιώντας γραμμικές λειτουργίες προγνωστικά και οι άγνωστες παράμετροι του μοντέλου υπολογίζονται από τα δεδομένα αυτά. Τέτοια μοντέλα καλούνται γραμμικά μοντέλα. Στην ουσία η εξαρτημένη μεταβλητή y είναι ένας γραμμικός συνδυασμός των ανεξάρτητων μεταβλητών x.
Το μοντέλο της πολλαπλής γραμμικής παλινδρόμησης έχει την εξής μορφή:

$$y = bo + b1x_1 + b2x_2 + b3x_3 + \dots + bix_i + e$$

Όπου τα x1, x2, xi αποτελούν τις ανεξάρτητες μεταβλητές που χρησιμοποιούνται για την τροφοδότηση του μοντέλου.

Τροφοδοτώντας το μοντέλο με τα σετ δεδομένων που δημιουργήσαμε (Εικόνα 6-1) και πραγματοποιώντας νέες προβλέψεις, λάβαμε τα παρακάτω αποτελέσματα για τη μετέπειτα αξιολόγηση της ακρίβειας του.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)


# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Εικόνα 6-1.

➔ **Filtering**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	8.08%	44.96%	-
MAE	2.90	81	-
MSE	12.80	11	-

Πίνακας 6-1.

➔ **Aggregation on Windows**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	9.89%	12.90%	26.30%
MAE	2.60	28	24
MSE	10.59	1.7	1.5

Πίνακας 6-2.

➔ **Window Joins**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	65.05%	22.88%	23.53%
MAE	1.53	437	450
MSE	4.28	291	298

Πίνακας 6-3.

6.2.2 Πολλαπλή πολυωνυμική παλινδρόμηση δευτέρου βαθμού (Polynomial regression)

Στην πολυωνυμική πολλαπλή παλινδρόμηση η εξαρτημένη μεταβλητή y συσχετίζεται με ένα πολυώνυμο των ανεξάρτητων x μεταβλητών. Το μοντέλο της πολλαπλής πολυωνυμικής παλινδρόμησης για μία ανεξάρτητη μεταβλητή είναι το εξής:

$$y = bo + bix_1 + b2x_1^2 + .. + bix_1^i + e$$

Ως i αναφέρεται ο βαθμός που ορίζεται για το πολυώνυμο.

Τροφοδοτώντας το μοντέλο με τα σετ δεδομένων που δημιουργήσαμε (Εικόνα 6-2) και πραγματοποιώντας νέες προβλέψεις, λάβαμε τα παρακάτω αποτελέσματα για τη μετέπειτα αξιολόγηση της ακρίβειας του.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X, X_test, y, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)

# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 2)
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

Εικόνα 6-2.

➔ **Filtering**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	11.42%	57.01%	-
MAE	4.84	134	-
MSE	36.56	395	-

Πίνακας 6-4.

➔ **Aggregation on Windows**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	13.58%	35.89%	46.49%
MAE	10.98	168	129
MSE	516.37	238	169

Πίνακας 6-5.

➔ **Window Joins**

	MST(MB/s)	EtL avg(s)	PtL avg(s)
MAPE	70.32%	46.36%	36.61%
MAE	2.82	1.09	0.90
MSE	54.15	4.06	3.10

Πίνακας 6-6.

6.2.3 Παλινδρόμηση με Δέντρα Απόφασης (Decison Tree Regression)

Σχηματικά αναπτύσσεται ένα δυαδικό δέντρο. Εντός αυτού, σε κάθε κόμβο του (node) εφαρμόζεται ένας έλεγχος που αφορά σε μια ανεξάρτητη μεταβλητή ξεχωριστά. Ανάλογα με το αποτέλεσμα του ελέγχου, μετακινούμαστε είτε προς την αριστερή είτε προς την δεξιά διακλάδωση του δέντρου. Κάποια στιγμή φτάνουμε σε τελικό κόμβο ή αλλιώς σε φύλλο (terminal node/leaf) όπου γίνεται μια πρόβλεψη.

Τροφοδοτώντας το μοντέλο με τα σετ δεδομένων που δημιουργήσαμε (Εικόνα 6-3) και πραγματοποιώντας νέες προβλέψεις, λάβαμε τα παρακάτω αποτελέσματα για τη μετέπειτα αξιολόγηση της ακρίβειας του.

```
# Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)
```

Εικόνα 6-3.

➔ **Filtering**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	9.13%	57.39%	-
MAE	3.25	113	-
MSE	16.48	19	-

Πίνακας 6-7.

➔ **Aggregation on Windows**

	MST(MB/s)	EtL avg(s)	PtL avg(s)
MAPE	11.88%	30.91%	32.79%
MAE	3.30	0.99	0.98
MSE	17	6.32	6.34

Πίνακας 6-8.

➔ **Window Joins**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	42.64%	33.29%	32.56%
MAE	1.49	709	699
MSE	6.13	761	741

Πίνακας 6-9.

6.2.4 Παλινδρόμηση με Τυχαία Δάση (Random forest regression)

Τα τυχαία δάση πρόκειται για μία συλλογή των δέντρων αποφάσεων. Ό αλγόριθμος παλινδρόμησης τυχαίων δασών (Random Forest Regression) επιλέγει τυχαία τις παρατηρήσεις και τα χαρακτηριστικά (features) για την κατασκευή αρκετών δέντρων απόφασης και στην συνέχεια υπολογίζει κατά μέσο όρο τα αποτελέσματα.

Τροφοδοτώντας το μοντέλο με τα σετ δεδομένων που δημιουργήσαμε (Εικόνα 6-4) και πραγματοποιώντας νέες προβλέψεις, λάβαμε τα παρακάτω αποτελέσματα για τη μετέπειτα αξιολόγηση της ακρίβειας του.

```
# Fitting Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X, y)
```

Εικόνα 6-4.

➔ **Filtering**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	8.07%	40.28%	-
MAE	3.26	97	-
MSE	17.21	15	-

Πίνακας 6-10.

➔ **Aggregation on Windows**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	9.04%	26.79%	29.61%
MAE	2.58	265	256
MSE	11.77	998	1003

Πίνακας 6-11.

➔ **Window Joins**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	37.58%	23.39%	23.52%
MAE	1.19	542	526
MSE	3.25	477	462

Πίνακας 6-12.

6.2.5 Παλινδρόμηση Xgboost

Η μέθοδος XGBoost είναι η συντομογραφία του “Extreme Gradient Boosting”. Gradient Boosting είναι μια τεχνική όπου προστίθενται νέα μοντέλα για τη διόρθωση των σφαλμάτων-υπολειμμάτων (διαφορά μεταξύ των προβλεπόμενων και των πραγματικών τιμών) που έγιναν από τα υπάρχοντα μοντέλα. Τα μοντέλα προστίθενται

διαδοχικά μέχρις ότου δεν μπορούν να γίνουν περαιτέρω βελτιώσεις.

Τροφοδοτώντας το μοντέλο με τα σετ δεδομένων που δημιουργήσαμε (Εικόνα 6-5) και πραγματοποιώντας νέες προβλέψεις, λάβαμε τα παρακάτω αποτελέσματα για τη μετέπειτα αξιολόγηση της ακρίβειας του.

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)

# Fitting XGBoost to the Training set
from xgboost import XGBRegressor
regressor = XGBRegressor()
regressor.fit(X_train, y_train)
```

Εικόνα 6-5.

➔ **Filtering**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	8.68%	51.12%	-
MAE	3.04	110	-
MSE	16.04	18	-

Πίνακας 6-13.

➔ **Aggregation on Windows**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	12.35%	20.18%	21.25%
MAE	3.27	236	237
MSE	15.23	999	1003

Πίνακας 6-14.

➔ **Window Joins**

	MST(MB/s)	EtL avg(ms)	PtL avg(ms)
MAPE	32.02%	23.24%	21.11%
MAE	1.01	493	442
MSE	2.98	432	375

Πίνακας 6-15.

6.3 Αξιολόγηση Αλγορίθμων

Είναι σημαντικό να τονιστεί ότι για την παραπάνω διαδικασία αξιολόγησης των αλγορίθμων και τον υπολογισμό των μετρικών MAE, MAPE και MSE χρησιμοποιήθηκε η τεχνική K-Fold Cross Validation με K=6. Σύμφωνα με αυτή το σετ δεδομένων των ανεξάρτητων μεταβλητών χωρίζεται σε K ίσα μικρότερα σετ και κάθε φορά επιλέγεται ένα από αυτά για τη πραγματοποίηση προβλέψεων και την αξιολόγηση του αλγορίθμου, ενώ τα υπόλοιπα χρησιμοποιούνται για την εκπαίδευση και δημιουργία του μοντέλου. Αυτό έχει σαν αποτέλεσμα τον υπολογισμό K μετρικών MAE, MAPE και MSE (6 στην περίπτωση μας), όπου σαν τελικές τιμές στη καταγραφή των αποτελεσμάτων επιλέχθηκε ο μέσος όρος αυτών. Έτσι με τη μεθοδολογία αυτή, εξασφαλίζεται η καλύτερη αξιολόγηση των μοντέλων με την απαλοιφή ενός μέρους της τυχαιότητας των αποτελεσμάτων χάρη στην εκπαίδευση των αλγορίθμων μηχανικής μάθησης και τη δοκιμή αυτών σε περισσότερα από 1 σετ δεδομένων.

Για κάθε λοιπόν διαφορετική εργασία (Filtering, Aggregation on Windows, Window joins) που εκτελέστηκε σε κατανεμημένο περιβάλλον Flink έγιναν προβλέψεις για το MST, τα Event time Latencies και τα Processing time Latencies (το μέσο όρο αυτών) με 5 διαφορετικά μοντέλα μηχανικής μάθησης. Σύμφωνα με τα αποτελέσματα της προηγούμενης ενότητας παρουσιάζεται παρακάτω η επιλογή του καλύτερου μοντέλου σε κάθε περίπτωση.

Επιλογή με βάση το MAE

	MST	EtL avg	PtL avg
Filtering	**Multiple linear regression (2.90 MB/s)**	**Multiple linear regression (81ms)**	**-**
Aggregation on Windows	**Random Forest Regression (2.58 MB/s)**	**Multiple linear regression (28ms)**	**Multiple linear regression (24ms)**
Window joins	**XgBoost (1.01 MB/s)**	**Multiple linear regression (437ms)**	**XgBoost (442ms)**

Πίνακας 6-16.

Επιλογή με βάση το Accuracy(100-MAPE)

	MST	EtL avg	PtL avg
Filtering	**Random Forest Regression (91.93%)**	**Random Forest Regression (59.72%)**	**-**
Aggregation on Windows	**Random Forest Regression (90.96%)**	**Multiple linear regression (87.10%)**	**XgBoost (78.75%)**

Window joins	**XgBoost (67.98%)**	**Multiple linear regression (77.12%)**	**XgBoost (78.89%)**

Πίνακας 6-17.

Επιλογή με βάση το MSE

	MST	EtL	PtL
Filtering	**Multiple linear regression (12.80 MB/s)**	**Multiple linear regression (11ms)**	**-**
Aggregation on Windows	**Multiple linear regression (10.59 MB/s)**	**Multiple linear regression (1.7ms)**	**Multiple linear regression (1.5ms)**
Window joins	**XgBoost (2.98 MB/s)**	**Multiple linear regression (291ms)**	**Multiple linear regression (298 ms)**

Πίνακας 6-18.

Από τα αποτελέσματα της αξιολόγησης, είναι προφανές ότι 3 από τα 5 μοντέλα ξεχωρίζουν από τα υπόλοιπα με την ακρίβεια των προβλέψεων τους και την ικανότητα τους να ελαχιστοποιήσουν τα σφάλματα. Αυτά είναι η πολλαπλή γραμμική παλινδρόμηση, η παλινδρόμηση με τυχαία δάση και η παλινδρόμηση XgBoost. Πιο αναλυτικά, στις εργασίες **Filtering** οι προβλέψεις για το **MST** με τη χρήση των παραπάνω μοντέλων ήταν πολύ κοντά στις πραγματικές. Ξεχώρισαν ωστόσο τα μοντέλα της **πολλαπλής γραμμικής παλινδρόμησης** και της **παλινδρόμησης με τυχαία δάση**. Η ίδια κατάσταση εντοπίζεται και στις προβλέψεις σχετικά με το **EtL**, αλλά τα σφάλματα των προβλέψεων ήταν αρκετά μεγαλύτερα με αποτέλεσμα η ακρίβεια του καλύτερου μοντέλου να είναι ίση με **59.72%**. Όσον αφορά το **PtL**, στα Filtering jobs οι τιμές του ήταν πάντα σχεδόν ίσες με **0**, οπότε δεν απαιτείται η χρήση κάποιου αλγορίθμου για πρόβλεψη.

Στα **Aggregation on Windows**, ξεχωρίζουν πάλι τα ίδια μοντέλα με το μοντέλο της **παλινδρόμησης με τυχαία δάση** να δίνει εξαιρετική ακρίβεια στις προβλέψεις σχετικά με το **MST** ίση με **90.96%** και την **πολλαπλή γραμμική παλινδρόμηση** εξίσου καλή ακρίβεια σχετικά με το **EtL** ίση με **87.10%**. Όσον αφορά το **PtL** η ακρίβεια του καλύτερου μοντέλου, δηλαδή του **XgBoost**, υπολογίστηκε ίση με **78.75%**. Παρατηρείται επομένως ότι για τις εργασίες αυτού του είδους σε Flink cluster επιτεύχθηκαν ικανοποιητικά ποσοστά ακρίβειας στις προβλέψεις και η μοντελοποίηση μπορεί να χαρακτηριστεί επιτυχής.

Τέλος, περνώντας στο πιο απαιτητικό είδος εργασιών τα **Window Joins**, μόνο ο αλγόριθμος **XgBoost** κατάφερε να πραγματοποιήσει προβλέψεις με μικρά σφάλματα για το **MST** με την ακρίβεια του ωστόσο να μην ξεπερνάει το 70% (**67.98%**). Καλύτερες επιδόσεις παρατηρήθηκαν για τα **Latencies** με τα μοντέλα της **πολλαπλής γραμμικής παλινδρόμησης** και της παλινδρόμησης **XgBoost** που πέτυχαν ικανοποιητικά αποτελέσματα με σχετικά μικρές αποκλίσεις (**77.12 %** για το **EtL** και **78.89%** για το **PtL**).

Κεφάλαιο 7

Επίλογος

7.1 Σύνοψη και Συμπεράσματα

Έχοντας πλέον στα χέρια μας μια καθαρή εικόνα των αποτελεσμάτων της αξιολόγησης του Κεφαλαίου 6, μπορούμε να οδηγηθούμε στον επίλογο της παρούσας διπλωματικής. Έχοντας μελετήσει σε βάθος το Flink framework και τον τρόπο με τον οποίο εκμεταλλεύεται ένα κατανεμημένο σύστημα για να επιτύχει γρήγορη επεξεργασία μεγάλου όγκου δεδομένων, έγινε η προσπάθεια με τη βοήθεια ενός κατανεμημένου συστήματος μεταφοράς μηνυμάτων (Kafka framework) να προσομοιωθούν περιπτώσεις εκτέλεσης εργασιών που ταυτίζονται με αληθινά σενάρια του επιχειρηματικού κόσμου. Μέσα από τις προσομοιώσεις αυτές και έχοντας επιλέξει συγκεκριμένες παραμέτρους προς εξέταση, αυτές που θεωρήθηκε ότι μπορούν να επηρεάσουν την ταχύτητα και την απόδοση του συστήματος, έγινε καταγραφή του μέγιστου ρυθμού επεξεργασίας δεδομένων που μπορούσε να αντέξει το σύστημα καθώς και των καθυστερήσεων που εντοπίστηκαν σε κάθε περίπτωση. Διαθέτοντας τα πειραματικά αυτά αποτελέσματα, απώτερος σκοπός ήταν ο εντοπισμός μοντέλων μηχανικής μάθησης που μπορούν να περιγράψουν με ακρίβεια τις διαφορετικές εργασίες επεξεργασίας δεδομένων που εκτελούνται σε Flink clusters και την πραγματοποίηση προβλέψεων. Οι προβλέψεις αυτές, αν είναι ακριβείς, μπορούν να βοηθήσουν σε πολλές περιπτώσεις στην εκτίμηση του μεγέθους και της παραμετροποίησης ενός κατανεμημένου Flink συστήματος που στόχο έχει να επεξεργαστεί μεγάλο όγκο δεδομένων. Ανάλογα με το είδος των δεδομένων αυτών καθώς και τις απαιτήσεις για ταχύτητα στην εκτέλεση των υπολογισμών και αντοχή σε σφάλματα, τα μοντέλα μπορούν να προσφέρουν μια αρχική εικόνα για το σχεδιασμό του cluster που μπορεί να προσφέρει τα επιθυμητά αποτελέσματα της εκάστοτε εφαρμογής.

Από την αξιολόγηση επομένως των μοντέλων της προηγούμενης ενότητας, καταλήγουμε στο συμπέρασμα ότι αποδείχθηκε δυνατή η πραγματοποίηση προβλέψεων με μικρά ποσοστά σφαλμάτων για το μέγιστο όγκο δεδομένων που μπορεί να αντέξει ένα Flink cluster που εκτελεί κάποια εργασία Filtering, Aggregation on Windows ή Window Joins. Ειδικότερα, στις 2 πρώτες κατηγορίες εργασιών η ακρίβεια των προβλέψεων ήταν ιδιαίτερα υψηλή, ενώ στην τελευταία κατηγορία απαιτούνται βελτιώσεις για την εξαγωγή ασφαλών συμπερασμάτων. Όσον αφορά τις καθυστερήσεις, που σαν στόχο έχουν να παρουσιάσουν μια εικόνα της ταχύτητας του συστήματος όταν αυτό αγγίξει τα όρια των δυνατοτήτων του, οι αλγόριθμοι μηχανικής μάθησης παρουσιάστηκαν και εκεί ικανοί να δώσουν εκτιμήσεις πολύ κοντά στις πραγματικές μετρήσεις. Εξαίρεση αποτέλεσαν οι εργασίες Filtering όπου τα μοντέλα

είχαν υψηλά ποσοστά σε σφάλματα, κάτι το οποίο οφείλεται ωστόσο και στις πολύ μικρές καθυστερήσεις που εντοπίζονται σε τέτοιου τύπου επεξεργασίες δεδομένων.

7.2 Μελλοντικές Επεκτάσεις

Είναι σαφές ότι η παραπάνω έρευνα αποτελεί μια πρώτη προσπάθεια χρήσης μηχανικής μάθησης για την πρόβλεψη της επίδοσης κατανεμημένων συστημάτων επεξεργασίας δεδομένων και ότι είναι πολλά τα σημεία που μπορούν να γίνουν βελτιώσεις. Αρχικά, τα σετ δεδομένων θεωρούνται αρκετά μικρά σε μέγεθος για να δώσουν τη δυνατότητα στα μοντέλα να εκπαιδευτούν σωστά και να δείξουν τις δυνατότητες τους. Αυτό οφείλεται ωστόσο και στους περιορισμένους πόρους που είχαμε στη διάθεση μας για το στήσιμο των Producers, Kafka cluster και Flink Cluster. Θα είχε ιδιαίτερο ενδιαφέρον τόσο η επέκταση των σετ δεδομένων για την εξέταση της κλιμάκωσης του συστήματος όσο και η δοκιμή των μοντέλων της παρούσας έρευνας σε μεγαλύτερα Flink clusters. Κάτι τέτοιο θα έδινε μια πιο καθαρή εικόνα για το αν οι προς εξέταση παράμετροι κάθε διαφορετικής εργασίας συνεχίζουν να επηρεάζουν με τον ίδιο τρόπο την επίδοση του κατανεμημένου συστήματος όσο αυτό κλιμακώνει.

Επιπλέον προσπάθειες που θα μπορούσαν να γίνουν είναι η χρήση αλγορίθμων μηχανικής μάθησης και σε άλλα κατανεμημένα συστήματα επεξεργασίας δεδομένων, όπως το Apache Spark και το Apache Storm. Θα μπορούσε δηλαδή να δημιουργηθεί ένα ενιαίο μοντέλο που θα μπορεί να περιγράψει κάθε σύστημα που εκτελεί υπολογισμούς σε μεγάλο όγκο δεδομένων. Σε ένα ψηφιακό κόσμο που όλες οι εφαρμογές απαιτούν πλέον τη διαχείριση πληροφοριών τεράστιου μεγέθους και την απόκριση σε πραγματικό χρόνο, καθίσταται αναγκαίο να πραγματοποιηθούν παραπάνω μελέτες γύρω από τα παραπάνω εργαλεία και να γίνει χρήση μαθηματικών μοντέλων για την περιγραφή τους.

Τέλος, τα μοντέλα μηχανικής μάθησης που εφαρμόστηκαν για τις προβλέψεις της επίδοσης της εκτέλεσης των εργασιών επιδέχονται πολλών βελτιώσεων. Στην περίπτωση μας επιλέχτηκαν σε κάθε περίπτωση οι προκαθορισμένες τιμές για την εκπαίδευση των αλγορίθμων μέσα από τα σετ δεδομένων που είχαμε στη διάθεση μας. Η εφαρμογή διαφορετικών τιμών με στόχο τη βελτιστοποίηση των μοντέλων αυτών και την ελαχιστοποίηση των σφαλμάτων θα μπορούσε να οδηγήσει σε ακόμα πιο ικανοποιητικά αποτελέσματα.

Βιβλιογραφία

[1] Apache Flink https://ci.apache.org/projects/flink/flink-docs-release-1.6/

[2] Apache Kafka https://kafka.apache.org

[3] Apache Zookeeper https://zookeeper.apache.org

[4] *"Introduction to Apache Flink"* by Ellen Friedman and Kostas Tzoumas

[5] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, Volker Markl: "*Benchmarking Distributed Stream Processing Engines"*

[6] High-throughput, low-latency, and exactly-once stream processing with Apache Flink https://www.ververica.com/blog/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink

[7] Ilaria Bartolini and Marco Patella: "*Comparing Performances of Big Data Stream Processing Platforms with RAM3S (extended abstract)"*

[8] Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, Kostas Tzoumas: "*Lightweight Asynchronous Snapshots for Distributed Dataflows"*

[9] K. MANI CHANDY, LESLIE LAMPORT: "*Distributed Snapshots: Determining Global States of Distributed Systems"*

[10] How To Size Your Apache Flink® Cluster: A Back-of-the-Envelope Calculation https://www.ververica.com/blog/how-to-size-your-apache-flink-cluster-general-guidelines

[11] An Overview of End-to-End Exactly-Once Processing in Apache Flink® (with Apache Kafka, too!) https://www.ververica.com/blog/end-to-end-exactly-once-processing-apache-flink-apache-kafka

[12] How Apache Flink manages Kafka consumer offsets https://www.ververica.com/blog/how-apache-flink-manages-kafka-consumer-offsets

[13] Kafka + Flink: A Practical, How-To Guide https://www.ververica.com/blog/kafka-flink-a-practical-how-to

[14] Kafka Producer and Consumer Examples Using Java https://dzone.com/articles/kafka-producer-and-consumer-example

[15] Running a Multi-Broker Apache Kafka 0.8 Cluster on a Single Node https://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/

[16] Kafka Tutorial: Creating Advanced Kafka Producers in Java
http://cloudurable.com/blog/kafka-tutorial-kafka-producer-advanced-java-examples/index.html

[17] Writing a Kafka Consumer in Java
https://dzone.com/articles/writing-a-kafka-consumer-in-java

[18] Shigeru Imai, Stacy Patterson, and Carlos A. Varela: "*Maximum Sustainable Throughput Prediction for Data Stream Processing over Public Clouds*"

[19] Yahoo! Inc., "Yahoo streaming benchmarks" https://github.com/yahoo/streaming-benchmarks

[20] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle:
"*The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing*"

[21] Javier Cervino, Evangelia Kalyvianaki, Joaquin Salvachua, Peter Pietzuch: "*Adaptive Provisioning of Stream Processing Systems in the Cloud*"

[22] A. Shukla and Y. Simmhan: "*Benchmarking distributed stream processing platforms for IoT applications*"

[23] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl and K. Tzoumas: *"Apache Flink: Stream and batch processing in a single engine"*

[25] Datadog Docs https://docs.datadoghq.com

[26] OpenStack Docs https://docs.openstack.org/rocky/

[27] O.-C. Marcu, A. Costan, G. Antoniu, and M. S. Perez: "*Understanding performance in big data analytics Frameworks*"

[28] S. Perera, A. Perera, and K. Hakimzadeh: "*Reproducible experiments for comparing apache flink and apache spark on public clouds*"

[29] M. A. Lopez, A. Lobato, and O. Duarte: "*A performance comparison of open-source stream processing platforms*"

[30] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer: *"Latency-aware elastic scaling for distributed data stream processing systems"*