Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

ΣΥΓΚΡΙΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ ΡΟΩΝ ΔΕΔΟΜΕΝΩΝ

Διπλωματική εργασία

Κωνσταντίνος Παππάς

Επιβλέπων:

Νεκτάριος Κοζύρης

Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.

Συν-επιβλέπων:

Νικόλαος Χαλβαντζής

Υποψήφιος Διδάκτορας ΣΗΜΜΥ Ε.Μ.Π.

Αθήνα, Νοέμβριος 2025

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

ΣΥΓΚΡΙΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ ΡΟΩΝ ΔΕΔΟΜΕΝΩΝ

Διπλωματική εργασία

Κωνσταντίνος Παππάς

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 7 Νοεμβρίου 2025.

Επιβλέπων:

Νεκτάριος Κοζύρης

Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.

Συν-επιβλέπων:

Νικόλαος Χαλβαντζής

Υποψήφιος Διδάκτορας ΣΗΜΜΥ Ε.Μ.Π.

(Ονόματα καθηγητών επιτροπής)

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Ιωάννης Κωνσταντίνου Αναπληρωτής Καθηγητής Π.Θ.	Δημήτριος Τσουμάκος Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2025

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

Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας αυτής της διπλωματικής εργασίας, και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην διπλωματική εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η διπλωματική εργασία προετοιμάστηκε από εμένα προσωπικά για τις απαιτήσεις του προγράμματος σπουδών του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών.

(Υπογραφή)

. . . . . .

Παππάς Κωνσταντίνος

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

Περίληψη

Η επεξεργασία ροών δεδομένων σε πραγματικό χρόνο αποτελεί κρίσιμη τεχνολογική απαίτηση για σύγχρονες εφαρμογές σε τομείς όπως τα χρηματοοικονομικά, τα τηλεπικοινωνιακά συστήματα και το Διαδίκτυο των Πραγμάτων (Internet of Things). Στο πλαίσιο αυτό, η ανάγκη για κατανεμημένα συστήματα ικανά να διαχειρίζονται μεγάλο όγκο και ταχύτητα δεδομένων με συνέπεια, επεκτασιμότητα και ανθεκτικότητα έχει καταστήσει πλατφόρμες όπως το Apache Flink και το Spark Structured Streaming βασικά εργαλεία της σύγχρονης μηχανικής δεδομένων.

Η παρούσα διπλωματική εργασία αποσκοπεί στη συγκριτική αξιολόγηση των δύο αυτών συστημάτων στο πεδίο της κατανεμημένης επεξεργασίας ροών. Αρχικά, παρουσιάζεται η εννοιολογική και αρχιτεκτονική θεμελίωση των συστημάτων, καθώς και τα βασικά χαρακτηριστικά τους σε επίπεδο υλοποίησης. Στη συνέχεια, εξετάζονται πέντε διαστάσεις σύγκρισης: η διαχείριση κατάστασης, η καθυστέρηση και η διαμέτρηση, η ανοχή σε σφάλματα, η αξιοποίηση πόρων και το κόστος επεξεργασίας, καθώς και ο χειρισμός καθυστερημένων δεδομένων.

Η αξιολόγηση βασίζεται σε θεωρητική ανάλυση και πρακτική ανάλυση, αντλώντας πληροφορίες από την επίσημη τεκμηρίωση και τη σχετική επιστημονική βιβλιογραφία. Μέσω της συστηματικής συγκριτικής προσέγγισης, αναδεικνύονται τα πλεονεκτήματα και οι περιορισμοί κάθε πλατφόρμας, με στόχο την τεκμηριωμένη υποστήριξη επιλογών για τη σχεδίαση και υλοποίηση συστημάτων ροής σε πραγματικό χρόνο.

Λέξεις-κλειδιά: Κατανεμημένα Συστήματα, Επεξεργασία Ροών, Apache Flink, Apache Spark, Ροές Δεδομένων, Διαχείριση Κατάστασης, Ανοχή σε Σφάλματα, Διαμέτρηση, Καθυστέρηση, Αξιοποίηση Πόρων, Κόστος Επεξεργασίας, Χειρισμός Καθυστερημένων Δεδομένων.

Abstract

Real-time stream data processing is a critical technological requirement for modern applications in domains such as finance, telecommunications systems, and the Internet of Things (IoT). In this context, the need for distributed systems capable of handling high data volume and velocity with consistency, scalability, and resilience has made platforms such as Apache Flink and Spark Structured Streaming essential tools in contemporary data engineering.

This thesis aims to provide a comparative evaluation of these two systems in the field of distributed stream processing. Initially, the conceptual and architectural foundations of the systems are presented, along with their key implementation characteristics. Subsequently, five comparison dimensions are examined: state management, latency and throughput, fault tolerance, resource utilization and processing cost, and handling of late-arriving data.

The evaluation is based on both theoretical analysis and practical insights, drawing from official documentation and relevant scientific literature. Through a systematic comparative approach, the strengths and limitations of each platform are highlighted, with the goal of providing well-documented support for selecting and designing real-time stream processing systems.

Keywords: Distributed Systems, Stream Processing, Apache Flink, Apache Spark, Data Streams, State Management, Fault Tolerance, Throughput, Latency, Resource Utilization, Processing Overhead, Handling Late Data.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

DAG	Directed Acyclic Graph

Κεφάλαιο 1

Εισαγωγή

1.1 Κατανεμημένα Συστήματα και Μεγάλα Δεδομένα

Η συνεχής αύξηση του όγκου των παραγόμενων δεδομένων, τόσο από επιχειρηματικές όσο και από επιστημονικές εφαρμογές, έχει καταστήσει αναγκαία την ανάπτυξη νέων υπολογιστικών προσεγγίσεων για την αποδοτική αποθήκευση και επεξεργασία τους. Η έννοια των Μεγάλων Δεδομένων (Big Data) περιγράφει αυτό το φαινόμενο, το οποίο χαρακτηρίζεται συνήθως από τα τρία παρακάτω:

- Όγκος (Volume): Αναφέρεται στην τεράστια ποσότητα δεδομένων που παράγονται και αποθηκεύονται καθημερινά.
- Ταχύτητα (Velocity): Περιγράφει την ταχύτητα με την οποία δημιουργούνται, μεταδίδονται και επεξεργάζονται τα δεδομένα.
- Ποικιλία (Variety): Αναφέρεται στους διαφορετικούς τύπους και μορφές δεδομένων (δομημένα, ημιδομημένα, αδόμητα).

Προκειμένου να αντιμετωπιστούν οι απαιτήσεις που επιβάλλει η διαχείριση και ανάλυση τέτοιου μεγέθους και ποικιλίας δεδομένων, έχουν επικρατήσει τα κατανεμημένα συστήματα (Distributed Systems) ως η βασική υπολογιστική υποδομή. Ένα κατανεμημένο σύστημα αποτελείται από πολλαπλούς υπολογιστικούς κόμβους (nodes) που συνεργάζονται για την εκτέλεση εργασιών, προσφέροντας δυνατότητες για αυξημένη υπολογιστική ισχύ, επεκτασιμότητα, αξιοπιστία και ανοχή σε σφάλματα.[3]

Σε ένα τέτοιο περιβάλλον, τα δεδομένα είναι κατανεμημένα μεταξύ των κόμβων και η επεξεργασία τους πραγματοποιείται με παράλληλο και ασύγχρονο τρόπο. Η υιοθέτηση αυτής της προσέγγισης επιτρέπει τη γρήγορη απόκριση σε ερωτήματα, την αποδοτική χρήση των διαθέσιμων πόρων και την υποστήριξη εφαρμογών που απαιτούν υψηλό ρυθμό ροής δεδομένων σε πραγματικό χρόνο.

Η ανάγκη για υποστήριξη εφαρμογών που λειτουργούν σε ροές δεδομένων (data streams) και όχι σε στατικά σύνολα, έχει οδηγήσει στην εξέλιξη εξειδικευμένων εργαλείων και υποδομών λογισμικού, όπως οι μηχανές επεξεργασίας ροών (stream processing engines). Οι μηχανές αυτές σχεδιάζονται ώστε να αξιοποιούν το μοντέλο των κατανεμημένων συστημάτων, προσφέροντας λύσεις που είναι ταυτόχρονα επεκτάσιμες και ανθεκτικές σε σφάλματα για την επεξεργασία ροών δεδομένων σε μεγάλη κλίμακα.

1.2 Ροές Δεδομένων και Επεξεργασία σε Πραγματικό Χρόνο

Η συνεχής ροή δεδομένων από ποικίλες πηγές, όπως αισθητήρες, κοινωνικά δίκτυα, εφαρμογές διαδικτύου και συστήματα συναλλαγών, έχει δημιουργήσει την ανάγκη για άμεση και διαρκή ανάλυση των δεδομένων αυτών. Η επεξεργασία σε πραγματικό χρόνο (real-time processing) αφορά την ικανότητα ανάλυσης και επεξεργασίας δεδομένων τη στιγμή που αυτά παράγονται, επιτρέποντας την άμεση λήψη αποφάσεων και την ανάδραση.[4]

Σε αντίθεση με τα παραδοσιακά συστήματα επεξεργασίας παρτίδων (batch processing), όπου τα δεδομένα συλλέγονται και επεξεργάζονται σε μεγάλες δόσεις μετά τη λήψη τους, η επεξεργασία ροών (stream processing) επιτρέπει την επεξεργασία των δεδομένων σε συνεχή ροή. Αυτό το μοντέλο προσφέρει σημαντικά πλεονεκτήματα σε εφαρμογές όπου η χρονική καθυστέρηση πρέπει να είναι ελάχιστη ή μηδενική, όπως στην ανίχνευση απάτης, την παρακολούθηση δικτύων, τη διαχείριση χρηματοπιστωτικών συναλλαγών και την υποστήριξη έξυπνων συστημάτων.

Η επεξεργασία ροών απαιτεί την υποστήριξη ειδικών μηχανισμών, όπως η διαχείριση κατάστασης (state management), η αντοχή σε σφάλματα (fault tolerance), και η κλιμάκωση (scalability), ώστε να μπορεί να ανταποκριθεί στις αυξανόμενες απαιτήσεις όγκου και ταχύτητας των δεδομένων. Οι μηχανές επεξεργασίας ροών, όπως το Apache Spark και το Apache Flink, παρέχουν πλατφόρμες που επιτρέπουν την υλοποίηση πολύπλοκων εφαρμογών πραγματικού χρόνου με αξιοπιστία και αποδοτικότητα.[5]

Η κατανόηση της φύσης των ροών δεδομένων και των βασικών προκλήσεων που σχετίζονται με την επεξεργασία τους αποτελεί θεμελιώδη παράμετρο για την επιλογή και τη βέλτιστη αξιοποίηση των κατάλληλων τεχνολογιών και εργαλείων.

1.3 Κατανεμημένα Συστήματα και Επεξεργαστές Ροών

Η επεξεργασία μεγάλου όγκου δεδομένων σε πραγματικό χρόνο απαιτεί τη χρήση κατανεμημένων συστημάτων, τα οποία διανέμουν το φορτίο εργασίας και τη διαχείριση δεδομένων σε πολλαπλούς υπολογιστικούς κόμβους. Τα κατανεμημένα συστήματα επιτρέπουν την κλιμάκωση των εφαρμογών, βελτιώνοντας την απόδοση, την αξιοπιστία και την ανθεκτικότητα σε σφάλματα.

Οι επεξεργαστές ροών (stream processors) αποτελούν εξειδικευμένα κατανεμημένα συστήματα σχεδιασμένα για την επεξεργασία συνεχών ροών δεδομένων με εγγυήσεις σχετικά με τη χρονική απόκριση και τη συνέπεια των αποτελεσμάτων. Σε αντίθεση με τα παραδοσιακά συστήματα παρτίδων, οι επεξεργαστές ροών πρέπει να αντιμετωπίζουν προκλήσεις όπως η διαχείριση κατάστασης, η αντοχή σε σφάλματα και η συντονισμένη εκτέλεση σε πολλαπλούς κόμβους.

Στο πλαίσιο αυτό, το Apache Spark και το Apache Flink αναδύονται ως δύο από τις πιο δημοφιλείς και ώριμες πλατφόρμες κατανεμημένης επεξεργασίας ροών. Το Apache Spark, με το Structured Streaming API, προσφέρει ένα αφαιρετικό μοντέλο βασισμένο σε μικροπαρτίδες (micro-batching)[6], ενώ το Apache Flink υλοποιεί την επεξεργασία δεδομένων με συνεχή ροή (true streaming)[5]. Η σύγκριση και αξιολόγηση αυτών των δύο συστημάτων αποτελεί κεντρικό άξονα αυτής της εργασίας.[3, 5]

Η βαθύτερη κατανόηση των αρχιτεκτονικών χαρακτηριστικών, των μηχανισμών διαχείρισης κατάστασης και των μοντέλων επεξεργασίας που υποστηρίζουν αυτοί οι επεξεργαστές είναι ουσιώδης για την αξιολόγηση της καταλληλότητας τους σε διαφορετικά σενάρια εφαρμογών και φορτίων εργασίας.

1.4 Διαστάσεις Σύγκρισης των Επεξεργαστών Ροών

Στην παρούσα εργασία, η συγκριτική αξιολόγηση του Apache Spark και του Apache Flink θα επικεντρωθεί σε πέντε θεμελιώδεις διαστάσεις απόδοσης και λειτουργικότητας, οι οποίες κρίνονται καίριες για την επιλογή κατάλληλου εργαλείου επεξεργασίας ροών δεδομένων.

Οι διαστάσεις αυτές αφορούν: τη διαχείριση κατάστασης (State Management), την καθυστέρηση και τη διαμέτρηση (Latency – Throughput), την ανοχή σε σφάλματα και την αποκατάσταση σφαλμάτων (Fault Tolerance – Fault Recovery), την αξιοποίηση πόρων και το κόστος επεξεργασίας (Resource Utilization – Processing Overhead) και την αντιμετώπιση καθυστερημένων δεδομένων (Late Data Handling). Καθεμία από αυτές αποτυπώνει διαφορετικές απαιτήσεις και προκλήσεις στον σχεδιασμό και την υλοποίηση συστημάτων πραγματικού χρόνου, ενώ στοχεύει στο να αναδείξει τα πλεονεκτήματα και τους περιορισμούς κάθε πλατφόρμας.[6, 7, 8]

Η πρώτη διάσταση αφορά τη διαχείριση κατάστασης (State Management). Η ικανότητα ενός συστήματος επεξεργασίας ροών να αποθηκεύει, ανακτά και ενημερώνει διαρκώς ένα σύνολο κατάστασης—είτε αυτό είναι απλός μετρητής, σύνθετες συναθροίσεις ή δομές δεδομένων με χρονοσήμανση—είναι κρίσιμη για πολύπλοκες εφαρμογές, όπως η παρακολούθηση χρηματοπιστωτικών συναλλαγών, τα συστήματα συστάσεων και η ανίχνευση απάτης. Η διερεύνηση αυτής της διάστασης θα εξετάσει τους τρόπους με τους οποίους το Spark και το Flink υλοποιούν την αποθήκευση κατάστασης (π.χ. state backends), τους μηχανισμούς στιγμιοτύπων (checkpointing και snapshotting), καθώς και τη διαχείριση της συνέπειας (π.χ. exactly-once semantics) και της απόδοσης κατά την ανάκτηση μετά από σφάλμα.[5, 6]

Η δεύτερη διάσταση αφορά την καθυστέρηση (latency) και τη διαμέτρηση (throughput). Η καθυστέρηση ορίζεται ως ο χρόνος που απαιτείται για να επεξεργαστεί ένα γεγονός από τη στιγμή που εισέρχεται στο σύστημα έως τη στιγμή που παράγεται το αποτέλεσμα, ενώ η διαμέτρηση αναφέρεται στον αριθμό γεγονότων που μπορούν να επεξεργαστούν ανά μονάδα χρόνου. Η σωστή ισορροπία μεταξύ χαμηλής καθυστέρησης και υψηλής διαμέτρησης είναι κρίσιμη, αφού ορισμένες εφαρμογές (π.χ. χρηματιστηριακές συναλλαγές) απαιτούν ελάχιστη χρονική υστέρηση, ενώ άλλες επιδιώκουν να διαχειριστούν τεράστιο όγκο δεδομένων με μέγιστη ταχύτητα.[4, 7, 8]

Η τρίτη διάσταση επικεντρώνεται στην ανοχή σε σφάλματα (fault tolerance) και την αποκατάσταση σφαλμάτων (fault recovery). Ένα σύστημα επεξεργασίας ροών πρέπει να διασφαλίζει ότι, σε περίπτωση αποτυχίας κόμβου ή δικτύου, δεν χάνεται σημαντικό τμήμα των δεδομένων ούτε επαναλαμβάνεται επεξεργασία ήδη ολοκληρωμένων γεγονότων. Η αξιολόγηση περιλαμβάνει τους μηχανισμούς στιγμιοτύπων και τις εγγυήσεις παράδοσης (Ακριβώς-Μία-Επεξεργασία και Τουλάχιστον-Μία-Επεξεργασία). Επίσης εξετάζεται ο χρόνος επαναφοράς (recovery time) και η συνέπεια της κατάστασης μετά την ανάκτηση.[5, 6, 7, 8]

Η τέταρτη διάσταση αναφέρεται στην αξιοποίηση πόρων (resource utilization) και το κόστος επεξεργασίας (processing overhead). Εδώ εξετάζεται πόση μνήμη, CPU, δίσκος και δίκτυο

απαιτούνται προκειμένου να επιτευχθεί συγκεκριμένη απόδοση. Επιπλέον, αναλύεται ο επιπλέον φόρτος που υποβάλλεται στο σύστημα από λειτουργίες όπως η διαχείριση κατάστασης, τα state backends, ο συγχρονισμός καταστάσεων και η εισαγωγή στιγμιοτύπων. Η σύγκριση εστιάζει στο ότι, ενώ κάποια πλατφόρμα μπορεί να προσφέρει υψηλή απόδοση, αυτή μπορεί να συνοδεύεται από αυξημένες απαιτήσεις πόρων, γεγονός που επηρεάζει το συνολικό κόστος λειτουργίας.[3, 7]

Η πέμπτη και τελευταία διάσταση αφορά τον χειρισμό καθυστερημένων δεδομένων (late data handling). Κατά την επεξεργασία ροών, δεδομένα ενδέχεται να φτάσουν με χρονική καθυστέρηση (π.χ. λόγω δικτυακών αποκλεισμών ή καθυστερήσεων στην παραγωγή τους). Η δυνατότητα ενός συστήματος να χειριστεί τέτοιες μεταγενέστερες εγγραφές χωρίς να αλλοιωθούν τα αποτελέσματα είναι κρίσιμη για την ακρίβεια της ανάλυσης. Στο πλαίσιο αυτό, αξιολογείται η υποστήριξη για παράθυρα χρόνου (time windows), η δυνατότητα αναπροσαρμογής (re-processing) και οι πολιτικές ανεκτικότητας (allowed lateness) που εφαρμόζει κάθε πλατφόρμα.[4, 7, 8]

Η περιγραφή αυτών των πέντε διαστάσεων θα καθοδηγήσει την αναλυτική σύγκριση που ακολουθεί στα επόμενα κεφάλαια, προκειμένου να καταδειχθούν συστηματικά τα πλεονεκτήματα και οι περιορισμοί του Apache Spark και του Apache Flink σε πραγματικά σενάρια επεξεργασίας ροών.

Κεφάλαιο 2

Διαστάσεις Σύγκρισης

2.1 Διαχείριση Κατάστασης

Η διαχείριση κατάστασης (state management) αναφέρεται στον τρόπο με τον οποίο ένας επεξεργαστής ροής διαχειρίζεται και αξιοποιεί την κατάσταση που σχετίζεται με κάθε γεγονός που εισάγεται κατά την επεξεργασία. Υπάρχουν δύο κύριες περιπτώσεις στη διαχείριση κατάστασης στη ροή δεδομένων: η επεξεργασία ροής με διατήρηση κατάστασης (stateful stream processing), όπου ο επεξεργαστής διατηρεί και ενημερώνει την κατάσταση μεταξύ των γεγονότων, και η επεξεργασία ροής χωρίς διατήρηση κατάστασης (stateless stream processing), όπου κάθε γεγονός επεξεργάζεται ανεξάρτητα, χωρίς τη διατήρηση οποιασδήποτε κατάστασης. [5, 6]

2.1.1 Επεξεργασία ροής χωρίς διατήρηση κατάστασης

Σχημα 2.1: Σχηματική Απεικόνιση της επεξεργασίας ροής χωρίς διατήρηση κατάστασης.[1]

Η επεξεργασία ροής χωρίς κατάσταση περιλαμβάνει την ανεξάρτητη διαχείριση κάθε γεγονότος, χωρίς αναφορά σε προηγούμενα ή μελλοντικά γεγονότα. Αυτή η προσέγγιση εφαρμόζεται σε περιπτώσεις όπου μόνο το πιο πρόσφατο γεγονός έχει σημασία και δεν απαιτείται ιστορικό πλαίσιο. [4]

Χαρακτηριστικά της επεξεργασίας ροής χωρίς διατήρηση κατάστασης

Η επεξεργασία ροής χωρίς διατήρηση κατάστασης είναι μια υπολογιστική προσέγγιση όπου κάθε γεγονός (event) αντιμετωπίζεται ως ανεξάρτητο από τα υπόλοιπα. Κατά την εκτέλεση, δεν αποθηκεύεται πληροφορία σχετικά με το παρελθόν ή την αλληλουχία των γεγονότων, γεγονός που καθορίζει και τα βασικά χαρακτηριστικά της. [7]

Ένα βασικό χαρακτηριστικό της είναι η απλότητα στην υλοποίηση, καθώς δεν απαιτείται σύνθετη διαχείριση κατάστασης. Επιπλέον, η ευκολία στην οριζόντια κλιμάκωση επιτρέπει την προσθήκη επεξεργαστών με ελάχιστο κόπο, γεγονός που την καθιστά ιδιαίτερα κατάλληλη για κατανεμημένα περιβάλλοντα. Η χαμηλή υπολογιστική και μνημονική επιβάρυνση, καθώς και η μικρή καθυστέρηση στην επεξεργασία, είναι απόρροια της απουσίας πρόσβασης ή ενημέρωσης κατάστασης.

Παράλληλα, η ανάκαμψη από σφάλματα είναι πιο άμεση, αφού δεν υπάρχει κατάσταση προς αποκατάσταση, ενώ η παραλληλία διευκολύνεται, καθώς δεν απαιτείται συγχρονισμός μεταξύ παράλληλων ροών ή διατήρηση συνέπειας κατάστασης.

Ωστόσο, αυτή η προσέγγιση δεν είναι κατάλληλη για σενάρια που απαιτούν συσχέτιση μεταξύ πολλών γεγονότων, όπως ομαδοποίηση (grouping), συνενώσεις (joins), ή υπολογισμούς που βασίζονται σε ιστορικό πλαίσιο. Δεν υποστηρίζει λειτουργίες που απαιτούν καταμέτρηση, εντοπισμό προτύπων ή τάσεων, ούτε μπορεί να εγγυηθεί μοναδικότητα, διατήρηση σειράς ή ακριβή μία επεξεργασία ανά γεγονός (exactly-once semantics).

Συνεπώς, η επεξεργασία ροής χωρίς κατάσταση ενδείκνυται για εφαρμογές με ελαφριά, ανεξάρτητα δεδομένα, όπως φιλτράρισμα, εμπλουτισμός με στατικά δεδομένα ή έλεγχος εγκυρότητας. Αντίθετα, για πιο σύνθετες ή αναλυτικές λειτουργίες, απαιτούνται τεχνικές που υποστηρίζουν διατήρηση και αξιοποίηση κατάστασης.

2.1.2 Επεξεργασία ροής με διατήρηση κατάστασης

Η επεξεργασία ροής με διατήρηση κατάστασης (stateful stream processing) περιλαμβάνει την ανάλυση μιας συνεχούς ροής δεδομένων σε πραγματικό χρόνο, διατηρώντας ταυτόχρονα μια κατάσταση ή ένα πλαίσιο που προκύπτει από προηγουμένως επεξεργασμένα δεδομένα. Αυτή η διατηρούμενη κατάσταση επιτρέπει στο σύστημα να αναγνωρίζει πρότυπα, να παρακολουθεί αλλαγές και να λαμβάνει αποφάσεις βασισμένες τόσο σε ιστορικές όσο και σε τρέχουσες πληροφορίες της ροής δεδομένων. [5]

Χαρακτηριστικά της επεξεργασίας ροής με διατήρηση κατάστασης

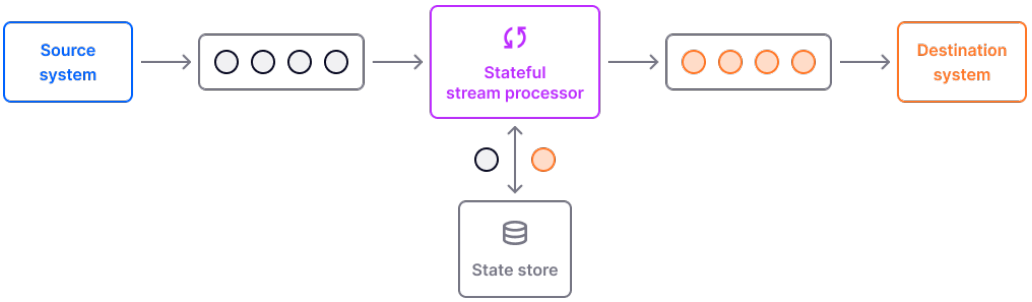
Η επεξεργασία ροής με διατήρηση κατάστασης αποτελεί βασική τεχνική για την υλοποίηση σύνθετων ροών δεδομένων, όπου η κατανόηση του πλαισίου και η αλληλεξάρτηση μεταξύ γεγονότων είναι κρίσιμη. Σε αυτή την προσέγγιση, το σύστημα διατηρεί πληροφορίες από προηγούμενα γεγονότα, οι οποίες χρησιμοποιούνται για την καθοδήγηση μελλοντικών ενεργειών ή υπολογισμών.

Ένα κύριο χαρακτηριστικό της είναι η ικανότητα διαχείρισης ιστορικών δεδομένων και πλαισίου, επιτρέποντας πιο πλούσια και ακριβή ανάλυση. Μέσω της κατάστασης, το σύστημα μπορεί να ανιχνεύει πρότυπα, να συσχετίζει γεγονότα, και να λαμβάνει αποφάσεις σε πραγματικό χρόνο. Αυτό καθιστά την τεχνική κατάλληλη για εφαρμογές όπως η ανίχνευση απάτης, η παρακολούθηση χρηστών και η βελτιστοποίηση ροών εργασιών. [6, 8]

Η εξατομίκευση αποτελεί ένα ακόμη βασικό χαρακτηριστικό, καθώς η διατήρηση πληροφορίας για προηγούμενες αλληλεπιδράσεις επιτρέπει τη δυναμική προσαρμογή της συμπεριφοράς του συστήματος σε κάθε χρήστη. Ενδεικτικά παραδείγματα είναι τα συστήματα προτάσεων και οι μηχανισμοί προσωποποιημένης ειδοποίησης.

Η ευελιξία στη λήψη αποφάσεων ενισχύεται σε περιβάλλοντα που απαιτούν συνδυασμό παλαιών και νέων δεδομένων, όπως οι χρηματοοικονομικές συναλλαγές, οι εφαρμογές IoT, και η βιομηχανική παρακολούθηση. Επιπλέον, η δυνατότητα παρακολούθησης της κατάστασης με την πάροδο του χρόνου ευνοεί σενάρια μηχανικής μάθησης και τεχνητής νοημοσύνης, καθώς επιτρέπει την προσαρμογή, μάθηση και συνεχή βελτίωση των αποτελεσμάτων.

Ωστόσο, η επεξεργασία με κατάσταση συνοδεύεται από χαρακτηριστικά που σχετίζονται με την αυξημένη πολυπλοκότητα στην υλοποίηση, καθώς απαιτεί διαχείριση και συγχρονισμό της κατάστασης μεταξύ πολλαπλών επεξεργαστών. Η επεκτασιμότητα είναι πιο απαιτητική, ιδιαίτερα σε κατανεμημένα περιβάλλοντα, καθώς η κατάσταση πρέπει να κατανεμηθεί ή να ανακτηθεί αποτελεσματικά. Επιπλέον, απαιτείται ισχυρή υποδομή αποθήκευσης, όπως κατανεμημένες βάσεις δεδομένων ή αποθηκευτικά συστήματα στη μνήμη (in-memory stores), τα οποία προσθέτουν υπολογιστικό και λειτουργικό βάρος στο σύστημα. [7, 8]



Σχημα 2.2: Σχηματική Απεικόνιση της επεξεργασίας ροής με διατήρηση κατάστασης.[1]

Η επεξεργασία ροής με διατήρηση κατάστασης είναι απαραίτητη σε σύνθετες εφαρμογές με υψηλές απαιτήσεις ακρίβειας, ευφυΐας και προσαρμογής, υπό την προϋπόθεση ότι το σύστημα έχει σχεδιαστεί κατάλληλα για να καλύψει τις αυξημένες ανάγκες σε πόρους, αξιοπιστία και απόδοση.

2.2 Καθυστέρηση - Διαμέτρηση

Η καθυστέρηση αποτελεί βασικό μέτρο στην επεξεργασία ροών δεδομένων και ορίζεται ως ο χρόνος που μεσολαβεί από τη στιγμή κατά την οποία ένα συμβάν εισέρχεται στο σύστημα έως ότου παράγεται το αντίστοιχο αποτέλεσμα. Υπάρχουν διαφορετικά επίπεδα καθυστέρησης που πρέπει να λαμβάνονται υπόψη: η ενδοσυστημική (processing latency), δηλαδή ο χρόνος που απαιτείται για την επεξεργασία του συμβάντος από τους εσωτερικούς μηχανισμούς του επεξεργαστή ροών, και η καθυστέρηση από άκρο σε άκρο (end-to-end latency), που περιλαμβάνει και την καθυστέρηση μεταφοράς των δεδομένων μέσω δικτύου, την αναμονή σε ουρές εισόδου και τις ενδεχόμενες αναμονές σε σημεία ανακατεύθυνσης. Η ελαχιστοποίηση της καθυστέρησης είναι κρίσιμη σε εφαρμογές όπου απαιτείται σχεδόν άμεση αντίδραση σε συμβάντα, όπως στην ανίχνευση απάτης, στη διαχείριση χρηματοοικονομικών συναλλαγών ή στην παρακολούθηση συστημάτων ασφάλειας. Ωστόσο, η προσπάθεια μείωσης της καθυστέρησης συχνά έρχεται σε αντίθεση με άλλες απαιτήσεις, όπως η ορθότητα των αποτελεσμάτων και η αξιοπιστία· για παράδειγμα, η εφαρμογή μηχανισμών επαναφοράς σε περίπτωση σφάλματος ή η παροχή ακριβούς κατάστασης μπορεί να επιβαρύνει τον χρόνο απόκρισης. [4]

Η διαμέτρηση αντιστοιχεί στον αριθμό συμβάντων που ένα σύστημα μπορεί να επεξεργαστεί ανά μονάδα χρόνου, συνήθως μετρημένη σε γεγονότα ανά δευτερόλεπτο (events/sec). Σε περιβάλλοντα μεγάλης κλίμακας, υψηλή διαμέτρηση είναι επιθυμητή για να εξασφαλιστεί ότι το σύστημα θα ανταποκριθεί στον όγκο των εισερχόμενων ροών χωρίς να δημιουργούνται συσσωρευμένα σημεία συμφόρησης (bottlenecks). Η επίτευξη υψηλής διαμέτρησης προϋποθέτει αποτελεσματική παράλληλη εκτέλεση, ορθολογική διαχείριση πόρων (π.χ. καταμερισμός μνήμης, CPU και I/O) και μηχανισμούς ανάσχεσης ροής (backpressure) για να αποτραπεί ο υπερβολικός φόρτος στα υποσυστήματα. Παράλληλα, η διαμέτρηση συνδέεται άμεσα με τη χωρητικότητα κλιμάκωσης (scaling capacity) του συστήματος· όσο μεγαλύτερος είναι ο αριθμός των διαθέσιμων επεξεργαστικών μονάδων (π.χ. κόμβων ή εκτελεστών), τόσο υψηλότερη μπορεί να είναι η συνολική διαμέτρηση, εφόσον ο τρόπος καταμερισμού των δεδομένων και του φορτίου είναι αποτελεσματικός. [7]

Ο συνδυασμός των δύο αυτών μεγεθών —καθυστέρηση και διαμέτρηση— δημιουργεί ένα θεμελιώδες δίλημμα σχεδιασμού (latency-throughput trade-off). Σε ορισμένες περιπτώσεις,

η αύξηση του μεγέθους των παρτίδων (batches) ή η εφαρμογή τεχνικών συγκέντρωσης δεδομένων (π.χ. window aggregation) βελτιώνει τη συνολική διαμέτρηση, αλλά ταυτόχρονα επιβαρύνει την καθυστέρηση, καθώς ο χρόνος αναμονής για τη συγκέντρωση ενός συνόλου συμβάντων αυξάνει. Αντίστροφα, μια πιο συνεχή ροή επεξεργασίας (χωρίς παρτίδες) μπορεί να διατηρήσει χαμηλή καθυστέρηση, όμως σε περίπτωση μεγάλης εισροής δεδομένων μπορεί να απαιτηθεί επιπλέον συντονισμός πόρων ώστε να διατηρηθεί η διαμέτρηση σε επαρκή επίπεδα. Η ισορροπία αυτών των δύο παραμέτρων εξαρτάται από τους στόχους της εφαρμογής — για παράδειγμα, ένα σύστημα παρακολούθησης υγείας ασθενών μπορεί να απαιτεί απόλυτα μικρή καθυστέρηση, ενώ μια ανάλυση καταγραφών (logs) για στατιστικά μπορεί να δίνει προτεραιότητα στη διαμέτρηση. [6]

Για τη μέτρηση και την αξιολόγηση αυτών των μεγεθών, χρησιμοποιούνται συνήθως εργαλεία αξιολόγησης απόδοσης (benchmarking) που καταγράφουν τον χρόνο απόκρισης για διάφορα επίπεδα φόρτου, καθώς και τον αριθμό των γεγονότων που επεξεργάζονται επιτυχώς ανά δευτερόλεπτο. Επιπλέον, ο σχεδιασμός μιας υποδομής επεξεργασίας ροών πρέπει να λαμβάνει υπόψη παράγοντες όπως η διακύμανση της εισροής (data variability), οι συνθήκες ανάσχεσης ροής, οι περιορισμοί του δικτύου και του υλικού, καθώς και η πολυπλοκότητα των λογικών κανόνων επεξεργασίας (π.χ. πολύπλοκα aggregations, join μεταξύ ροών). Συνεπώς, κάθε πλατφόρμα επιδιώκει να παρέχει παραμετρικές ρυθμίσεις (π.χ. ρύθμιση μεγέθους παραθύρου, προσαρμογή επιπέδου παραλληλίας) που επιτρέπουν στον χρήστη να βελτιστοποιήσει την απόδοση ανάλογα με τις απαιτήσεις σε καθυστέρηση και διαμέτρηση. Τέλος, η τήρηση συμφωνημένων επιπέδων υπηρεσιών (SLA) απαιτεί συνεχή παρακολούθηση και δυναμική προσαρμογή των πόρων για να διασφαλιστεί ότι η υποδομή ανταποκρίνεται στα επιθυμητά περιθώρια απόκρισης και φόρτου.

2.3 Ανοχή σε σφάλματα

Η ανοχή σε σφάλματα αποτελεί κρίσιμη πτυχή στη σχεδίαση συστημάτων επεξεργασίας ροών δεδομένων, καθώς διασφαλίζει τη συνεχή λειτουργία και την αξιοπιστία ανεξάρτητα από απρόβλεπτες αποτυχίες κόμβων, δικτύου ή άλλων υποσυστημάτων. Η απώλεια δεδομένων ή η επανάληψη επεξεργασίας γεγονότων μπορεί να έχει σοβαρές επιπτώσεις σε εφαρμογές που απαιτούν ακρίβεια και σταθερότητα, όπως οι χρηματοπιστωτικές συναλλαγές, η παρακολούθηση υποδομών και τα συστήματα παραγγελιών σε πραγματικό χρόνο. [5, 8]

Σε ένα κατανεμημένο περιβάλλον, οι αποτυχίες ενδέχεται να προκληθούν από διαφοροποιημένους παράγοντες, όπως αστοχία μονάδας επεξεργασίας, διακοπή σύνδεσης μεταξύ κόμβων,

ή ανεπαρκείς πόροι (π.χ. μνήμη, CPU). Η ανοχή σε σφάλματα επιτυγχάνεται μέσω μηχανισμών που καταγράφουν περιοδικά την εσωτερική κατάσταση του συστήματος σε επίμονο αποθηκευτικό μέσο, διασφαλίζοντας ότι, σε περίπτωση σφάλματος, το σύστημα μπορεί να επανεκκινήσει από το τελευταίο συνεπές σημείο χωρίς απώλεια ή διπλή καταγραφή δεδομένων.

Οι βασικές έννοιες που εμπλέκονται σε αυτό το πεδίο είναι οι εξής:

Στιγμιότυπα Κατάστασης και Checkpointing

Τα στιγμιότυπα (snapshots ή checkpoints) αποτελούν στιγμές στο χρόνο κατά τις οποίες καταγράφονται τόσο η εσωτερική κατάσταση των λειτουργιών επεξεργασίας όσο και τα μεταδεδομένα που απαιτούνται για την αποκατάσταση σε περίπτωση σφάλματος. Η διαδικασία δημιουργίας στιγμιοτύπων περιλαμβάνει τον συγχρονισμό των σημείων κατάστασης όλων των κατανεμημένων κόμβων—διασφαλίζοντας ότι οι ενδιάμεσες μεταβάσεις δεν οδηγούν σε ασυνέπεια—και την αποθήκευσή τους σε αξιόπιστο αποθηκευτικό μέσο (π.χ. δίσκος δικτύου, κατανεμημένο αρχείο). [7]

Εγγυήσεις Παράδοσης

Οι εγγυήσεις παράδοσης (delivery semantics) [4] καθορίζουν με ποιον τρόπο διαχειρίζεται το σύστημα την περίπτωση αποτυχίας. Συγκεκριμένα, διακρίνονται τρεις βασικοί τύποι:

- Τουλάχιστον-Μία-Επεξεργασία (At-Least-Once): Κάθε γεγονός επεξεργάζεται τουλάχιστον μία φορά. Σε περίπτωση επαναφοράς, κάποια γεγονότα μπορεί να επεξεργαστούν ξανά, με αποτέλεσμα πιθανή διπλή καταγραφή ή διπλή ενέργεια.
- Το-Πολύ-Μία-Επεξεργασία (At-Most-Once): Κάθε γεγονός επεξεργάζεται το πολύ μία φορά. Σε περίπτωση σφάλματος, κάποια γεγονότα μπορεί να χαθούν, καθώς δεν επαναπροωθούνται μετά την αποτυχία.
- Ακριβώς-Μία-Επεξεργασία (Exactly-Once): Κάθε γεγονός επεξεργάζεται ακριβώς μία φορά, διασφαλίζοντας ότι ούτε χάνεται ούτε επαναλαμβάνεται η επεξεργασία του, υποστηρίζοντας έτσι την αυστηρότερη μορφή συνέπειας.

Η επιλογή του κατάλληλου μοντέλου εξαρτάται από τις απαιτήσεις της εφαρμογής: εφαρμογές που δεν μπορούν να ανεχτούν διπλοεπεξεργασία (π.χ. οικονομικές μεταφορές) προτιμούν Ακριβώς-Μία-Επεξεργασία, ενώ άλλες με λιγότερο αυστηρές ανάγκες μπορεί να επιλέξουν Τουλάχιστον-Μία-Επεξεργασία ή Το-Πολύ-Μία-Επεξεργασία για μείωση κόστους.

Αποκατάσταση Σφαλμάτων

Σε περίπτωση αποτυχίας, το σύστημα επανεκκινεί από το πιο πρόσφατο συνεπές στιγμιότυπο. Η διαδικασία αποκατάστασης (fault recovery) περιλαμβάνει την ανάκτηση όλων των σχετικών δεδομένων (state, μεταδεδομένα, log επεξεργασίας) από τον αποθηκευτικό φορέα και τη συνέχιση της επεξεργασίας από το σημείο εκείνο. Η ταχύτητα και η ακρίβεια της αποκατάστασης επηρεάζουν άμεσα το συνολικό χρόνο αποκατάστασης recovery time και την ικανότητα εξυπηρέτησης των υπηρεσιών χωρίς μακρά διακοπή.

Σημεία Αποθήκευσης Κατάστασης (Savepoints)

Τα σημεία αποθήκευσης κατάστασης αποτελούν χειροκίνητα ενεργοποιούμενα στιγμιότυπα κατάστασης, επιτρέποντας στον χρήστη να αποθηκεύσει την τρέχουσα κατάσταση του συστήματος πριν από σημαντικές αλλαγές, όπως αναβάθμιση λογισμικού ή μεταβολή της ρύθμισης παραλληλίας. Σε αντίθεση με τα αυτόματα στιγμιότυπα, τα σημεία αποθήκευσης κατάστασης παρέχουν μεγαλύτερο έλεγχο στον προγραμματιστή, καθώς μπορούν να ανακτηθούν ακόμη και αν τα αυτόματα στιγμιότυπα έχουν διαγραφεί ή αντικατασταθεί.

Αυτοματοποιημένη Ανίχνευση Αποτυχιών και Επαναδρομολόγηση

Ένα πλήρες σύστημα ανοχής σε σφάλματα πρέπει να αναγνωρίζει αυτόματα όταν ένας κόμβος ή μια διεργασία αποτυγχάνει και να αναδιανέμει το φόρτο σε εναλλακτικούς πόρους. Η ανίχνευση αποτυχίας (failure detection) υλοποιείται συνήθως μέσω πρωτοκόλλων heartbeat[9] ή gossip[10], ενώ η επαναδρομολόγηση (re-scheduling) του φορτίου πραγματοποιείται με βάση πολιτικές που διατηρούν ισορροπία ανάμεσα στην απόδοση και την ελαχιστοποίηση των καθυστερήσεων.

Συμμετρία μεταξύ Συνέπειας και Απόδοσης

Οι μηχανισμοί ανοχής σφαλμάτων επιβαρύνουν τη συνολική απόδοση του συστήματος, καθώς απαιτούν επιπλέον ενέργειες—όπως η αποθήκευση κατάστασης, η ταυτόχρονη εγγραφή σε δίσκο ή δικτυακό φορέα, και η ανταλλαγή μηνυμάτων συγχρονισμού μεταξύ κόμβων. Κατά συνέπεια, υπάρχει ένα θεμελιώδες δίλημμα σχεδιασμού ανάμεσα στην αυστηρότητα της συνέπειας και τη συνολική απόδοση (consistency-performance trade-off). Συστήματα που επιδιώκουν να παρέχουν exactly-once semantics συνήθως θυσιάζουν μέρος της διαμέτρησης ή αυξάνουν την καθυστέρηση, ενώ επιλογές με χαλαρότερους κανόνες συνέπειας (π.χ. Τουλάχιστον-Μία-Επεξεργασία) επιτρέπουν ταχύτερη επεξεργασία με την πιθανότητα επανάληψης δεδομένων.

Επαλήθευση και Παρακολούθηση

Η αποτελεσματική ανοχή σφαλμάτων προϋποθέτει την ύπαρξη μηχανισμών παρακολούθησης (monitoring) και επαλήθευσης (verification) των διαδικασιών αποκατάστασης. Μέσω καταγραφής συμβάντων, μετρικών καθυστέρησης αποκατάστασης και ποσοστών αποτυχίας, ο διαχειριστής μπορεί να αναλύει την αξιοπιστία του συστήματος, να προσαρμόζει παραμέτρους (π.χ. συχνότητα στιγμιοτύπων) και να επαληθεύει ότι οι απαιτήσεις των συμφωνημένων επιπέδων υπηρεσιών ικανοποιούνται. [8]

Συνοψίζοντας, η ανοχή σε σφάλματα συνιστά τον ακρογωνιαίο λίθο των αξιόπιστων συστημάτων επεξεργασίας ροών και απαιτεί την υιοθέτηση συνδυαστικών μηχανισμών για την καταγραφή, αποθήκευση και αποκατάσταση της κατάστασης, την παροχή σωστών εγγυήσεων παράδοσης και την ισορροπία μεταξύ συνέπειας και απόδοσης. Αυτοί οι μηχανισμοί θα αναλυθούν επιμέρους στα επόμενα κεφάλαια, όπου θα συγκριθούν η υλοποίηση και οι επιδόσεις τους στα Apache Spark και Apache Flink.

2.4 Αξιοποίηση Πόρων - Κόστος Επεξεργασίας

Η αξιοποίηση πόρων (Resource Utilization) και το κόστος επεξεργασίας (Processing Overhead) αποτελούν κεντρικά στοιχεία σχεδιασμού και λειτουργίας συστημάτων επεξεργασίας ροών δεδομένων. Κατά την εκτέλεση μιας διαδρομής επεξεργασίας ροών δεδομένων (stream processing pipeline), οι διαθέσιμοι πόροι—όπως η μονάδα κεντρικής επεξεργασίας (CPU), η μνήμη (RAM memory), οι δίσκοι εισόδου/εξόδου (I/O) και το δίκτυο (network)—πρέπει να διαχειρίζονται με τρόπο που επιτρέπει την απρόσκοπτη ροή δεδομένων, χωρίς να προκαλούνται σημεία συμφόρησης ή υπερβολική σπατάλη πόρων. [7]

Το κόστος επεξεργασίας αναφέρεται στο επιπρόσθετο φορτίο που επιβάλλεται στο σύστημα όταν εκτελούνται συγκεκριμένες ενέργειες, όπως η διαχείριση κατάστασης , η καταγραφή (logging), η σειριοποίηση (serialization), η διαδικασία στιγμιοτύπων και ο συγχρονισμός μεταξύ κατανεμημένων κόμβων (coordination). Κάθε μία από αυτές τις ενέργειες επιβαρύνει τους πόρους: για παράδειγμα, η καταγραφή στιγμιοτύπων κατάστασης ενδέχεται να αυξήσει τη χρήση δίσκων και δικτύου, ενώ η σειριοποίηση δεδομένων μπορεί να καταναλώσει επιπλέον CPU και μνήμη.

Η ισορροπία μεταξύ αποδοτικής επεξεργασίας και ελάχιστου κόστους επεξεργασίας είναι απαραίτητη για τη διατήρηση της συνολικής απόδοσης. Εάν το κόστος επεξεργασίας είναι πολύ υψηλό, μειώνεται η διαμέτρηση και αυξάνεται η καθυστέρηση, ενώ αν οι πόροι δεν αξιοποιούνται πλήρως, το σύστημα ενδέχεται να λειτουργεί αναποτελεσματικά, με χαμηλή απόδοση ανά μονάδα κόστους (cost-effectiveness). Έτσι, ο σχεδιασμός μιας διαδρομής επεξεργασίας

ροών δεδομένων πρέπει να λάβει υπόψη τόσο την αποδοτικότητα εκτέλεσης των εργασιών όσο και τη δυνατότητα δυναμικής προσαρμογής των πόρων (elasticity).

Η παρακολούθηση των πόρων σε πραγματικό χρόνο είναι κρίσιμη για την ανίχνευση ανισορροπιών. Μέσω μετρήσεων, όπως ποσοστό χρήσης CPU, στατιστικά μνήμης (π.χ. heap usage), ρυθμός δίσκου (disk I/O rate) και φόρτος δικτύου (network throughput), ο διαχειριστής μπορεί να εντοπίζει σημεία συμφόρησης και να αναπροσαρμόζει την παραλληλία ή την κατανομή των εργασιών. Σε συστήματα με αυτόματη κλιμάκωση (autoscaling), οι μετρήσεις αυτές τροφοδοτούν πολιτικές που αυξομειώνουν τον αριθμό των κόμβων ή τις εκτελεστικές μονάδες, ώστε να διασφαλίζεται ότι οι απαιτήσεις σε απόδοση ικανοποιούνται χωρίς υπερβολική σπατάλη πόρων.

Ένας επιπλέον παράγοντας στο κόστος επεξεργασίας είναι η πολυπλοκότητα των λογικών κανόνων που εφαρμόζονται σε κάθε γεγονός, όπως σύνθετες συναθροίσεις, ένωση ροών ή υπολογισμοί παραθύρων. Όσο πιο πολύπλοκοι είναι οι υπολογισμοί, τόσο μεγαλύτερη είναι η κατανάλωση CPU και μνήμης. Γι' αυτό, οι πλατφόρμες συχνά παρέχουν εργαλεία βελτιστοποίησης, όπως αυτόματη επιλογή στρατηγικών εκτέλεσης (execution strategies) και προδιαθέσεις χειρισμού παραθύρων (window configurations), που επιτρέπουν στον χρήστη να μειώσει το κόστος επεξεργασίας χωρίς να θυσιάσει απαραίτητα τη λειτουργικότητα.

Τέλος, η αξιολόγηση του συνολικού κόστους επεξεργασίας πρέπει να περιλαμβάνει το συνολικό κόστος υποδομών (infrastructure cost), δηλαδή τη χρέωση για χρησιμοποιημένους πόρους (π.χ. ωριαίο κόστος CPU, κόστος αποθήκευσης ανά GB, κόστος δικτύου ανά GB μεταφοράς). Η κατανόηση του πώς οι ενέργειες επεξεργασίας—όπως συχνότητα στιγμιοτύπων, μέγεθος κατάστασης, και ρυθμίσεις παράλληλης εκτέλεσης—επηρεάζουν το κόστος αυτό, επιτρέπει στον χρήστη να σχεδιάσει τον τρόπο λειτουργίας του συστήματος με γνώμονα την οικονομική αποδοτικότητα.

Συνοψίζοντας, η ενδελεχής κατανόηση της αξιοποίησης πόρων και των πηγών κόστους επεξεργασίας αποτελεί προϋπόθεση για τη δημιουργία κλιμακούμενων, αποδοτικών και οικονομικά βιώσιμων υποδομών επεξεργασίας ροών δεδομένων. Η παρακολούθηση και η δυναμική προσαρμογή των πόρων, σε συνδυασμό με κατάλληλες βελτιστοποιήσεις στις λογικές επεξεργασίας, διασφαλίζουν ότι το σύστημα θα ανταποκριθεί στις απαιτήσεις απόδοσης χωρίς περιττά έξοδα.

2.5 Χειρισμός Καθυστερημένων Δεδομένων

Όταν η επεξεργασία ροών δεδομένων βασίζεται σε χρονικά παράθυρα, ανακύπτει το ζήτημα των καθυστερημένων δεδομένων (late data), δηλαδή των εγγραφών που φτάνουν στο σύστημα μετά τη λήξη του παραθύρου ή μετά την αρχική επεξεργασία του σχετικού χρονικού διαστήματος. Ο χειρισμός των καθυστερημένων δεδομένων είναι κρίσιμος για την ακρίβεια των αποτελεσμάτων, καθώς επιλογές που αγνόησαν ή απέκλεισαν τέτοια δεδομένα μπορεί να οδηγήσουν σε ελλιπή ή λανθασμένα συμπεράσματα.

Καταρχάς, η διάκριση μεταξύ χρονικού σημείου δεδομένου (event time) και χρονικού σημείου επεξεργασίας (processing time) παίζει βασικό ρόλο: ο πρώτος αναφέρεται στον χρονικό δείκτη που συνοδεύει κάθε γεγονός από την πηγή παραγωγής, ενώ ο δεύτερος στον χρόνο κατά τον οποίο το γεγονός φτάνει και επεξεργάζεται από το σύστημα. Η διαφορά μεταξύ των δύο—λόγω καθυστερήσεων δικτύου, επεξεργασίας στον παραγωγό ή άλλων συνθηκών—προκαλεί αθροίσματα ή αποτελέσματα που δεν καλύπτουν πλήρως την πραγματική ροή γεγονότων. [4]

Για τον λόγο αυτό, υιοθετούνται τεχνικές όπως τα watermarks, που αντιπροσωπεύουν εκτιμήσεις για το πότε θεωρείται ότι συμπεριλαμβάνονται τα περισσότερα δεδομένα ενός παραθύρου. Τα watermarks επιτρέπουν στο σύστημα να προχωρήσει σε κλείσιμο του παραθύρου όταν έχει περάσει επαρκής χρόνος—ύστερα από τον οποίο θεωρείται ότι όσα δεδομένα δεν έχουν φτάσει θα θεωρηθούν καθυστερημένα. Όμως, ακόμη και τότε, απαιτείται η υποστήριξη για πολιτικές επιτρεπόμενης καθυστέρησης (allowed lateness) ή περίοδος χάριτος (grace period), ώστε να καθοριστεί πόσο χρόνο μετά το αρχικό κλείσιμο επιτρέπεται να γίνει επεξεργασία καθυστερημένων εγγραφών και να ενημερωθούν τα αποτελέσματα. [7]

Ο σχεδιασμός πολιτικών χειρισμού καθυστερημένων δεδομένων συνδέεται στενά με την ακρίβεια και τη συνέπεια των αποτελεσμάτων. Αν οι καθυστερημένες εγγραφές αγνοηθούν πλήρως, υπάρχει ο κίνδυνος τα στατιστικά και οι υπολογισμοί συγκέντρωσης (π.χ. αθροίσματα, μετρήσεις, μέσοι όροι) να αποκλίνουν από την πραγματική κατάσταση. Από την άλλη, εάν επιτραπεί ανοιχτό παράθυρο για μεγάλο χρονικό διάστημα, προκαλείται αύξηση απαιτήσεων σε μνήμη και πόρους αποθήκευσης, καθώς θα πρέπει να φυλάσσονται τα δεδομένα και να υποστηρίζεται η συνεχής ενημέρωση των ήδη δημοσιευμένων αποτελεσμάτων.

Επιπλέον, σε περιπτώσεις όπου τα δεδομένα φτάνουν σε υπερβολικά μεγάλο χρονικό διάστημα μετά την αρχική επεξεργασία (π.χ. λόγω δικτυακών αποκλεισμών ή εκτός λειτουργίας παραγωγού), πρέπει να εξεταστεί η πολιτική απόρριψης (drop policy), όπου οι εξαιρετικά

καθυστερημένες εγγραφές απορρίπτονται ή κατευθύνονται προς ειδικές παράπλευρες εξόδους (side outputs) για μελλοντική ανάλυση, χωρίς να επηρεάζουν τα κυρίως αποτελέσματα. [7]

Τέλος, η παρακολούθηση της διακύμανσης της καθυστέρησης (data skewness) και η δυναμική προσαρμογή των παραθύρων ή των watermarks μπορούν να βελτιώσουν την ισορροπία μεταξύ ακρίβειας και απόδοσης. Η αποτελεσματική διαχείριση των καθυστερημένων δεδομένων διασφαλίζει ότι οι μετρήσεις ροών παραμένουν αξιόπιστες, συντελώντας καθοριστικά στην επιτυχία πραγματικών εφαρμογών, όπως η ανίχνευση απατών, η ανάλυση δικτύων τηλεπικοινωνιών και η παρακολούθηση βιομηχανικών διεργασιών.

Κεφάλαιο 3

Αρχιτεκτονική και Χαρακτηριστικά του Apache Flink

3.1 Εισαγωγή

Το Apache Flink αποτελεί ένα από τα πλέον εξελιγμένα και ευέλικτα συστήματα επεξεργασίας ροών σε κατανεμημένα συστήματα. Σε αντίθεση με λύσεις που βασίζονται αποκλειστικά σε παρτίδες ή σε προσεγγίσεις μικρο–παρτίδων, το Flink σχεδιάστηκε εξαρχής για επεξεργασία δεδομένων σε πραγματικό χρόνο, παρέχοντας χαμηλή καθυστέρηση, υψηλή διαμέτρηση και πλούσιο μοντέλο διαχείρισης κατάστασης. Η αρχιτεκτονική του συνδυάζει έναν κατανεμημένο περιβάλλον εκτέλεσης (runtime) με πλούσιο API, επιτρέποντας τόσο έμπειρους προγραμματιστές όσο και μηχανικούς δεδομένων να υλοποιούν πολύπλοκες διαδρομές ανάλυσης ροών.

Σε επίπεδο υποδομής, το Flink στηρίζεται στο διαχωρισμό ρόλων μεταξύ JobManager και TaskManager, διασφαλίζοντας τόσο τον συντονισμό των εργασιών όσο και την αποκέντρωση της εκτέλεσης. Το execution graph που παράγεται κατά τη μετάφραση του προγράμματος περιγράφει με ακρίβεια τη ροή των δεδομένων, ενώ το δίκτυο επικοινωνίας υποστηρίζει υψηλό ρυθμό ανταλλαγής μηνυμάτων και μηχανισμούς ανάσχεσης ροής. Επιπλέον, το Flink διακρίνει ρητά το χρονικό σημείο δεδομένου από το χρονικό σημείο επεξεργασίας, παρέχοντας πλούσιες δυνατότητες χρονικών παραθύρων και συγχρονισμού γεγονότων.

Η παρούσα μελέτη εστιάζει στη διερεύνηση της αρχιτεκτονικής και των βασικών χαρακτηριστικών του Flink σε σχέση με τις πέντε διαστάσεις σύγκρισης που ορίστηκαν στο Κεφάλαιο 2: τη διαχείριση κατάστασης, τη σχέση καθυστέρησης–διαμέτρησης, την ανοχή σε σφάλματα, την αξιοποίηση πόρων και τον χειρισμό καθυστερημένων δεδομένων. Σε κάθε υποενότητα

θα παρουσιαστεί με αφαιρετικό, τεχνικό ύφος το πώς το Flink οργανώνει και υλοποιεί τους αντίστοιχους μηχανισμούς, δίνοντας έμφαση στην κατανεμημένη κλίμακα, την ευελιξία ρύθμισης και τη δυνατότητα προσαρμογής σε δυναμικές συνθήκες φόρτου.

3.2 Βασική Αρχιτεκτονική Υποδομή

Το Apache Flink υλοποιεί την εκτέλεση ροών δεδομένων σε κατανεμημένα περιβάλλοντα χρησιμοποιώντας ένα διεισδυτικό μοντέλο κυρίου–εργάτη (master–worker). Η υποδομή διαχωρίζεται σε δύο κύριες συνιστώσες—τον JobManager και τους TaskManagers—οι οποίες συνεργάζονται μέσω ενός καλοσχεδιασμένου execution graph και μιας υψηλής απόδοσης στοίβας δικτύου (network stack). Αυτή η αρχιτεκτονική επιτρέπει την κλιμάκωση σε δεκάδες ή και εκατοντάδες κόμβους, διατηρώντας ταυτόχρονα τις εγγυήσεις συνέπειας και ανοχής σε σφάλματα.

3.2.1 JobManager, TaskManager και Execution Graph

Ο JobManager λειτουργεί ως κεντρικός συντονιστής: λαμβάνει το πρόγραμμα που έχει συνταχθεί από τον χρήστη, το μεταφράζει σε ένα κατευθυνόμενο ακυκλικό γράφο εργασιών (execution graph) και αναλαμβάνει τον προγραμματισμό (scheduling) και την παρακολούθηση της εκτέλεσης. Το γράφημα αυτό περιγράφει τους κόμβους (tasks) και τις ακμές (data exchanges) της ροής, επιτρέποντας στο Flink να βρει βέλτιστες διαδρομές επικοινωνίας και να διαχειριστεί τη διάταξη (parallelism) κάθε σταδίου.[7]

Κάθε TaskManager αποτελεί έναν κόμβο εκτέλεσης που διαθέτει μία ή περισσότερες υποδοχές εργασιών (task slots). Σε αυτά εκτελούνται οι εργασίες (tasks) που καθορίζει ο JobManager. Οι TaskManagers, εκτός από την διαχείριση των υποδοχών εκτέλεσης, ώστε να επιτυγχάνεται ευέλικτη κατανομή των πόρων (CPU, μνήμη), διαχειρίζονται την αποθήκευση της κατάστασης στη μνήμη ή σε state backends, εκτελούν τοπικά στιγμιότυπα και αναφέρουν τακτικά την υγεία και την πρόοδο τους στον JobManager. Η συνεργασία των TaskManagers με τον JobManager εξασφαλίζει ότι σε περίπτωση αποτυχίας ενός κόμβου, οι υπόλοιποι μπορούν να ανακατανεμηθούν και να ανακτήσουν τη λειτουργία από το τελευταίο συνεπές στιγμιότυπο.

3.2.2 Επικοινωνία και Στοίβα Δικτύου

Η απόδοση ενός συστήματος επεξεργασίας ροών σε κατανεμημένα περιβάλλοντα εξαρτάται σε μεγάλο βαθμό από την αποτελεσματικότητα της ανταλλαγής δεδομένων. Το Flink υλοποιεί ένα ασύρματο μοντέλο ανταλλαγής δεδομένων, όπου οι παραγωγοί (producers) και οι καταναλωτές (consumers) συνδέονται με ενδιάμεσες μνήμες (buffers) σε κάθε ακμή του execution graph. Αυτές οι ενδιάμεσες μνήμες διαχειρίζονται μικρές "παρτίδες" (segments), επιτρέποντας την ταυτόχρονη ανάγνωση και εγγραφή χωρίς σκληρό συγχρονισμό.

Η ενσωματωμένη στοίβα δικτύου βασίζεται σε ένα υψηλής απόδοσης πρωτόκολλο TCP (ή, εναλλακτικά, RDMA όπου υποστηρίζεται), με διαχείριση ανάσχεσης ροής ώστε να αποφεύγεται η υπερφόρτωση των κόμβων. Κάθε TaskManager διατηρεί ένα σύνολο δικτυακών νημάτων (threads) που διακινούν τις μικρές παρτίδες σε πραγματικό χρόνο, εξισορροπώντας την πίεση εισόδου/εξόδου (I/O) και εξασφαλίζοντας ελάχιστο κόστος επεξεργασίας. Ο μηχανισμός ανάσχεσης ροής ενεργοποιεί διακοπές ανάγνωσης όταν οι ενδιάμεσες μνήμες πλησιάζουν στο όριό τους, εμποδίζοντας την αύξηση της καθυστέρησης και των απαιτήσεων μνήμης. [5, 7]

3.3 Προγραμματιστικό Πρότυπο

Το Apache Flink παρέχει ένα ευέλικτο και ισχυρό προγραμματιστικό πρότυπο για την κατασκευή εφαρμογών επεξεργασίας ροών σε κατανεμημένα περιβάλλοντα. Η καρδιά αυτού του προτύπου είναι το DataStream API, το οποίο επιτρέπει στον προγραμματιστή να περιγράψει τη ροή των δεδομένων και τους μετασχηματισμούς της με αφαιρετικό τρόπο, ενώ το Flink αναλαμβάνει τη χαρτογράφηση σε κατανεμημένους κόμβους, την εκτέλεση και τη διαχείριση σφαλμάτων.

3.3.1 DataStream API και Μετασχηματισμοί (Transformations)

Το DataStream API εκφράζει την επεξεργασία ροών ως αλυσίδα μετασχηματισμών επί ενός DataStream. Κάθε μετασχηματισμός (π.χ. map, filter, keyBy, window, join) λειτουργεί με αφαιρετικό τρόπο, επιτρέποντας την εκφώνηση πολύπλοκων λογικών κανόνων χωρίς να απαιτείται χειροκίνητη φροντίδα για την κατανομή των δεδομένων και τον συντονισμό των κόμβων. Το Flink δημιουργεί αυτόματα το αντίστοιχο execution graph, διαχειρίζεται τις συγκατοικήσεις (operator chains) για ελαχιστοποίηση κόστους επεξεργασίας και ενσωματώνει σημεία ελέγχου (checkpoints) στα κατάλληλα σημεία της ροής για συνεπή αποκατάσταση.[5, 7]

3.3.2 Σύγκριση Χρονικού Σημείου Δεδομένου-Επεξεργασίας

Μία κρίσιμη διάκριση στο Flink είναι η υποστήριξη δύο διαφορετικών χρονικών εννοιών:

- **Χρονικό Σημείο Δεδομένου** (Event Time): Ο χρόνος που αποδίδεται σε κάθε γεγονός από την πηγή του. Επιτρέπει ακριβή ανάλυση βάσει της αληθινής χρονικής στιγμής δημιουργίας, ανεξάρτητα από δικτυακές ή επεξεργαστικές καθυστερήσεις.
- **Χρονικό Σημείο Επεξεργασίας** (Processing Time): Ο χρόνος κατά τον οποίο το γεγονός φτάνει και επεξεργάζεται από τον Flink. Παρέχει απλούστερη υλοποίηση με ελάχιστο κόστος επεξεργασίας, αλλά χωρίς εγγυήσεις σχετικά με την ακρίβεια σε περίπτωση χρονικών αποκλίσεων.

Η υποστήριξη χρονικού σημείου δεδομένου επιτυγχάνεται μέσω της χρήσης watermarks και χρονικών παραθύρων, καθιστώντας δυνατές παραθυρικές συναθροίσεις και συσχετίσεις που αντιστοιχούν στην πραγματική ροή των γεγονότων. Αντίθετα, η χρήση χρονικού σημείου επεξεργασίας μπορεί να είναι επαρκής σε σενάρια όπου οι καθυστερήσεις δικτύου είναι σταθερές ή μη κρίσιμες. Ο προγραμματιστής μπορεί να επιλέξει την κατάλληλη χρονική διάσταση ανάλογα με τις απαιτήσεις ακρίβειας, καθυστέρησης και απλότητας της εφαρμογής.

3.4 Διαχείριση κατάστασης

Το Apache Flink προσφέρει μια ολοκληρωμένη και ευέλικτη προσέγγιση στη διαχείριση κατάστασης, η οποία είναι κρίσιμη για την επεξεργασία ροών με διατήρηση κατάστασης. Μέσω της αποδοτικής διαχείρισης τόσο της κατάστασης με κλειδί (keyed state) όσο και της κατάστασης του τελεστή (operator state), το Flink εξασφαλίζει επεκτασιμότητα, ανοχή σε σφάλματα και συνέπεια. Αυτός ο σχεδιασμός επιτρέπει στο Flink να υποστηρίζει ένα ευρύ φάσμα σεναρίων επεξεργασίας ροών, από απλές συναθροίσεις έως πολύπλοκες εφαρμογές βασισμένες σε γεγονότα. [5, 7]

Κατάσταση με Κλειδί και Κατάσταση Τελεστή

Το Flink διακρίνει δύο βασικούς τύπους κατάστασης:

Η κατάσταση με κλειδί συσχετίζεται με ροές που έχουν οριστεί με κλειδί και περιορίζεται σε ατομικά κλειδιά, επιτρέποντας τη διαχείριση της κατάστασης ξεχωριστά για κάθε κλειδί σε διαφορετικά τμήματα. Το Flink προσφέρει διάφορες λειτουργίες για τη διαχείριση της κατάστασης με κλειδί: το ValueState, που διατηρεί μία μόνο τιμή ανά κλειδί· το ListState,

που αποθηκεύει μια λίστα στοιχείων για κάθε κλειδί· το ReducingState, που εφαρμόζει μια συνάρτηση σύνοψης για τη διατήρηση μιας συγκεντρωτικής κατάστασης· και το MapState, που αποθηκεύει αντιστοιχίσεις κλειδιών-τιμών για κάθε κλειδί. Από την άλλη πλευρά, η κατάσταση τελεστή περιορίζεται στον ίδιο τον τελεστή και είναι κοινή μεταξύ των στιγμιοτύπων του. Αυτή η μορφή κατάστασης χρησιμοποιείται συνήθως σε σενάρια όπως η διατήρηση των δεικτών θέσης (offsets) σε έναν Kafka consumer. Το Flink διασφαλίζει ότι η κατάσταση τελεστή μπορεί να ανακατανεμηθεί σωστά όταν αυξάνεται η παραλληλία μιας εργασίας.

State Backends και Αποθήκευση (Storage) Κατάστασης

Το Flink προσφέρει αξιόπιστες επιλογές backend για τη διαχείριση της κατάστασης. Το heap-based state backend αποθηκεύει την κατάσταση απευθείας στη μνήμη της JVM, προσφέροντας ταχύτατη πρόσβαση, αλλά περιορίζεται από το διαθέσιμο μέγεθος μνήμης. Αντίθετα, το RocksDB state backend βασίζεται σε δίσκο και αποτελεί ένα ενσωματωμένο key-value store, βελτιστοποιημένο για τη διαχείριση μεγάλων ποσοτήτων κατάστασης και την υποστήριξη επαυξητικών στιγμιοτύπων (incremental snapshots). Η διαχειριζόμενη κατάσταση αποθηκεύεται και αποκαθίσταται αυτόματα από το περιβάλλον εκτέλεσης του Flink, εξασφαλίζοντας συνέπεια ακόμη και σε περίπτωση σφαλμάτων. [7]

Πρόσβαση και Διαχείριση Κατάστασης (State Access and Management)

Η κατάσταση στο Flink είναι προσβάσιμη προγραμματιστικά μέσω του RuntimeContext στους τελεστές . Οι προγραμματιστές ορίζουν τον τύπο και το πεδίο της κατάστασης μέσω των StateDescriptors και την προσπελαύνουν μέσω μεθόδων που παρέχονται από το πλαίσιο (context). Τα βασικά βήματα περιλαμβάνουν τη δημιουργία της απαιτούμενης κατάστασης χρησιμοποιώντας περιγραφείς (descriptors) και την πρόσβαση σε αυτήν για ανάγνωση ή ενημέρωση κατά την εκτέλεση εργασιών επεξεργασίας ροών.

Εργασίες Συντήρησης και Έλεγχος Ταυτόχρονης Πρόσβασης (Maintenance Tasks and Concurrency Control)

Το Flink διαχειρίζεται αρχεία κατάστασης και στιγμιότυπα ασύγχρονα, ώστε να βελτιστοποιεί την απόδοση κατά την επεξεργασία ροών. Η ασύγχρονη συντήρηση (asynchronous maintenance) πραγματοποιείται μέσω εργασιών παρασκηνίου, οι οποίες εκτελούν λειτουργίες όπως συμπίεση (compaction), καθαρισμό (purging) και συγχώνευση (merging) των αρχείων κατάστασης, χωρίς να επηρεάζεται η κύρια ροή επεξεργασίας δεδομένων. Όσον αφορά τον έλεγχο ταυτόχρονης πρόσβασης (concurrency control), κάθε εργασία διαχειρίζεται την πρόσβαση στη δική της τοπική κατάσταση ανεξάρτητα, ενώ το Flink εξασφαλίζει κατανεμημένη συνέπεια μέσω ενός αυστηρά συντονισμένου μοντέλου εκτέλεσης (execution model).

3.5 Καθυστέρηση – Διαμέτρηση

Το Apache Flink έχει σχεδιαστεί εξαρχής με στόχο την επίτευξη χαμηλής καθυστέρησης και υψηλής διαμέτρησης σε σενάρια συνεχούς και κατανεμημένης επεξεργασίας ροών. Ο τρόπος με τον οποίο το Flink διαχειρίζεται αυτές τις δύο κρίσιμες μετρικές απόδοσης είναι άρρηκτα συνδεδεμένος με την αρχιτεκτονική του και τους αλγορίθμους εκτέλεσης που εφαρμόζει.

Σε ό,τι αφορά την καθυστέρηση, το Flink υποστηρίζει true streaming μοντέλο εκτέλεσης, κατά το οποίο τα δεδομένα επεξεργάζονται στοιχειοσειρά-προς-στοιχειοσειρά (record-by-record) χωρίς να απαιτείται η συγκέντρωση ή προσωρινή αποθήκευση μεγάλου όγκου γεγονότων. Αυτό σημαίνει ότι τα δεδομένα προωθούνται και επεξεργάζονται αμέσως μόλις παραληφθούν, μειώνοντας την από άκρο σε άκρο καθυστέρηση. Η καθυστέρηση μπορεί να διατηρηθεί σε εξαιρετικά χαμηλά επίπεδα, ιδιαίτερα όταν συνδυάζεται με το μοντέλο χρονικού σημείου δεδομένου και την επεξεργασία βασισμένη σε τελεστές χαμηλής καθυστέρησης (low-latency operators), όπως οι συναρτήσεις επεξεργασίας (process functions) και οι προσαρμοσμένοι χειριστές (custom operators). [7]

Η επίτευξη χαμηλής καθυστέρησης ενισχύεται επίσης από τη χρήση του ασύρματου μοντέλου ανταλλαγής δεδομένων, όπου τα αποτελέσματα μιας διεργασίας μπορούν να αποσταλούν απευθείας στην επόμενη χωρίς αναμονή για την ολοκλήρωση της συνολικής παρτίδας. Η στοίβα δικτύου του Flink είναι βελτιστοποιημένη ώστε να ελαχιστοποιεί τους χρόνους μετάδοσης και να υποστηρίζει υψηλό επίπεδο παραλληλισμού χωρίς να εισάγει σημαντικά σημεία συμφόρησης.

Στην περίπτωση της διαμέτρησης, το Flink βασίζεται σε μηχανισμούς παραλληλισμού και κατανεμημένης εκτέλεσης για να επιτύχει υψηλό ρυθμό επεξεργασίας. Κάθε τελεστής μπορεί να εκτελείται σε πολλαπλές υποδοχές εργασιών ταυτόχρονα, επιτρέποντας τη δυναμική κλιμάκωση των εφαρμογών. Επιπλέον, το Flink υποστηρίζει τις συγκατοικήσεις , έναν μηχανισμό συγχώνευσης πολλών απλών τελεστών σε μια ενιαία διεργασία, μειώνοντας έτσι το κόστος επικοινωνίας και αυξάνοντας τη συνολική απόδοση. [7]

Η διαμέτρηση επηρεάζεται επίσης από την επιλογή του state backend, τη διαμόρφωση των ενδιάμεσων μνημών και τη στρατηγική των στιγμιοτύπων. Η χρήση ασύγχρονων στιγμιοτύπων σε συνδυασμό με τη δυνατότητα για σταδιακή αποθήκευση κατάστασης (incremental state storage) επιτρέπει στο σύστημα να συνεχίζει την επεξεργασία ροών με ελάχιστη διακοπή, ακόμη και κατά τη διάρκεια κρίσιμων λειτουργιών όπως η αποθήκευση κατάστασης ή η ανάκτηση από σφάλματα.

Τέλος, το Apache Flink παρέχει ενσωματωμένο σύστημα μετρικών (metrics system) που επιτρέπει την παρακολούθηση της καθυστέρησης και της διαμέτρησης σε πραγματικό χρόνο, τόσο σε επίπεδο εργασίας όσο και σε επίπεδο τελεστή. Οι πληροφορίες αυτές είναι κρίσιμες για τη βελτιστοποίηση των εφαρμογών, καθώς δίνουν τη δυνατότητα στον χρήστη να εντοπίσει πιθανά σημεία συμφόρησης και να προσαρμόσει αναλόγως τις παραμέτρους εκτέλεσης.

Συνοψίζοντας, το Flink υλοποιεί μια αρχιτεκτονική που συνδυάζει επεξεργασία σε πραγματικό χρόνο, υψηλή παραλληλία και προσαρμοστικότητα, επιτυγχάνοντας έναν εξαιρετικά ευνοϊκό συμβιβασμό μεταξύ καθυστέρησης και διαμέτρησης, κατάλληλο για απαιτητικά σενάρια κατανεμημένης επεξεργασίας ροών.

3.6 Ανοχή σε Σφάλματα

Η αξιοπιστία αποτελεί κρίσιμο χαρακτηριστικό για κάθε σύστημα κατανεμημένης επεξεργασίας ροών, καθώς η φύση των δεδομένων σε πραγματικό χρόνο και η αστάθεια των υποκείμενων υποδομών καθιστούν αναπόφευκτα τα σφάλματα. Το Apache Flink προσφέρει ένα ιδιαίτερα ανθεκτικό και εξελιγμένο μοντέλο ανοχής σε σφάλματα, το οποίο στηρίζεται στην ιδιότητά του να παρέχει εγγυήσεις για Ακριβώς-Μία-Επεξεργασία των δεδομένων, ακόμη και σε περιβάλλοντα υψηλής παραλληλίας και μεγάλης κλίμακας.

Ο θεμέλιος λίθος της ανοχής σε σφάλματα στο Flink είναι ο μηχανισμός των στιγμιοτύπων. Κατά την εκτέλεση μίας εργασίας επεξεργασίας ροών, το σύστημα δημιουργεί περιοδικά στιγμιότυπα της κατάστασης των τελεστών με διατήρηση κατάστασης (stateful operators) με χρήση του πρωτοκόλλου Asynchronous Barrier Snapshotting. Το πρωτόκολλο αυτό εισάγει φραγμούς (barriers) μέσα στις ροές των δεδομένων, επιτρέποντας την ασύγχρονη λήψη στιγμιοτύπων χωρίς να διακόπτεται η επεξεργασία. [5, 7]

Η διαδικασία αυτή υποστηρίζεται από τα state backends, τα οποία είναι υπεύθυνα για την αποθήκευση της κατάστασης των εφαρμογών. Οι πιο διαδεδομένες επιλογές είναι το RocksDB State Backend, που αποθηκεύει την κατάσταση σε τοπικούς δίσκους, και το HashMap State Backend, που λειτουργεί αποκλειστικά στη μνήμη. Και οι δύο επιλογές υποστηρίζουν σταδιακή αποθήκευση κατάστασης, μειώνοντας τον όγκο δεδομένων που αποθηκεύονται και βελτιώνοντας τον χρόνο αποκατάστασης.

Σε περίπτωση σφάλματος — είτε σε επίπεδο υποεργασίας, κόμβου, είτε ακόμη και σε ολόκληρη εργασία — το Flink μπορεί να επανεκκινήσει την εκτέλεση από το πιο πρόσφατο

επιτυχημένο στιγμιότυπο, αποκαθιστώντας την κατάσταση των τελεστών και επαναλαμβάνοντας την κατανάλωση των δεδομένων από τις πηγές (sources), με τρόπο που διασφαλίζει την ακρίβεια της επεξεργασίας.

Ένα ιδιαίτερο πλεονέκτημα του Flink είναι ότι ο χειρισμός των στιγμιοτύπων και της επαναφοράς τους γίνεται με τρόπο διαφανή για τον χρήστη. Ο προγραμματιστής δεν χρειάζεται να ενσωματώσει ρητούς μηχανισμούς ανάκτησης ή αποθήκευσης κατάστασης στον κώδικά του, καθώς το σύστημα φροντίζει αυτόματα για την αξιοπιστία της διαδρομής των δεδομένων (data pipeline).

Το Flink υποστηρίζει επίσης μηχανισμούς εξωτερικευμένων στιγμιοτύπων (externalized checkpoints), που διατηρούν τα στιγμιότυπα κατά την ακύρωση ή αποτυχία μιας εργασίας, διευκολύνοντας την αποκατάσταση ακόμα και μετά από παρεμβάσεις χρήστη ή συντήρηση συστήματος. Επιπλέον, το σύστημα σημείων αποθήκευσης κατάστασης του Flink επιτρέπει την αποθήκευση κατάστασης κατ' απαίτηση, με στόχο την ασφαλή αναβάθμιση ή μετεγκατάσταση εφαρμογών.

Συνοψίζοντας, η αρχιτεκτονική ανοχής σε σφάλματα του Apache Flink προσφέρει ένα ισχυρό και αποτελεσματικό υπόβαθρο για την αξιόπιστη επεξεργασία ροών δεδομένων σε κατανεμημένα περιβάλλοντα. Η δυνατότητα αυτόματης και ακριβούς ανάκτησης κατάστασης διασφαλίζει την ακεραιότητα των αποτελεσμάτων και καθιστά το Flink κατάλληλο για κρίσιμες επιχειρησιακές εφαρμογές όπου τα σφάλματα δεν είναι αποδεκτά.

3.7 Αξιοποίηση Πόρων – Κόστος Επεξεργασίας

Η αξιοποίηση πόρων και το κόστος επεξεργασίας στο Apache Flink καθορίζονται από τη σχεδίαση του περιβάλλοντος εκτέλεσης και τους μηχανισμούς που υλοποιεί για την αποδοτική κατανομή εργασιών και τη διαχείριση κατάστασης. Σε κάθε TaskManager, η μνήμη διαιρείται σε περιοχές για τις ενδιάμεσες μνήμες δικτύου (network buffers), τη διατήρηση κατάστασης και την εκτέλεση των τελεστών. Η ορθή παραμετροποίηση των ορίων μνήμης εξασφαλίζει ότι δεν δημιουργούνται σημεία συμφόρησης ούτε υπερφόρτωση του JVM.

Ο κατανεμημένος παραλληλισμός επιτυγχάνεται μέσω πολλαπλών υποδοχών εργασιών και δυναμικής κλιμάκωσης των ρυθμίσεων παραλληλισμού. Κατά την κατανομή των υποεργασιών, το Flink βελτιστοποιεί τη συγκατοίκηση, συγχωνεύοντας διαδοχικούς τελεστές σε ένα ενιαίο νήμα εκτέλεσης, ώστε να μειωθεί το κόστος μεταφοράς δεδομένων και να αυξηθεί η διαμέτρηση χωρίς επιπλέον κατανάλωση CPU.

Η διαχείριση της κατάστασης μέσω state backends (π.χ. RocksDB ή in-memory HashMap) εισάγει πρόσθετη επιβάρυνση: τα ασύγχρονα στιγμιότυπα και η σταδιακή αποθήκευση κατάστασης καθιστούν δυνατή τη συνεχή επεξεργασία χωρίς παύση, ωστόσο απαιτούν πόρους I/O και δικτύου για τη μεταφορά και αποθήκευση των αλλαγών κατάστασης. Η επιλογή ανάμεσα σε memory-heavy και disk-based backends επηρεάζει τόσο τη μνήμη που δεσμεύεται όσο και την καθυστέρηση των διαδικασιών δημιουργίας στιγμιοτύπων. [7]

Στο επίπεδο δικτύου, οι ενδιάμεσες μνήμες δικτύου και η εσωτερική στοίβα δικτύου του Flink διαχειρίζονται τη ροή μηνυμάτων μεταξύ των TaskManagers. Ορισμένες παράμετροι—όπως το μέγεθος και ο αριθμός των ενδιάμεσων μνημών —επηρεάζουν άμεσα τη χρήση μνήμης και την καθυστέρηση μετάδοσης (transmission latency), ενώ οι μηχανισμοί ανάσχεσης ροής αποτρέπουν την κατάρρευση των υποσυστημάτων όταν η εισροή δεδομένων ξεπερνά τις δυνατότητες επεξεργασίας.

Τέλος, το Flink παρέχει ένα ολοκληρωμένο σύστημα μετρικών , μέσω του οποίου παρακολουθούνται σε πραγματικό χρόνο οι δείκτες χρήσης CPU, μνήμης, δίσκου και δικτύου, καθώς και οι αποδόσεις των τελεστών. Οι πληροφορίες αυτές επιτρέπουν την προσαρμογή παραμέτρων, όπως η συχνότητα στιγμιοτύπων ή η αύξηση του παραλληλισμού, με στόχο την ελαχιστοποίηση του κόστους και τη διατήρηση της απόδοσης σε επίπεδα που ικανοποιούν τα απαιτητικά συμφωνημένα επιπέδα υπηρεσιών.

3.8 Χειρισμός Καθυστερημένων Δεδομένων

Στο Apache Flink, ο χειρισμός καθυστερημένων δεδομένων βασίζεται σε ένα εκτενές μοντέλο χρονικών παραθύρων και στη χρήση watermarks για συγχρονισμό βάσει χρονικού σημείου δεδομένου. Καθώς τα γεγονότα φτάνουν με καθυστέρηση —λόγω δικτυακών καθυστερήσεων ή επεξεργασίας στους παραγωγούς— το Flink επιτρέπει την προσαρμοσμένη διαχείριση επιπλέον εγγραφών χωρίς να θυσιάζεται η συνέπεια των αποτελεσμάτων.

Η βασική τεχνική είναι η εισαγωγή watermarks, τα οποία αντιπροσωπεύουν μια εκτίμηση του χρόνου που έχει «διανυθεί» στο χρονικό σημείο δεδομένου και σηματοδοτούν πότε θεωρείται ότι έχει παραληφθεί το μεγαλύτερο μέρος των δεδομένων για ένα παράθυρο. Σε συνδυασμό με αυτά, οι πολιτικές επιτρεπόμενης καθυστέρησης ορίζουν το χρονικό περιθώριο —πέραν της αρχικής λήξης του παραθύρου— εντός του οποίου το Flink δέχεται ακόμα καθυστερημένα γεγονότα και ενημερώνει τα αποτελέσματα. [7, 11]

Παράλληλα, το Flink υποστηρίζει την αποστολή περισσοτέρων δεδομένων σε παράπλευρες εξόδους, όταν η καθυστέρηση υπερβαίνει το όριο επιτρεπόμενης καθυστέρησης. Αυτές οι

πλευρικές ροές επιτρέπουν τη μεμονωμένη ανάλυση ή επεξεργασία εξαιρετικά καθυστερημένων εγγραφών, χωρίς να επηρεάζεται η κύρια ροή αποτελεσμάτων.

Το σύστημα καθαρισμού παραθύρων (window cleanup) διαχειρίζεται τη μνήμη αποθηκεύοντας τις μετρήσεις έως ότου περάσει η επιτρεπόμενη καθυστέρηση και στη συνέχεια απελευθερώνει τους πόρους, διασφαλίζοντας ότι η πλατφόρμα παραμένει αποδοτική σε μεγάλες κλίμακες.

Τέλος, το Flink παρέχει εργαλεία παρακολούθησης των watermarks και των καθυστερημένων δεδομένων μέσω του ενσωματωμένου συστήματος μετρικών, δίνοντας τη δυνατότητα στους διαχειριστές να παρακολουθούν τον ρυθμό καθυστερήσεων και να προσαρμόζουν δυναμικά τις πολιτικές επιτρεπόμενης καθυστέρησης ή τα όρια των παραθύρων, ώστε να διατηρείται η ακρίβεια και η απόδοση της εργασίας επεξεργασίας ροών.

Κεφάλαιο 4

Αρχιτεκτονική και Χαρακτηριστικά του Apache Spark

4.1 Εισαγωγή

Το Apache Spark Structured Streaming αποτελεί εκτεταμένη επέκταση του Spark για την επεξεργασία ροών δεδομένων με το ίδιο προγραμματιστικό πρότυπο που χρησιμοποιείται για παρτίδες (batches). Αντί της παραδοσιακής λογικής των Resilient Distributed Datasets (RDDs), το Structured Streaming στηρίζεται σε DataFrames και Datasets, παρέχοντας διαύλους δεδομένων με εγγυήσεις συνέπειας και ανθεκτικότητας. Η βασική ιδέα είναι να αντιμετωπίζει τη ροή ως αέναο πίνακα ενημερώσεων, όπου κάθε εκτέλεση ερωτήματος αντιστοιχεί σε μία μικρο-παρτίδα (micro-batch) ή—σε προηγμένες εκδόσεις—σε συνεχές μοντέλο (continuous processing). [8]

Η προσέγγιση αυτή επιτρέπει την εγγενή υποστήριξη ιδιοτήτων όπως exactly-once semantics για την επεξεργασία με διατήρηση κατάστασης, χωρίς να αλλάζει ο τρόπος συγγραφής των ερωτημάτων σε SQL ή DataFrame API. Επιπλέον, το Spark παρέχει ενσωματωμένους συνδετήρες (connectors) για δημοφιλείς πηγές και αποδέκτες δεδομένων (π.χ. Kafka, Kinesis, HDFS), ενώ αξιοποιεί την κατανεμημένη μηχανή εκτέλεσης (execution engine) του για την κλιμάκωση και την ανοχή σε σφάλματα. [6]

Στην παρούσα ενότητα θα περιγραφεί η αρχιτεκτονική του Spark Structured Streaming, εστιάζοντας στον τρόπο με τον οποίο οργανώνει τον πόρο των Driver και Executors, τη δημιουργία του DAG, καθώς και τις υποδομές shuffle και δικτύου. Η ανάλυση θα θέσει τις βάσεις για την κατανόηση των μηχανισμών διαχείρισης κατάστασης, καθυστέρησης-διαμέτρησης, ανοχής σε σφάλματα, αξιοποίησης πόρων και χειρισμού καθυστερημένων δεδομένων, οι οποίοι θα αναπτυχθούν στα επόμενα υποκεφάλαια. [3]

4.2 Βασική Αρχιτεκτονική Υποδομή

Η αρχιτεκτονική του Apache Spark Structured Streaming βασίζεται στον ίδιο κατανεμημένο πυρήνα εκτέλεσης του Spark, επεκτείνοντας τον ώστε να υποστηρίζει επεξεργασία ροών σε πραγματικό χρόνο. Το μοντέλο βασίζεται στην έννοια των μικρο-παρτίδων ή σε συνεχή ροή, διατηρώντας τα πλεονεκτήματα του Spark SQL και του βελτιστοποιημένου κατανεμημένου σχεδιασμού. Η ροή των δεδομένων διαχειρίζεται ως μια άπειρη ακολουθία εγγραφών, η οποία κατανέμεται, φιλτράρεται, συσχετίζεται και αποθηκεύεται με συνέπεια και ανοχή σε σφάλματα.

Η βασική υποδομή του Spark Structured Streaming περιλαμβάνει τρία κύρια δομικά στοιχεία: τον Driver, τους Executors και τον μηχανισμό ανακατανομής (shuffle), τα οποία συνεργάζονται για την εκτέλεση εργασιών σε ένα κατανεμημένο περιβάλλον. Η παρακάτω ενότητα αναλύει αυτά τα στοιχεία.

4.2.1 Driver, Executors και DAG

Στο Spark Structured Streaming, η διαχείριση και ο συντονισμός της επεξεργασίας ροών ανατίθενται στον Driver, ο οποίος δημιουργεί και διαχειρίζεται το καθολικό Directed Acyclic Graph (DAG) των εργασιών. Το λογικό ερώτημα μετασχηματίζεται αρχικά σε έναν πλέγμα εξαρτήσεων DataFrame, το οποίο στη συνέχεια εναλλάσσεται σε DAG εκτέλεσης. Κάθε κόμβος στον DAG αντιστοιχεί σε μια μετατροπή δεδομένων ή σε ένα στάδιο μικρο-παρτίδας (micro-batch stage), ενώ οι ακμές αναπαριστούν την ροή των διανεμημένων τμημάτων (partitions).

Οι Executors αναλαμβάνουν την εκτέλεση των επιμέρους εργασιών που προκύπτουν από τον DAG. Κάθε Executor διαθέτει ένα σύνολο πόρων—νήματα επεξεργασίας, μνήμη και ενδιάμεσες μνήμες δικτύου—ώστε να υλοποιεί παράλληλα υποεργασίες . Ο Driver κατανέμει τις εργασίες βάσει των ρυθμίσεων παραλληλισμού και επιτηρεί την ολοκλήρωση ή ενδεχόμενη επανεκκίνησή τους σε περιπτώσεις σφαλμάτων. [3]

4.2.2 Shuffle Service και Στοίβα Δικτύου

Κατά τη διάρκεια της επεξεργασίας μικρο-παρτίδων, τα δεδομένα ενδέχεται να απαιτούν ανακατανομή μεταξύ των Executors, διαδικασία γνωστή ως shuffle. Το Shuffle Service του Spark παρέχει έναν ανεξάρτητο μηχανισμό αποθήκευσης και ανάκτησης μετρήσεων κατατμημένων δεδομένων (partitioned data), διασφαλίζοντας ότι τα αποτελέσματα των προηγούμενων σταδίων είναι διαθέσιμα ακόμη και σε περίπτωση επανεκκίνησης κόμβων. Η χρήση ενός εξωτερικού shuffle service αφαιρεί την αποθήκευση δεδομένων από τη μνήμη των Executors, βελτιώνοντας την αξιοπιστία και την ανθεκτικότητα. [8]

Η εσωτερική στοίβα δικτύου του Spark διαχειρίζεται τη σειριοποίηση, τη συμπίεση και τη μεταφορά των τμημάτων μέσω TCP συνδέσεων. Κάθε Executor διατηρεί ενδιάμεσες μνήμες εισόδου/εξόδου, που βελτιστοποιούν τη ροή μηνυμάτων και μειώνουν δραστικά τον χρόνο επαναφοράς από ανάσχεση ροής. Η ρύθμιση των παραμέτρων ενδιάμεσης μνήμης και η κατάτμηση των δεδομένων καθορίζουν άμεσα την επίδοση της υποδομής δικτύου και επιτρέπουν στο σύστημα να ανταποκρίνεται σε μεγάλους όγκους δεδομένων με χαμηλή καθυστέρηση.

4.3 Προγραμματιστικό Πρότυπο

Το Spark Structured Streaming παρέχει ένα ενιαίο, δηλωτικό πλαίσιο για την επεξεργασία ροών σε κατανεμημένες υποδομές. Χρησιμοποιώντας το ίδιο DataFrame API με το Spark SQL, απλοποιεί την ανάπτυξη εφαρμογών επεξεργασίας ροών, ενώ εξασφαλίζει επεκτασιμότητα, ανοχή σε σφάλματα και διαχείριση κατάστασης χωρίς πρόσθετη πολυπλοκότητα.

4.3.1 DataFrame API και Structured Streaming Queries

Στο Spark Structured Streaming, κάθε ροή δεδομένων αναπαρίσταται ως Streaming DataFrame, δηλαδή μια διαρκής, κατανεμημένη συλλογή στήλης-δομημένων εγγραφών με προκαθορισμένο σχήμα (schema). Οι χρήστες συνθέτουν ερωτήματα ροής με τις ίδιες δηλωτικές λειτουργίες (select, filter, groupBy, join, κ.ά.) που ισχύουν σε παρτίδες, χωρίς να διαχειρίζονται χειροκίνητα τη χαρτογράφηση τμημάτων ή την αποθήκευση κατάστασης. [6]

Κατά την υποβολή ενός ερωτήματος, ο βελτιστοποιητής Catalyst μετασχηματίζει τη σειρά των λογικών πράξεων σε ένα κατανεμημένο φυσικό σχεδιασμό (physical plan), τον οποίο ο Driver αναπτύσσει σε Executors. Η επεξεργασία ροής, αν και εμφανίζεται ως αέναη,

αντιμετωπίζεται εσωτερικά ως ακολουθία από εργασίες μικρο-παρτίδων, με τις εγγυήσεις του Spark SQL (π.χ. schema enforcement, predicate pushdown) διατηρούμενες πλήρως.

4.3.2 Μοντέλο Μικρο-παρτίδων - Συνεχής Επεξεργασία

Το προεπιλεγμένο μοντέλο λειτουργίας του Structured Streaming είναι το μοντέλο μικρο-παρτίδων, όπου η ροή χωρίζεται σε συνεχείς μικρο-παρτίδες με βάση έναν καθορισμένο χρονικό διάστημα πυροδότησης (trigger interval). Κάθε μικρο-παρτίδα αντιστοιχεί σε εκτέλεση εργασίας Spark SQL, ανανεώνοντας μόνο το τμήμα της κατάστασης που επεξεργάστηκε πρόσφατα. Αυτή η προσέγγιση εξασφαλίζει πλήρη ακεραιότητα (exactly-once semantics) και ευελιξία στην υποστήριξη σύνθετων μετασχηματισμών. [8]

Ως εναλλακτική, το Spark παρέχει πειραματικό μοντέλο συνεχούς επεξεργασίας, όπου η διάσπαση σε παρτίδες παύει να υφίσταται και η επεξεργασία γίνεται εγγραφή-προς-εγγραφή (record-by-record). Αυτό μειώνει δραστικά την καθυστέρηση, υπερβαίνοντας τον περιορισμό του ελάχιστου χρονικού διαστήματος πυροδότησης, αλλά επιτρέπει περιορισμένο σετ μετασχηματισμών και απαιτεί προσεκτικότερη διαχείριση πόρων για να διατηρηθεί η συνέπεια και η ανθεκτικότητα του DAG.

4.4 Διαχείριση Κατάστασης

Στο Spark Structured Streaming, η κατάσταση αποτελεί κρίσιμο στοιχείο για την υποστήριξη λειτουργιών όπως συναθροίσεις βάσει χρονικών παραθύρων (windowed aggregations), συνενώσεις ροών (stream–stream joins) και αφαίρεση διπλοτύπων (deduplication). Αντί να απαιτείται χειροκίνητη διαχείριση, το σύστημα ενσωματώνει μηχανισμούς για συνεπή, ανεκτική σε σφάλματα εξέλιξη της κατάστασης σε κάθε μικρο-παρτίδα. [6, 8]

Κατά την εκτέλεση ερωτημάτων με διατήρηση κατάστασης, το Spark δημιουργεί εσωτερικά δομές κατάστασης σε κάθε Executor, χωρισμένες ανά τμήματα κλειδιών (keyed state). Οι ενημερώσεις καταγράφονται τοπικά μέχρι την ολοκλήρωση κάθε μικρο-παρτίδας, οπότε και αποθηκεύονται σε αξιόπιστο αποθηκευτικό μέσο (HDFS ή cloud storage). Η αποκατάσταση μετά από σφάλμα στηρίζεται σε αυτά τα στιγμιότυπα, διασφαλίζοντας exactly-once semantics για την κατάσταση.

Για μεγάλης κλίμακας καταστάσεις, το Spark αξιοποιεί εσωτερικό αποθηκευτικό χώρο κατάστασης το οποίο υποστηρίζει σταδιακές ενημερώσεις (incremental updates) και αποδοτική πρόσβαση σε δεδομένα κλειδιών. Η διεργασία στιγμιοτύπων πραγματοποιείται ασύγχρονα:

ενώ τρέχει η επεξεργασία νέων μικρο-παρτίδων, οι μεταβολές της τρέχουσας κατάστασης συμπιέζονται και γράφονται στο υπόβαθρο, μειώνοντας την επιβάρυνση στην καθυστέρηση.

Παράλληλα, το Spark παρέχει δυνατότητες περιορισμού της μνήμης που καταλαμβάνει η κατάσταση, μέσω πολιτικών κατάστασηςTTL (time-to-live) και "state pruning", ώστε οι παλαιές εγγραφές να απομακρύνονται αυτόματα. Αυτές οι ρυθμίσεις είναι απαραίτητες σε συνεχείς ροές με ανεξέλεγκτο όγκο δεδομένων, εξασφαλίζοντας τη σταθερότητα του συμπλέγματος (cluster).

Τέλος, η παρακολούθηση της κατάστασης γίνεται μέσω του ενσωματωμένου συστήματος μετρικών, που εκθέτει μετρικές όπως μέγεθος κατάστασης ανά τμήμα, χρόνοι στιγμιοτύπων και ρυθμοί συλλογής σκουπιδιών (Garbage Collection), δίνοντας στον χρήστη εικόνα για τη δυναμική συμπεριφορά και επιτρέποντας προσαρμογή παραμέτρων χωρίς διακοπή της ροής.

4.5 Καθυστέρηση – Διαμέτρηση

Στο Spark Structured Streaming, η επίδοση μετριέται κυρίως μέσω δύο συντελεστών: της καθυστέρησης από άκρο σε άκρο (end-to-end latency) και της διαμέτρησης. Δεδομένου ότι η ροή υλοποιείται ως διαδοχικές μικρο-παρτίδες, η ελάχιστη καθυστέρηση δεν μπορεί να είναι μικρότερη από τον ορισμένο χρονικό διάστημα πυροδότησης. Στην πράξη, συνιστώνται τιμές μεταξύ 200 ms και 1 δευτερολέπτου, ώστε να διατηρείται χαμηλό κόστος διαχείρισης εργασιών παρτίδων και στιγμιοτύπων, χωρίς σημαντική αύξηση του χρόνου απόκρισης. [6]

Παράλληλα, το Spark αξιοποιεί πλήρως την κατανεμημένη αρχιτεκτονική του: κάθε μικροπαρτίδα κατακερματίζεται σε τμήματα που ανατίθενται σε πολλαπλούς Executors. Η εκτέλεση σε παράλληλες υπο-εργασίες, σε συνδυασμό με τη βελτιστοποίηση της μηχανής Spark SQL και τον μηχανισμό συγχώνευσης τελεστών (operator fusion), εξασφαλίζει σταθερή διαμέτρηση άνω των 500 χιλ. καταγραφών/δευτ., ανάλογα με τη σύνθεση των ερωτημάτων και τη ρύθμιση πόρων.

Για σενάρια που απαιτούν ακόμη χαμηλότερη καθυστέρηση, το μοντέλο συνεχούς επεξεργασίας επιτρέπει επεξεργασία κάθε εγγραφής επί τόπου, περιορίζοντας την πρόσθετη χρονική υστέρηση από το μοντελο μικρο-παρτίδων. Ωστόσο, λόγω της τρέχουσας ωριμότητας του μοντέλου, υποστηρίζει μόνο σταθερές, απλές συναθροίσεις χωρίς πλούσια λογική διατήρησης κατάστασης.

Τέλος, η παρακολούθηση των συγκεκριμένων μετρικών —όπως η καταγραφή χρόνων εκτέλεσης εργασιών παρτίδων, η χρήση JVM heap και οι ρυθμοί εισόδου/εξόδου—παρέχει συνεχή

ανατροφοδότηση, επιτρέποντας την προσαρμογή των ρυθμίσεων χρονικού διαστήματος πυροδότησης, παραλληλισμού και τμημάτων shuffle για τη βέλτιστη ισορροπία μεταξύ καθυστέρησης και διαμέτρησης. [8]

4.6 Ανοχή σε Σφάλματα

Στο Spark Structured Streaming, η ανοχή σε σφάλματα βασίζεται στο μοντέλο των μικρο-παρτίδων και στις λειτουργίες στιγμιοτύπων που προσφέρει η μηχανή Spark SQL. Κάθε εργασία μικρο-παρτίδων εκτελείται ως ανεξάρτητη εργασία, της οποίας το αποτέλεσμα και η κατάσταση εσωτερικών τελεστών ομαδοποιούνται σε έναν κατάλογο στιγμιοτύπων (π.χ. σε HDFS ή S3). [8, 6]

Κατάσταση Στιγμιοτύπων και Δείκτες Θέσης

Οι ενημερώσεις κατάστασης των μετασχηματισμών με διατήρηση κατάστασης αποθηκεύονται αυτόματα στον κατάλογο στιγμιοτύπων στο τέλος κάθε παρτίδας. Παράλληλα, για πηγές όπως Kafka, καταγράφονται δείκτες θέσης μέσω καταγραφών προεγγραφής (write-ahead logs), διασφαλίζοντας ότι, σε περίπτωση αποτυχίας, η κάθε παρτίδα επαναλαμβάνεται από το τελευταίο αποθετήριο χωρίς απώλεια ή διπλή κατανάλωση.

Exactly-Once Semantics

Με τον συνδυασμό στιγμιοτύπων κατάστασης και καταγραφής δεικτών θέσης, το Structured Streaming παρέχει exactly-once semantics στις εξόδους πινάκων και υποδοχείς δεδομένων (sinks) που το υποστηρίζουν (π.χ. transactional Kafka sinks, idempotent βάσεις). Αυτή η εγγύηση ισχύει ακόμη και σε ενδιάμεσες αποτυχίες κόμβων ή driver, καθότι η επανεκκίνηση βασίζεται στο πιο πρόσφατο πλήρες στιγμιότυπο.

Αποκατάσταση Εργασιών

Σε κάθε αποτυχία εργασίας ή executor, ο driver εντοπίζει το σφάλμα, αποσύρει τις τρέχουσες εκτελέσεις και επανεκκινεί τις εργασίας μικρο-παρτίδων από το τελευταίο στιγμιότυπο. Ο σχεδιασμός του DAG επιτρέπει αυτόματη επαναφορά των καθολικών εξαρτήσεων χωρίς χειροκίνητες παρεμβάσεις, ελαχιστοποιώντας το χρόνο διακοπής (downtime).

Εξωτερικά Στιγμιότυπα και Μεταδεδομένα

Το Spark υποστηρίζει Εξωτερικά Στιγμιότυπα, όπου τα αρχεία στιγμιοτύπων διατηρούνται μετά την ακύρωση ή αναβάθμιση της εφαρμογής. Επιπλέον, ο driver μπορεί να διαχειριστεί

πολλαπλούς καταλόγους στιγμιοτύπων, επιτρέποντας την προσαρμογή πολιτικών διατήρησης αρχείων και τη διαχείριση χώρου.

Περιορισμοί και Συμβιβασμοί

Ενώ το μοντέλο παρτίδων εξασφαλίζει σταθερή συνέπεια, εισάγει ένα ελάχιστο κόστος στη διαμέτρηση και την καθυστέρηση κάθε μίκρο-παρτίδας. Ο προγραμματιστής μπορεί να ρυθμίσει τη συχνότητα των στιγμιοτύπων ώστε να ισορροπεί μεταξύ μικρότερου χρόνου αποκατάστασης και μικρότερου επιπλέον κόστους I/O.

Συνολικά, ο συνδυασμός κατάστασης στιγμιοτύπων, διαχείρισης δεικτών θέσης και αυτόματης επαναφοράς εργασιών παρτίδων καθιστά το Spark Structured Streaming ικανό να παρέχει αξιόπιστη επεξεργασία ροών σε κατανεμημένα περιβάλλοντα με ισχυρές εγγυήσεις συνέπειας.

4.7 Αξιοποίηση Πόρων – Κόστος Επεξεργασίας

Στο Spark Structured Streaming, η διαχείριση πόρων γίνεται μέσω του ενιαίου μοντέλου μνήμης (Unified Memory Management) του Spark, που διαχωρίζει δυναμικά την χρήση μνήμης για εκτέλεση (JVM heaps, Tungsten off-heap) και για αποθήκευση (caching, state). Αυτή η ευέλικτη κατανομή επιτρέπει στο σύστημα να αποφεύγει τη σπατάλη πόρων σε περιπτώσεις μεταβαλλόμενου φορτίου. [3, 8]

Η εκτέλεση κάθε μικρο-παρτίδας ενεργοποιεί διεργασίες σειριοποίησης και αποσειριοποίησης, παραγωγή κώδικα (code generation) μέσω τής μηχανής Tungsten και εκτενή χρήση CPU για τον βελτιστοποιημένο φυσικό σχεδιασμό. Η συγχώνευση τελεστών μειώνει την εναλλαγή μεταξύ εργασιών, αλλά οι πολύπλοκοι μετασχηματισμοί—όπως συνενώσεις ή συναθροίσεις βάσει παραθύρων —μπορούν να αυξήσουν σημαντικά το κόστος.

Στο επίπεδο δικτύου, το shuffle των τμημάτων απαιτεί εγγραφή και ανάγνωση μεγάλου όγκου εγγραφών, εισάγοντας κόστος I/O και ανάγκη για διεργασίες ανάσχεσης ροής. Η ρύθμιση του αριθμού τμημάτων shuffle και των ενδιάμεσων μνημών έχει άμεσο αντίκτυπο στην διαμέτρηση και στη μνήμη δικτύου που δεσμεύεται.

Τα στιγμιότυπα προσθέτουν επιπλέον φορτίο I/O: στο τέλος κάθε μικρο-παρτίδας, η κατάσταση των τελεστών με διατήρηση κατάστασης και οι δείκτες θέσης αποθηκεύονται στο εξωτερικό μέσο (π.χ. HDFS, S3). Η χρήση δημιουργίας ασύγχρονων στιγμιοτύπων περιορίζει την παύση της ροής, αλλά απαιτεί διακριτούς πόρους δικτύου και δίσκου.

Στο επίπεδο μνήμης, η εσωτερική διαχείριση συλλογής σκουπιδιών της JVM μπορεί να γίνει σημείο συμφόρησης εάν οι εργασίες εκτέλεσης και αποθήκευσης μοιράζονται ανεπαρκώς το ίδιο heap. Η επιλογή off-heap memory για Tungsten μπορεί να ελαττώσει το κόστος συλλογής σκουπιδιών, αλλά αυξάνει την πολυπλοκότητα ρύθμισης.

Τέλος, το Spark παρέχει μετρικά σε πραγματικό χρόνο για CPU, μνήμη, I/O και δίκτυο, επιτρέποντας την προσαρμογή του παραλληλισμού, των μεγεθών των παρτίδων και της συχνότητας χρονικών διαστημάτων πυροδότησης ώστε να επιτευχθεί η επιθυμητή ισορροπία μεταξύ απόδοσης και πόρων.

4.8 Χειρισμός Καθυστερημένων Δεδομένων

Στο Spark Structured Streaming, η διαχείριση εκπρόθεσμων εγγραφών ενσωματώνεται στο μοντέλο των παραθύρων και των watermarks ώστε να διασφαλίζεται η συνέπεια των συναθροίσεων χωρίς ανεξέλεγκτη αύξηση της κατάστασης.

Η υποδομή βασίζεται στην κλήση withWatermark(eventTime, delayThreshold), όπου ορίζεται ο χρονοδείκτης γεγονότος (eventTime) και το μέγιστο επιτρεπτό καθυστερημένο διάστημα (delayThreshold). Κατά την εκτέλεση κάθε μικρο-παρτίδας, το σύστημα διατηρεί, για κάθε χρονοπαράθυρο, μόνο τα δεδομένα με χρονικό σημείο μεγαλύτερο της διαφοράς currentWatermark - delayThreshold. [8, 6]

Οι εγγραφές που φτάνουν μετά το watermark, θεωρούνται καθυστερημένες πέραν του ορίου και απορρίπτονται από τα παράθυρα που έχουν ήδη κλείσει, αποφεύγοντας περαιτέρω αυξήσεις στην κατανάλωση μνήμης. Σε συναθροίσεις με ενημέρωση ή προσθήκη (append mode), οι εκπρόθεσμες εγγραφές δεν ανατρέπουν τα υπολογισμένα αποτελέσματα, προστατεύοντας την ακεραιότητα των δημοσιευμένων αποτελεσμάτων.

Για σενάρια όπου είναι απαραίτητη η διαχείριση εξαιρετικά καθυστερημένων εγγραφών (π.χ. αναδρομικές διορθώσεις), προτείνεται η έξοδος σε παράπλευρη έξοδο πριν την εφαρμογή withWatermark, επιτρέποντας μεταγενέστερη, επιτόπου ανάλυση χωρίς να επηρεάζεται η κύρια ροή.

Τέλος, ο καθαρισμός των παραθύρων γίνεται αυτόματα μόλις ξεπεραστεί το άθροισμα watermark + delayThreshold, απελευθερώνοντας όλους τους πόρους που δεσμεύτηκαν για την κατάσταση των συγκεκριμένων παραθύρων. Αντίστοιχα, η ρύθμιση του watermark delayThreshold αποτελεί έναν κρίσιμο συμβιβασμό: μικρότερες τιμές ελαχιστοποιούν τον χώρο

κατάστασης αλλά αυξάνουν την πιθανότητα απόρριψης ετεροχρονισμένων δεδομένων, ενώ μεγαλύτερες τιμές επιτρέπουν πληρέστερη κάλυψη γεγονότων με κόστος πρόσθετης μνήμης.

Κεφάλαιο 5

Συγκριτική Αξιολόγηση

5.1 Θεωρητική Αξιολόγηση

5.1.1 Διαχείριση Κατάστασης

Η διαχείριση κατάστασης αποτελεί κεντρική διάσταση σύγκρισης μεταξύ του Apache Flink και του Spark Structured Streaming. Παρά το κοινό ζητούμενο —τη συνεπή, ανεκτική σε σφάλματα εξέλιξη της κατάστασης— τα δύο συστήματα υλοποιούν διαφορετικές προσεγγίσεις:

Apache Flink

Το Flink υιοθετεί μοντέλο πραγματικής ροής (true streaming) με κατάσταση κλειδιών και τελεστών. Η κατάσταση κατανέμεται και παρακολουθείται ατομικά ανά κλειδί, με αφαιρετικά StateDescriptors και παραμετροποιήσιμα state backends (π.χ. RocksDB, in-memory). Η λήψη στιγμιοτύπων γίνεται ασύγχρονα, μέσω του πρωτοκόλλου Asynchronous Barrier Snapshotting, επιτρέποντας exactly-once semantics χωρίς διακοπή της ροής. Επιπλέον, τα σημεία αποθήκευσης κατάστασης παρέχουν χειροκίνητη διαχείριση εκδόσεων κατάστασης για αναβάθμιση ή μεταφορά, ενώ οι πολιτικές κατάστασης TTL επιτρέπουν τον αυτόματο καθαρισμό παλαιών εγγραφών.

Spark Structured Streaming

Το Spark Structured Streaming βασίζεται σε μικρο-παρτίδες. Κάθε εργασία παρτίδων αποθηκεύει την κατάσταση των τελεστών με διατήρηση κατάστασης στο τέλος της εκτέλεσης, με δημιουργία στιγμιοτύπων σε αξιόπιστο μέσο (π.χ. HDFS, S3), συνδυάζοντας διατήρηση

δεικτών θέσης και στιγμιοτύπων κατάστασης για exactly-once semantics. Τα εσωτερικά αποθηκευτικά συστήματα κατάστασης (state store) υποστηρίζουν σταδιακές ενημερώσεις (incremental updates) και ασύγχρονη δημιουργία στιγμιοτύπων, αλλά η αποθήκευση και ανάκτηση αφορά ολόκληρη την κατάσταση της παρτίδας, καθιστώντας τη λανθάνουσα καθυστέρηση ανά παρτίδα αναπόφευκτη. Το Spark παρέχει επίσης state TTL και "state pruning", όμως οι δυνατότητες προσαρμογής του backend είναι λιγότερο ευέλικτες σε σύγκριση με το Flink.

Συμπέρασμα

Σε επίπεδο κόστους, το μοντέλο του Flink επιβαρύνει περισσότερο το δίκτυο και το I/O λόγω των επαυξητικών, ανά-κλειδί στιγμιοτύπων, αλλά διατηρεί συνεχή επεξεργασία χωρίς κόστος που προκύπτει από την δημιουργία παρτίδων. Αντίθετα, το Spark συγκεντρώνει I/O ανά παρτίδα, μειώνοντας τις μικρές εγγραφές, αλλά εισάγει καθυστέρηση ανά παρτίδα και αυξημένο αποτύπωμα στην μνήμη όταν το μέγεθος παρτίδας αυξάνει.

Τέλος, όσον αφορά την ευκολία χρήσης, το Spark κρύβει εξ' ολοκλήρου την διαχείριση κατάστασης πίσω από το DataFrame API, απαιτώντας ελάχιστες ρυθμίσεις από τον χρήστη, ενώ το Flink παρέχει μεγαλύτερη ευελιξία και διαφάνεια στις στρατηγικές state backends, στιγμιοτύπων και σημείων αποθήκευσης κατάστασης, απευθυνόμενο σε σενάρια που απαιτούν λεπτομερή βελτιστοποίηση της ροής και της ανάκτησης.

Συμπερασματικά, το Flink υπερτερεί σε διατήρηση κατάστασης σε ροές (streaming-native state management) και λεπτομερή παραμετροποίηση, ενώ το Spark Structured Streaming προσφέρει απλότητα χρήσης και αξιοπιστία λόγω παρτίδων (batch-oriented), με κόστος στην καθυστέρηση και στο μοντέλο στιγμιοτύπων.

5.1.2 Καθυστέρηση – Διαμέτρηση

Η διαχείριση της καθυστέρησης και της διαμέτρησης αποτελεί καθοριστικό κριτήριο για συστήματα επεξεργασίας ροών σε κατανεμημένα περιβάλλοντα. Παρά το κοινό αίτημα για χαμηλή καθυστέρηση και υψηλή διαμέτρηση, οι δύο πλατφόρμες υιοθετούν διαφορετικά μοντέλα:

Apache Flink

Το Flink επεξεργάζεται δεδομένα σε ροή μίας εγγραφής τη φορά (record-at-a-time), επιτυγχάνοντας καθυστέρηση από άκρο σε άκρο σε επίπεδα μερικών δεκάδων χιλιοστών του δευτερολέπτου. Η χρήση ανταλλαγής δεδομένων μέσω διαδρομής (pipelined data exchanges) και συγκατοικήσεων διασφαλίζει ότι δεν δημιουργούνται καθυστερήσεις λόγω περιορισμών από παρτίδες. Παράλληλα, η κλιμακούμενη κατανομή τμημάτων και η αποδοτική υποδομή

ανάσχεσης ροής διατηρούν σταθερή διαμέτρηση ακόμη και υπό υψηλό φορτίο, φτάνοντας σε εκατομμύρια γεγονότα ανά δευτερόλεπτο.

Spark Structured Streaming

Το Spark βασίζεται σε μικρο-παρτίδες, όπου η ελάχιστη καθυστέρηση εξαρτάται απευθείας από το χρονικό διάστημα πυροδότησης. Τυπικές τιμές 200 ms–1 δευτερολέπτου οδηγούν σε υψηλή διαμέτρηση (εκατοντάδες χιλιάδες ή περισσότερα γεγονότα/δευτ.), αλλά δεν μπορούν να υποστηρίξουν απαιτήσεις εξαιρετικά χαμηλής καθυστέρης κάτω των 10–20 ms. Το πειραματικό μοντέλο συνεχούς επεξεργασίας μειώνει την καθυστέρηση σε επίπεδα κάτω του ms, αλλά περιορίζει τη λειτουργικότητα και την ωριμότητα των τελεστών με διατήρηση κατάστασης.

Συμπέρασμα

Συνεπώς, το Flink προσφέρει άμεση ανταπόκριση χαμηλής καθυστέρησης με ελαφρώς υψηλότερο κόστος από συνεχή ανάσχεση ροής και λεπτομερής ανταλλαγή μηνυμάτων (fine-grained messaging). Από την άλλη, το Spark παρέχει σταθερή διαμέτρηση και απλοποιημένη βελτιστοποίηση εργασιών παρτίδας, με κόστος περιορισμένης ελαστικότητας στην καθυστέρηση.

Συνολικά, η επιλογή εξαρτάται από τις απαιτήσεις της εφαρμογής: για εργασίας επικεντρωμένες στην καθυστέρηση (latency-sensitive workloads), το Flink υπερτερεί· για εργασίες επικεντρωμένες στην διαμέτρηση (throughput-oriented pipelines) με πιο απλό μοντέλο ανάπτυξης, το Spark Structured Streaming προσφέρει καλύτερη προβλεψιμότητα και ευκολότερη βελτιστοποίηση.

5.1.3 Ανοχή σε Σφάλματα

Η ικανότητα ανάκτησης από σφάλματα χωρίς απώλεια ή διπλή επεξεργασία δεδομένων αποτελεί αναγκαίο στοιχείο για συστήματα επεξεργασίας ροών σε κατανεμημένα περιβάλλοντα. Αν και τόσο το Apache Flink όσο και το Spark Structured Streaming προσφέρουν exactly-once semantics, οι αρχιτεκτονικές τους διαφέρουν ουσιωδώς.

Στο Flink, ο μηχανισμός ανοχής σφαλμάτων βασίζεται στο πρωτόκολλο Asynchronous Barrier Snapshotting. Κατά την εισαγωγή φραγμών (barriers) σε συνεχή ροή, το σύστημα καταγράφει μόνο τις αλλαγές κατάστασης από το προηγούμενο στιγμιότυπο, υποστηρίζοντας σταδιακά στιγμιότυπα και μειώνοντας το κόστος I/O. Η επανεκκίνηση μετά από σφάλμα γίνεται από το πιο πρόσφατο επιτυχημένο στιγμιότυπο, με ελάχιστο παράθυρο επανεπεξεργασίας,

και τα σημεία αποθήκευσης κατάστασης επιτρέπουν χειροκίνητη διαχείριση και αναβάθμιση των εργασιών.

Αντίθετα, το Spark Structured Streaming στηρίζεται στο μοντέλο των μικρο-παρτίδων: κάθε μικρο-παρτίδα ολοκληρώνει τόσο την επεξεργασία όσο και τη λήψη στιγμιότυπου της κατάστασης και των δεικτών θέσης—συχνά μέσω συνδυασμένων καταγραφών προεγγραφής για Kafka sources. Σε περίπτωση αποτυχίας, επανεκκινείται ολόκληρη η τελευταία παρτίδα, επαναλαμβάνοντας κατανάλωση και ανακατασκευή της κατάστασης. Τα εξωτερικά στιγμιότυπα διατηρούνται μετά από ακυρώσεις ή αναβάθμιση, διευκολύνοντας την επιστροφή σε προηγούμενη εκδοχή.

Συμπέρασμα

Συνεπώς, το Flink επιτυγχάνει πιο γρήγορο χρόνο αποκατάστασης (recovery time) και ελαχιστοποιεί το I/O με τα σταδιακά στιγμιότυπα, αλλά εισάγει επιπλέον μηχανισμούς συγχρονισμού μέσω των φραγμών στην πραγματική ροή. Το Spark προσφέρει απλούστερη διαχείριση στιγμιοτύπων —ενσωματωμένη στην εκτέλεση μικρο-παρτίδων—χωρίς ανάγκη επιπλέον πρωτοκόλλων, όμως η επανεκκίνηση εργασιών παρτίδας οδηγεί σε μεγαλύτερο παράθυρο επανεπεξεργασίας και υψηλότερη κατανάλωση πόρων κατά την αποκατάσταση.

Η επιλογή μεταξύ των δύο εξαρτάται από τη συχνότητα σφαλμάτων, το αποδεκτό παράθυρο χρόνου διακοπής και την επιθυμητή ισορροπία ανάμεσα στην αποδοτικότητα του I/O και την ταχύτητα ανάκαμψης.

5.1.4 Αξιοποίηση Πόρων – Κόστος Επεξεργασίας

Η διαχείριση πόρων και το συνολικό κόστος επεξεργασίας αποτελούν κρίσιμες παραμέτρους για την αποδοτικότητα σε κατανεμημένη επεξεργασία ροών. Το Apache Flink και το Spark Structured Streaming ακολουθούν διαφορετικές στρατηγικές:

Apache Flink

Το Flink κατακερματίζει τη μνήμη σε ενδιάμεσες μνήμες δικτύου, διαχειρίσιμη μνήμη (managed memory) για τελεστές και state backend αποθήκη. Η χρήση συγκατοίκησης μειώνει τις εσωτερικές μεταφορές, ενώ οι ασύγχρονες εργασίες παρασκηνίου (asynchronous background tasks) (compaction, checkpointing) εκτελούνται χωρίς παύση της ροής. Η επιλογή state backend καθορίζει το I/O και την καθυστέρηση πρόσβασης, με τα σταδιακά στιγμιότυπα να περιορίζουν την έκταση των λειτουργιών εγγραφής. Ωστόσο, η συνεχής ροή μηνυμάτων

και ο λεπτομερής συγχρονισμός ανάσχεσης ροής ενδέχεται να αυξήσουν το δίκτυο και το κόστος CPU.

Spark Structured Streaming

Το Spark βασίζεται στο ενοποιημένο μοντέλο μνήμης (Unified Memory Management) που «μοιράζει» δυναμικά χώρο μεταξύ εκτέλεσης και αποθήκευσης. Κάθε μικρο-παρτίδα ενεργοποιεί I/O σχετικό με παρτίδες (shuffle, checkpoint), εκμεταλλευόμενο τη μηχανή Tungsten για διανυσματοποιημένη επεξεργασία (vectorized processing) και παραγωγή κώδικα. Η ομαδοποίηση δεδομένων σε παρτίδες μειώνει τον αριθμό των I/O κλήσεων αλλά οδηγεί σε απότομες αυξήσεις ανά παρτίδα, απαιτώντας ρύθμιση μεγέθους παρτίδας και τμημάτων shuffle. Η δημιουργία στιγμιοτύπων προσθέτει επιπλέον κόστος στο δίσκο ή στο δίκτυο, αλλά εκτελείται ασύγχρονα.

Συμπέρασμα

Συνεπώς, στο Flink, η λεπτομερής παραμετροποίηση state backends και οι αδιάκοπες διεργασίες ανάσχεσης ροής επιτυγχάνουν ομαλή συνεχή εκτέλεση με προβλέψιμο κόστος ανά γεγονός, αλλά αυξάνουν τη πολυπλοκότητα ρύθμισης. Στο Spark, η συγκέντρωση I/O σε μικρο-παρτίδες προσφέρει υψηλή διαμέτρηση με εύκολη βελτιστοποίηση εργασιών παρτίδας, όμως οδηγεί σε μη ομαλή χρήση πόρων και απαιτεί προσεκτική ρύθμιση των παραμέτρων ώστε να αποφευχθούν σημεία συμφόρησης σε μνήμη και δίκτυο.

Η επιλογή μεταξύ των δύο εξαρτάται από τη συχνότητα των μικρο-παρτίδων, το μέγεθος των παραθύρων, τις απαιτήσεις για ομαλή κατανομή του φορτίου και την ικανότητα διαχείρισης απότομων αυξήσεων χρήσης πόρων.

5.1.5 Χειρισμός Καθυστερημένων Δεδομένων

Η ικανότητα διαχείρισης εγγραφών που φτάνουν μετά το αρχικό παράθυρο επεξεργασίας είναι κρίσιμη για την ακρίβεια και την αξιοπιστία των εφαρμογών ροών δεδομένων.

Apache Flink

Το Flink ενσωματώνει watermarks και επιτρεπόμενη καθυστέρηση ως βασικά συστατικά του μοντέλου χρονικού σημείο γεγονότος του. Τα watermarks προωθούνται παράλληλα με τα γεγονότα, επιτρέποντας στο σύστημα να «κλείνει» ένα παράθυρο όταν θεωρεί ότι έχει συλλεχθεί η πλειονότητα των σχετικά γεγονότων. Με τις ρυθμίσεις επιτρεπόμενης καθυστέρησης, το Flink διατηρεί ανοιχτά παράθυρα για επιπλέον χρόνο, ανανεώνοντας τα αποτελέσματα όταν

καθυστερημένα δεδομένα φτάνουν εντός του ορίου. Πολύ εκπρόθεσμες εγγραφές μπορούν είτε να απορριφθούν είτε να κατευθυνθούν σε παράπλευρες εξόδους για ξεχωριστή επεξεργασία, διασφαλίζοντας ότι δεν κινδυνεύει η σταθερότητα της κύριας ροής.

Spark Structured Streaming

Στο Structured Streaming, η διαχείριση καθυστερημένων δεδομένων υλοποιείται μέσω watermark APIs σε συνδυασμό με το μοντέλο μικρο-παρτίδων. Ο καθορισμός withWatermark(eventTime, delayThreshold) επιτρέπει στον χρήστη να ορίσει πότε ένα παράθυρο θεωρείται κλειστό και πόσο καθυστερημένα γεγονότα αποδέχεται. Οι εγγραφές μετά το όριο αποκόπτονται αυτομάτως από τις παρτίδες που έχουν ήδη υποβληθεί, αποφεύγοντας αναδρομικές ενημερώσεις. Για πιο ευέλικτη διαχείριση, προτείνεται η χρήση διακριτών παράπλευρων εξόδων πριν το withWatermark ώστε τα πολύ εκπρόθεσμα δεδομένα να υποβάλλονται όποτε απαιτείται.

Συμπέρασμα

Συνεπώς, το Flink προσφέρει πιο δυναμική και συνεχή επανυποβολή παραθύρων, επιτρέποντας ενημερώσεις ακόμη και μετά το αρχικό κλείσιμο, με ελεγχόμενο αποτύπωμα πόρων. Αντιθέτως, το Spark βασίζεται στο όρια παρτίδας και το όριο του watermark, παρέχοντας απλούστερους κανόνες απόρριψης αλλά λιγότερη ευελιξία σε περιπτώσεις όπου η αναδρομική διόρθωση απαιτείται πέραν ενός προκαθορισμένου καθυστέρησης.

Συνολικά, για εφαρμογές με υψηλές απαιτήσεις ακριβείας σε καθυστερημένα δεδομένα, το Flink προσφέρει μεγαλύτερη ελευθερία κινήσεων· για διαδρομές όπου η απλότητα των κανόνων απόρριψης είναι επαρκής, το Spark Structured Streaming παρέχει μια πιο ευθύγραμμη προσέγγιση.

5.2 Πειραματική Αξιολόγηση

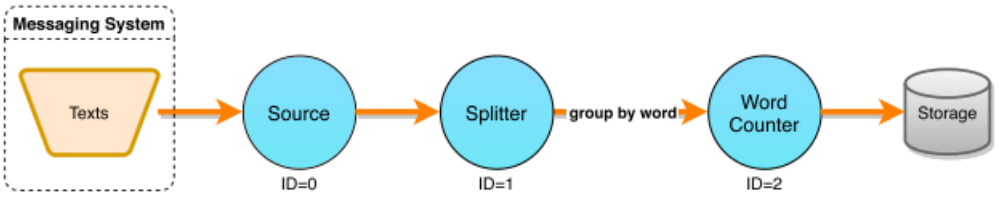
Για την εκτέλεση των πειραματικών αξιολογήσεων χρησιμοποιήθηκε ένα εσωτερικό σύμπλεγμα πέντε κόμβων, το οποίο έχει διαμορφωθεί ώστε να υποστηρίζει τόσο το Apache Flink όσο και το Spark Structured Streaming υπό παρόμοιες συνθήκες. Ο κεντρικός κόμβος λειτουργεί ως Driver (Spark) ή JobManager (Flink), ενώ οι υπόλοιποι τέσσερις κόμβοι ανατίθενται ως Executors ή TaskManagers αντίστοιχα. Κάθε κόμβος διαθέτει 8 GB μνήμης RAM, επαρκή για την αποθήκευση κατάστασης και την εκτέλεση των διεργασιών χωρίς εμφάνιση εικονικών σφαλμάτων μνήμης (OOM). Η παραμετροποίηση των πλατφορμών—όπως ο βαθμός παραλληλισμού, οι ρυθμίσεις ενδιάμεσων μνημών, οι θύρες δικτύου και οι πολιτικές δημιουργίας

στιγμιοτύπων—είχε επιλεγεί ώστε να παραμένει όσο το δυνατόν πιο συνεπής μεταξύ των δύο συστημάτων, εξασφαλίζοντας ότι οι μετρήσεις καθυστέρησης, διαμέτρησης, ανοχής σε σφάλματα κ.λπ. ανταναχλούν αποκλειστικά τις σχεδιαστικές διαφορές των πλατφορμών και όχι διαφορές στην υποκείμενη υποδομή. Για την εξαγωγή των δεδομένων χρησιμοποιήθηκαν τα εργαλεία Prometheus και Grafana και για την παρουσίαση τους χρησιμοποιήθηκε κώδικας Python.

5.2.1 Περιγραφή πειραμάτων

1) WordCount Benchmark

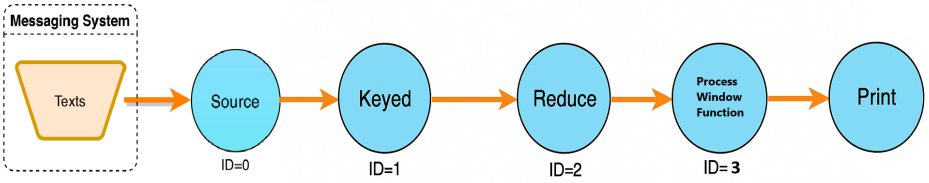
Η εφαρμογή WordCount (WC) [2] (βλ. Σχήμα 5.1) είναι μια δημοφιλής συνθετική εφαρμογή που χωρίζει τις προτάσεις ενός μεγάλου κειμένου σε λέξεις και μετρά τον αριθμό εμφανίσεων κάθε λέξης σε ολόκληρο το σώμα του κειμένου. Ενώ ο τελεστής (Splitter) που κάνει τον διαχωρισμό των λέξεων είναι χωρίς κατάσταση, οι λέξεις αποστέλλονται στα στιγμιότυπα του τελεστή που κάνει την καταμέτρηση (WordCounter) με τέτοιο τρόπο ώστε όλες οι εμφανίσεις της ίδιας λέξης να παραδίδονται στο ίδιο στιγμιότυπο του τελεστή καταμέτρησης. Το γράφημα είναι ένας αγωγός με μια κατανομή τύπου group-by στη μέση.



Σχημα 5.1: Γραφική Απεικόνιση της διαδικασίας του Πειράματος 1. [2]

2) Yahoo Streaming Benchmark

Η εφαρμογή Yahoo Streaming (YS) [12] (βλ. Σχήμα 5.2) λαμβάνει μία ροή γεγονότων που αναπαριστά εμφανίσεις διαφημίσεων. Ο τελεστής Source καταναλώνει αυτά τα γεγονότα σε μορφή JSON και τα αναλύει ώστε να εξάγει το αναγνωριστικό της καμπάνιας. Κάθε αναλυμένο γεγονός μετατρέπεται σε ένα ζεύγος που αποτελείται από το αναγνωριστικό της καμπάνιας και την ακέραια τιμή 1, η οποία αντιπροσωπεύει μία εμφάνιση. Ο τελεστής keyed ομαδοποιεί τα γεγονότα βάσει του αναγνωριστικού καμπάνιας, ώστε οι εμφανίσεις που ανήκουν στην ίδια καμπάνια να επεξεργάζονται μαζί. Ένα κυλιόμενο χρονικό παράθυρο διάρκειας 10 δευτερολέπτων εφαρμόζεται σε κάθε ομάδα, συγκεντρώνοντας τον αριθμό των εμφανίσεων σε αυτό το διάστημα. Ο τελεστής Reduce αθροίζει τις εμφανίσεις για κάθε καμπάνια, παράγοντας τον συνολικό αριθμό εμφανίσεων ανά καμπάνια μέσα στο παράθυρο. Ο τελεστής



Σχημα 5.2: Γραφική Απεικόνιση της διαδικασίας του Πειράματος 2.

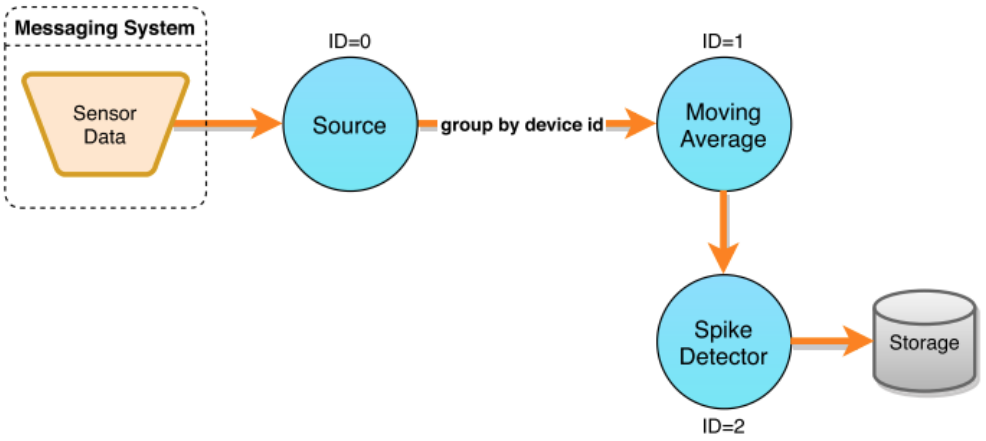
ProcessWindowFunction μορφοποιεί τα αποτελέσματα σε αναγνώσιμες συμβολοσειρές, εκπέμποντας μηνύματα που αναφέρουν το αναγνωριστικό της καμπάνιας και τον αντίστοιχο αριθμό εμφανίσεων. Τέλος, ο τελεστής Print εμφανίζει αυτά τα αποτελέσματα στην κονσόλα, παρέχοντας περιοδικά μετρήσεις εμφανίσεων ανά καμπάνια σε παράθυρα διάρκειας 10 δευτερολέπτων.

3) Spike Detection Benchmark

Η εφαρμογή Spike Detection (SD) [2] (βλ. Σχήμα 5.3) λαμβάνει μια ροή μετρήσεων από αισθητήρες με σκοπό την παρακολούθηση αιχμών. Ο τελεστής MovingAverage λαμβάνει αυτά τα γεγονότα ομαδοποιημένα βάσει του αναγνωριστικού του αισθητήρα και διατηρεί για κάθε αναγνωριστικό ένα κινούμενο παράθυρο. Όταν λαμβάνεται ένα νέο γεγονός, ο τελεστής προσθέτει τη νέα τιμή στο αντίστοιχο παράθυρο και εκπέμπει ένα νέο γεγονός με το αναγνωριστικό της συσκευής, την τρέχουσα τιμή και τον κινούμενο μέσο όρο. Ο τελεστής SpikeDetector λαμβάνει αυτά τα γεγονότα και, με βάση ένα όριο που έχει καθοριστεί κατά την αρχικοποίηση, ελέγχει αν το τρέχον γεγονός αποτελεί αιχμή ή όχι, υπολογίζοντας τη σχετική διαφορά μεταξύ της τρέχουσας τιμής και του μέσου όρου του τελευταίου παραθύρου, και εκπέμπει εκείνες τις τιμές που υπερβαίνουν το όριο.

4) Bargain Index Benchmark

Η εφαρμογή Bargain Index (BI) [2] (βλ. Σχήμα 5.4) αναλύει μετοχές διαθέσιμες προς πώληση, των οποίων η ποσότητα και η τιμή βρίσκονται κάτω από τον μέσο όρο που παρατηρήθηκε στο τελευταίο χρονικό παράθυρο. Για τον σκοπό αυτό, η εφαρμογή υπολογίζει έναν δείκτη ευκαιρίας (bargain index), μια βαθμωτή τιμή που εκφράζει το μέγεθος της ευκαιρίας. Το

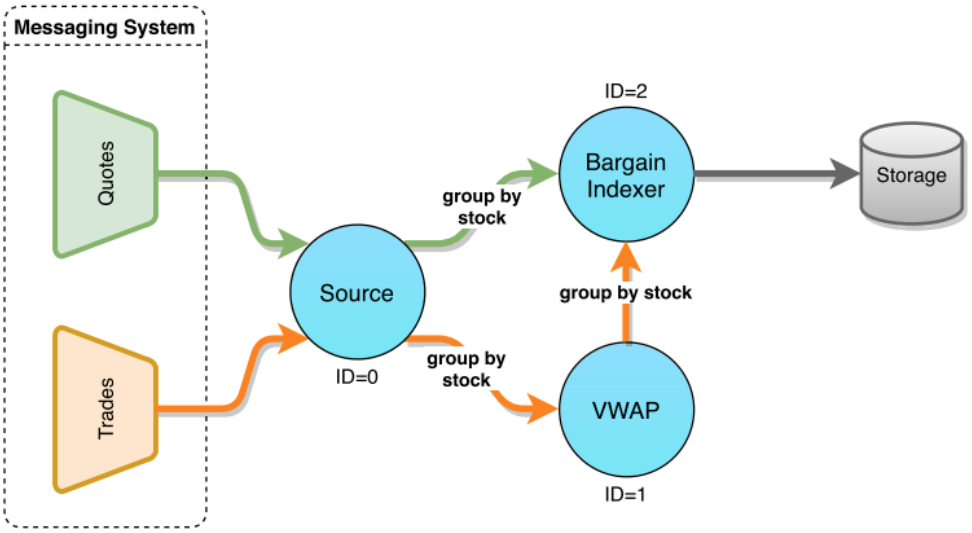


Σχημα 5.3: Γραφική Απεικόνιση της διαδικασίας του Πειράματος 3. [2]

σύστημα τροφοδοτείται με μια ροή συναλλαγών και προσφορών από μια χρηματοπιστωτική αγορά. Μια συναλλαγή (trade) είναι μια ολοκληρωμένη αγοραπωλησία, ενώ μια προσφορά (quote) είναι μια πρόταση αγοράς/πώλησης (που στη χρηματοοικονομική ορολογία αποκαλείται bid/ask). Ο τελεστής Source λαμβάνει και τους δύο τύπους εισόδων και τους διαχωρίζει σε δύο ξεχωριστές ροές που αναλύονται παράλληλα. Ο τελεστής VWAP υπολογίζει τον μέσο όρο τιμής σταθμισμένο ως προς τον όγκο (volume-weighted average price) των συναλλαγών για το ίδιο σύμβολο μετοχής, σε ένα παράθυρο των τελευταίων 15 συναλλαγών που έχουν ληφθεί. Ο τελεστής BargainIndexer λαμβάνει τις προσφορές και υπολογίζει τον δείκτη ευκαιρίας (ο οποίος απαιτεί τη σύγκριση της τιμής και της ποσότητας της προσφοράς με τον τελευταίο VWAP που υπολογίστηκε). Μόνο προσφορές με δείκτη ευκαιρίας μεγαλύτερο από ένα προκαθορισμένο κατώφλι εμφανίζονται ως αποτέλεσμα.

Πινακας 5.1: Πειραματικές Διαμορφώσεις

Α/Α	**Πείραμα**	**Χαρακτηριστικά**
1	WC	Διαμέτρηση 10000 γεγονότα ανά δευτερόλεπτο
2	WC	Διαμέτρηση 50000 γεγονότα ανά δευτερόλεπτο
3	YS	Μέγεθος Γεγονότος 1000KB
4	YS	Μέγεθος Γεγονότος 1000000KB
5	SD	Διαμέτρηση 10000 γεγονότα ανά δευτερόλεπτο
6	SD	Διαμέτρηση 50000 γεγονότα ανά δευτερόλεπτο
7	BI	Διαμέτρηση 10000 γεγονότα ανά δευτερόλεπτο
8	BI	Διαμέτρηση 50000 γεγονότα ανά δευτερόλεπτο



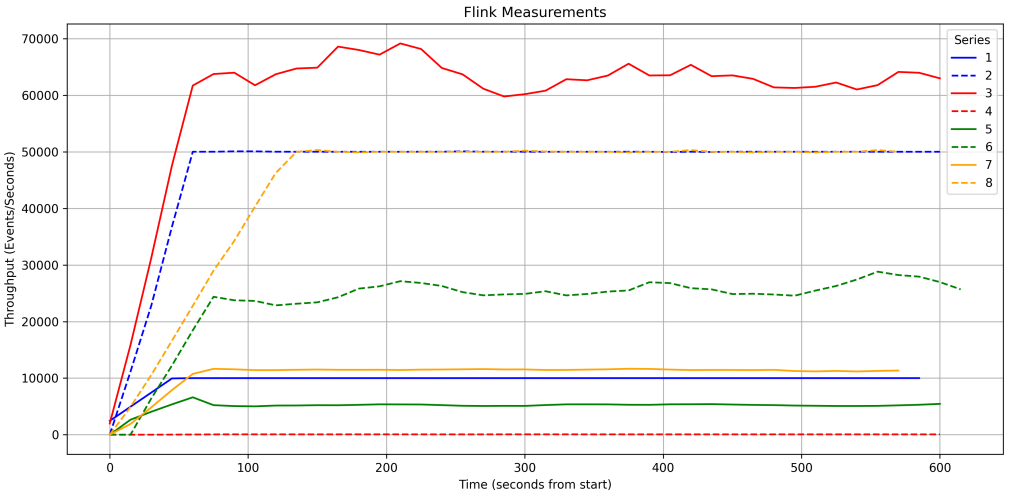
Σχημα 5.4: Γραφική Απεικόνιση της διαδικασίας του Πειράματος 4. [2]

5.2.2 Διαμέτρηση

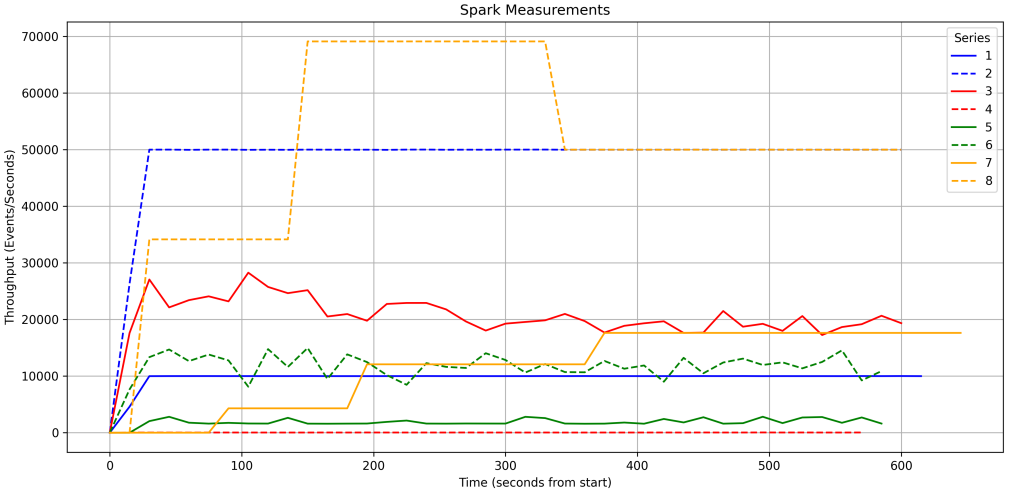
Η συγκριτική αξιολόγηση της διαμέτρησης μεταξύ των δύο συστημάτων επεξεργασίας ροών, Apache Flink και Apache Spark, ανέδειξε ουσιαστικές διαφορές στην απόδοση και τη σταθερότητα των δύο πλαισίων. Το Flink παρουσίασε σαφώς υψηλότερες τιμές διαμέτρησης στα περισσότερα πειραματικά σενάρια, ενώ το Spark εμφάνισε εμφανείς περιορισμούς, ιδιαίτερα σε συνθήκες αυξημένου φόρτου. Στα πειράματα που προσομοιώνουν υψηλό ρυθμό εισερχόμενων γεγονότων, όπως αυτά με περισσότερα από 50.000 γεγονότα ανά δευτερόλεπτο, το Flink κατόρθωσε να διατηρήσει σταθερά επίπεδα επεξεργασίας που ξεπερνούσαν τις 60.000 έως και 70.000 γεγονότα ανά δευτερόλεπτο. Αντίθετα, το Spark δεν ξεπέρασε σταθερά τα 50.000 γεγονότα ανά δευτερόλεπτο, γεγονός που καταδεικνύει την αδυναμία του να κλιμακώσει την απόδοσή του σε συνθήκες υψηλής πίεσης.

Η σταθερότητα αποτελεί επίσης κρίσιμο παράγοντα διαφοροποίησης των δύο συστημάτων. Στο Flink παρατηρήθηκε ότι, μετά την αρχική περίοδο εκκίνησης, η διαμέτρηση σταθεροποιείται σε υψηλά επίπεδα με μικρές διακυμάνσεις. Ενδεικτικό είναι ότι στα πειράματα της κατηγορίας BI η απόδοση διατηρήθηκε σχεδόν αμετάβλητη κοντά στις 50.000 γεγονότα ανά δευτερόλεπτο καθ' όλη τη διάρκεια της μέτρησης. Αντιθέτως, το Spark παρουσίασε απότομες διακυμάνσεις στην αρχική φάση, με έντονες αυξομειώσεις, και τελικά κατέληξε σε παρόμοια σταθερά επίπεδα απόδοσης.

Η ανάλυση των πειραμάτων WC ανέδειξε τη παρόμοια συμπεριφορά των δύο συστημάτων απέναντι σε αυξανόμενους ρυθμούς εισερχόμενων γεγονότων. Με ρυθμό 10.000 και 50.000



Σχημα 5.5: Διάγραμμα διαμέτρησης πειραμάτων Apache Flink.



Σχημα 5.6: Διάγραμμα διαμέτρησης πειραμάτων Apache Spark.

γεγονότα ανά δευτερόλεπτο, και τα δύο συστήματα ανταποκρίθηκαν ικανοποιητικά, χωρίς να εμφανιστούν ενδείξεις συμφόρησης, με μοναδική αμελητέα διαφορά τον χρόνο σταθεροποίησης διαμέτρησης.

Στα πειράματα που εξετάζουν την επίδραση του μεγέθους των γεγονότων (YS), η διαμέτρηση και των δύο πλαισίων άλλαξε σημαντικά. Η αύξηση του μεγέθους των μηνυμάτων επηρέασε δυσανάλογα το Spark, το οποίο παρουσίασε έντονη μείωση στην αποδοτικότητά του σε σχέση με το Flink. Παρόλο που και το Flink επηρεάστηκε από το μεγάλο μέγεθος των μηνυμάτων, η απόδοση του στα μηνύματα των 1000KB παρέμεινε φανερά ανώτερη από το Spark, γεγονός που καταδεικνύει καλύτερη προσαρμοστικότητα στις αυξημένες απαιτήσεις.

Αντίστοιχα, στα σενάρια με πιο σύνθετους υπολογισμούς, όπως τα πειράματα SD και BI, το Flink διατήρησε υψηλή διαμέτρηση, επιδεικνύοντας ικανότητα να διαχειριστεί το αυξημένο υπολογιστικό φορτίο χωρίς σημαντικές απώλειες στην απόδοση. Αντιθέτως, το Spark κατέγραψε σαφώς χαμηλότερες επιδόσεις, γεγονός που υποδηλώνει περιορισμούς στην αποτελεσματική διαχείριση απαιτητικών σεναρίων πραγματικού χρόνου.

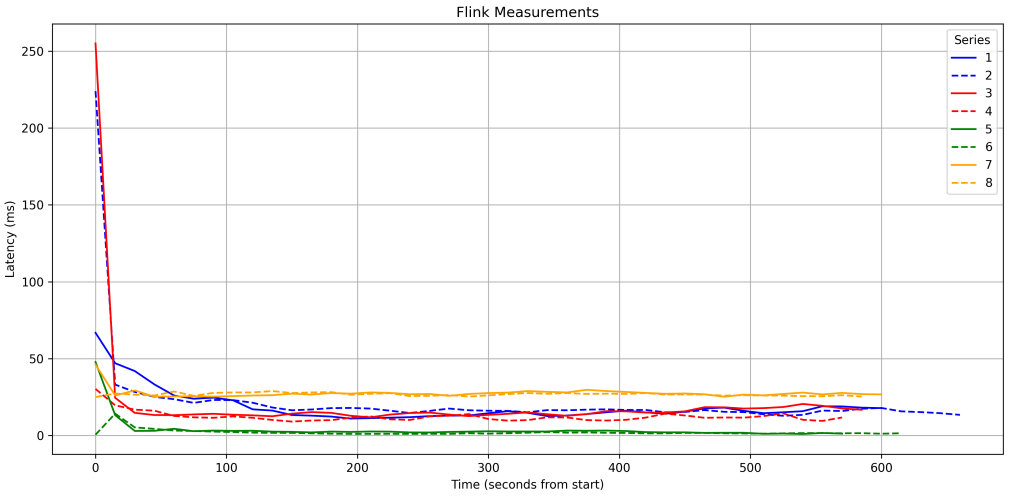
Συνοψίζοντας, η πειραματική ανάλυση καταδεικνύει την υπεροχή του Apache Flink έναντι του Apache Spark ως προς τη διαμέτρηση. Η αρχιτεκτονική του Flink, η οποία είναι σχεδιασμένη εξαρχής για εγγενή επεξεργασία ροών, ευνοεί την αποδοτικότερη αξιοποίηση των διαθέσιμων πόρων και τη διατήρηση υψηλών ρυθμών επεξεργασίας σε διαφορετικά σενάρια. Το Spark, αν και αποδίδει ικανοποιητικά σε χαμηλότερα φορτία, παρουσιάζει σαφείς περιορισμούς όταν καλείται να επεξεργαστεί δεδομένα υψηλού όγκου και απαιτητικότητας, κάτι που περιορίζει τη χρηστικότητά του σε περιβάλλοντα όπου η ταχύτητα επεξεργασίας σε πραγματικό χρόνο αποτελεί κρίσιμο παράγοντα.

5.2.3 Καθυστέρηση

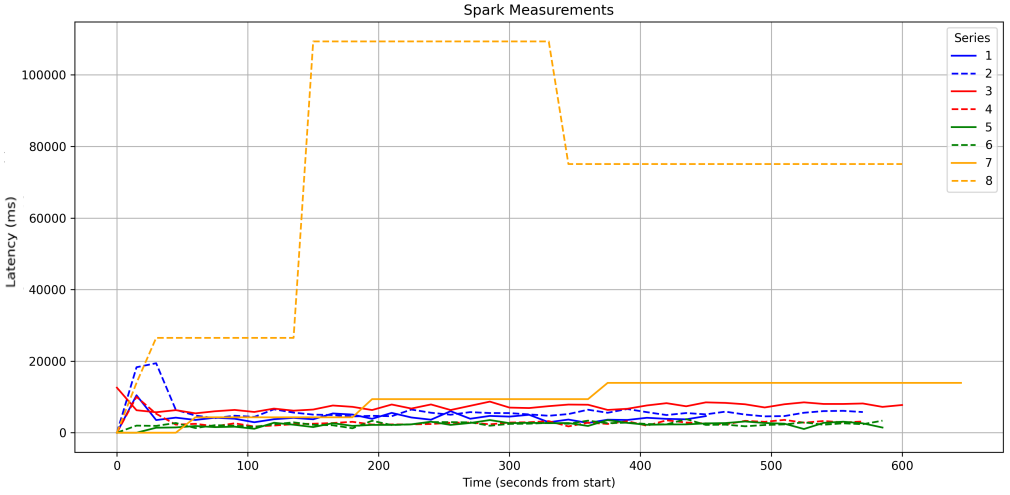
Η ανάλυση των μετρήσεων καθυστέρησης ανέδειξε ακόμη πιο έντονες διαφοροποιήσεις μεταξύ Apache Flink και Apache Spark. Το Flink σε όλα τα σενάρια παρουσιάζει εξαιρετικά χαμηλές τιμές καθυστέρησης, οι οποίες σταθεροποιούνται γρήγορα μετά την αρχική φάση εκκίνησης. Στα περισσότερα πειράματα οι τιμές παραμένουν κάτω από τα 30 ms, ενώ σε ορισμένες περιπτώσεις φτάνουν ακόμη και σε επίπεδα της τάξης των λίγων χιλιοστών του δευτερολέπτου. Παρά το γεγονός ότι στην αρχή ορισμένων πειραμάτων καταγράφηκαν υψηλότερες τιμές, αυτές μειώθηκαν ραγδαία και η καθυστέρηση διατηρήθηκε στη συνέχεια σε σταθερά και προβλέψιμα επίπεδα. Η συμπεριφορά αυτή υποδηλώνει ότι το Flink μπορεί να ανταποκριθεί αξιόπιστα σε εφαρμογές πραγματικού χρόνου, όπου η χαμηλή καθυστέρηση αποτελεί κρίσιμο παράγοντα.

Αντιθέτως, το Spark εμφάνισε σημαντικά υψηλότερες τιμές καθυστέρησης, οι οποίες χαρακτηρίζονται από έντονες διακυμάνσεις και, σε ορισμένες περιπτώσεις, από ακραίες αυξήσεις που ξεπερνούν κατά πολύ τα αποδεκτά όρια για εφαρμογές πραγματικού χρόνου. Στα πειράματα της κατηγορίας BI, η καθυστέρηση κατέγραψε δραματική αύξηση, φτάνοντας σε ακραίες τιμές που ξεπερνούσαν ακόμη και τα 100 δευτερόλεπτα. Αυτή η συμπεριφορά καταδεικνύει την αδυναμία του Spark να ανταποκριθεί σε απαιτητικά σενάρια ροών όπου η άμεση απόκριση είναι απαραίτητη.

Επιπλέον, σε αντίθεση με το Flink, το οποίο σταθεροποιεί γρήγορα την καθυστέρησή του, το Spark παρουσιάζει συχνά προοδευτική αύξηση με την πάροδο του χρόνου. Το γεγονός



Σχημα 5.7: Διάγραμμα καθυστέρησης πειραμάτων Apache Flink.



Σχημα 5.8: Διάγραμμα καθυστέρησης πειραμάτων Apache Spark.

αυτό συνδέεται με τον κορεσμό των διαθέσιμων πόρων και με την περιορισμένη δυνατότητα κλιμάκωσης υπό υψηλό φόρτο. Αντίθετα, το Flink φαίνεται να αξιοποιεί αποτελεσματικότερα τους διαθέσιμους πόρους, διατηρώντας τη σταθερότητα της καθυστέρησης ακόμη και σε πιο απαιτητικά σενάρια με αυξημένη πολυπλοκότητα.

Συνολικά, η πειραματική διαδικασία καταδεικνύει ότι το Apache Flink υπερτερεί σαφώς του Apache Spark ως προς την καθυστέρηση. Η αρχιτεκτονική του, που έχει σχεδιαστεί εξαρχής για εγγενή επεξεργασία ροών, το καθιστά ικανό να προσφέρει χαμηλή και σταθερή καθυστέρηση, χαρακτηριστικό απαραίτητο για εφαρμογές πραγματικού χρόνου. Αντίθετα, το Spark, αν και επαρκές σε σενάρια όπου η καθυστέρηση δεν είναι πρωταρχικός περιορισμός,

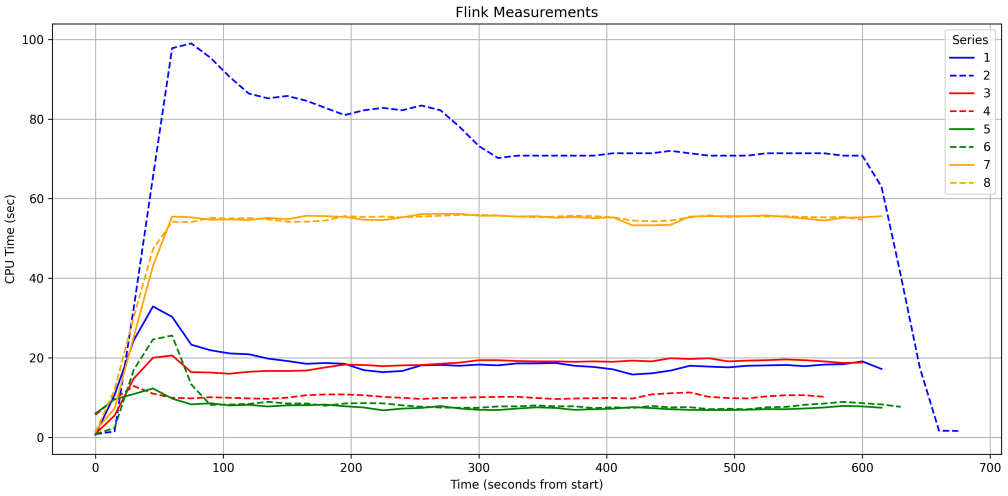
εμφανίζει σοβαρούς περιορισμούς σε περιβάλλοντα που απαιτούν γρήγορη επεξεργασία και άμεση απόκριση.

5.2.4 Χρήση CPU

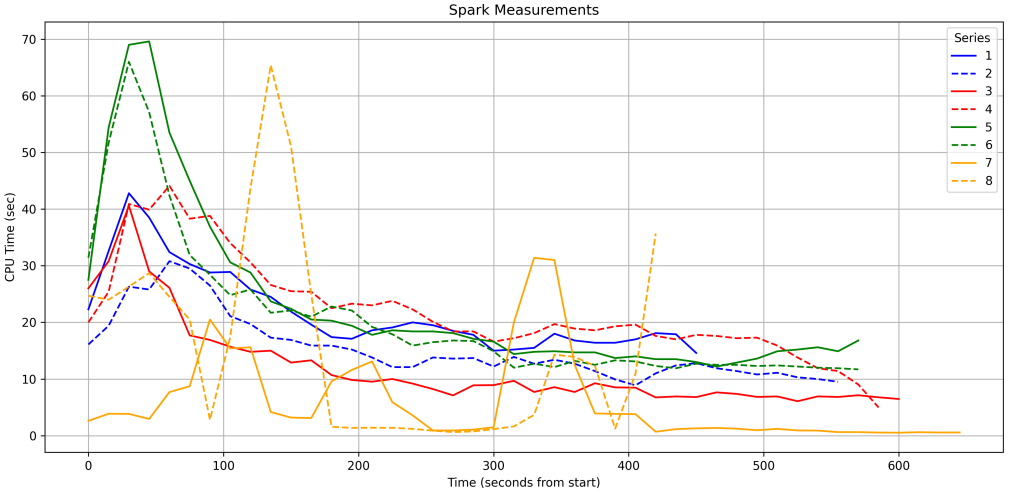
Η ανάλυση της κατανάλωσης επεξεργαστικών πόρων αποκαλύπτει ουσιαστικές διαφορές μεταξύ του Apache Flink και του Apache Spark. Στην περίπτωση του Flink, παρατηρείται μια αρχική απότομη αύξηση στη χρήση της CPU, η οποία κορυφώνεται στα πρώτα δευτερόλεπτα της εκτέλεσης. Ωστόσο, στη συνέχεια οι μετρήσεις σταθεροποιούνται γρήγορα σε σταθερά επίπεδα, διατηρώντας μια σχετικά ομαλή πορεία μέχρι το τέλος των πειραμάτων. Το γεγονός αυτό δείχνει ότι το Flink αξιοποιεί αποτελεσματικά τους διαθέσιμους πόρους και καταφέρνει να διατηρήσει την επεξεργαστική του κατανάλωση σε προβλέψιμα όρια, ακόμη και σε απαιτητικά σενάρια. Η συμπεριφορά αυτή συνδέεται με τον εγγενή σχεδιασμό του συστήματος για συνεχή ροή δεδομένων και χαμηλή καθυστέρηση, καθώς και με τη δυνατότητα αποδοτικής διαχείρισης των threads και των task managers.

Αντίθετα, το Spark εμφανίζει πιο ασταθή συμπεριφορά. Στα πρώτα στάδια των πειραμάτων παρατηρείται έντονη διακύμανση στη χρήση της CPU, με απότομες κορυφώσεις που σε ορισμένες περιπτώσεις ξεπερνούν τα 60 ή 70 δευτερόλεπτα CPU χρόνου. Παρόλο που στη συνέχεια η κατανάλωση τείνει να μειώνεται, η συνολική πορεία παραμένει ακανόνιστη, με συνεχείς αυξομειώσεις και απουσία σταθεροποίησης σε σταθερά επίπεδα. Οι απότομες διακυμάνσεις καταδεικνύουν ότι το Spark δεν επιτυγχάνει την ίδια συνέπεια στη διαχείριση πόρων, γεγονός που συνδέεται με την αρχιτεκτονική του βασισμένη σε μικρο-παρτίδες. Η μέθοδος αυτή εισάγει επιπλέον επιβάρυνση στο σύστημα, καθώς οι πόροι καταναλώνονται περιοδικά και όχι με συνεχή και σταθερό ρυθμό, όπως στο Flink.

Η συνολική εικόνα από τις μετρήσεις δείχνει ότι το Flink υπερέχει ως προς την αποδοτική και σταθερή χρήση της CPU. Η σταθεροποίηση σε προβλέψιμα επίπεδα χρήσης το καθιστά πιο αξιόπιστο για εφαρμογές μεγάλης κλίμακας και με αυξημένες απαιτήσεις σε πραγματικό χρόνο. Το Spark, αν και μπορεί να αποδώσει ικανοποιητικά σε περιπτώσεις όπου η πλήρης αξιοποίηση των πόρων δεν είναι κρίσιμη, παρουσιάζει περιορισμούς σε σενάρια που απαιτούν συνεχή και ομαλή χρήση της CPU. Οι ασταθείς μετρήσεις ενδέχεται να οδηγήσουν σε προβλήματα κλιμάκωσης και μειωμένη ενεργειακή αποδοτικότητα, κάτι που αποτελεί σημαντικό μειονέκτημα σε περιβάλλοντα παραγωγής.



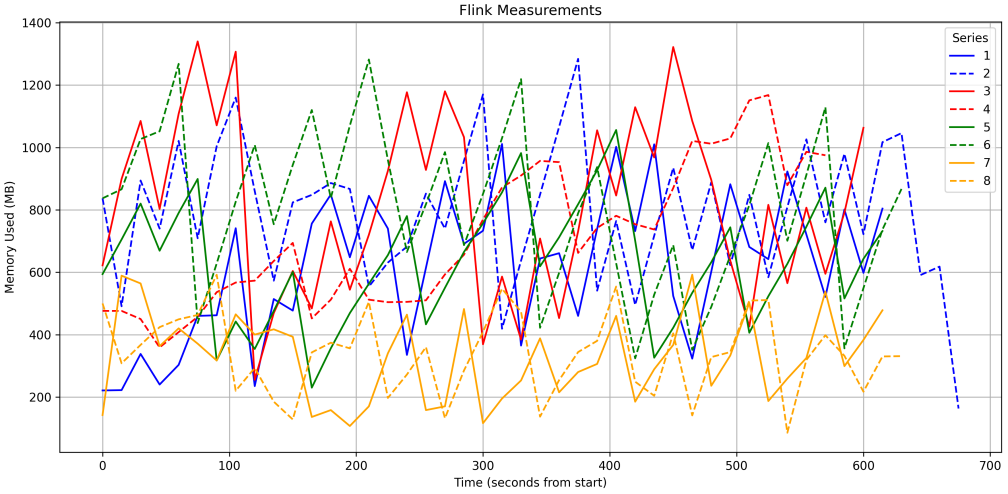
Σχημα 5.9: Διάγραμμα χρήσης CPU πειραμάτων Apache Flink.



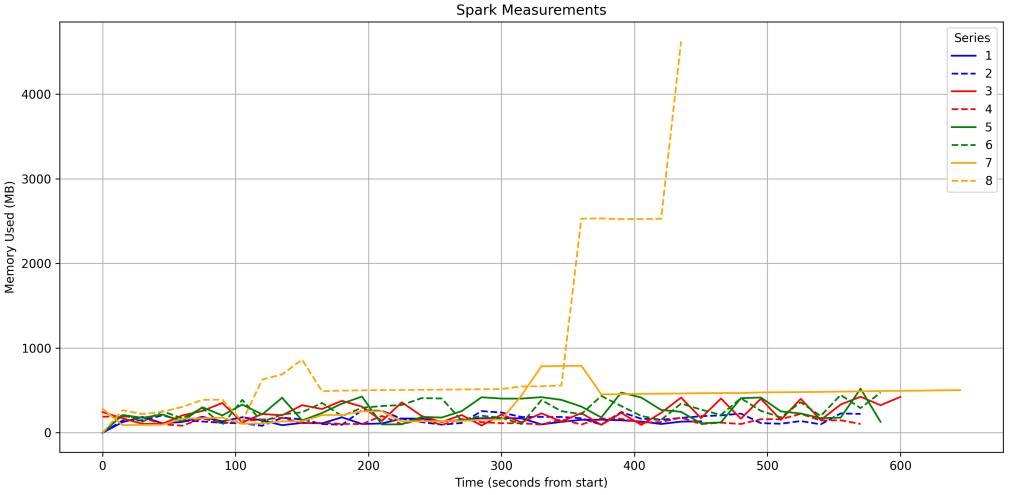
Σχημα 5.10: Διάγραμμα χρήσης CPU πειραμάτων Apache Spark.

5.2.5 Χρήση μνήμης

Η ανάλυση της χρήσης μνήμης αποκαλύπτει διαφορετικά μοτίβα κατανάλωσης μεταξύ του Apache Flink και του Apache Spark. Στην περίπτωση του Flink, παρατηρείται μια έντονη διακύμανση στη χρήση της μνήμης σε όλη τη διάρκεια των πειραμάτων. Οι τιμές παρουσιάζουν συνεχείς αυξομειώσεις, κυμαινόμενες μεταξύ μερικών εκατοντάδων MB έως και πάνω από 1 GB. Αυτή η συμπεριφορά αντικατοπτρίζει τον τρόπο με τον οποίο το Flink διαχειρίζεται τα δεδομένα σε πραγματικό χρόνο, διατηρώντας πληροφορία κατάστασης στη μνήμη και πραγματοποιώντας συχνές λειτουργίες συλλογής και εκκαθάρισης. Παρά τις αυξομειώσεις, το σύστημα καταφέρνει να παραμένει λειτουργικά σταθερό, χωρίς ενδείξεις ανεξέλεγκτης



Σχημα 5.11: Διάγραμμα χρήσης μνήμης πειραμάτων Apache Flink.



Σχημα 5.12: Διάγραμμα χρήσης μνήμης πειραμάτων Apache Spark.

κατανάλωσης μνήμης ή διαρροών. Ωστόσο, η συνεχής εναλλαγή στα επίπεδα κατανάλωσης μπορεί να επιβαρύνει το περιβάλλον εκτέλεσης, ιδίως σε σενάρια μεγάλης κλίμακας όπου η μνήμη αποτελεί κρίσιμο πόρο.

Αντίθετα, το Spark εμφανίζει μια διαφορετική εικόνα. Στην πλειονότητα των μετρήσεων, η χρήση της μνήμης παραμένει σχετικά χαμηλή και σταθερή, με τιμές που σπάνια ξεπερνούν τα μερικές εκατοντάδες MB. Ωστόσο, σε συγκεκριμένες περιπτώσεις καταγράφονται απότομες αυξήσεις, οι οποίες σε ορισμένα σενάρια φτάνουν ακόμη και τα 4 GB. Αυτές οι αιφνίδιες κορυφώσεις οφείλονται στην προσέγγιση μικρο-παρτίδων του Spark, όπου η προσωρινή συσσώρευση δεδομένων οδηγεί σε έντονη αύξηση της κατανάλωσης μνήμης. Παρά το γεγονός ότι οι τιμές επανέρχονται σε χαμηλότερα επίπεδα μετά τις κορυφώσεις, η συμπεριφορά αυτή

υποδεικνύει μειωμένη προβλεψιμότητα, καθώς το σύστημα μπορεί να απαιτήσει αιφνιδιαστικά πολύ περισσότερους πόρους.

Συγκρίνοντας τα δύο συστήματα, γίνεται φανερό ότι το Flink αξιοποιεί πιο σταθερά τη διαθέσιμη μνήμη, αλλά με συνεχείς διακυμάνσεις που αντικατοπτρίζουν την φύση διατήρησης κατάστασης των εφαρμογών του. Το Spark, αντίθετα, διατηρεί χαμηλότερα επίπεδα κατανάλωσης στο μεγαλύτερο μέρος του χρόνου, αλλά με έντονες και απρόβλεπτες αιχμές που ενδέχεται να προκαλέσουν προβλήματα σε περιβάλλοντα περιορισμένων πόρων. Έτσι, το Flink αποδεικνύεται πιο αξιόπιστο σε όρους συνέπειας, ενώ το Spark εμφανίζει καλύτερη μέση απόδοση αλλά με τον κίνδυνο ξαφνικών και υψηλών απαιτήσεων σε μνήμη.

Κεφάλαιο 6

Συμπεράσματα

Η συγκριτική αξιολόγηση των συστημάτων Apache Flink και Apache Spark Structured Streaming στον τομέα της κατανεμημένης επεξεργασίας ροών δεδομένων βασίστηκε τόσο στη θεωρητική ανάλυση όσο και στην εμπειρική διερεύνηση μέσω πειραμάτων, λαμβάνοντας υπόψη τις πέντε κρίσιμες διαστάσεις που καθορίζουν την απόδοση και την καταλληλότητα τέτοιων συστημάτων:

- Διαχείριση Κατάστασης
- Καθυστέρηση και Διαμέτρηση
- Ανοχή σε Σφάλματα
- Αξιοποίηση Πόρων και Κόστος Επεξεργασίας
- Χειρισμός Καθυστερημένων Δεδομένων

Τα αποτελέσματα της ανάλυσης και των πειραμάτων ανέδειξαν με σαφήνεια τα συγκριτικά πλεονεκτήματα και τους περιορισμούς κάθε πλατφόρμας.

6.1 Apache Flink

Το Flink απέδειξε ότι αποτελεί μία από τις πλέον ισχυρές επιλογές για σενάρια πραγματικού χρόνου. Το μοντέλο true streaming που υιοθετεί επιτρέπει την επεξεργασία των δεδομένων σε επίπεδο εγγραφής (record-at-a-time), προσφέροντας εξαιρετικά χαμηλή καθυστέρηση από

άκρο σε άκρο. Στην πειραματική αξιολόγηση, το Flink κατέγραψε τη χαμηλότερη καθυστέρηση και την υψηλότερη διαμέτρηση σε όλα τα σενάρια, επιβεβαιώνοντας την ικανότητά του να διαχειρίζεται μεγάλο όγκο και ταχύτητα δεδομένων.

Αναφορικά με τη διαχείριση κατάστασης, το Flink διαθέτει εξελιγμένους μηχανισμούς μέσω state backends και snapshots, προσφέροντας ακριβείς εγγυήσεις παράδοσης, ακόμα και σε συνθήκες αποτυχιών. Η ανοχή σε σφάλματα υλοποιείται με συνέπεια, ενώ η υποστήριξη του για χειρισμό καθυστερημένων δεδομένων είναι ιδιαίτερα ευέλικτη μέσω τεχνικών όπως τα watermarks και η επιτρεπόμενη καθυστέρηση (allowed lateness).

Ωστόσο, το Flink παρουσίασε αυξημένη κατανάλωση μνήμης, ειδικά σε σενάρια μεγάλης κλίμακας, γεγονός που ενδέχεται να αυξήσει το κόστος υποδομής και να απαιτεί προσεκτική παραμετροποίηση.

Ενδεικνυόμενες Εφαρμογές:

- Χρηματοπιστωτικές συναλλαγές σε πραγματικό χρόνο
- Συστήματα ανίχνευσης απάτης
- IoT εφαρμογές υψηλής ευαισθησίας χρονισμού
- Έξυπνα βιομηχανικά συστήματα και παρακολούθηση
- Ροές μηχανικής μάθησης και ανάλυση προτύπων σε πραγματικό χρόνο

6.2 Apache Spark Structured Streaming

Το Spark βασίζεται στο μοντέλο επεξεργασίας μικρο-παρτίδων (micro-batching), το οποίο απλοποιεί σημαντικά την ανάπτυξη και παραμετροποίηση των εφαρμογών. Το προγραμματιστικό πρότυπο DataFrame API προσφέρει υψηλό επίπεδο αφαιρετικότητας και ευκολία χρήσης, γεγονός που καθιστά το Spark ιδιαίτερα προσιτό, ειδικά σε ομάδες με περιορισμένη εμπειρία σε κατανεμημένα συστήματα.

Στην πειραματική αξιολόγηση, το Spark εμφάνισε υψηλότερη καθυστέρηση σε όλα τα σενάρια, λόγω του εγγενούς μοντέλου μικρο-παρτίδων. Παρόλα αυτά, η συνολική κατανάλωση μνήμης ήταν χαμηλότερη συγκριτικά με το Flink, γεγονός που καθιστά το Spark πιο κατάλληλο για περιβάλλοντα περιορισμένων πόρων.

Αναφορικά με τη διαχείριση κατάστασης και την ανοχή σε σφάλματα, το Spark παρέχει ικανοποιητική υποστήριξη, αν και λιγότερο ευέλικτη από το Flink. Ο χειρισμός καθυστερημένων δεδομένων υλοποιείται μέσω στατικών παραθύρων και watermarks, χωρίς όμως την πλήρη δυναμικότητα που προσφέρει το Flink.

Ενδεικνυόμενες Εφαρμογές:

- Εφαρμογές όπου η χαμηλή καθυστέρηση δεν είναι κρίσιμη
- Αθροιστική ανάλυση δεδομένων και στατιστικά διαγράμματα
- Ανάλυση καταγραφών συστημάτων (log analysis)
- Εκπαιδευτικές ή πειραματικές υλοποιήσεις επεξεργασίας ροών

6.3 Συμπεράσματα Πειραματικής Αξιολόγησης

Η πειραματική αξιολόγηση, μέσω των οχτώ σεναρίων που υλοποιήθηκαν, επιβεβαίωσε ότι το Flink υπερέχει σε περιβάλλοντα που απαιτούν ελάχιστη καθυστέρηση και υψηλή διαμέτρηση, ιδίως σε σενάρια με μεγάλα μεγέθη γεγονότων και σύνθετες λειτουργίες, όπως η συνολική και τοπική ομαδοποίηση. Το Spark, αν και πιο εύχρηστο και οικονομικό σε πόρους, περιορίζεται σε εφαρμογές όπου η άμεση απόκριση δεν αποτελεί απαραίτητη προϋπόθεση.

6.4 Συνολική Εκτίμηση

Η επιλογή του κατάλληλου συστήματος εξαρτάται άμεσα από τις απαιτήσεις της εκάστοτε εφαρμογής. Το Flink αποτελεί την προτιμότερη λύση για κρίσιμες, πραγματικού χρόνου εφαρμογές υψηλής πολυπλοκότητας, όπου απαιτούνται αυστηρές εγγυήσεις απόδοσης και συνέπειας. Το Spark, από την άλλη, παρέχει μία πιο απλοποιημένη, οικονομική και εύκολη στην υλοποίηση προσέγγιση για περιπτώσεις όπου η χαμηλή καθυστέρηση δεν είναι απολύτως απαραίτητη.

Η εμπεριστατωμένη κατανόηση των πλεονεκτημάτων και των περιορισμών κάθε πλατφόρμας, σε συνδυασμό με τις ανάγκες του εκάστοτε σεναρίου, αποτελούν θεμέλιο για τον ορθολογικό σχεδιασμό και την υλοποίηση αποδοτικών και αξιόπιστων συστημάτων επεξεργασίας ροών δεδομένων.

6.5 Σχετικές Έρευνες

A Survey on the Evolution of Stream Processing Systems

Η έρευνα «A Survey on the Evolution of Stream Processing Systems» [13] αποτελεί μια εκτενή και ιδιαίτερα σημαντική ανασκόπηση της εξέλιξης των συστημάτων επεξεργασίας ροών δεδομένων τα τελευταία είκοσι χρόνια. Ο βασικός σκοπός της μελέτης είναι η συστηματική αποτύπωση της πορείας των τεχνολογιών stream processing από την πρώτη γενιά συστημάτων (2000–2010) έως τα πιο σύγχρονα πλαίσια (2011–2023). Οι συγγραφείς επιχειρούν να αναδείξουν τον τρόπο με τον οποίο οι αρχικές ερευνητικές ιδέες, όπως η επεξεργασία παραθύρων χρόνου και η παράλληλη εκτέλεση, εξελίχθηκαν σε πλήρως κατανεμημένα και ανθεκτικά συστήματα, ικανά να υποστηρίξουν εφαρμογές πραγματικού χρόνου σε περιβάλλοντα cloud.

Η μελέτη δίνει ιδιαίτερη έμφαση στη διαχείριση εκτός σειράς δεδομένων, στη διαχείριση κατάστασης, στην ανοχή σε σφάλματα και στην ελαστικότητα των συστημάτων. Επιπλέον, επιχειρεί να ενοποιήσει την ποικιλία όρων και ορισμών που χρησιμοποιούνται στη βιβλιογραφία, προτείνοντας ένα συνεκτικό πλαίσιο κατηγοριοποίησης των συστημάτων επεξεργασίας ροών με βάση την αρχιτεκτονική και το προγραμματιστικό τους μοντέλο. Τα συμπεράσματα των συγγραφέων υποδεικνύουν ότι τα σύγχρονα συστήματα, όπως τα Apache Flink, Google Dataflow και Apache Spark Streaming, έχουν καταφέρει να ενοποιήσουν την επεξεργασία παρτίδων και ροών, ενσωματώνοντας προηγμένα μοντέλα διαχείρισης κατάστασης και μηχανισμούς ανθεκτικότητας που εξασφαλίζουν αξιοπιστία και υψηλή απόδοση.

Σε αντίθεση με αυτήν τη γενική και ιστορικά προσανατολισμένη ανασκόπηση, η παρούσα διπλωματική εργασία εστιάζει σε μια πιο εφαρμοσμένη και πειραματικά τεκμηριωμένη προσέγγιση. Ενώ η παραπάνω έρευνα στοχεύει στη θεωρητική κατηγοριοποίηση και στη συνολική επισκόπηση του πεδίου, η παρούσα εργασία επικεντρώνεται στη συγκριτική αξιολόγηση συγκεκριμένων συστημάτων επεξεργασίας ροών, με μετρήσιμα κριτήρια απόδοσης όπως η καθυστέρηση, η διαμέτρηση, η αξιοποίηση πόρων και η ανοχή σε σφάλματα. Επομένως, η παρούσα μελέτη διαφοροποιείται ουσιαστικά ως προς το είδος και το βάθος της ανάλυσης, καθώς μετατοπίζει το ενδιαφέρον από τη θεωρητική ταξινόμηση προς την πειραματική αξιολόγηση και τη σύγκριση των τεχνολογιών στην πράξη.

Evaluation of Stream Processing Frameworks

Η μελέτη «Evaluation of Stream Processing Frameworks» [14] επικεντρώνεται στην πειραματική αξιολόγηση και συγκριτική ανάλυση των κυριότερων πλαισίων επεξεργασίας ροών δεδομένων, όπως τα Apache Flink, Apache Spark Streaming, Structured Streaming και

Kafka Streams. Ο κύριος σκοπός της έρευνας είναι να μετρηθεί και να συγκριθεί η απόδοση αυτών των συστημάτων υπό διαφορετικά σενάρια και φόρτους εργασίας, προκειμένου να καθοριστούν οι παράγοντες που επηρεάζουν την καθυστέρηση , τη διαμέτρηση και την κατανάλωση πόρων. Για τον σκοπό αυτόν, οι συγγραφείς ανέπτυξαν ένα εκτεταμένο πείραμα benchmarking, βασισμένο σε δεδομένα αισθητήρων κυκλοφορίας από το δίκτυο NDW της Ολλανδίας, το οποίο περιλαμβάνει διαφορετικά είδη φόρτων εργασίας: σταθερής ροής, αιχμών εκκίνησης (startup bursts) και περιοδικών εκρήξεων δεδομένων (periodic bursts).

Η μελέτη αποδεικνύει ότι τα συστήματα event-driven, όπως το Flink και το Kafka Streams, υπερτερούν σε εφαρμογές που απαιτούν χαμηλή καθυστέρηση και πραγματικό χρόνο, ενώ τα συστήματα μικρο-παρτίδων, όπως τα Spark Streaming και Structured Streaming, προσφέρουν υψηλότερη διαμέτρηση με το κόστος αυξημένης καθυστέρησης. Ειδικότερα, το Flink διατηρεί την καλύτερη απόδοση υπό αυστηρούς περιορισμούς καθυστέρησης, ενώ το Structured Streaming επιτυγχάνει τον υψηλότερο ρυθμό επεξεργασίας γεγονότων ανά δευτερόλεπτο. Παράλληλα, οι συγγραφείς αναλύουν τον ρόλο της ρύθμισης παραμέτρων (parameter tuning), της διαχείρισης κατάστασης και των παραθύρων (windowing) στην τελική απόδοση των συστημάτων, παρέχοντας κατευθυντήριες γραμμές για την επιλογή του κατάλληλου framework ανάλογα με τις απαιτήσεις κάθε εφαρμογής.

Η παρούσα διπλωματική εργασία σχετίζεται θεματικά με την προσέγγιση της μελέτης των van Dongen και Van den Poel, καθώς επίσης επιδιώκει τη συγκριτική αξιολόγηση συστημάτων επεξεργασίας ροών, εστιάζοντας στα ίδια βασικά κριτήρια — καθυστέρηση, διαμέτρηση, ανοχή σε σφάλματα και αξιοποίηση πόρων. Ωστόσο, ενώ η παραπάνω εργασία των εξετάζει ένα σύνολο διαφορετικών frameworks υπό κοινές συνθήκες φόρτου, η παρούσα έρευνα εστιάζει πιο στοχευμένα στη συγκριτική μελέτη των Apache Flink και Spark Structured Streaming, αναλύοντας εις βάθος τη συμπεριφορά τους σε πειραματικά περιβάλλοντα και αναδεικνύοντας συγκεκριμένες τεχνικές διαφοροποιήσεις μεταξύ τους. Έτσι, η παρούσα εργασία μπορεί να θεωρηθεί ως επικεντρωμένη επέκταση της λογικής της προαναφερθείσας μελέτης, παρέχοντας λεπτομερέστερη ανάλυση σε μικρότερο αλλά κρίσιμο υποσύνολο των πλέον αντιπροσωπευτικών συστημάτων.

A Survey of Distributed Data Stream Processing Frameworks

Η μελέτη «A Survey of Distributed Data Stream Processing Frameworks» [15]αποτελεί μία από τις πιο ολοκληρωμένες και τεχνικά λεπτομερείς ανασκοπήσεις στο πεδίο των κατανεμημένων συστημάτων επεξεργασίας ροών δεδομένων (Distributed Data Stream Processing Systems – DSPS). Σκοπός της μελέτης είναι η δημιουργία μιας συστηματικής ταξινομίας και συγκριτικής ανάλυσης των κυριότερων πλαισίων επεξεργασίας ροών, τόσο ανοιχτού κώδικα

όσο και εμπορικών, με στόχο την κατανόηση των διαφορών στον σχεδιασμό, στην αρχιτεκτονική και στις λειτουργικές τους δυνατότητες. Οι συγγραφείς παρουσιάζουν ένα ενιαίο θεωρητικό πλαίσιο το οποίο περιγράφει τη γενική αρχιτεκτονική ενός DSPS, τα βασικά δομικά του στοιχεία (στρώματα εισαγωγής, επεξεργασίας, αποθήκευσης, διαχείρισης πόρων και εξόδου) και προτείνουν ένα σύνολο χαρακτηριστικών για την αξιολόγηση των DSPEs (Data Stream Processing Engines), όπως το μοντέλο προγραμματισμού, η διαχείριση κατάστασης, η ανοχή σε σφάλματα, και οι εγγυήσεις επεξεργασίας μηνυμάτων.

Η μελέτη προσφέρει μία εκτενή συγκριτική παρουσίαση των κυριότερων ανοιχτού κώδικα συστημάτων επεξεργασίας ροών, όπως τα Apache Storm, Apache Flink, Apache Spark Streaming, Kafka Streams και IBM Streams, εστιάζοντας τόσο στα θεωρητικά μοντέλα λειτουργίας τους όσο και στις πρακτικές τους επιδόσεις. Μέσα από αυτήν την ανάλυση, οι συγγραφείς καταλήγουν ότι, αν και τα περισσότερα σύγχρονα συστήματα συγκλίνουν προς την ενοποίηση των προσεγγίσεων batch και stream processing, εξακολουθούν να διαφέρουν σημαντικά ως προς την αρχιτεκτονική παραλληλισμού, το επίπεδο ανοχής σε σφάλματα και τη στρατηγική διαχείρισης κατάστασης. Επίσης, η μελέτη τονίζει τη σημασία της επιλογής του κατάλληλου πλαισίου με βάση το εκάστοτε πεδίο εφαρμογής, παρουσιάζοντας συγκεκριμένα παραδείγματα από τομείς όπως η ανάλυση κοινωνικών δικτύων, η παρακολούθηση αισθητήρων και η επεξεργασία ειδησεογραφικών ροών σε πραγματικό χρόνο.

Η συγκεκριμένη εργασία σχετίζεται άμεσα με τη θεματική της παρούσας διπλωματικής, καθώς προσφέρει ένα θεμελιώδες υπόβαθρο αναφοράς για την αξιολόγηση των πλαισίων επεξεργασίας ροών δεδομένων, παρέχοντας το θεωρητικό πλαίσιο πάνω στο οποίο μπορεί να βασιστεί η πειραματική ανάλυση. Ωστόσο, ενώ η παραπάνω μελέτη επικεντρώνεται στην ταξινομική και θεωρητική σύγκριση των συστημάτων, η παρούσα εργασία προχωρά ένα βήμα παραπέρα, επιχειρώντας να εξετάσει εμπειρικά την απόδοση και τη συμπεριφορά ορισμένων από τα ίδια συστήματα, όπως των Apache Flink και Spark Structured Streaming, μέσα από μετρήσιμες παραμέτρους όπως η καθυστέρηση και η διαμέτρηση. Με αυτόν τον τρόπο, η διπλωματική εργασία συμπληρώνει τη συνθετική προσέγγιση της παραπάνω μελέτης συνεισφέροντας με πειραματική επικύρωση και πρακτική ανάλυση των αποτελεσμάτων που η προαναφερθείσα εργασία ανέδειξε σε θεωρητικό επίπεδο.

Βιβλιογραφία

[1] T. Sawicki, "Navigating stateful stream processing." `https://quix.io/blog/navigating-stateful-stream-processing`, Mar. 2024. Accessed: 2024-10-25.

[2] M. V. Bordin, D. Griebler, G. Mencagli, C. F. Geyer, and L. G. L. Fernandes, "Dspbench: A suite of benchmark applications for distributed data stream processing systems," *IEEE Access*, vol. 8, pp. 222900–222917, 2020.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[4] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *IEEE Data Engineering Bulletin*, vol. 38, no. 4, pp. 28–38, 2015.

[6] M. Armbrust, T. Das, R. S. Xin, M. Zaharia, *et al.*, "Structured streaming: A declarative api for real-time applications in apache spark," in *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, 2018.

[7] "Apache flink documentation." `https://nightlies.apache.org/flink/`, 2025. Accessed: 2025-06.

[8] "Apache spark structured streaming documentation." `https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html`, 2025. Accessed: 2025-06.

[9] Wikipedia contributors, "Heartbeat (computing) — wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/Heartbeat_(computing)`, 2025. Accessed: 2025-06-20.

[10] Wikipedia contributors, "Gossip protocol — wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/Gossip_protocol`, 2025. Accessed: 2025-06-20.

[11] Apache Flink, "Event time and watermarks." `https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/time/`, 2023.

[12] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, *et al.*, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pp. 1789–1792, IEEE, 2016.

[13] M. Fragkoulis, P. Carbone, V. Kalavri, and A. Katsifodimos, "A survey on the evolution of stream processing systems," *The VLDB Journal*, vol. 33, no. 2, pp. 507–541, 2024.

[14] G. Van Dongen and D. Van den Poel, "Evaluation of stream processing frameworks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1845–1858, 2020.

[15] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," *IEEE Access*, vol. 7, pp. 154300–154316, 2019.

[16] S. W. Kaisler and A. J. Sillitti, *Stream Processing with Apache Flink*. Springer, 2021.