ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Μελέτη επίδοσης συστημάτων διαχείρισης δεδομένων μεγάλης κλίμακας σε υπολογιστικά νέφη

Ταξιάρχης Κούσκουρας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Ιωάννης Κωνσταντίνου
Επίκουρος Καθηγητής

Λαμία 22/3 έτος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Μελέτη επίδοσης συστημάτων διαχείρισης δεδομένων μεγάλης κλίμακας σε υπολογιστικά νέφη

Ταξιάρχης Κούσκουρας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Ιωάννης Κωνσταντίνου
Επίκουρος Καθηγητής

Λαμία 22/3 έτος 2021

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Research on the performance of large-scale data management systems in cloud computing

Taxiarchis Kouskouras

FINAL THESIS

ADVISOR

Ioannis Konstantinou
Assistant professor

Lamia 22/3 year 2021

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 22/3/2021

Ο Δηλών

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Τα δεδομένα μεγάλου όγκου αποτελούν καθημερινότητα πολλών επιχειρήσεων και οργανισμών. Για την αποθήκευση και την ανάλυση τους έχουν δημιουργηθεί διάφορα υπολογιστικά πλαίσια. Η αποδοτική εκτέλεση ερωτημάτων πάνω στα δεδομένα και η εξαγωγή σημαντικών σχέσεων από τα δεδομένα που παράγονται αποτελεί κύριο στόχο αυτών των υπολογιστικών πλαισίων. Η μείωση του χρόνου εκτέλεσης των ερωτημάτων βασίζεται πάνω σε βελτιστοποίησής στο λογικό επίπεδο , στο φυσικό επίπεδο αλλά και από τον αλγόριθμο Join που θα επιλεχθεί για κάθε σχέση. Σε αυτήν την πτυχιακή εργασία θα εξετάσουμε δυο δημοφιλείς βελτιστοποιητές ερωτημάτων που χρησιμοποιούνται από μεγάλες εταιρίες όπως η Amazon, η Alibaba, Το Facebook κλπ. . Το πρώτο από τα δυο υπολογιστικά πλαίσια που θα μελετήσουμε είναι αυτό του Apache Spark με τον βελτιστοποιητή ερωτημάτων Catalyst και το δεύτερο είναι το Apache Hive με τον βελτιστοποιητή ερωτημάτων Calcite. Κάθε ένας βελτιστοποιητής ερωτημάτων διαθέτει δικούς του κανόνες και μηχανισμούς, με τους οποίους επιλέγει διαφορετικούς αλγόριθμους Join. Ακόμα μελετάμε τα πλάνα που παράγουν οι δυο μηχανές για την συστοιχία υπολογιστών μας σε διάφορα ερωτήματα χρησιμοποιώντας το γνωστό Benchmark TPC-DS αντιπροσωπευτικά για συστήματα μεγάλου όγκου δεδομένων και υποστήριξης αποφάσεων. Τέλος εξετάζουμε γιατί το υπολογιστικό πλαίσιο Apache Spark και ο βελτιστοποιητής Catalyst εκτέλεσαν τα ερωτήματα αυτά γρηγορότερα στην συστοιχία υπολογιστών που δημιουργήσαμε.

ABSTRACT

High volume data is a daily occurrence for many companies and organizations. Various computing frameworks have been created for their storage and analysis. The efficient execution of queries and the extraction of important relationships from the data generated is the main goal of these computing frameworks. The reduction of query execution time is based on optimization at the logical and physical level but also by the Join algorithm that will be selected for each relationship. In this thesis we will look at two popular query optimizers used by large companies such as Amazon, Alibaba, Facebook, etc. The first of the two computing frameworks we will study is that of Apache Spark with the Catalyst query optimizer and the second is Apache Hive with the Calcite query optimizer. Each query optimizer has its own rules and mechanisms, with which it selects different Join algorithms. We will study the plans produced by the two frameworks for our cluster of computers in various queries using the well-known Benchmark TPC-DS representative for large data and decision support systems. Finally, we look at why the Apache Spark computing framework and the Catalyst optimizer executed these queries faster on the cluster of computers we created.

Table of Contents

Table of Figures

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

Κίνητρο 1.1

Καθημερινά περισσότερες επιχειρήσεις διαλέγουν το διαδίκτυο για να επιχειρήσουν, περισσότερες συσκευές συνδέονται στο διαδίκτυο και παράγουν δεδομένα ,όλο και πιο πολλοί χρήστες χρησιμοποιούν ιστοσελίδες που αποθηκεύουν δεδομένα σχετικά με την συμπεριφορά τους . Οι ιστοσελίδες κοινωνικής δικτύωσης και τα ψηφιακά μαγαζιά παίζουν μεγάλο ρόλο σε αυτήν την εκτίναξη της παραγωγής δεδομένων, πράγμα που έχει ωθήσει τους επιστήμονες που ασχολούνται με δεδομένα μεγάλου όγκου να δημιουργήσουν καινούργια μεγέθη μέτρησης δεδομένων, καινούργιους αλγόριθμους αποθήκευσης και εξερεύνησης μεγάλου όγκου δεδομένων και τρόπους να αποθηκεύσουν και να εξάγουν χρήσιμες πληροφορίες μέσα από αυτά. Τα δεδομένα και οι σχέσεις ή τα αποτελέσματα που παράγονται από αυτά θεωρούνται πολύ σημαντικά αφού μέσα από αυτά μπορούν να δημιουργηθούν μοντέλα πρόβλεψης . Τέτοια μοντέλα είναι πολύ σημαντικά για τις επιχειρήσεις αλλά και για άλλους οργανισμούς αφού μπορούν να προβλέψουν τάσεις και συμπεριφορές. Πάνω στην έρευνα στον τομέα των δεδομένων μεγάλου όγκου ερευνητές ανέπτυξαν διάφορα υπολογιστικά πλαίσια για την καλύτερη και γρηγορότερη εκτέλεση ερωτημάτων πάνω σε τέτοια δεδομένα. Από αυτά τα υπολογιστικά πλαίσια άλλα ειδικεύονται στον τρόπο αποθήκευσης των δεδομένων σε κατανεμημένα συστήματα όπως μια συστοιχία υπολογιστών, άλλα στην εκτέλεση μεγάλων εργασιών πάνω σε κατανεμημένα δεδομένα έτσι ώστε να μειωθεί ο χρόνος που χρειάζεται για την περάτωση τους και άλλα ειδικεύονται στην βελτιστοποίηση ερωτημάτων έτσι ώστε να γίνουν οι κατάλληλοι μετασχηματισμοί των ερωτημάτων για πιο αποδοτικές εκτελέσεις. Μέσω τον υπολογιστικών αυτών πλαισίων και των βελτιστοποιητών ερωτημάτων παράγονται διάφορα πλάνα εκτέλεσής. Αυτά τα πλάνα εκτέλεσης δρούν τόσο σε λογικό επίπεδο, με συγκεκριμένες βελτιστοποιήσεις σε συγκεκριμένες εργασίες που έχουν μελετηθεί και έχει αποδεικτή ότι επιφέρουν καλύτερους χρόνους εκτέλεσης αλλά και σε φυσικό επίπεδο όπως με τον διαμοιρασμό και την αποθήκευση επαναλαμβανόμενων εργασιών και την επιλογή αλγόριθμου Join. Η διαδικασία της επιλογής αλγορίθμου Join αποτελεί ένα σημαντικό κομμάτι στην μείωση του χρόνου εκτέλεσης . Διαφορετικοί αλγόριθμοι Join προκαλούν διαφορετικοί χρήση των πόρων της συστοιχίας υπολογιστών έτσι με την χρήση πιο αποδοτικών αλγορίθμων Join μειώνεται η χρήση του δικτύου που είναι από τους πιο σημαντικούς πόρους σε μια συστοιχία υπολογιστών για την εκτέλεση μιας εργασίας. Οι πιο γνωστοί βελτιστοποιητές ερωτημάτων είναι ο Catalyst του Apache Spark και ο Apache Calcite που χρησιμοποιείται από το Apache Hive. Οι δύο αυτοί βελτιστοποιητές ερωτημάτων είναι διαφορετικοί και πολλές φορές προτείνουν διαφορετικά σενάρια εκτέλεσης για το ίδιο ερώτημα. Για να μετρηθεί η αποδοτικότητα αυτών των υπολογιστικών πλαισίων σε συστοιχίες υπολογιστών έχουν δημιουργηθεί από διάφορους οργανισμούς ειδικά Benchmarks, τα οποία προσφέρουν δεδομένα και ερωτήματα κατάλληλα για συστήματα μεγάλου όγκου δεδομένων. Τα Benchmark αυτά είναι φτιαγμένα έτσι ώστε να εξετάζουν διαφορετικές πτυχές ενός συστήματος έτσι ώστε να μπορεί να ελεγχθεί η αποτελεσματικότητα ενός συστήματος με δεδομένα και ερωτήματα που προσομοιώνουν εργασίες που συμβαίνουν στον πραγματικό κόσμο. Αξίζει να μελετηθεί η συμπεριφορά των δυο αυτών βελτιστοποιητών ερωτημάτων πάνω στα ίδια δεδομένα με σκοπό να δούμε την απόδοση των δυο υπολογιστικών πλαισίων για την συστοιχία υπολογιστών μας.

Συνεισφορά 1.2

Κατά την διάρκεια της πτυχιακής εργασίας μελετήθηκε η συμπεριφορά δυο δημοφιλών υπολογιστικών πλαισίων και των βελτιστοποιητών τους σε διαφορετικούς τύπους δεδομένων και ερωτημάτων.

- Με την χρήση του γνωστού και αξιόπιστου Benchmark υποστήριξης λήψης αποφάσεων TPC-DS για συστήματα μεγάλης κλίμακας δεδομένων δημιουργήθηκαν με την χρήση του εργαλείου dsdgen τα κατάλληλα δεδομένα για τους πίνακες του benchmark.
- Επίσης με την χρήση του TPC-DS και του εργαλείου dsqgen δημιουργήθηκαν τα κατάλληλα ερωτήματα για τα δεδομένα που παραχθήκαν
- Δημιουργήθηκαν 3 εικονικές μηχανές στο υπολογιστικό νέφος του δικτυού Okeanos
- Για κάθε μια από τις εικονικές μηχανές εγκαταστάθηκε το λογισμικό Apache Spark, Apache Hive και Apache Hadoop και έγιναν οι κατάλληλες παραμετροποιήσεις για τα τεχνικά χαρακτηριστικά της συστοιχίας υπολογιστών μας
- Με την χρήση του κατανεμημένου συστήματος αποθήκευσης δεδομένων του υπολογιστικού πλαισιού Apache Hadoop το HDFS διαμοιραστήκαν στην συστοιχία των υπολογιστών τα δεδομένα του TPC-DS που παράχθηκαν
- Με την χρήση του υπολογιστικού πλαισιού Apache Spark τα δεδομένα μετατράπηκαν στην μορφή Parquet και αποθηκεύτηκαν οι πίνακες στο κατανεμημένο σύστημα αρχείων HDFS
- Επιλέχθηκαν 4 ερωτήματα από τα ερωτήματα που παράχθηκαν αντιπροσωπευτικά για τις τέσσερις κατηγορίες του Benchmark TPC-DS
- Με την χρήση του υπολογιστικού πλαισιού Apache Spark εκτελέστηκαν τα ερωτήματα που επιλέχθηκαν και παράχθηκαν τα φυσικά και λογικά πλάνα για κάθε ερώτημα
- Με την χρήση του υπολογιστικού πλαισιού Apache Hive εκτελέστηκαν τα ερωτήματα που επιλέχθηκαν και παράχθηκαν τα φυσικά πλάνα για κάθε ερώτημα καθώς η μηχανή Apache Hive δεν εμφανίζει στον χρήστη το λογικό πλάνο που ακολούθησε
- Για κάθε ένα από τα ερωτήματα και με την βοήθεια των φυσικών πλάνων συγκρίναμε τους διαφορετικούς αλγορίθμους Join καθώς και τις διαφοροποίησης στα φυσικά πλάνα που παρήγαγαν τα δυο υπολογιστικών πλαισίων
- Καταλήξαμε βάση των μετρήσεων ότι για την συστοιχία υπολογιστών μας ότι το υπολογιστικό πλαίσιο Apache Spark ήταν γρηγορότερο στην εκτέλεση των ερωτημάτων από το Apache Hive
- Τέλος προσπαθήσαμε να εξηγήσουμε και να μελετήσουμε ποιοι παράγοντες έπαιξαν ρόλο στις γρηγορότερες εκτελέσεις του υπολογιστικού πλαισίου Apache Spark.

ΚΕΦΑΛΑΙΟ 2 Βιβλιογραφική Επισκόπηση

Εισαγωγή 2.1

Λόγο της στροφής της κοινωνίας αλλά και των επιχειρήσεων στον ψηφιακό κόσμο, τόσο πιο πολύ μεγαλώνει και η σχεδίαση συστημάτων μεγάλης κλίμακας δεδομένων έτσι ώστε να μπορούν να υποστηριχτούν αυτές οι εργασίες. Οι ερευνητές στον κλάδο των δεδομένων μεγάλου όγκου και των συστημάτων διαχείρισης αυτού του είδους των δεδομένων προσπαθούν να δημιουργήσουν νέα υπολογιστικά πλαίσια, νέα Benchamark για τον έλεγχο αυτών των συστημάτων υπο συγκεκριμένο φόρτο εργσίας και να δημιουργήσουν αποδοτικότερα συστήματα για την αποθήκευση και την εκτέλεση εργασιών σε δεδομένα μεγάλου όγκου.

Σχετικές εργασίες 2.2

Λόγο του όγκου των δεδομένων που παράγονται με περισσότερα δεδομένα να παράγονται κάθε μέρα και την ανάγκη αυτά τα δεδομένα να επεξεργαστούν όσο πιο γρήγορα γίνεται οι ερευνητές στον τομέα των δεδομένων μεγάλου όγκου σχεδιάζουν συστήματα ανάλυσης δεδομένων. Ερευνητές αποφάσισαν να συγκρίνουν την αποτελεσματικότητα δυο ευρέως γνωστά συστήματα εκτέλεσης ερωτημάτων SQL πάνω στο υπολογιστικό πλαίσιο Hadoop[1] . Τα δυο αυτά συστήματα είναι το Apache Hive και το Impala. Τα πειράματα που εκτελέστηκαν στην συστοιχία υπολογιστών χρησιμοποιήσαν δεδομένα αποθηκευμένα στις μορφές Parquet και ORC και χρησιμοποιήθηκε το Benchmark TPC-H του οργανισμού TPC αλλά και δυο φορτία εργασιών παρόμοια με το Benchmark TPC-DS του οργανισμού TPC. Ακόμη γίναν και πειράματα για την απόδοση εισόδου/εξόδου στην αποθήκευση αρχείων βάση στήλης. Από τα πειραματικά αποτελέσματα οι ερευνητές βγάλαν τα εξής συμπεράσματα:

- Για τα πειράματα που χρησιμοποιήσαν το Benchmark TPC-H βρέθηκε ότι το σύστημα Impala είναι γρηγορότερο 3,3 με 4,4 φορές από το Hive με μηχανή εκτέλεσης το MapReduce του Hadoop και 2,1 με 2,8 γρηγορότερο από το Hive με μηχανή εκτέλεσης το Tez.
- Για τα πειράματα που χρησιμοποιήσαν φόρτο εργασίας ανάλογο με αυτό του TPC-DS το Impala ήταν γρηγορότερο κατά 8,2 με 10 φορές από το Hive χρησιμοποιώντας ως μηχανή το MapReduce του Hadoop ενώ ήταν και γρηγορότερο κατά 4,3 φορές από το Hive με μηχανή εκτέλεσης του Tez

Όσο αναφορά τα συστήματα ανάλυσης δεδομένων έχουν δημιουργηθεί συστήματα ανάλυσης από διαφορετική σκοπιά. Η πρώτη κατηγορία είναι γενικού σκοπού ενώ η δεύτερη αποτελείται από συστήματα ειδικού σκοπού. Σε άλλη έρευνα οι ερευνητές δημιουργώντας μια συστοιχία υπολογιστών αποφάσισαν να εκτελέσουν μια σειρά από πειράματα μεταξύ των δυο κατηγοριών των συστημάτων ανάλυσης για να συγκρίνουν την αποδοτικότητα των δυο συστημάτων υπό τον φόρτο δεδομένων μεγάλου όγκου με την χρήση συνθετικών δεοδμένων αλλά και πραγματικών [2]. Έτσι χρησιμοποιώντας το υπολογιστικό πλαίσιο Apache Spark που αποτελεί ένα σύστημα γενικού σκοπού , αποφάσισαν να χρησιμοποιήσουν την λειτουργία μηχανικής εκμάθησης του Spark (Spark MLlib) σε σύγκριση με το σύστημα

ειδικού σκοπού μηχανικής μάθησης TensorFlow της Google υποβάλλοντας τα συστήματα να εκτελέσουν γνωστούς αλγόριθμου μηχανικής μάθησης πάνω σε δεδομένα . Επίσης αποφάσισαν αν συγκρίνουν την εκτέλεση ερωτημάτων SQL με το σύστημα γενικού σκοπού Apache Spark χρησιμοποιώντας την Spark-SQL με τις μηχανές ειδικού σκοπού εκτέλεσης ερωτημάτων SQL Apache Hive και Presto.Βάση των πειραματικών αποτελεσμάτων οι ερευνητές κατέληξαν στα εξής αποτελέσματα:

- το TensorFlow αποδείχτηκε υπολογιστικά αποδοτικότερο από ότι η μηχανή Spark ενώ η Spark αποδείχτηκε ότι έχει καλύτερη απόδοση ανάγνωσης των δεδομένων σε σχέση με το TensorFlow. Τα αποτελέσματα αυτά αποδόθηκαν στην διαφορά των αρχιτεκτονικών των δυο συστημάτων
- Στην σύγκριση των υπολογιστικών πλαισίων Apache Spark SQL, Apache Hive και Presto πάνω σε ερωτήματα SQL. Τα δεδομένα και τα ερωτήματα δημιουργήθηκαν χρησιμοποιώντας το benchmark TPC-H του οργανισμού TPC. Οι ερευνητές κατέληξαν ότι η Spark-SQL και η Hive έχουν παρόμοια αποτελέσματα καθ' όλη την διάρκεια του πειράματος
- Η Spark και η Hive είναι γρηγορότερη κατά μέσο όρο 1,2 φορές από την Presto αν και το υπολογιστικό πλαίσιο Presto βασίζεται σε μια MPP (Massively Parallel Processing) αρχιτεκτονική.

Έτσι βασιζόμενοι σε αυτές τις έρευνες αποφασίσαμε να συγκρίνουμε την απόδοση αι την συμπεριφορά των δυο υπολογιστικών πλαισίων Apache Spark και Apache Hive σε μια μικρή συστοιχία υπολογιστών χρησιμοποιώντας το γνωστό benchmark TPC-DS

ΚΕΦΑΛΑΙΟ 3 Το TPC-DS Benchmark

Το TPC-DS Benchmark 3.1

Το Transaction Performance Council 3.1.α

Το Transaction Performance Council είναι ένας ανεξάρτητος μη-κερδοσκοπικός οργανισμός, που σκοπό έχει την δημιουργία αξιόπιστων benchmarks με τα οποία μπορούν να υποβληθούν σε έλεγχο βάσης δεδομένων και συστήματα μεγάλου όγκου δεδομένων. Πολλές μεγάλες εταιρίες αποτελούν μέλη του TPC όπως η Intel, η IBM, η Microsoft κλπ.

Ιδιότητες και σκοπός του Benchmark 3.1.β

Το TPC-DS Είναι ένα benchmark για συστήματα υποστήριξης αποφάσεων και έχει φτιαχτεί για συστήματα τα οποία οι κατασκευαστές του χαρακτήρισαν ότι διαθέτουν τις εξής ιδιότητες[3]:

- Εξετάζουν δεδομένα σε μεγάλο όγκο
- Καλούνται να απαντήσουν σε πραγματικές επιχειρηματικές ερωτήσεις
- Εκτελούν ερωτήματα που έχουν διαφορετική πολυπλοκότητα και απαιτήσεις
- Χαρακτηρίζονται από υψηλό φόρτο στην CPU και ανάγκη για συνεχή επικοινωνία μέσω του δικτύου
- Περιοδικά συγχρονίζονται με OLTP βάσεις δεδομένων μέσο των συναρτήσεων συντήρησης δεδομένων.
- Τρέχουν σε συστήματα διαχείρισης μεγάλου όγκου δεδομένων όπως είναι οι RDBMS αλλά και τα συστήματα που βασίζονται στο Spark/Hadoop

Σκοπός του TPC-DS είναι να προσφέρει δεδομένα και ερωτήματα που συναντούμε στον πραγματικό κόσμο για τέτοιου είδους συστήματα.

Το μοντέλο του TPC-DS 3.2

Το μοντέλο που χρησιμοποιεί και προσομοιώνει το benchmark του TPC-DS είναι αυτό μιας μεγάλης επιχείρησης λιανεμπορίου. Η επιχείρηση έχει πολλά καταστήματα και προσφέρει επιλογές αγοράς τόσο από φυσικά καταστήματα όσο και επιλογές αγοράς από το διαδίκτυο. Ακόμα το benchmark περιέχει πίνακες που αναπαριστούν κλασσικές εργασίες μιας τέτοιας επιχείρησης όπως είναι οι αγορές και οι επιστροφές αλλά και συστήματα καταγραφής εμπορευμάτων και προώθησης.

Ένα καλό σύστημα υποστήριξης αποφάσεων πρέπει να διαθέτει διαφορετικά ερωτήματα που έχουν διαφορετικό σκοπό και φόρτο. Έτσι το TPC-DS περιλαμβάνει ερωτήματα που ανήκουν σε τέσσερις κατηγορίες.

- Reporting ερωτήματα
- Ad hoc ερωτήματα
- Iterative OLAP ερωτήματα
- Data mining ερωτήματα

Τα Reporting ερωτήματα είναι δημιουργημένα έτσι ώστε να απαντάνε σε γνωστές και προκαθορισμένες ερωτήσεις που αφορούν την οικονομική υγεία της επιχείρησης ,χαρακτηρίζονται από στατικότητα.

Τα Ad hoc ερωτήματα είναι φτιαγμένα έτσι ώστε το σύστημα καλείται να απαντήσει σε αυτοσχέδια ερωτήματα που στοχεύουν στην άμεση και συγκεκριμένη απάντηση επιχειρηματικών ερωτήσεων. Χαρακτηρίζονται από δυναμικότητα και η κύρια διαφορά που έχουν με τα Reporting ερωτήματα είναι η περιορισμένη γνώση που διαθέτει γι' αυτά ο διαχειριστής του συστήματος.

Τα Iterative OLAP ερωτήματα επιτρέπουν στο σύστημα να αναλύσει δεδομένα μέσα από τα οποία μπορούν να παραχθούν νέες τάσεις και σημαντικές σχέσεις. Η διαφορά τους με την κατηγορία των Ad hoc ερωτημάτων είναι ότι έχουν οδηγό την συμπεριφορά του χρήστη.

Τα Data mining ερωτήματα αποσκοπούν στην διαδικασία εξόρυξης γνώσης μέσα από δεδομένα μεγάλου όγκου. Οι σχέσεις που δημιουργούνται μέσα από την γνώση μπορούν να οδηγήσουν σε πρόβλεψη μελλοντικών τάσεων, συμπεριφορών και δίνουν την δυνατότητα σε επιχειρήσεις να πάρουν αποφάσεις που είναι στηριγμένες στην γνώση που προέρχεται μέσα από αυτό το είδος των ερωτημάτων. Τέτοιου είδους ερωτημάτων περιέχουν πολλά Joins και συναρτήσεις συνάθροισης (aggregation functions).

Το σχήμα του TPC-DS 3.3

Το Snowflake schema 3.3.α

Το σχήμα που ακολουθεί το TPC-DS Benchmark για την βάση δεδομένων του και τους πίνακες του είναι το Snowflake Schema που είναι μια παραλλαγή του Star schema. Το Snowflake schema είναι η λογική τοποθέτηση των πινάκων μιας βάσης δεδομένων και χρησιμοποιείται για την μοντελοποιήση αποθηκών δεδομένων (Data Warehouses) . Το σχήμα αυτό δημιουργήθηκε έχοντας ως στόχο την κανονικοποιήση (μείωση της επανάληψης κοινών πεδίων στους πίνακες που δημιουργεί πρόβλημα στην διαγραφή, επεξεργασία και ανανέωση εγγραφών) και έτσι μεγάλοι πίνακες διαιρούνται σε περισσότερους μικρούς πίνακες. Το σχήμα έχει ως κέντρο του τους fact πίνακες που περιέχουν την κύρια πληροφορία. Γύρω από κάθε fact πίνακα υπάρχουν οι dimension πίνακες που περιέχουν συμπληρωματικές πληροφορίες και συσχετίζονται με έναν ή περισσότερους fact πίνακες. Οι dimension πίνακες μπορεί να έχουν σχέση και με άλλους dimension πίνακες. Έτσι αν ενώσουμε με βέλη τους πίνακες που έχουν σχέση μεταξύ τους θα δημιουργηθούν σχήματα που μοιάζουν με χιονονιφάδες εξού και η ονομασία του σχήματος Snowflake schema.

Αναπαράσταση των fact πινάκων στου σχήματος 3.3.β

Για την αναπαράσταση των λειτουργειών πώλησης και επιστροφής το TPC-DS χρησιμοποιεί 3 βασικά κανάλια πωλήσεων το διαδίκτυο, τα φυσικά καταστήματα και τον κατάλογο. Για αυτά τα κανάλια χρησιμοποιούνται οι fact πίνακες Store_Sales, Store_Returns, Catalog_Sales, Catalog_Sales, Web_Sales, Web_Returns και Inventory μαζί με άλλους dimensional πίνακες. Το σχήμα της βάσης περιέχει 7 fact πίνακες και 17 dimensional πίνακες.

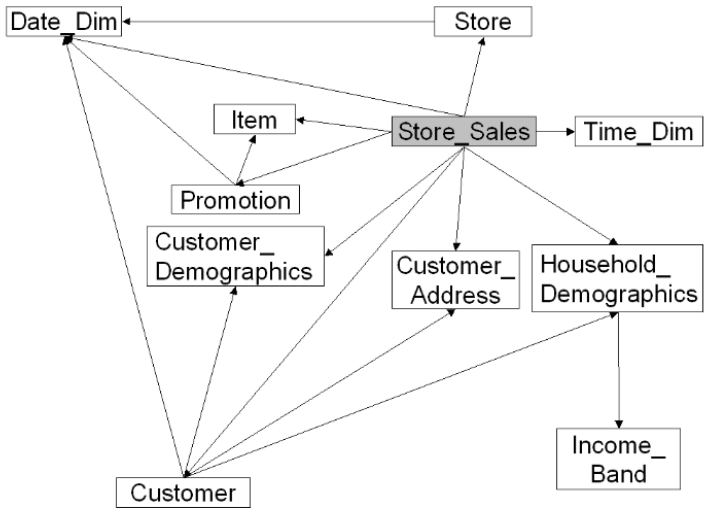

Figure 1 αναπαράσταση του fact πίνακα Store_Sales και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 1 φαίνεται η σχέση του fact πίνακα Store_Sales με τους 10 dimension πίνακες που χρησιμοποιεί και η σχέση αυτών των πινάκων μεταξύ τους

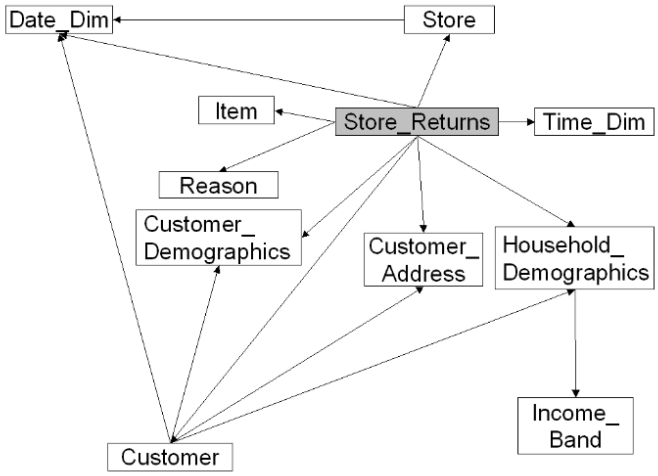

Figure 2 αναπαράσταση του fact πίνακα Store_Returns και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 2 φαίνεται η σχέση του fact πίνακα Store_Returns με τους 10 dimension πίνακες που χρησιμοποίει και η σχέση αυτών των πινάκων μεταξύ τους

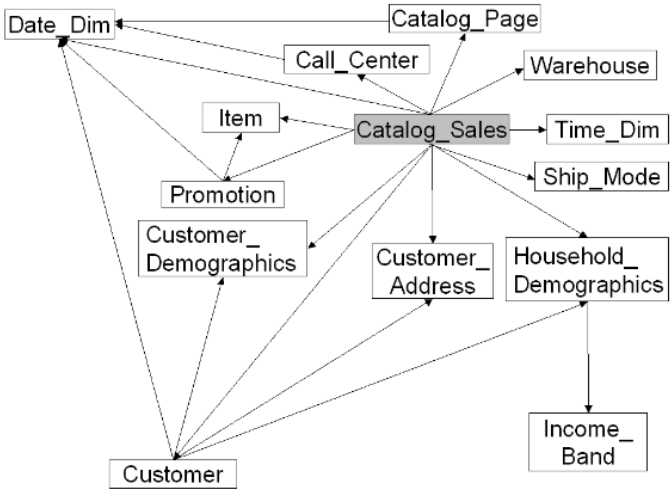

Figure 3 αναπαράσταση του fact πίνακα Catalog_Sales και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 3 φαίνεται η σχέση του fact πίνακα Catalog_Sales με τους 13 dimension πίνακες που χρησιμοποιεί και η σχέση αυτών των πινάκων μεταξύ τους

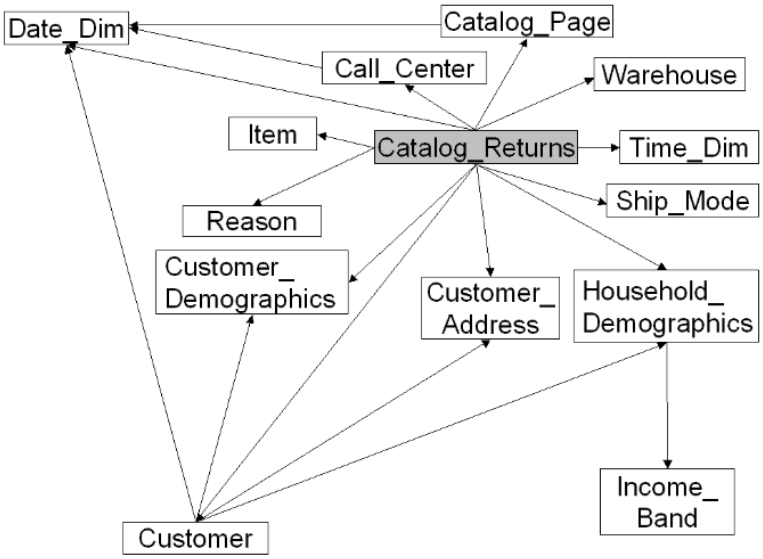

Figure 4 αναπαράσταση του fact πίνακα Catalog_Returns και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 4 φαίνεται η σχέση του fact πίνακα Catalog_Returns με τους 13 dimension πίνακες που χρησιμοποιεί και η σχέση αυτών των πινάκων μεταξύ τους

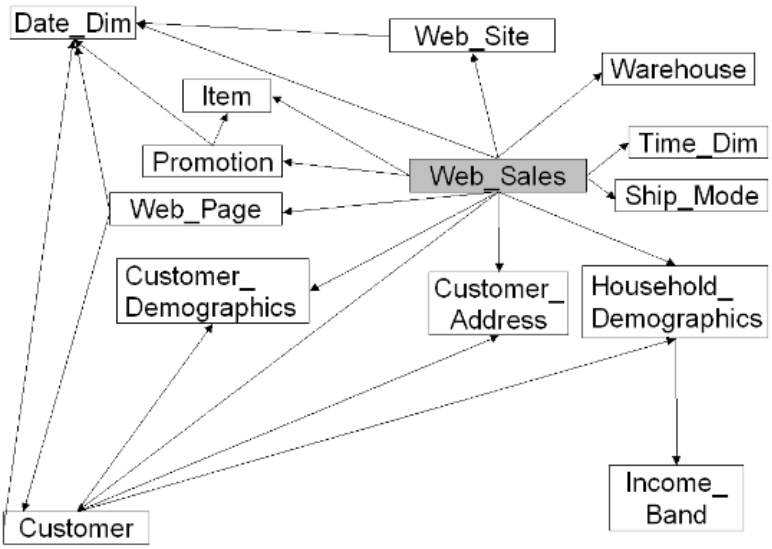

Figure 5 αναπαράσταση του fact πίνακα Web_Sales και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 5 φαίνεται η σχέση του fact πίνακα Web_Sales με τους 13 dimension πίνακες που χρησιμοποιεί και η σχέση αυτών των πινάκων μεταξύ τους

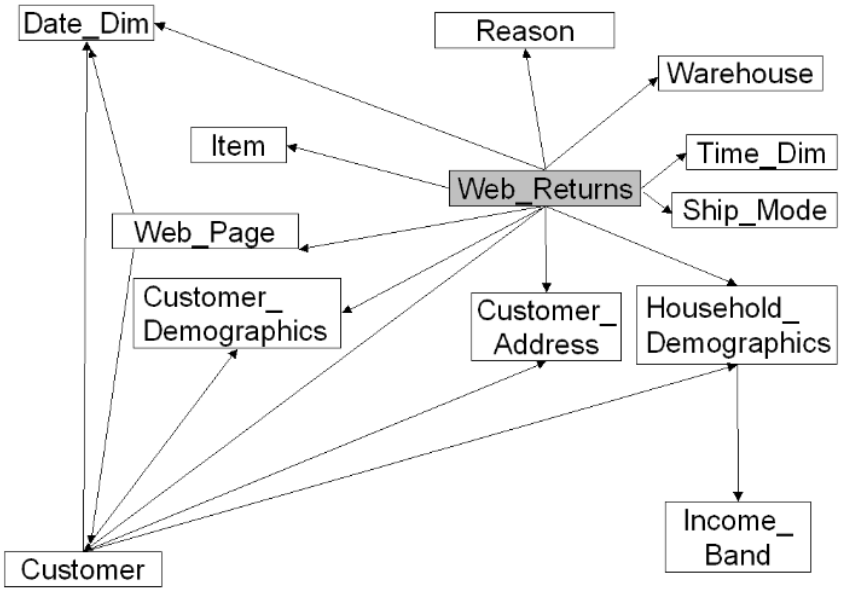

Figure 6 αναπαράσταση του fact πίνακα Web_Returns και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 6 φαίνεται η σχέση του fact πίνακα Web_Returns με τους 12 dimension πίνακες που χρησιμοποιεί και η σχέση αυτών των πινάκων μεταξύ τους

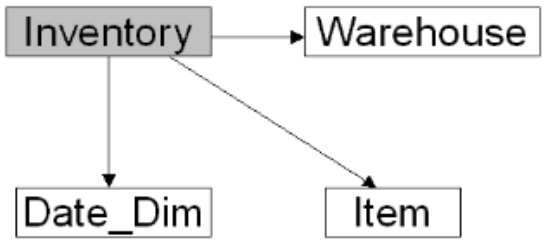

Figure 7 αναπαράσταση του fact πίνακα Inventory και των dimension πινάκων που χρησιμοποιεί από το documentation του TPC-DS http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf

Στην Εικόνα 7 φαίνεται η σχέση του fact πίνακα Inventory με τους 3 dimension πίνακες που χρησιμοποιεί

ΚΕΦΑΛΑΙΟ 4 Τα Υπολογιστικά πλαίσια

Υπόβαθρο 4.1

Το προγραμματιστικό μοντέλο Map/Reduce 4.1.α

Το προγραμματιστικό μοντέλο Map/Reduce είναι ένα προγραμματιστικό μοντέλο που επιτρέπει την παράλληλη εκτέλεση μιας εργασίας μέσα σε μια συστοιχία υπολογιστών. Αρχικά υλοποιήθηκε από την Google [4] και έπειτα δημιουργήθηκαν και άλλα παρόμοια ανοιχτού κώδικα έργα όπως το Apache Hadoop. Με την χρήση του μοντέλου Map/Reduce μπορούμε να εκτελέσουμε μια εργασία μεγάλου όγκου δεδομένων σε λιγότερό χρόνο . Η διαδικασία εκτέλεσης γίνεται σε δύο βασικά στάδια το Map και το Reduce. Ο Προγραμματιστής καλείται να γράψει τον κώδικα για αυτά τα δύο στάδια ξεκινώντας από το στάδιο του Map που παίρνει ως είσοδο ένα ζευγάρι τιμών <κλειδί, τιμή> και παράγει ως έξοδο ένα ενδιάμεσο συνδυασμό τιμών <κλειδί, τιμή> ,έπειτα το στάδιο Reduce καλείται να συγκεντρώσει τα δεδομένα από τα μηχανήματα που εκτέλεσαν το στάδιο του Map και να εκτελέσει την τελική διεργασία πάνω τους έτσι ώστε να έχουμε το τελικό αποτέλεσμα. Ακόμα όπως και οι Mappers μπορούν να υπάρχουν πολύ Reducers λόγο του μεγάλου μεγέθους τέτοιων εργασιών. Τέλος υπάρχει ένας υπολογιστής Master που επιβλέπει και κατευθύνει τους υπόλοιπους υπολογιστές Workers προς την περάτωση της εργασίας.

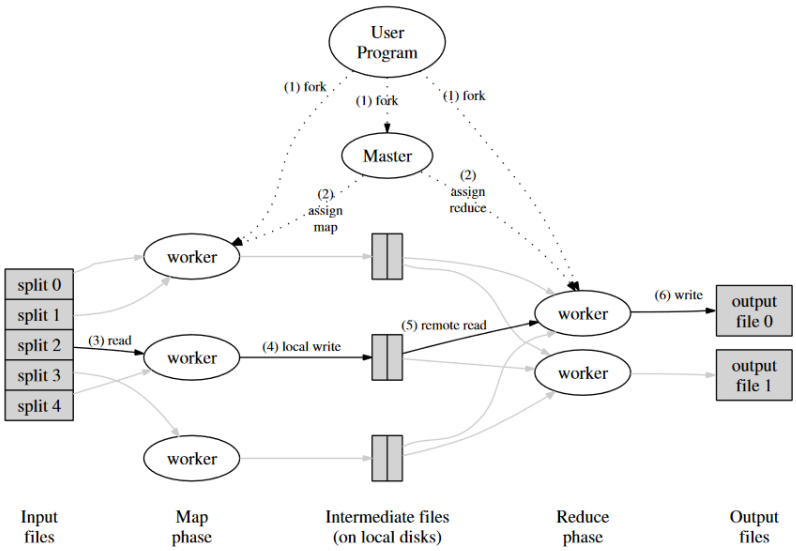

Figure 8 Αναπαράσταση μιας εργασίας Map/Reduce σε έναν Cluster . Εικόνα από το paper της Google MapReduce: Simplified Data Processing on Large Clusters

Στην Εικόνα 8 μπορούμε να δούμε την οπτική αναπαράσταση μιας εργασίας Map/Reduce μέσα σε μια συστοιχία υπολογιστών. Αρχικά ο υπολογιστής Master υποβάλει τον κώδικα που έγραψε ο προγραμματιστής προς εκτέλεση και ορίζει ποιοι από τους υπολογιστές Workers θα έχουν τον ρόλο του Mapper και ποιοι του Reducer. Για τους υπολογιστές που έχουν τον ρόλο του Mapper ο Master ορίζει ποια κομμάτια των δεδομένων τους αναλογούν για να εκτελέσουν την εργασία που τους έχει ανατεθεί. Έπειτα οι Workers εκτελούν την εργασία και αποθηκεύουν τα αποτελέσματα σε buffers από τους οποίους τα δεδομένα θα μπορούν να διαβαστούν από τους υπολογιστές με τον ρόλο του Reducer , ο Master ενημερώνεται για την θέση αυτών των buffer. Μετά για τους υπολογιστές που έχουν τον ρόλο του Reducer ο Master τους ενημερώνει για το που θα βρουν τα δεδομένα των Mappers και φροντίζει να στείλει τα δεδομένα με κοινά κλειδιά στον ίδιο Reducer. Τέλος οι Reducer εκτελούν την εργασία που τους έχει ανατεθεί παράγουν το τελικό αποτέλεσμα και ενημερώνουν τον Master. Σε περίπτωση που κάποιο μηχάνημα υποστεί βλάβη τα δεδομένα και η εργασία του ανατίθενται σε άλλον Mapper ή Reducer.

Το κατανεμημένο σύστημα αρχείων HDFS 4.1.β

Το HDFS αποτελεί το κατανεμημένο σύστημα αποθήκευσης του υπολογιστικού πλαισιού Apache Hadoop.Είναι βασισμένο στο κατανεμημένο σύστημα αποθήκευσης της Google GFS .Οι σχεδιαστές του HDFS δώσαν ιδιαίτερο βάρος στην διαθεσιμότητα των δεδομένων και την ανεκτικότητα στα σφάλματα. Με την χρήση διάφορων μηχανισμών όπως το σύστημα αναπαραγωγής (Replication system) κάθε κομμάτι (Block) των δεδομένων αντιγράφεται σε παραπάνω από ένα κόμβους (Node) της συστοιχίας υπολογιστών (Cluster). Το υπολογιστικό πλαίσιο δημιουργήθηκε έτσι ώστε λειτουργεί πάνω σε φτηνό υλικό (Hardware) που είναι ευρέως διαθέσιμο στην αγορά και με την χρήση της γλώσσας Java στην οποία έχει προγραμματιστεί μπορεί να εκτελέσει εργασίες σε μηχανήματα διαφορετικά μεταξύ τους (πχ που έχουν διαφορετικό λογισμικό) έτσι αποτελεί μια ελκυστική και προσιτή λύση για πολλούς οργανισμούς και εταιρίες. Κάθε αρχείο που εισάγεται στο σύστημα χωρίζεται σε κομμάτια (Blocks) τα οποία διανέμονται στα διάφορα μηχανήματα. Το HDFS ακολουθεί την γνωστή αρχιτεκτονική Master-Slave, καθώς υπάρχει ένα μηχάνημα Name που έχει ηγετικές υποχρεώσεις και πολλά μηχανήματα DataNodes τα οποία ακολουθούν εντολές που τους ανατίθενται. Ο NameNode διαχειρίζεται τα αρχεία του γενικού συστήματος αρχείων και εκτελεί πάνω τους εργασίες όπως άνοιγμα, κλείσιμο, μετονομασία αρχείων και υπο-καταλόγων , διαιρεί τα αρχεία σε κομμάτια και αποφασίζει σε ποιος DataNode θα αποθηκευτούν και διαχειρίζεται την πρόσβαση των πελατών (Clients) στα αρχεία . Οι DataNodes διαχειρίζονται το τοπικό τους σύστημα αρχείων και εκτελούν διάφορες εργασίες αρχείων όπως δημιουργία block ,διαγραφή block και επιτρέπουν σε πελάτες να γράψουν και να διαβάσουν αρχεία πάντα με την εντολή του NameNode. Ακόμη οι δημιουργοί του HDFS δράσανε με το σκεπτικό «Η μετακίνηση των υπολογισμών είναι φθηνότερη από την μετακίνηση των δεδομένων» έτσι το υπολογιστικό πλαίσιο έχει σχεδιαστεί να προσπαθεί να αναθέσει εργασίες σε μηχανήματα που διαθέτουν τα κατάλληλα δεδομένα για την περάτωση της. Τέλος εκτός από την εκτέλεση εργασιών Map/Reduce από το ίδιο το Apache Hadoop άλλα υπολογιστικά πλαίσια μπορούν να χρησιμοποιήσουν μόνο το κατανεμημένο σύστημα αποθήκευσης HDFS του υπολογιστικού πλαισίου και να παρέχουν τις δικές τους μηχανές εκτέλεσης Map/Reduce εργασιών. Ένα τέτοιο παράδειγμα είναι η υλοποίηση που υιοθετήσαμε με το Apache Spark ως μηχανή εκτέλεσης εργασιών και το HDFS ως το σύστημα κατανεμημένων αρχείων.

Αλγόριθμοι Join 4.1.γ

Για την εκτέλεση κατανεμημένων εργασιών Join δημιουργήθηκαν από ερευνητές ειδικοί αλγόριθμοι Join κατάλληλη για διαφορετικές περιπτώσεις. Σημαντικό ρόλος παίζει το μέγεθος των πινάκων και η ικανότητα της συστοιχίας υπολογιστών να αποθηκεύσουν στην μνήμη κάποιων από τους πίνακες . Ιδιαίτερη προσπάθεια καταβάλλεται έτσι ώστε να χρησιμοποιείται το δίκτυο όσο το δυνατόν λιγότερο γίνεται αφού αποτελεί τον πιο «ακριβό» πόρο σε μια συστοιχία υπολογιστών. Έτσι σε αυτό το υπο-κεφάλαιο θα μελετήσουμε τους πιο σημαντικούς από τους αλγορίθμους Join [5].

Ο αλγόριθμος Repartition Join είναι ο αλγόριθμος που χρησιμοποιείται περισσότερο από όλους τους αλγόριθμους Join για κατανεμημένα συστήματα. Ο Repartition Join παρομοιάζεται με τον αλγόριθμο sort-merge join των RDBMS βάσεων. Ο αλγόριθμος μπορεί να εκτελεστεί σε μια εργασία Map/Reduce. Σε κάθε κομμάτι των δεδομένων η κάθε εργασία Map προσθέτει στο κομμάτι των δεδομένων του μια «ταμπέλα» (tag) έτσι ώστε να γνωρίζει το υπολογιστικό πλαίσιο από ποιόν πίνακα προέρχονται τα δεδομένα. Έπειτα τα δεδομένα εξάγονται βάση του κλειδιού στο οποίο γίνονται Join στην μορφή (κλειδί, τιμή) . Μετά τα εξαγόμενα δεδομένα χωρίζονται, ταξινομούνται και ενώνονται από το υπολογιστικό πλαίσιο που εκτελεί τις εργασίες Map/Reduce. Έπειτα το υπολογιστικό πλαίσιο φροντίζει τα ζευγάρια δεδομένων με κοινό κλειδί Join να αποστέλλονται στους ίδιους Reducer. Για κάθε κλειδί Join το υπολογιστικό πλαίσιο φορτώνει σε δυο διαφορετικούς buffers τα δεδομένα των πινάκων ανάλογα με τον πίνακα από το οποίον προέρχεται. Τέλος συνενώνει τις εγγραφές.

Ο αλγόριθμος Broadcast Join βασίζεται στο ότι ένας από των δυο πινάκων είναι μικρότερος από τον άλλον και επίσης είναι αρκετά μικρός έτσι ώστε να μπορεί να φορτωθεί στην κύρια μνήμη. Με αυτό τον τρόπο ο μικρότερος πίνακας μπορεί να μεταφερθεί σε όλους του κόμβους της συστοιχίας υπολογιστών και έτσι δεν χρειάζεται να μεταφερθούν και οι δυο πίνακες μέσω του δικτύου αποφεύγοντας την διαδικασία ταξινόμησης και για τους δυο πίνακες. Ο αλγόριθμος μπορεί να εκτελεστεί σε μια εργασία Map. Έτσι Αρχικά σε κάθε Map εργασία αρχικά ελέγχεται εάν υπάρχει ο πίνακας που έχει επιλεχθεί για να διαμοιραστεί στο τοπικό σύστημα αποθήκευσης. Εάν δεν υπάρχει τότε η ζητάει τον πίνακα και τον αποθηκεύει στο τοπικό της σύστημα αρχείων. Έπειτα δημιουργεί δυναμικά τον πίνακα Hash πάνω στα δεδομένα του μικρού πίνακα που επιλέχθηκαν για να διαμοιραστούν ή στο κομμάτι των δεδομένων του μεγάλου πίνακα που αναλογεί στην εργασία Map ανάλογα με το ποιο από τα δύο έχει μικρότερο μέγεθος. Έπειτα εξάγεται μια λίστα με τα κλειδιά Join του πίνακα και με την χρήση του πίνακα Hash συνενώνονται οι εγγραφές.

Ο αλγόριθμος Semi-Join επιλέγεται όταν λόγο του μεγάλου μεγέθους ενός από των δυο πινάκων πολλές εγγραφές του μεγάλου πίνακα δεν έχουν κοινά κλειδιά με τον μικρότερο πίνακα. Μια εργασία Join που ακολουθεί τον αλγόριθμο Semi-Join εκτελείται σε τρία στάδια. Το πρώτο στάδιο του αλγορίθμου αποτελείται από μία ολοκληρωμένη Map/Reduce εργασία στο στάδιο του Map κάθε εργασία Map δημιουργεί για κάθε κομμάτι του μικρότερου πίνακα που του αναλογεί έναν πίνακα Hash με τα μοναδικά κλειδιά Join και στο στάδιο Reduce ενώνει όλα τα κλειδιά σε ένα αρχείο που είναι αρκετά μικρό έτσι ώστε να μπορεί να φορτωθεί στην μνήμη . Έπειτα το επόμενο στάδιο είναι ένα στάδιο Map το οποίο μοιάζει στον αλγόριθμο Broadcast Join φορτώνει το αρχείο που παράχθηκε στο πρώτο στάδιο στην μνήμη και διατρέχοντας τα κομμάτια δεδομένων στα οποία κάθε Map εργασία κατέχει εξάγει τα δεδομένα με κοινό Join κλειδί σε ένα αρχείο. Με το πέρας του σταδίου Map υπάρχει μια λίστα από τέτοια αρχεία. Στο τελευταίο στάδιο οι εγγραφές με κοινό κλειδί Join συνενώνονται χρησιμοποιώντας τον αλγόριθμο Broadcast Join.

Ο αλγόριθμος Per-Split Semi-Join διαθέτει τρία στάδια. Το πρώτο και το τρίτο στάδιο αποτελούνται μόνο από εργασίες Map ενώ το δεύτερο από ολοκληρωμένες εργασίες Map/Reduce. Στο πρώτο στάδιο του αλγορίθμου ο μικρότερος από τους δυο πίνακες χωρίζεται σε κομμάτια και ανατίθεται σε μια εργασία Map μέσω της οποίας παράγονται αρχεία με τα μοναδικά κλειδιά Join κάθε κομματιού δεδομένων του πίνακα. Στο δεύτερο στάδιο το Map κομμάτι της εργασίας Map/Reduce φορτώνει στην κύρια μνήμη κομμάτια δεδομένων από τον μεγαλύτερο πίνακα και ελέγχει κάθε αρχείο από το προηγούμενο στάδιο του αλγορίθμου για τυχόν κοινά κλειδιά Join. Έπειτα βάζει μια ταμπέλα για κάθε εγγραφή που βρίσκει κοινό κλειδί Join έτσι ώστε να χρησιμοποιηθεί από την εργασία Reduce του σταδίου και να μαζέψει όλα τα κομμάτια του μεγάλου πίνακα που θα ενωθούν με ένα κομμάτι δεδομένων του μικρού πίνακα. Στο τελευταίο στάδιο οι εγγραφές συνενώνονται.

Για κάθε έναν από αυτούς του αλγορίθμους υπάρχουν τρόποι με τους οποίους μπορεί να βελτιωθεί η απόδοση τους χρησιμοποιώντας κάποιες τεχνικές προ-υπολογισμού κάποιου κομματιού του αλγορίθμου έτσι ώστε να εκτελούνται οι εργασίες Join γρηγορότερα.

Apache Spark 4.2

Η δημιουργία του Apache Spark 4.2.α

Το Apache Spark είναι μια γρήγορη ενοποιημένη μηχανή ανάλυσης και επεξεργασία δεδομένων μεγάλου όγκου που προσφέρει και δυνατότητες μηχανικής μάθησης. Η δημιουργία του ξεκίνησε στα εργαστήρια RADLab (σήμερα RISELab) στο Πανεπιστήμιο Berkley από τον Matei Zaharia και άλλους ερευνητές , αργότερα ο βασικός του κώδικας έγινε δωρεά στο Apache software foundation όπου διατηρείται και ανανεώνεται έως και σήμερα ως έργο ανοιχτού κώδικα. Ο λόγος της δημιουργίας της μηχανής Spark ήταν όταν ερευνητές του πανεπιστήμιού Berkley έχοντας δουλέψει σε ένα παρόμοιο έργο ανοιχτού κώδικα το Apache Hadoop MapReduce παρατήρησαν την δυσκολία εκμάθησης του, καθώς και την αναποτελεσματικότητα του σε συγκεκριμένες εργασίες όπως οι επαναλαμβανόμενες διεργασίες αλλά και οι διαδραστικές εργασίες. Έτσι αποφάσισαν να δημιουργήσουν ένα υπολογιστικό πλαίσιο που θα ήταν εύκολο στην εκμάθηση , απλούστερο και γρηγορότερο. Ακόμα και στα αρχικά στάδια της μηχανής όπως φαίνεται και από τις πρώτες δημοσιεύσεις η μηχανή Spark είναι 10 με 20 φορές γρηγορότερο από μηχανή Hadoop για ορισμένες εργασίες. Σήμερα αυτός ο αριθμός έχει μεγαλώσει υπολογίζοντας ότι η μηχανή Spark είναι περίπου 100 φορές γρηγορότερη στην μνήμη και 10 φορές γρηγορότερη στον δίσκο από την μηχανή Hadoop.

Πως δουλεύει το Apache Spark 4.2.β

Το Apache Spark είναι μια μηχανή cluster computing [6] η οποία ακολουθεί την αρχιτεκτονική του Master-Worker. Υπάρχει ένα μηχάνημα στην συστοιχία υπολογιστών που έχει τον ρόλο του Master και πολλά μηχανήματα με τον ρόλο του Worker. Αρχικά το πρόγραμμα που έχει γράψει ο προγραμματιστής υποβάλλεται στο μηχάνημα με τον ρόλο του Master. Έπειτα ο Master εκτελεί το πρόγραμμα και δημιουργεί το SparkContext και με αυτό αποστέλλει τον κώδικα στα άλλα μηχανήματα της συστοιχίας υπολογιστών. Όπως φαίνεται στην Εικόνα 9 υπάρχει ένας cluster manager οποίος ορίζει τους πόρους που αναλογούν σε κάθε εφαρμογή (η Spark διαθέτει τον δικό της cluster manager αλλά η μηχανή είναι έτσι φτιαγμένη ώστε να αγνοεί ποιος cluster manager χρησιμοποιείται με αποτέλεσμα να μπορεί να χρησιμοποιεί και άλλους cluster manager όπως ο Mesos, ο Yarn κλπ.). Ο κόμβος από τον οποίον υποβλήθηκε ο κώδικας (Driver program) επιτελεί χρέη Master και είναι υπεύθυνος για να σχεδιάσει τις εργασίες Map/Reduce , να τις αναθέσει στους Workers και να επιβλέπει την εκτέλεση τους. Οι εργασίες αυτές αναπαριστώνται με έναν μηχανισμό που ονομάζεται DAG (Directed Acyclic Graph ένας κατευθυνόμενος ακυκλικός Γράφος όπου οι κόμβοι είναι RDDs και οι ακμές είναι οι εργασίες που εφαρμόζονται πάνω τους.). Έπειτα ο DAGScheduler με βάση τον Γράφο DAG εξετάζει σε ποιόν Worker μια εργασία θα χρησιμοποιήσει τους λιγότερους πόρους (πχ διαθέτει ήδη τα αρχεία που χρειάζονται για μια εργασία και δεν θα χρειαστεί να χρησιμοποίηση το δίκτυο) και έπειτα χωρίζει αυτήν την εργασία σε στάδια Map και σε στάδια Reduce. Έπειτα ο TaskScheduler μαζί με την βοήθεια του cluster manager αναθέτουν την κάθε εργασία στον Worker που αποφάσισε ο DAGScheduler.Ο TaskScheduler έχει την ευθύνη ελέγχου των εργασιών και αποκατάστασης πιθανών βλαβών που θα προκύψουν. Οι Workers εκτελούν τις εργασίες που τους ανατίθενται με την βοήθεια των Executors οι οποίοι διαθέτουν κομμάτι από τος τοπικούς πόρους του μηχανήματος Worker (CPU, μνήμη κλπ.)

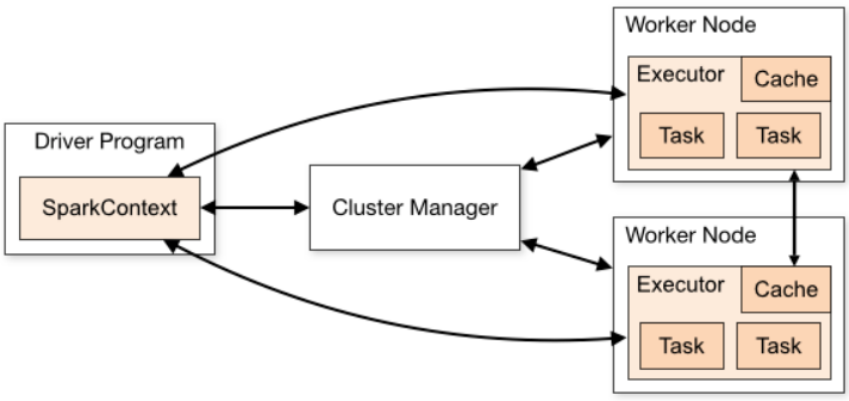

Figure 9 Αναπαράσταση εκτέλεσης μιας εφαρμογής σε έναν cluster του Apache Spark. Εικόνα από την ιστοσελίδα του framework Apache Spark https://spark.apache.org/docs/latest/cluster-overview.html

Το δομικό υλικό της Apache Spark 4.2.γ

Το βασικό δομικό υλικό του apache spark είναι τα RDD. Τα resilient distributed dataset ή RDD αποτελούν τον τρόπο με το οποίον το Apache Spark απεικονίζει τα δεδομένα έτσι ώστε να εκτελεί παράλληλες εργασίες. Ουσιαστικά είναι συλλογές αντικειμένων που έχουν αποθηκευτεί κατανεμημένα σε ένα σύνολο υπολογιστών , σε αυτά δεν επιτρέπεται η επεξεργασία ή η αλλαγή παρά μόνο η ανάγνωση. Δεν χρειάζεται η φυσική αποθήκευση κάποιου RDD αφού ,αποθηκεύεται η ακολουθία των εργασιών που χρειάζεται για να φτιαχτεί κάποιο συγκεκριμένο RDD σε ένα αρχείο που ονομάζεται handler. Έτσι σε περίπτωση απώλειας κάποιου RDD ή κάποιου κόμβου του συστήματος η μηχανή Spark συμβουλεύεται το αρχείο handler του RDD για να το δημιουργήσει ξανά. Τέλος αξίζει να σημειωθεί ότι τα RDD χαρακτηρίζονται από τους δημιουργούς τους ως «βαριεστημένα» και «εφήμερα» [7] αφού δημιουργούνται μόνο όταν είναι πραγματική ανάγκη σε κάποιο παράλληλο υπολογισμό και όχι πιο πριν πχ σε κάποια εργασία που ενεργοποιεί την εκτέλεση μια εργασίας όπως πχ μια εργασία Count ή Collect (εργασίες που χρειάζονται τα δεδομένα για να εκτελεστούν)και έτσι λόγο αυτής της καθυστέρησης στην δημιουργία τους μπορούν να εφαρμοστούν πάνω τους απλές βελτιστοποιήσεις. Ακόμη προσφέρονται συναρτήσεις που αποθηκευτούν RDD στην μνήμη (πχ με την λειτουργία cache()) ή στο δίσκο όταν η μνήμη μας δεν είναι αρκετή έτσι ώστε να μην ξανά χρειαστεί να υπολογιστούν από την αρχή

Η Spark-SQL 4.2.δ

Η Spark-SQL είναι το μοντέλο που χρησιμοποιεί η Apache Spark για να επεξεργαστεί δομημένα δεδομένα. Η Spark-SQL μας δίνει την ικανότητα να εκτελέσουμε λειτουργίες SQL σε διαμοιρασμένα δεδομένα, αποθηκευμένα σε εξωτερικές πηγές όπως HDFS κ.α. σε διάφορες μορφές όπως Parquet, Avro, Json κ.α. . Εσωτερικά στην Spark-SQL προσφέρονται πληροφορίες τόσο για τα δεδομένα όσο και για τους υπολογισμούς που πραγματοποιούνται ώστε να μπορούν να γίνουν κάποιες βελτιστοποιήσεις σε αυτούς .Ακόμα μας δίνεται η δυνατότητα να ενσωματώσουμε το Hive μαζί με το spark-SQL έτσι ώστε να μπορούν να εκτελεστούν ερωτήματα σε δεδομένα που είναι αποθηκευμένα σε πίνακες Hive. Η δημιουργοί της Spark-SQL έθεσαν τους εξής στόχους για την δημιουργία της [7]

- Υποστηρίζει σχεσιακή επεξεργασία τόσο στα RDDs όσο και σε δεδομένα από εξωτερικές πηγές
- Προσφέρει υψηλή απόδοση χρησιμοποιώντας εδραιωμένες τεχνικές DBMS (Database management systems)
- Υποστηρίζει ημι-δομημένα δεδομένα, νέες πηγές δεδομένων και εξωτερικές βάσεις δεδομένων
- Επιτρέπει την επέκταση χρησιμοποιώντας εξελιγμένους αλγόριθμους ανάλυσης όπως επεξεργασίας γράφων και μηχανική μάθηση

Η Spark-SQL προσφέρει μια αναβάθμιση τον RDD τα Dataframes. Τα Dataframes έχουν βάση τα RDD για αυτό μοιράζονται πολλές ιδιότητες τους. Η διαφορά των Dataframes με τα RDD είναι ότι τα Dataframes είναι μια κατανεμημένη συλλογή σειρών και έχουν γνώση επί του σχήματος τους. Έτσι τα Dataframes είναι στην Spark ότι οι πίνακες σε μια σχεσιακή βάση δεδομένων. Λόγο της κατασκευής τους τα Dataframes υπάγονται σε βελτιστοποίησης που κάνουν την εκτέλεση ενός ερωτήματος αποδοτικότερη.

Ο βελτιστοποιητής ερωτημάτων Catalyst 4.2.ε

Ο βελτιστοποιητής ερωτημάτων Catalyst [7] δημιουργήθηκε μαζί με την Spark-SQL έτσι ώστε να προσφέρει βελτιστοποιήσεις στα ερωτήματα που υποβάλλονται για εκτέλεση και έτσι να κάνει την Spark-SQL πιο αποδοτική. Δημιουργήθηκε με την χρήση της γλώσσας scala και χρησιμοποιώντας στοιχεία της γλώσσας κάνει πολύ εύκολη την διαδικασία επέκτασης του Catalyst από προγραμματιστές προσθέτοντας νέους τύπους , νέους κανόνες κτλ. . Η μηχανή βελτιστοποιήσης ερωτημάτων Catalyst προσφέρει βελτιώσεις τόσο βάση κανόνων (Rule based optimization) μεταθέτοντας και μετασχηματίζοντας κανόνες , όσο και βελτιστοποιήσεις βάση του κόστους (Cost based optimization) δημιουργώντας διαφορετικά πλάνα για το ίδιο ερώτημα και επιλέγοντας στο τέλος το λιγότερο κοστοβόρο. Το βασικό δομικό υλικό του Catalyst είναι τα δέντρα , έτσι οι δημιουργοί παρέχουν μια βασική βιβλιοθήκη αναπαράστασης των εργασιών ενός ερωτήματος ως δέντρα και διάφορες ενέργειες που επιτρέπουν την επιβολή κανόνων σε αυτά. Ακόμα προσφέρονται βιβλιοθήκες που εστιάζουν στην επεξεργασία σχεσιακών ερωτημάτων και ένα σύνολο από σετ κανόνων που εφαρμόζονται στα διάφορα στάδια της εκτέλεσης του ερωτήματος. Όπως φαίνεται και στην Εικόνα 10 ένα δέντρο στον Catalyst αποτελείται από αντικείμενα κόμβων (node) που έχουν τύπο και ένα ή περισσότερα παιδιά. Τα αντικείμενα αυτά είναι αμετάβλητα και παρέχονται ειδικές συναρτήσεις για να τα επεξεργαστούμε.

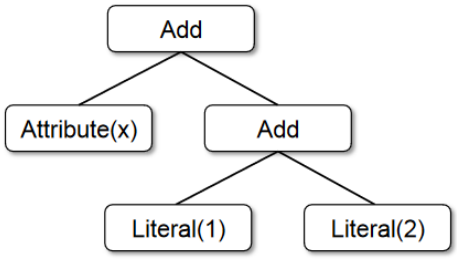

Figure 10 Απεικόνιση ενός δέντρου στον Catalyst. Εικόνα από το paper Spark SQL: Relational Data Processing in Spark

Οι κανόνες είναι ο τρόπος που χρησιμοποιεί ο Catalyst για να επεξεργαστεί τα δέντρα έτσι ώστε να δημιουργήσει βελτιστοποιήσεις. Αυτό γίνεται χρησιμοποιώντας ένα σετ κανόνων που έχουν γραφτεί και την ιδιότητα pattern-matching της γλώσσας scala. Αναδρομικά κάθε δέντρο ελέγχεται για τον αν κάποιο κομμάτι του δέντρου του ταιριάζει με κάποιον κανόνα και μετασχηματίζεται στην μορφή που ο κανόνας αυτός προστάζει. Για παράδειγμα το δέντρο της Εικόνας 10 θα μπορούσε να μετασχηματιστεί στο παρακάτω δέντρο που απεικονίζει η Εικόνα 11 εάν υπήρχε κανόνας που έλεγε ότι:

case: add(literal(c1), literal(c2)) => literal(c1+c2)

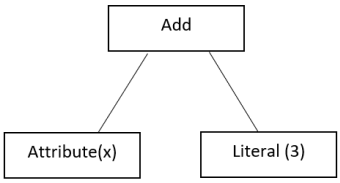

Figure 11 Αναπαράσταση του νέου δέντρου στον Catalyst μετά τον μετασχηματισμό του

Τέλος Οι σχεδιαστές του Catalyst χωρίζουν την διαδικασία βελτιστοποίησης σε 4 στάδια. Αυτά τα στάδια είναι 1)Ανάλυση λογικών πλάνων για την επίλυση αναφορών 2)Βελτιστοποίηση λογικών πλάνων 3)Φυσική σχεδίαση 4)Παραγωγή κώδικα για να μεταγλωττιστούν κομμάτια του query σε java bytecode.

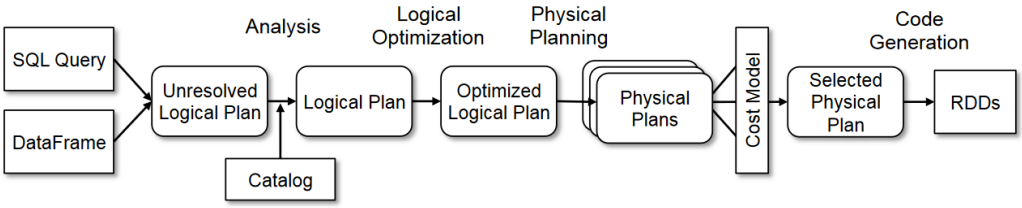

Figure 12 Αναπαράσταση των σταδίων βελτιστοποίησης ενός ερωτήματος στον Catalyst. Εικόνα από το paper Spark SQL: Relational Data Processing in Spark

Η Ανάλυση αποτελεί το βασικό και πρωταρχικό στάδιο της βελτιστοποίησης κάποιου ερωτήματος. Όπως μπορούμε να δούμε και στην Εικόνα 12 αρχικά η μηχανή βελτιστοποίησης δέχεται ως είσοδο ένα AST (Abstract syntax tree) από κάποιο SQL ερώτημα ή ένα Dataframe. Η είσοδος που δέχεται για ανάλυση ο βελτιστοποιητής περιέχει στοιχεία που δεν έχουν ελεγχθεί , έτσι ξεκινά από το unresolved logical plan (λογικό πλάνο στο οποίο δεν έχουν ακόμα ελεγχθεί αν τα στοιχεία υπάρχουν) και με την βοήθεια του στοιχείου του καταλόγου δημιουργεί το logical plan. Ο βελτιστοποιητής χρησιμοποιεί τον κατάλογο για να ελέγξει αν υπάρχουν συγκεκριμένες στήλες σε ένα πίνακα, τον τύπο αυτών των στηλών , να αλλάξει τον τύπο όπου χρειάζεται και να αποδώσει έναν μοναδικό αριθμό ID στα στοιχεία στηλών έτσι ώστε να αναγνωρίζει τα όμοια στοιχεία και να προχωρήσει σε βελτιστοποιήσεις.

Κατά την διάρκεια του σταδίου βελτιστοποίησης λογικού πλάνου (Logical optimization) η μηχανή ακολουθεί μια διαδικασία εφαρμογής κανόνων που έχουν μελετηθεί και αποτελούν στάνταρ κανόνες βελτιστοποίησής. Οι κανόνες αυτοί περιλαμβάνουν project pruning, predicate pushdowns , απλοποίηση Boolean εκφράσεων κ.α. . Έτσι από το λογικό πλάνο (logical plan) παράγεται το βελτιστοποιημένο λογικό πλάνο (Optimizer logical plan)

Για το στάδιο της φυσικής σχεδίασης (physical planning) ο βελτιστοποιητής Catalyst παράγει διαφορετικά πλάνα και βάσει ενός μοντέλου κόστους επιλέγει αυτό που έχει την μικρότερη τιμή. Με είσοδο το βελτιστοποιημένο λογικό πλάνο το στάδιο της φυσικής σχεδίασης εφαρμόζει και βελτιστοποιήσεις κόστους βάση κανόνων όπως pipelining projections ή συναρτήσεις filters σε εργασίες Map του Spark. Σε αυτό το στάδιο επιλέγεται και ο αλγόριθμος Join που θα εφαρμοστεί για το ερώτημα

Το τελευταίο στάδιο της διαδικασίας βελτιστοποίησης του Catalyst περιλαμβάνει το στάδιο παραγωγή κώδικας για να μετατραπούν κομμάτια του ερωτήματος σε java bytecode.

Apache Hive 4.3

Η δημιουργία του Apache Hive 4.3.α

Το Apache hive είναι ένα ανοιχτού κώδικα data warehouse έργο που προσφέρει την δυνατότητα εκτέλεσης ερωτημάτων και ανάλυσης σε δεδομένα που βρίσκονται κατανεμημένα σε διάφορα υπολογιστικά συστήματα. Το Hive δημιουργήθηκε από προγραμματιστές του facebook, όταν αποφάσισαν να χρησιμοποιήσουν το δημοφιλές υπολογιστικό πλαίσιο για κατανεμημένη επεξεργασία δεδομένων Apache Hadoop λόγο τον συνεχώς αυξανόμενο όγκο των δεδομένων που καλούνταν να επεξεργαστούν κάθε μέρα. Οι προγραμματιστές του facebook ανακάλυψαν ότι η συγγραφή εργασιών Map/Reduce που προσφέρει το Hadoop ακόμα και για τα πιο απλές εργασίες αποτελεί μια χρονοβόρα διαδικασία , ειδικά για προγραμματιστές που δεν είχαν εμπειρία με το προγραμματιστικό μοντέλο Map/Reduce . Έτσι αποφάσισαν να δημιουργήσουν ένα υπολογιστικό πλαίσιο που θα επιτρέπει να εκτελούνται SQL ερωτήματα πάνω σε κατανεμημένα δεδομένα μετατρέποντας τα στις κατάλληλες Map/Reduce εργασίες.

Η HiveQL και τρόποι αποθήκευσης δεδομένων στην Hive 4.3.β

Η HiveQL είναι η γλώσσα με την οποία η μηχανή Hive εκτελεί ερωτήματα σε κατανεμημένα δεδομένα. Έχει πολλές ομοιότητες με την SQL οπότε επιτρέπει σε προγραμματιστές που έχουν εμπειρία με αυτήν να την χρησιμοποιήσουν εύκολα. Παρόλες τις ομοιότητες με την κλασσική SQL η HiveQL προσφέρει καινούργιες δυνατότητες όπως είναι οι νέοι τύποι πέραν τον βασικών (int, float, string) πχ τα maps , τα lists και τα structs κλπ. . Επίσης προσφέρει και την δυνατότητα εκτελέσεις κώδικα με την βοήθεια άλλον γλωσσών υψηλού επιπέδου όπως η python

Οι τρόποι με τους οποίους το Hive αποθηκεύει δεδομένα είναι οι εξής[8]:

- Tables
- Partitions
- Buckets

Τα Tables συσχετίζονται ,μέσω τον metadata τους, λόγο της κατανεμημένης φύσης αυτών των συστημάτων με ένα μονοπάτι (path). Στην περίπτωση μας του συστήματος κατανεμημένης αποθήκευσης HDFS. Ένα table μπορεί να είναι partitioned ή όχι. Μέσω της HiveQL τα διαχειριζόμαστε όπως τα κλασσικά Tables των σχεσιακών βάσεων δεδομένων.

Τα Partitions είναι ένας τρόπος που προσφέρει η Hive για διάσπαση ενός πίνακα σε κομμάτια έτσι ώστε να γίνονται πιο γρήγορες οι εργασίες εκτέλεσης ερωτημάτων και τέλος πιο αποδοτικό το σύστημα. Ο διαχωρισμός αυτός σε κομμάτια γίνεται από τον χρήστη πάνω σε μια κοινή τιμή σε κάποια στήλη ή συνδυασμό στηλών πχ αν υπάρχει μια στήλη που αποθηκεύει την ώρα έστω hour , θα μπορούσαμε να κάνουμε partition όλες τις εγγραφές που έχουν κοινή ώρα. Έτσι ένα ερώτημα που ενδιαφέρεται συγκεκριμένα μόνο για τις εγγραφές που έχουν τιμή hour = 12 θα χρειαστεί να κοιτάξει μόνο το ανάλογο partition και όχι όλο το dataset που περιέχει πολλά δεδομένα άχρηστα για το συγκεκριμένο ερώτημα. Τα Partitions αποθηκεύονται σε υπό-καταλόγους του Table από το οποίο προέρχονται.

Τα Buckets είναι ένας ακόμα τρόπος διαχωρισμού των δεδομένων σε μικρότερα πιο διαχειρίσιμα κομμάτια. Αποτελεί ακόμα ένα μέτρο πέρα από τα Partitions, έτσι ώστε εάν υπάρχουν Partitions που έχουν συγκεντρώσει το μεγαλύτερο ποσοστό ή ένα μεγάλο ποσοστό δεδομένων σε σχέση με τα άλλα Partitions να μπορεί να χωριστεί εκ νέου σε μικρότερα κομμάτια. Η HiveQL δίνει την δυνατότητα επιλογής δεδομένων από συγκεκριμένο bucket ενός Partition

Η αρχιτεκτονική της Hive 4.3.γ

Τα βασικά στοιχεία μηχανισμό της αρχιτεκτονικής του Hive είναι[8]:

- Το Metastore
- Ο Driver
- Ο Query Compiler
- Η Execution engine

Το Metastore αποθηκεύει τα καίριά δεδομένα που χρειάζεται το hive για να εκτελέσει τα ερωτήματα. Ουσιαστικά εκτελεί καθήκοντα καταλόγου αφού αποθηκεύει πληροφορίες για τους πίνακες όπως τους τύπους των στηλών και τα ονόματα τους , τα Partitions , καθώς και τις τοποθεσίες στις οποίες είναι αποθηκευμένα στο HDFS. Λόγο της ταχύτητας που πρέπει να παρέχονται αυτές οι πληροφορίες στον μεταγλωττιστή επιλέχθηκε από τους προγραμματιστές του Hive το Metastore να φιλοξενηθεί σε μια κλασσική βάση RDBMS και όχι στο κατανεμημένο σύστημα αποθήκευσης. Τέλος για να προληφθεί το πρόβλημα ανταπόκρισής όσο τα δεδομένα μεγαλώνουν οι δημιουργεί του hive αποτρέπουν την επικοινωνία των Mappers και των Reducers με το Metastore , παρέχοντας τους από τον compiler ένα xml αρχείο που περιέχει όλες τις πληροφορίες που θα χρειαστούν κατά την εκτέλεση.

Ο Driver επιτηρεί και διευθύνει την εκτέλεση των εργασιών που υποβάλλονται καθ' όλη την διάρκεια εκτέλεσης τους. Ο Driver είναι υπεύθυνος για την διατήρηση στατιστικών που προκύπτουν από την εκτέλεση. Τέλος ο Driver υποβάλει το ερώτημα για μεταγλώττιση στον Query compiler.

Ο Query compiler τροφοδοτείται από τον Driver και αρχικά εκτελεί τους τυπικούς σημασιολογικούς ελέγχους. Έπειτα και σε περίπτωση μη σφάλματος με τις πληροφορίες που του παρέχει το Metastore παράγει το execution plan. Η διαδικασία μεταγλώττισής έχει ως εξής:

1) Ο Αναλυτής δημιουργεί ένα AST(Abstract syntax tree) για το ερώτημα

2) Το ερώτημα υποβάλλεται στους κλασσικούς σημασιολογικούς ελέγχους και ελέγχους τύπων. Με τις πληροφορίες που του δίνει το Metastore για τους πίνακες παράγει το λογικό πλάνο (logical plan) ,έπειτα ελέγχει για την συμβατότητα μεταξύ των τύπων στο ερώτημα που υποβλήθηκε. Για την μετατροπή του AST σε μια απεικόνιση ακυκλικού γράφου DAG ο compiler χρησιμοποιεί μια ενδιάμεση απεικόνιση με την χρήση του query block tree QB.

3) Το κομμάτι της βελτιστοποιήσης του Apache Hive ανατίθεται σε ένα άλλο υπολογιστικό πλαίσιο επεξεργασίας δεδομένων το Apache Calcite που με την βοήθεια του γίνονται βελτιστοποιήσεις με βάση το κόστος (Cost based optimization – CBO)

Το execution engine είναι η μηχανή που εκτελεί τις εργασίες Map/Reduce που παράγει ο μεταγλωττιστής . Στα αρχικά στάδια του Hive υποστηριζόταν το Apache Hadoop ,αλλά λόγο της ανάπτυξης υπολογιστικών πλαισίων που εκτελούν εργασίες πιο γρήγορα από το Hadoop οι προγραμματιστές ανέπτυξαν το hive έτσι ώστε να υποστηρίζει και άλλα υπολογιστικά πλαίσια όπως το Apache Tez και το Apache Spark για πιο αποδοτικές εκτελέσεις.

Ο βελτιστοποιητής ερωτημάτων Apache Calcite 4.3.δ

Ο Calcite ως ένα αυτόνομο υπολογιστικό πλαίσιο [9] διαθέτει αναλυτή (parser) και επικυρωτή (validator) για τον έλεγχο των ερωτημάτων. Από τον αναλυτή και τον επικυρωτή του Calcite παράγεται ένα δέντρο σχεσιακών τελεστών που με την σειρά του δίνεται ως είσοδο στο κομμάτι του Calcite που γίνεται η βελτιστοποίηση των ερωτημάτων. Έπειτα με την χρήση των Metadata που παρέχονται από τους Metadata Provides και την βοήθεια των Pluggable Rules που μπορεί να παρέχονται από κάποιο εξωτερικό υπολογιστικό πλαίσιο ο Calcite περνάει στο στάδιο της βελτιστοποίησης ερωτημάτων. Λόγο της ευελιξίας που παρέχεται από το υπολογιστικό πλαίσιο άλλα υπολογιστικά πλαίσια που έχουν δικούς τους αναλυτές και επικυρωτές μπορούν να χρησιμοποιήσουν την ίδια διαδικασία παραγωγής δέντρου σχεσιακών τελεστών έτσι ώστε να μπορέσουν να χρησιμοποιήσουν το κομμάτι βελτιστοποίησης ερωτημάτων του Calcite. Η Εικόνα 13 αποτελεί μια εικονική απεικόνιση της αρχιτεκτονικής του Calcite

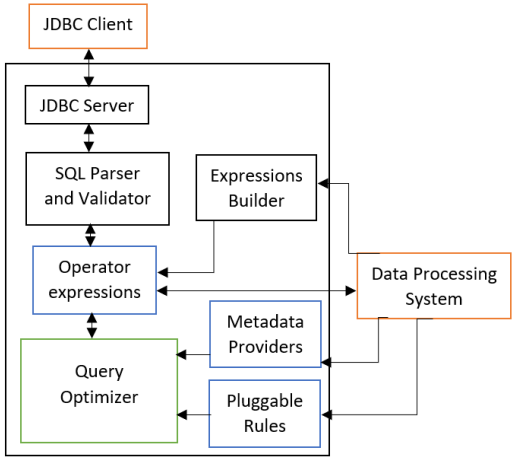

Figure 13 Αναπαράσταση της αρχιτεκτονικής του βελτιστοποιητή ερωτημάτων Calcite

Το κομμάτι του βελτιστοποιητή Αποτελεί το βασικό κομμάτι της αρχιτεκτονικής του Calcite. Δέχεται ως είσοδο ένα δέντρο σχεσιακών τελεστών τον τρόπο δηλαδή που χρησιμοποιεί ο Calcite για να αναπαραστήσει τα δεδομένα. Τα τρία στοιχεία του βελτιστοποιητή είναι οι κανόνες, Οι Metadata Providers και τα Planner Engines. Ο βελτιστοποιητής βελτιώνει τα ερωτήματα εφαρμόζοντας μια σειρά από κανόνες βελτιώσεις πάνω στα κομμάτια των δέντρων τους. Η όλη διαδικασία γίνεται υπό την επίβλεψη ενός μοντέλου που βασίζεται στο κόσοτς (Cost based) και με την βοήθεια του Planner engine παράγει ένα διαφορετικό πλάνο εκτελέσεις με την ίδια σημασία αλλά με λιγότερο κόστος έτσι ώστε να διαλεχθεί το λιγότερο κοστοβόρο. Τα κομμάτια του βελτιστοποιητή όπως η κανόνες κλπ. είναι επεκτάσιμα και μπορούν να προστεθούν καινούργιοι κανόνες , cost models, στατιστικά και σχεσιακοί τελεστές.

Στο στάδιο της εφαρμογής των κανόνων ο βελτιστοποιητής μέσα από ένα σετ κανόνων που διαθέτει ελέγχει το δέντρο για να βρει αν κάποιο υπό-δέντρο του ταιριάζει σε κάποιον κανόνα και να εφαρμόσει σε αυτό τον μετασχηματισμό που του αναλογεί. Αν και το Calcite διαθέτει το δικό του εκτενές σετ κανόνων πολλά υπολογιστικά πλαίσια προτιμούν να εισάγουν και να χρησιμοποιούν τα δικά τους σετ κανόνων. Πολύ από αυτούς τους

μετασχηματισμούς γίνονται σε διαφορετικά συστήματα γι' αυτό ο Calcite χρησιμοποιεί το concept των Adapters που ουσιαστικά υλοποιεί μια εργασία από το δικό του backend στο backend ενός άλλου υπολογιστικού πλαισιού.

Οι Metadata providers είναι ένα από τα κύρια στοιχεία του βελτιστοποιητή. Οι Metadata Providers δημιουργήθηκαν έτσι ώστε με την βοήθεια των πληροφοριών που παρέχουν να βοηθήσουν τον Planner να παράξει πλάνα με μικρότερο κόστος και να παρέχει πληροφορίες στους κανόνες για το δέντρο όταν αυτοί εφαρμόζονται. Αν και ο Calcite διαθέτει τους δικούς του Metadata Providers που προσφέρουν συναρτήσεις που μετράν το κόστος εκτέλεσης ενός υπό-δέντρου , των αριθμό των σειρών και το μέγεθος των δεδομένων μιας έκφρασης κ.α. επιτρέπει σε άλλα υπολογιστικά πλαίσια να συνδέσουν τους δικούς του Metadata Providers με τις δικές του συναρτήσεις.

Η Planner Engine έχει την ευθύνη να ενεργοποίηση τους κανόνες μέχρι να φτάσει σε ένα επιθυμητό επίπεδο. Η μηχανή διαθέτει 2 Planner Engines αν και ο Calcite επιτρέπει σε άλλα υπολογιστικά πλαίσια να συνδέσουν τις δικές του Planner Engines. Ο πρώτος Planner που παρέχει ο Calcite δρα με βάση την μείωση του κόστους. Χρησιμοποιώντας δυναμικό προγραμματισμό δημιουργεί και ελέγχει διάφορα πλάνα με την βοήθεια των κανόνων που του παρέχονται. Αρχικά κάθε έκφραση (expression) του δέντρου προστίθεται στον Planner καθώς και τα δεδομένα του μαζί με τα χαρακτηριστικά του. Όταν μια έκφραση έστω Ε1 μετασχηματίζεται στην έκφραση Ε2 βάση ενός κανόνα Κ ο Planner θα προσθέσει την νέα έκφραση σε ένα σύνολο εκφράσεων ισοδυναμίας έστω Σ στο οποίο υπήρχε ο Ε1. Ο Planner μετράει το κόστος κάθε έκφρασης και την συγκρίνει με τις άλλες εκφράσεις που έχουν προστεθεί σε αυτόν. Αν βρεθεί κάποια ομοιότητα στο πλάνο πχ της Ε2 με μια έκφραση Ε3 που βρίσκεται σε διαφορετικό σύνολο θα ενωθούν τα δυο σετ σε 1 αυτή η διαδικασία συνεχίζεται μέχρι να φτάσουμε ένα επιθυμητό επίπεδο. Ο πρώτος Planner διαθέτει δυο προσεγγίσεις με τις οποίες φτάνει στο επιθυμητό επίπεδο. Ο πρώτος είναι να εξερεύνησή όλων τον χώρο αναζήτησης μέχρι να εφαρμοστούν όλοι οι κανόνες σε όλες τις εκφράσεις και ο δεύτερος που ακολουθεί μια ευριστική προσέγγισή που σταματάει την προσπάθεια βελτίωσης όταν στην τελευταία αλλαγή από τον Planner η βελτίωση στο κόστος δεν περνάει ένα κατώτατο όριο έστω θ. Οι συναρτήσεις κόστους που επιτρέπουν στον Planner να αποφασίσει πιο πλάνο να επιλέξει παρέχονται από τους Metadata provider , αυτοί του Calcite παρέχουν υπολογισμό κόστους που παίρνει υπόψιν την χρήση της CPU, της μνήμης και την επικοινωνία του συστήματος με την χρήση του δικτύου για κάθε έκφραση. Ο δεύτερος Planner ακολουθεί μια προσέγγιση διεξοδικής εφαρμογής κανόνων μέχρι να μην υπάρχει κομμάτι που να μπορεί να γίνει μετατροπή αυτός ο δεύτερος Planner χρησιμοποιείται για γρήγορη εκτέλεση χωρίς να παίρνει υπόψιν το κόστος εκτέλεσης.

Διαδικασία χρήσης του βελτιστοποιητή Calcite από την μηχανή Hive 4.3.ε

Ο Hive χρησιμοποιεί τον βελτιστοποιητή ερωτημάτων Apache Calcite για την βελτιστοποίηση των ερωτημάτων και για να παράξει καλύτερα πλάνα παρέχοντας το δικό του Cost Based μοντέλο[10]. Το μοντέλο αυτό μετράει το κόστος βάση την χρήση της CPU, την επικοινωνία μέσο του δικτύου, το μέσο μέγεθος ενός tuple σε μια σχέση και την πληθικότητα. Τα δυο τελευταία χρησιμοποιούνται για να υπολογιστεί εάν μια σχέση μπορεί να φορτωθεί εξ ολοκλήρου στην κύρια μνήμη ή όχι έτσι ώστε να επιλεχθεί ο κατάλληλος αλγόριθμος Join. Για το μοντέλο κόστους της Hive δίνεται παραπάνω βάση η χρήση της CPU και η επικοινωνία μέσο του δικτύου. Τέλος η διαδικασία που ακολουθεί ο Hive είναι η εξής όπως μπορούμε να δούμε και στην Εικόνα 14:

1) Υποβάλλεται στο Hive ένα ερώτημα για εκτέλεση

2) Δημιουργείται ένα AST για το ερώτημα, γίνονται οι απαραίτητοι έλεγχοί του δέντρου και υποβάλετε στον Calcite για την διαδικασία βελτιστοποίησης

3) Ο Calcite από το δέντρο που λαμβάνει δημιουργεί ένα δέντρο σχεσιακών τελεστών

4) Ο βελτιστοποιητής ερωτημάτων παράγει ένα βελτιστοποιημένο σχεσιακό δέντρο και το στέλνει πίσω στο Hive

5) Το Hive μετατρέπει το δέντρο που δέχεται σε ένα δικό του σχεσιακό δέντρο

6) Το δέντρο υποβάλετε προς εκτέλεση στην μηχανή εκτέλεσης που έχει επιλεχθεί (Tez, Spark κλπ.) και δημιουργείται ο ακυκλικός Γράφος εκτέλεσης. Το ερώτημα εκτελείται από την μηχανή εκτέλεσης που έχει επιλεχθεί

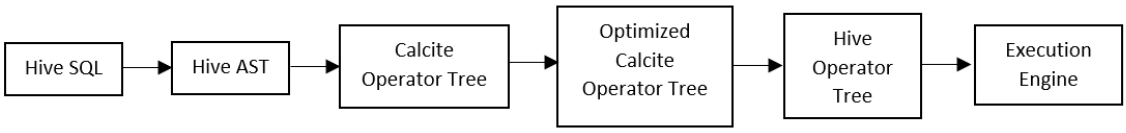

Figure 14 Αναπαράσταση της διαδικασίας εκτέλεσης και βελτιστοποίησης ενός ερωτήματος με την συνεργασία της μηχανής Hive και του βελτιστοποιητή Calcite

ΚΕΦΑΛΑΙΟ 5 Πειραματική αποτίμηση Spark και Hive

Εισαγωγή 5.1

Πρόλογος 5.1.α

Για να γίνει η πειραματική αποτίμηση των δυο υπολογιστικών πλαισίων επιλέχθηκαν τέσσερα ερωτήματα του Benchmark TPC-DS αντιπροσωπευτικά για την κάθε μια από τις τέσσερις κατηγορίες που παρέχει. Έτσι για τα πειράματα μας επιλέξαμε τα ερωτήματα 27,33,69 και 85. Σε αυτό το κεφάλαιο θα εξετάσουμε τα πλάνα που παράγουν τα υπολογιστικά πλαίσια για τα ερωτήματα αυτά και θα μελετήσουμε τους αλγορίθμους Joins, του χρόνους και την αποδοτικότητα του κάθε υπολογιστικού πλαισιού για την συστοιχία υπολογιστών μας

Περιγραφή υπολογιστικής υποδομής 5.1.β

Για την μελέτη των πλάνων που παράγουν οι δυο μηχανές σημαντικό ρόλο παίζουν τα τεχνικά χαρακτηριστικά της συστοιχίας υπολογιστώ μας στον οποίον εκτελούνται τα ερωτήματα. Για την αποθήκευση των δεδομένων και την εκτέλεση των ερωτημάτων δημιουργήθηκαν 3 εικονικές μηχανές (Virtual Machines). Οι εικονικές μηχανές βρίσκονται στο υπολογιστικό νέφος της υπηρεσίας Okeanos [11]. Το υπολογιστικό νέφος Okeanos δημιουργήθηκε με σκοπό να προσφέρει υπολογιστικές υπηρεσίες και υπηρεσίες αποθήκευσης για τους Έλληνες ερευνητές. Είναι στηριγμένο πάνω σε διάφορα έργα ανοιχτού κώδικα (Linux, Google Zaneti, Python/Django Κλπ.) με τα οποία μπορεί και προσφέρει μια γρήγορη , εύκολη και ασφαλή πρόσβαση σε εικονικούς πόρους. Κάθε ερευνητής αποκτά πρόσβαση σε πόρους όπως CPU, Ram , εικονικούς δίσκους και εικονικές μηχανές που μπορεί να διαχειριστεί μέσω μιας διαδικτυακής διεπαφής χρήστη. Η συστοιχία υπολογιστών στην οποία πραγματοποιήθηκαν τα πειράματα μεταξύ των δυο υπολογιστικών πλαισίων διαθέτει παρόμοια χαρακτηριστικά για κάθε ένα υπολογιστή που αποτελεί μέλος της. Κάθε εικονική μηχανή διαθέτει περίπου 200 GB αποθηκευτικού χώρου , 8 GB RAM και 8 πυρήνες CPU. Οι εικονικές μηχανές βρίσκονται στο ίδιο δίκτυο έτσι ώστε να μπορούν να επικοινωνούν και να εκτελούν εργασίες παράλληλα. Για την συστοιχία μας ένας υπολογιστής επιβλέπει τους άλλους κατέχοντας τον ρόλο του Master ενώ οι άλλοι δύο εκτελούν εργασίες υπό την εποπτεία του Master ως Workers όπως φαίνεται και στην εικόνα 15

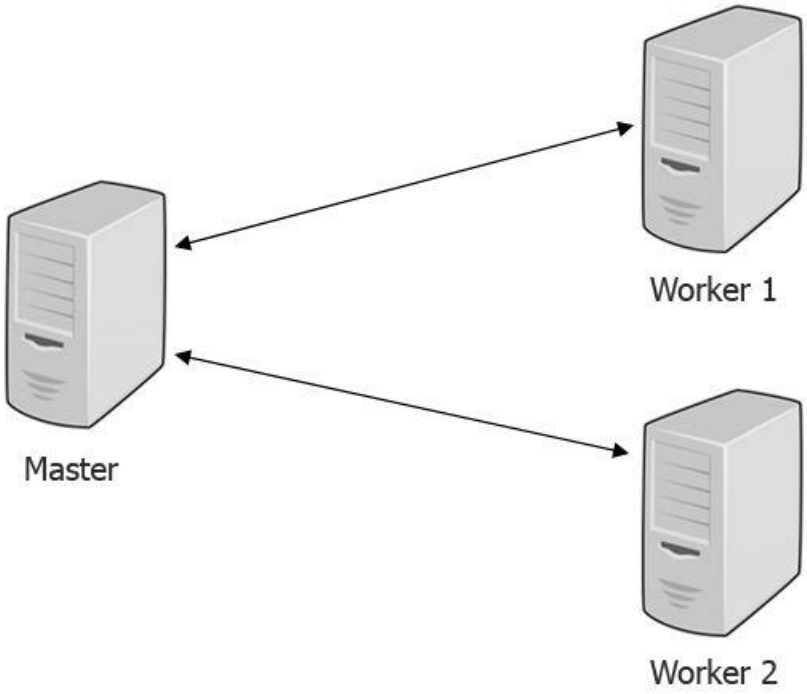

Figure 15 Η τοπολογία της συστοιχίας υπολογιστών που χρησιμοποιήθηκαν

Περιγραφή λογισμικού υποδομής 5.1.γ

Το λειτουργικό σύστημα το οποίο διαθέτουν οι εικονικές μηχανές είναι το Ubuntu Linux .Για το σύστημα κατανεμημένης αποθήκευσης δεδομένων χρησιμοποιήθηκε μια κλασσική λύση στην αποθήκευση δεδομένων σε τέτοια συστήματα το HDFS του υπολογιστικού πλαίσιού Apache Hadoop. Έτσι για κάθε ένα από τις εικονικές μηχανές εγκαταστάθηκε και έγινε η κατάλληλη παραμετροποίηση του λογισμικού Apache Hadoop της έκδοσης 2.7.7 . Η συστοιχία υπολογιστών μας διαθέτει μια εικονική μηχανή που έχει χρέη Master ενώ τα άλλα δύο εκτελούν χρέη Worker. Για την δημιουργία των πινάκων και έπειτα την εκτέλεση των ερωτημάτων χρησιμοποιήθηκε το λογισμικό Apache Spark της έκδοσης 3.0.0 . Κάθε εικονική μηχανή διαθέτει ένα αντίγραφο του λογισμικού Apache Spark και ομοίως με το HDFS μια εικονική μηχανή έχει χρέη Master ενώ τα άλλα έχουν χρέη Worker. Αρχικά με την χρήση του Spark και του Hive δημιουργήσαμε τους κατάλληλους πίνακες και τους γεμίσαμε με τα δεδομένα που παραχθήκαν στα προηγούμενα βήματα. Τα δεδομένα μετατρέπονται σε μια μορφή αρχείων που έχει ως βάση την στήλη του πίνακα γνωστό ως Parquet, που θα εξοικονομήσει χώρο και θα κάνει πιο γρήγορες κάποιές εργασίες αναζήτησης , και έπειτα αποθηκεύτηκαν στο HDFS. Κάθε εικονική μηχανή διαθέτει το υπολογιστικό πλαίσιο Apache Hive της έκδοσης 2.3.7. Το Metastore του Hive βρίσκεται σε μια βάση MySql στον Master κόμβο της συστοιχίας υπολογιστών. Από εκεί γίνεται προσβάσιμο σε όλα τα άλλα μέλη της συστοιχίας υπολογιστών με την χρήση ενός thrift server προτόκολου που τρέχει πάνω στον Master κόμβο. Για την σύγκριση των δυο υπολογιστικών πλαισίων χρησιμοποιήθηκε το TPC-DS τόσο για τα 100 GB δεδομένων που παράχθηκαν όσο και για τα ερωτήματα. Τέλος επιλέχθηκαν 4 ερωτήματα από τα 99 ερωτήματα που παράχθηκαν ένα για κάθε κατηγορία ερωτημάτων του TPC-DS Benchmark

Σύγκριση των πλάνων των δυο υπολογιστικών πλαισίων 5.1.δ

Και τα δυο υπολογιστικά πλαίσια προσφέρουν στον χρήστη την δυνατότητα να κατευθύνει την μηχανή ως προς την επιλογή του αλγορίθμου αν αυτός έχει γνώσει για τον τύπο των δεδομένων και πως αυτά είναι κατανεμημένα στους πίνακες. Αυτό γίνεται με έναν ειδικό μηχανισμό που ονομάζεται Hints. Έτσι θα μπορούσαμε κατά την εκτέλεση ενός ερωτήματος να διαλέξουμε ως χρήστης εμείς τον αλγόριθμο πχ τον Broadcast Join για ένα συγκεκριμένο ερώτημα προσθέτοντας στο ερώτημα την εντολή BROADCAST(όνομα _πίνακα) για την μηχανή Apache Spark και MAPJOIN(όνομα _πίνακα) για την μηχανή Apache Hive. Για να μελετήσουμε καλύτερα τον τρόπο επιλογής των αλγορίθμων τον βελτιστοποιητή ερωτημάτων Catalyst και Calcite δεν χρησιμοποιήθηκε αυτός ο μηχανισμός. Η επιλογή των αλγορίθμων Join για τις δύο μηχανές επηρεάζεται από πολλούς παράγοντες όπως τον τύπο του Join, το μέγεθος των πινάκων, τον βελτιστοποιητή και το μοντέλο που χρησιμοποιεί.

Εκτέλεση ερωτήματος της κατηγορίας OLAP Iterative ερωτήματα 5.2

Το Ερώτημα 27 5.2.α

Το ερώτημα 27 που θα εξετάσουμε ανάγεται στην κατηγορία των OLAP Iterative ερωτημάτων του benchmark TPC-DS. Όπως βλέπουμε και στην εικόνα 16 το συγκεκριμένο ερώτημα δρα βάσει του χρήστη για αυτό και βλέπουμε την χρήση των πινάκων Customer, Customer_Address και Customer_Demographics.

```
select
  cd_gender,
  cd_marital_status,
  cd_education_status,
  count(*) cnt1,
  cd_purchase_estimate,
  count(*) cnt2,
  cd_credit_rating,
  count(*) cnt3
 from
  customer c,customer_address ca,customer_demographics
 where
  c.c_current_addr_sk = ca.ca_address_sk and
  ca_state in ('KY','TX','AR') and
  cd_demo_sk = c.c_current_cdemo_sk and
  exists (select *
          from store_sales,date_dim
          where c.c_customer_sk = ss_customer_sk and
                ss_sold_date_sk = d_date_sk and
                d_year = 2001 and
                d_moy between 1 and 1+2) and
    (not exists (select *
            from web_sales,date_dim
            where c.c_customer_sk = ws_bill_customer_sk and
                  ws_sold_date_sk = d_date_sk and
                  d_year = 2001 and
                  d_moy between 1 and 1+2) and
     not exists (select *
            from catalog_sales,date_dim
            where c.c_customer_sk = cs_ship_customer_sk and
                  cs_sold_date_sk = d_date_sk and
                  d_year = 2001 and
                  d_moy between 1 and 1+2))
 group by cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
          cd_credit_rating
 order by cd_gender,
          cd_marital_status,
          cd_education_status,
          cd_purchase_estimate,
          cd_credit_rating
 limit 100;
```

Figure 16 Ερώτημα 27 του Benchmark TPC-DS της κατηγορίας OLAP Iterative queries

Εκτέλεση ερωτήματος 27 από το υπολογιστικό πλαίσιο Spark 5.2.β

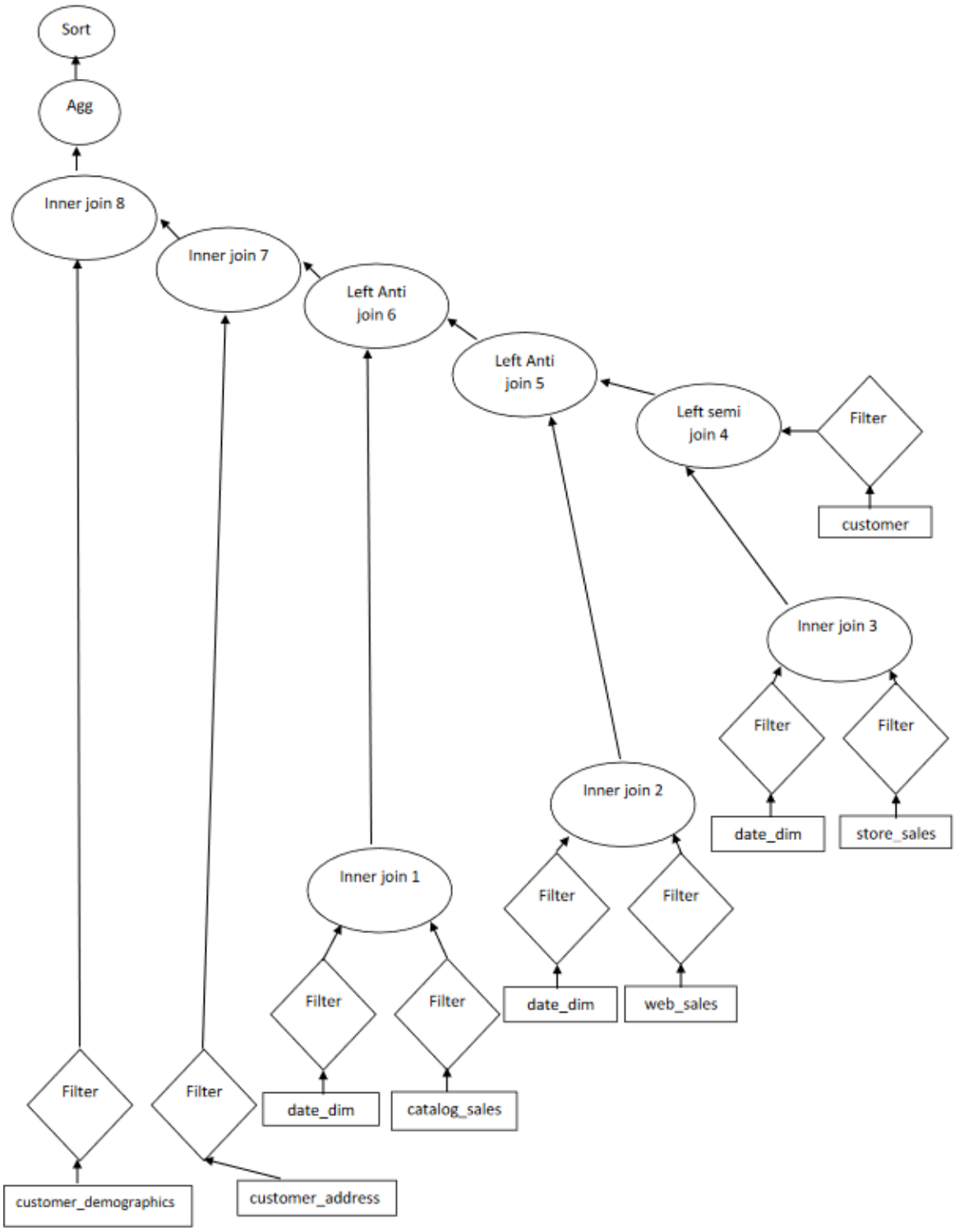

Figure 17 Φυσικό πλάνο εκτέλεσης του ερωτήματος 27 του TPC-DS που παράγει ο βελτιστοποιητής ερωτημάτων του Apache Spark

Από το φυσικό πλάνο που παράγει η μηχανή Apache Spark από την Εικόνα 17 βλέπουμε τον τύπο των Joins και σε ποιους πίνακες γίνονται. Το πρώτο join είναι τύπου inner Join και γίνεται στο dimension πίνακα date_dim και στον fact πίνακα catalog_sales για τις εγγραφές των πινάκων που ισχύει η έκφραση cs_sold_date_sk= d_date_sk. Το δεύτερο Join είναι και αυτό τύπου inner Join στον πίνακα date_dim και στον fact πίνακα web_sales για τις εγγραφές των πινάκων που ισχύει η έκφραση ws_sold_date_sk = d_date_sk. Για το τρίτο Join ισχύει το ίδιο με τα προηγούμενα Joins με την διαφορά ότι στην σχέση ως fact πίνακας χρησιμοποιείται ο πίνακας store_sales και η έκφραση που θα πρέπει να ισχύει είναι ss_sold_date_sk= d_date_sk. Για το τέταρτο Join που είναι τύπου left-semi Join βλέπουμε ότι παίρνει ως είσοδο το τρίτο Join και τον dimension πίνακα customer και η έκφραση που θα πρέπει να ισχύει είναι c_customer_sk= ss_customer_sk. Για το πέμπτο Join που παίρνει ως είσοδο το δεύτερο και το τέταρτο Join και είναι τύπου left anti Join πρέπει να ισχύει η σχέση c_customer_sk= ws_bill_customer_sk. Για το έκτο Join που είναι ίδιου τύπου με το προηγούμενο παίρνει ως είσοδο το πρώτο και το πέμπτο Join και η σχέση που πρέπει να ισχύει είναι η c_customer_sk= cs_ship_customer_sk. Το έβδομο Join που είναι τύπου inner Join παίρνει ως είσοδο το προηγούμενο Join που αναλύσαμε και τον dimension πίνακα customer_address το Join γίνεται βάσει της σχέσης c_current_addr_sk = ca_address_sk. Το τελευταίο Join είναι του ίδιου τύπου με το προηγούμενο και παίρνει ως είσοδο το έβδομο Join και τον dimension πίνακα customer_demographics ενώ η σχέση που πρέπει να ικανοποιεί το Join είναι η cd_demo_sk = c_current_cdemo_sk. Τα αποτελέσματα με την διαδικασία των Joins περνάν από μια συνάρτηση συνάθροισης και έπειτα ταξινομούνται για να πάρουμε τα τελικά αποτελέσματα.

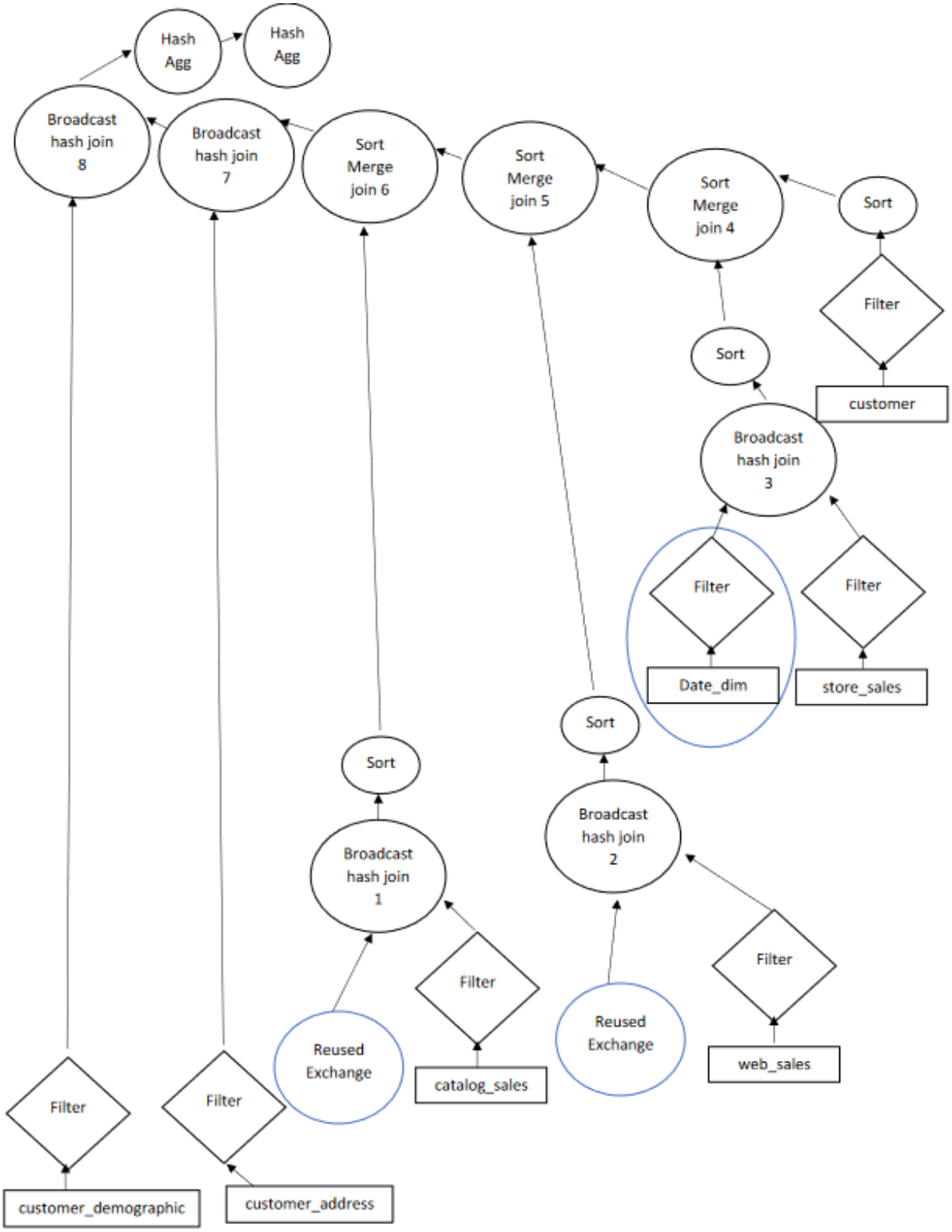

Figure 18 Εκτέλεση του ερωτήματος 27 από την μηχανή Apache Spark

Από την φυσική εκτέλεση του ερωτήματος βλέπουμε περισσότερες πληροφορίες για το πως γίνεται η πραγματική εκτέλεση ενός ερωτήματος στην συστοιχία υπολογιστών μας. Όπως βλέπουμε και στο Εικόνα 18 το πρώτο Join επαναχρησιμοποιεί την εργασία ανάγνωσης και φιλτραρίσματος του dimension πίνακα Date_Dim τον οποίο η μηχανή έχει υπολογίσει νωρίτερα και αποφασίζει να στείλει στους κόμβους για να τον επαναχρησιμοποιήσουν αφού χρησιμοποιείται πολλές φορές κατά την διάρκεια εκτέλεσης του ερωτήματος. Παρατηρούμε ότι ο αλγόριθμος που ακολουθείται για την υλοποίηση του πρώτου Join είναι ο Broadcast Join που εξηγήσαμε στο υπο-κεφάλαιο 4.1.γ . Ουσιαστικά ακολουθώντας αυτή την υλοποίηση η μηχανή διαμοιράζει τον μικρότερο από τους δύο πίνακες στους κόμβους της συστοιχίας υπολογιστών μας μέσα από το δίκτυο με την προϋπόθεση ότι αυτός ο πίνακας είναι αρκετά μικρός έτσι ώστε να φορτωθεί στην μνήμη. Έτσι αποστέλλεται μόνο ένας πίνακας με την χρήση του δικτύου και εξοικονομείται χρόνος αφού η χρήση του δικτύου καθιστά τις εργασίες πιο χρονοβόρες και θεωρείται ένας «ακριβός» πόρος. Σε αυτήν την περίπτωση η μηχανή αποστέλλει τον dimension πίνακα αφού συνήθως αυτοί είναι πιο μικροί από τους fact πίνακες. Το δεύτερο και τρίτο Join ακολουθούν και αυτά τον αλγόριθμο με την ίδια λογική διαμοιράζοντας τον dimension πίνακα τους. Έπειτα βλέπουμε ότι η εργασία ταξινόμησης κατεβαίνει στα χαμηλότερα στάδια του πλάνου στους κατάλληλους πίνακες στους οποίους θα εφαρμοστεί ο αλγόριθμος Sort Merge Join ή αλλιώς Repartition Join.Στο τέταρτο Join βλέπουμε ότι η μηχανή επέλεξε τον αλγόριθμο Sort Merge Join που επιλέγεται όταν το μέγεθος των πινάκων απαγορεύει την επιλογή του αλγόριθμου Broadcast Join έτσι τα δεδομένα πρώτα ταξινομούνται τοπικά βάση του κλειδιού Join τους και έπειτα μέσα από το δίκτυο αποστέλλονται οι εγγραφές με το ίδιο κλειδί Join στον ίδιο worker έτσι ώστε να εκτελέσει την διαδικασία συνένωσης. Όπως βλέπουμε και στο σχήμα ακολουθούν ακόμα άλλα δυο Joins που ακολουθούν τον αλγόριθμο Sort Merge Join και έπειτα τα δυο τελευταία που ακολουθούν τον αλγόριθμο Broadcast Join η αλλαγή αυτή στο τέλος από Sort-Merge σε Broadcast μπορεί αν εξηγηθεί να παρατηρήσουμε ότι τα δυο τελευταία Sort-Merge join είναι τύπου Left Anti Join που σημαίνει ότι αν υπάρχουν πολλά κοινά στοιχεία στους πίνακες μειώνεται το μέγεθος του αποτελέσματος του Join αφού επιστρέφει τις εγγραφές του αριστερού πίνακα που δεν έχουν κοινές τιμές με τον δεξιό πίνακα στην σχέση που δίνεται στο Join. Τέλος βλέπουμε ότι η μηχανή εκτελεί δυο συναρτήσεις «συνάθροισης» (Aggregation functions) που σκοπό έχουν να υποβάλουν τα τελικά δεδομένα σε μια πράξη πχ πρόσθεση (sum), εύρεσης μέσου όρου (average) κλπ. . Η πρώτη συνάρτηση από αυτές αποτελεί μια συνάρτησης «μερικής συνάθροισης» (partial aggregation) που υποβάλει ένα υποσύνολο από τα δεδομένα σε μια συνάρτηση λόγο του μεγάλου όγκου των συνολικών δεδομένων. Έπειτα τα δεδομένα διαμοιράζονται μέσα από το δίκτυο και εφαρμόζεται σε αυτά και στα δεδομένα που έμεινα η επόμενη συνάρτηση συνάθροισης. Στο συγκεκριμένο ερώτημα αρχικά τα δεδομένα υποβάλλονται σε μια εργασία “partial count” και έπειτα σε μια συνάρτηση “count” έτσι ώστε να παραχθούν τα τελικά αποτελέσματα.

Εκτέλεση ερωτήματος 27 από το υπολογιστικό πλαίσιο Hive 5.2.γ

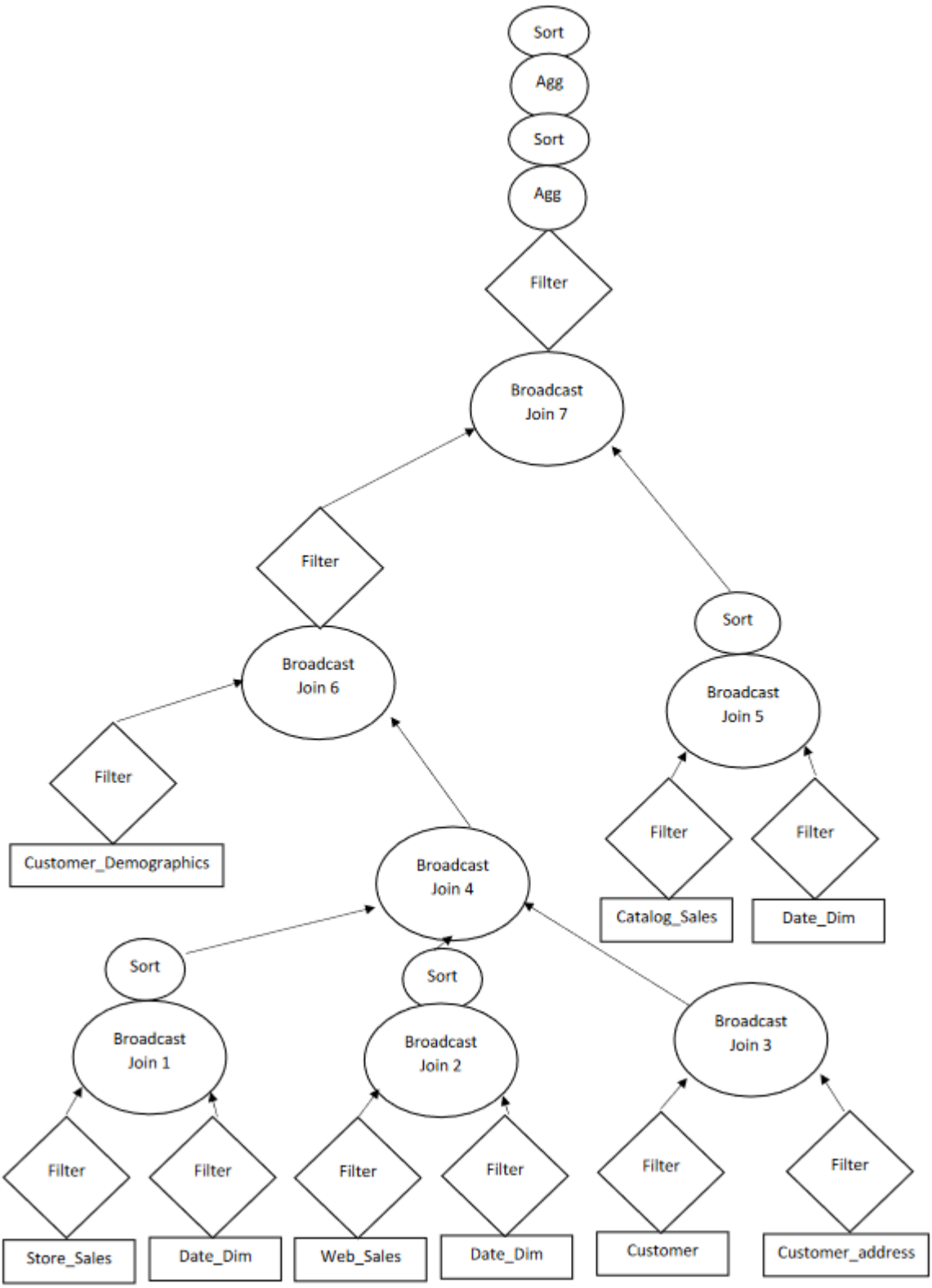

Figure 19 Εκτέλεση του ερωτήματος 27 από την μηχανή Apache Hive

Η μηχανή Apache Hive δεν παρέχει κάποιο τρόπο ώστε να μπορούμε να προβάλουμε το φυσικό πλάνο που παράγει για κάποιο ερώτημα ο βελτιστοποιητής Calcite αλλά μόνο την φυσική εκτέλεση . Όπως βλέπουμε και από την Εικόνα 19 ο Optimizer Calcite επιλέγει τον αλγόριθμο Broadcast Join για να συνενώσει τους πίνακες Store_Sales και Date_Dim και έπειτα ταξινομεί τα αποτελέσματα. Το ίδιο κάνει και για το επόμενο Join για τους πίνακες Web_Sales και Date_Dim. Για το τρίτο Join ακολουθείται η ίδια διαδικασία Join για τους δυο dimension πίνακες Customer και Customer_address. Το τέταρτο Join είναι ένα Multi-way Join ακολουθώντας τον αλγόριθμο Broadcast Join μεταξύ τριών πινάκων, η μηχανή αποφασίζει να εκτελέσει αυτό το Join χρησιμοποιώντας παραπάνω από 2 πίνακες λόγου του κοινού Κλειδιού Join c_customer_sk που διαθέτουν τα Joins που ενώνονται σε ένα . Το πέμπτο Join ακολουθεί και αυτό τον αλγόριθμο Broadcast Join μεταξύ των πινάκων Catalog_Sales και Date_Dim και έπειτα τα αποτελέσματα ταξινομούνται. Το έκτο Join εφαρμόζεται στον πίνακα Customer_Demographics και στο αποτέλεσμα του τέταρτου Join και έπειτα περνάει από μια συνάρτηση φιλτραρίσματος. Το τελευταίο Join χρησιμοποιεί και αυτό τον αλγόριθμο Broadcast Join παίρνοντας ως είσοδο τα αποτελέσματα του πέμπτου και του έκτου Join και έπειτα τα φιλτράρει χρησιμοποιώντας μια συνάρτηση φιλτραρίσματος. Τέλος τα αποτελέσματα περνάν από δυο συναρτήσεις συνάθροισης και κάθε φορά τα αποτελέσματα που βγαίνουν από αυτές παίρναν από ταξινόμηση . Από το πλάνο συμπεραίνουμε ότι αποτελούν και τα δυο συναρτήσεις συσσωματώσεις count η πρώτη σε κάποιο στάδιο map και η άλλη σε κάποιο στάδιο reduce οπότε καταλαβαίνουμε ότι γίνεται με την ίδια λογική όπως στην εκτέλεση του φυσικού πλάνου της Apache Spark δηλαδή στην αρχή σε κάποια δεδομένα και στο τέλος στο σύνολο τους για το τελικό αποτέλεσμα.

Εκτέλεση ερωτήματος της κατηγορίας Ad hoc 5.3

Το Ερώτημα 33 5.3.α

Το ερώτημα 33 ανάγεται στην κατηγορία ad hoc. Όπως βλέπουμε και στην εικόνα 20 το ερώτημα σκοπεύει στην απάντηση επιχειρησιακών ερωτήσεων και έτσι βλέπουμε την χρήση πινάκων όπως αυτών του Store και Store_Sales

```
select
    sum(ss_net_profit) as total_sum
   ,s_state
   ,s_county
   ,grouping(s_state)+grouping(s_county) as lochierarchy
   ,rank() over (
    partition by grouping(s_state)+grouping(s_county),
    case when grouping(s_county) = 0 then s_state end
    order by sum(ss_net_profit) desc) as rank_within_parent
 from
    store_sales
   ,date_dim       d1
   ,store
 where
    d1.d_month_seq between 1209 and 1209+11
 and d1.d_date_sk = ss_sold_date_sk
 and s_store_sk  = ss_store_sk
 and s_state in
              ( select s_state
                from  (select s_state as s_state,
                 rank() over ( partition by s_state order by sum(ss_net_profit) desc) as ranking
                          from   store_sales, store, date_dim
                          where  d_month_seq between 1209 and 1209+11
                 and d_date_sk = ss_sold_date_sk
                 and s_store_sk  = ss_store_sk
                          group by s_state
                         ) tmp1
                where ranking <= 5
              )
 group by rollup(s_state,s_county)
 order by
   lochierarchy desc
  ,case when lochierarchy = 0 then s_state end
  ,rank_within_parent
 limit 100;
```

Figure 20 Ερώτημα 33 του benchmark TPC-DS της κατηγορίας ad hoc

Εκτέλεση ερωτήματος 33 από το υπολογιστικό πλαίσιο Spark 5.3.β

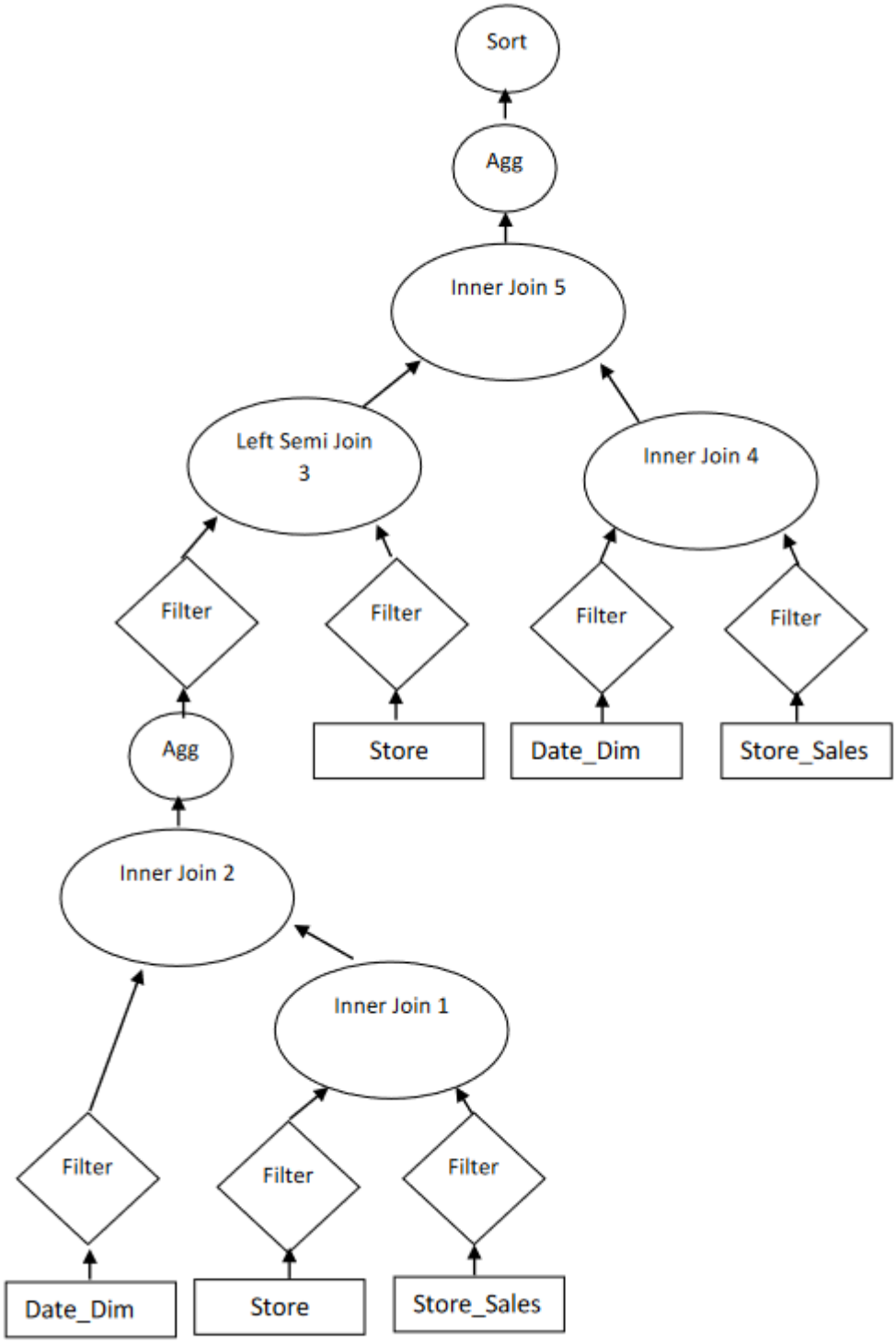

Figure 21 Φυσικό πλάνο εκτέλεσης του ερωτήματος 33 που παράγει ο βελτιστοποιητής ερωτημάτων του Apache Spark

Όπως βλέπουμε από το φυσικό πλάνο που παράγει η μηχανή Apache Spark στην Εικόνα 21 το πρώτο Join είναι τύπου inner Join και δέχεται ως είσοδο τον dimension πίνακα Store και τον fact πίνακα Store_Sales και η σχέση που θα πρέπει να ισχύει είναι η s_store_sk = ss_store_sk. Το δεύτερο Join είναι και αυτό τύπου Inner Join που δέχεται ως είσοδο τον dimension πίνακα Date_Dim και του προηγούμενο Join και η σχέση που θα πρέπει να ισχύει είναι η d_date_sk = ss_sold_date_sk. Έπειτα το αποτέλεσμα περνάει μέσα σε μια συνάρτηση συνάθροισης. Το αποτέλεσμα φιλτράρετε και περνάει ως είσοδος στο επόμενο Join που είναι τύπου Left Semi Join ως δεύτερη είσοδος δέχεται τον dimension πίνακα Store η σχέση που θα πρέπει να ισχύει είναι η s_state= s_state. Στο τέταρτο Join που είναι τύπου Inner Join και δέχεται ως είσοδο το dimension πίνακα Date_Dim και τον fact πίνακα Store_Sales και η σχέση που θα πρέπει να ισχύει είναι η d_date_sk = ss_sold_date_sk. Το τελευταίο Join είναι τύπου Inner Join και παίρνει ως είσοδο το τρίτο και το τέταρτο Join με την σχέση που θα πρέπει να ισχύει είναι η s_store_sk = ss_store_sk. Τέλος τα αποτελέσματα περνάν μέσα από μια συνάρτηση συνάθροισης και έπειτα ταξινομούνται.

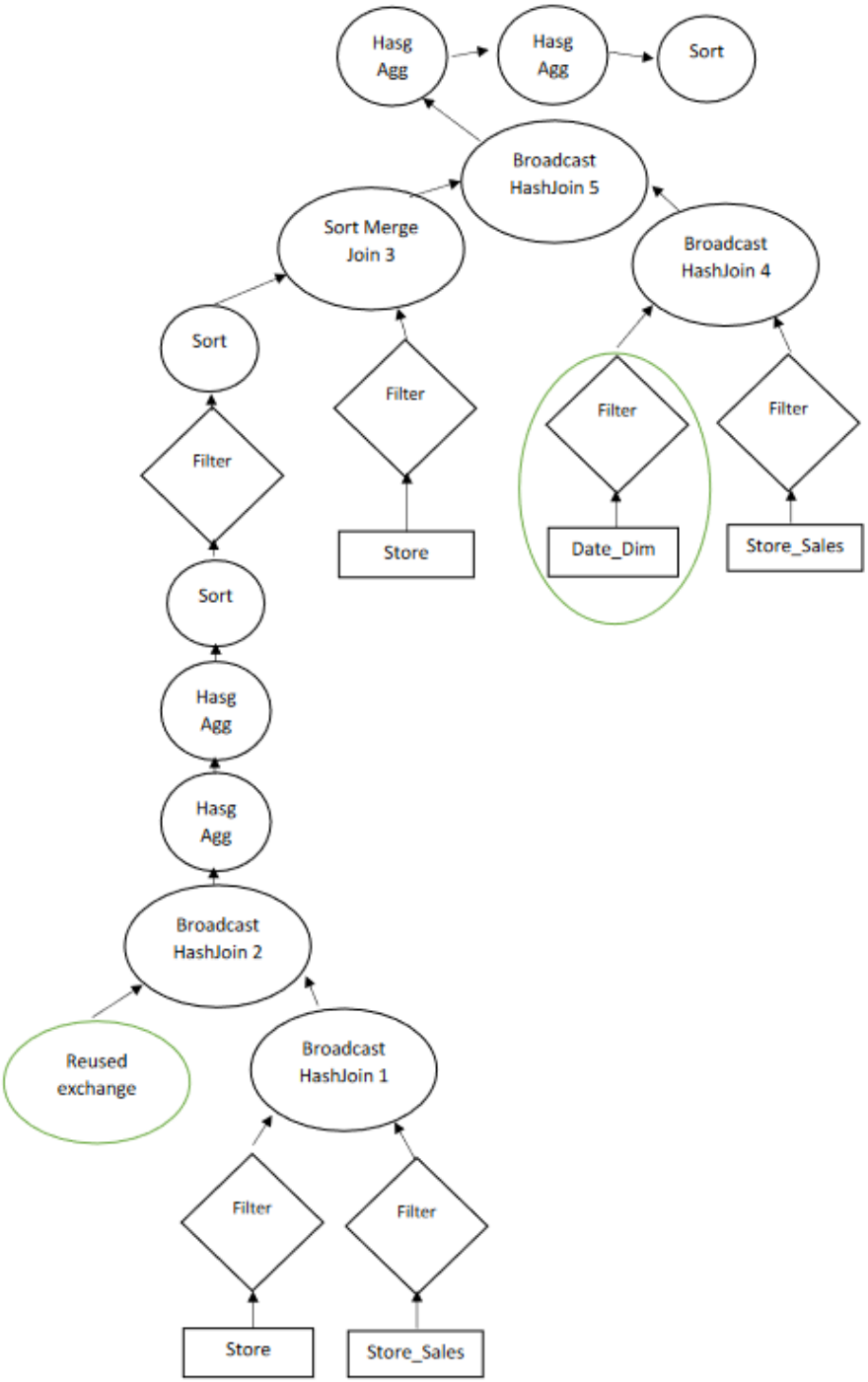

Figure 22 Φυσικό πλάνο εκτέλεσης του ερωτήματος 33 που παράγει ο βελτιστοποιητής ερωτημάτων του Apache Spark

Για την εκτέλεση του ερωτήματος από την μηχανή Apache Spark στην Εικόνα 22 παρατηρούμε ότι το πρώτο Join χρησιμοποιεί τον αλγόριθμο Broadcast Join στους πίνακες Store και Store_Sales ο αλγόριθμος διαμοιράζει τον μικρότερο από τους δυο πίνακες και εκτελεί το Join, στην περίπτωση μας ο πίνακας που διαμοιράζεται είναι ο Store επειδή είναι πίνακας dimension. Όπως βλέπουμε και από το σχήμα η μηχανή αποφασίζει να διαμοιράσει την διεργασία ανάγνωσης και φιλτραρίσματος του πίνακα Date_Dim που έχει υπολογιστεί σε προηγούμενη εργασία και υποβάλετε ως είσοδο μαζί με το πρώτο Join στο δεύτερο Broadcast Join ο μικρότερος πίνακας σε αυτήν την περίπτωση είναι ο Date_Dim επειδή όμως έχει διαμοιραστεί πιο πριν για η μηχανή μπορεί να προσπεράσει αυτό το βήμα και να συνεχίσει την υπόλοιπη διαδικασία του Broadcast Join. Έπειτα το αποτέλεσμα του δεύτερου Join υποβάλετε σε δυο συναρτήσεις συνάθροισης, στην πρώτη όπως φαίνεται και από το πλάνο το όνομα της εργασίας είναι “partial sum” και η επόμενη “sum” οπότε το πρώτο sum γίνεται σε ένα υποσύνολο των δεδομένων, έπειτα τα δεδομένα στέλνονται μέσα από το δίκτυο και εφαρμόζεται η δεύτερη συνάρτηση συνάθροισης στο σύνολο τους έτσι ώστε να παραχθεί το τελικό αποτέλεσμα. Μετά τα δεδομένα ταξινομούνται, φιλτράρονται , ξανά ταξινομούνται και υποβάλλονται ως είσοδο στο τρίτο Sort Merge Join μαζί με τον fact πίνακα Store. Το τέταρτο Broadcast Join δέχεται ως είσοδο τον dimension πίνακα Date_Dim που έχει υπολογιστεί πιο πριν και έχει διαμοιραστεί στους άλλους κόμβους και τον fact πίνακα Store_Sales. Το τελευταίο Broadcast Join δέχεται ως είσοδο το τρίτο και το τέταρτο Join και έπειτα τα αποτελέσματα του υποβάλλονται σε δυο συναρτήσεις συνάθροισης. Η πρώτη είναι “partial sum” και η δεύτερη “sum” οπότε ισχύει η ίδια λογική με πριν. Τέλος τα αποτελέσματα ταξινομούνται.

Εκτέλεση του ερωτήματος 33 από το υπολογιστικό πλαίσιο Hive 5.3.γ

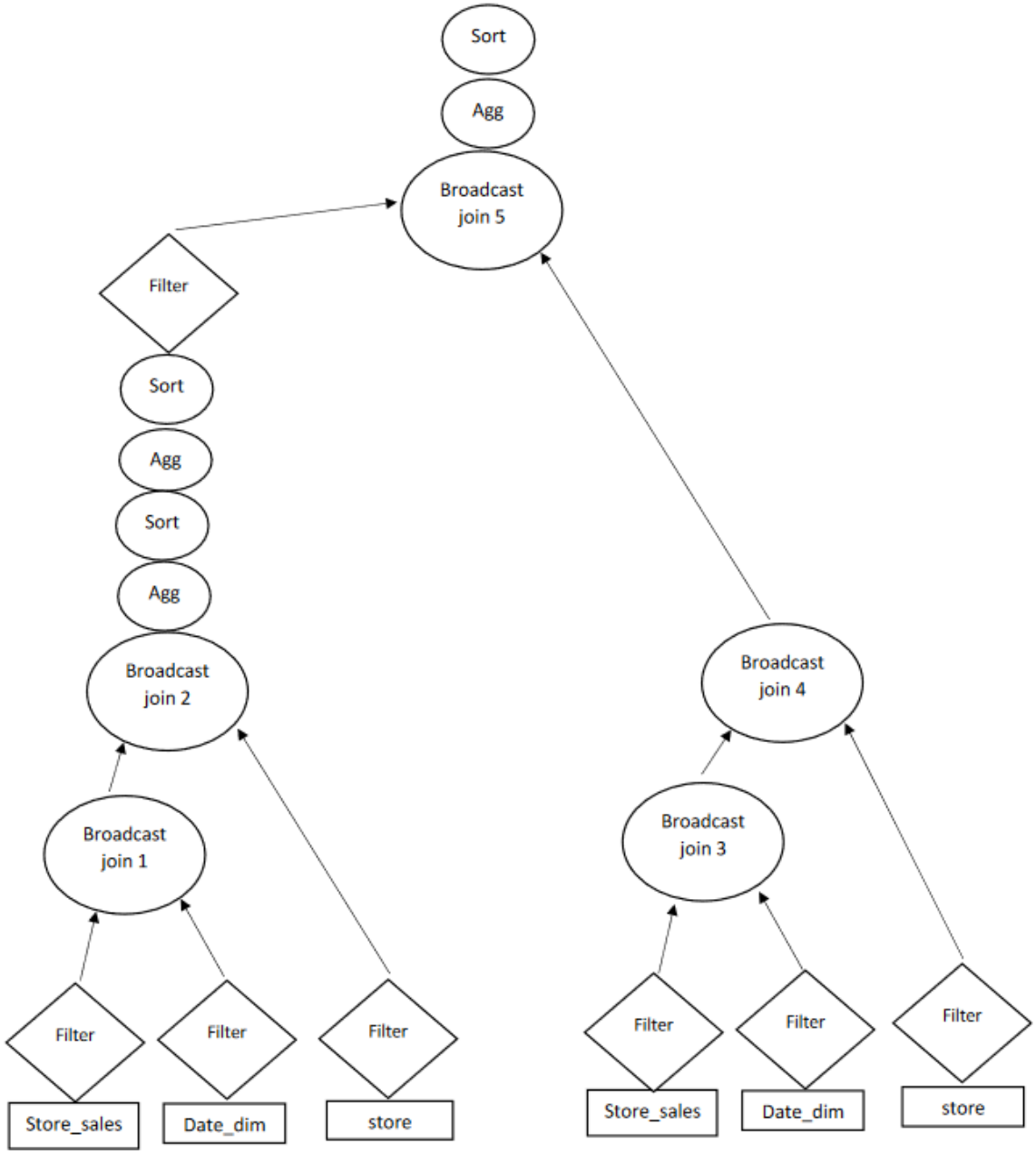

Figure 23 Εκτέλεση του ερωτήματος 33 από την μηχανή Apache Hive

Για την εκτέλεση του ερωτήματος από την μηχανή Apache Hive που φαίνεται στην Εικόνα 23 το πρώτο Join ακολουθεί τον αλγόριθμο Broadcast Join στους πίνακες Date_Dim και Store_Sales επειδή ο Date_Dim είναι μικρότερος από τον Store_Sales επιλέγεται να διαμοιραστεί μέσα από το δίκτυο. Το δεύτερο Join ακολουθεί και αυτόν τον αλγόριθμο Broadcast Join πάνω στο προηγούμενο Join και τον πίνακα Store που διαμοιράζεται λόγο του μικρότερου μεγέθους του. Έπειτα τα τελικά δεδομένα υποβάλλονται σε δυο συναρτήσεις συνάθροισης "sum" στις οποίες ακολουθείται η ίδια λογική με αυτές της μηχανής Apache Spark, κάθε φορά που βγαίνουν τα δεδομένα από τις συναρτήσεις ταξινομούνται και τέλος τα τελικά δεδομένα φιλτράρονται. Το τρίτο και το τέταρτο Join ακολουθούν την ίδια λογική με αυτά του πρώτου και του δεύτερου Join χωρίς όμως το κομμάτι των συναρτήσεων συνάθροισης. Το πέμπτο και τελευταίο Join που ακολουθεί τον αλγόριθμο Broadcast Join δέχεται ως είσοδο το δεύτερο και το τέταρτο Join. Έπειτα τα τελικά δεδομένα υποβάλλονται σε μια συνάρτηση συνάθροισης "sum" και ταξινομούνται.

Εκτέλεση ερωτήματος της κατηγορίας Data mining ερωτήματα 5.4

Το Ερώτημα 69 5.4.α

Το ερώτημα 69 ανάγεται στην κατηγορία των Data mining ερωτημάτων του TPC-DS. Σκοπό έχει δηλαδή να εξάγει σημαντικές σχέσεις και μελλοντικές τάσεις έτσι βλέπουμε στην Εικόνα 24 να χρησιμοποιούνται πίνακες όπως Item,Catalog_Sales,Call_Center κλπ.

```
with v1 as(
 select i_category, i_brand,
        cc_name,
        d_year, d_moy,
        sum(cs_sales_price) sum_sales,
        avg(sum(cs_sales_price)) over
          (partition by i_category, i_brand,
                        cc_name, d_year)
          avg_monthly_sales,
        rank() over
          (partition by i_category, i_brand,
                        cc_name
           order by d_year, d_moy) rn
 from item, catalog_sales, date_dim, call_center
 where cs_item_sk = i_item_sk and
       cs_sold_date_sk = d_date_sk and
       cc_call_center_sk= cs_call_center_sk and
       (
         d_year = 1999 or
         ( d_year = 1999-1 and d_moy =12) or
         ( d_year = 1999+1 and d_moy =1)
       )
 group by i_category, i_brand,
          cc_name , d_year, d_moy),
 v2 as(
 select v1.cc_name
        ,v1.d_year
        ,v1.avg_monthly_sales
        ,v1.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum
 from v1, v1 v1_lag, v1 v1_lead
 where v1.i_category = v1_lag.i_category and
       v1.i_category = v1_lead.i_category and
       v1.i_brand = v1_lag.i_brand and
       v1.i_brand = v1_lead.i_brand and
       v1. cc_name = v1_lag. cc_name and
       v1. cc_name = v1_lead. cc_name and
       v1.rn = v1_lag.rn + 1 and
       v1.rn = v1_lead.rn - 1)
  select  *
 from v2
 where  d_year = 1999 and
        avg_monthly_sales > 0 and
        case when avg_monthly_sales > 0 then abs(sum_sales - avg_monthly_sales) / avg_monthly_sales else null end > 0.1
 order by sum_sales - avg_monthly_sales, psum
 limit 100;
```

Figure 24 Ερώτημα 69 του benchmark TPC-DS της κατηγορίας Data mining

Εκτέλεση του ερωτήματος 69 από το υπολογιστικό πλαίσιο Spark 5.4.β

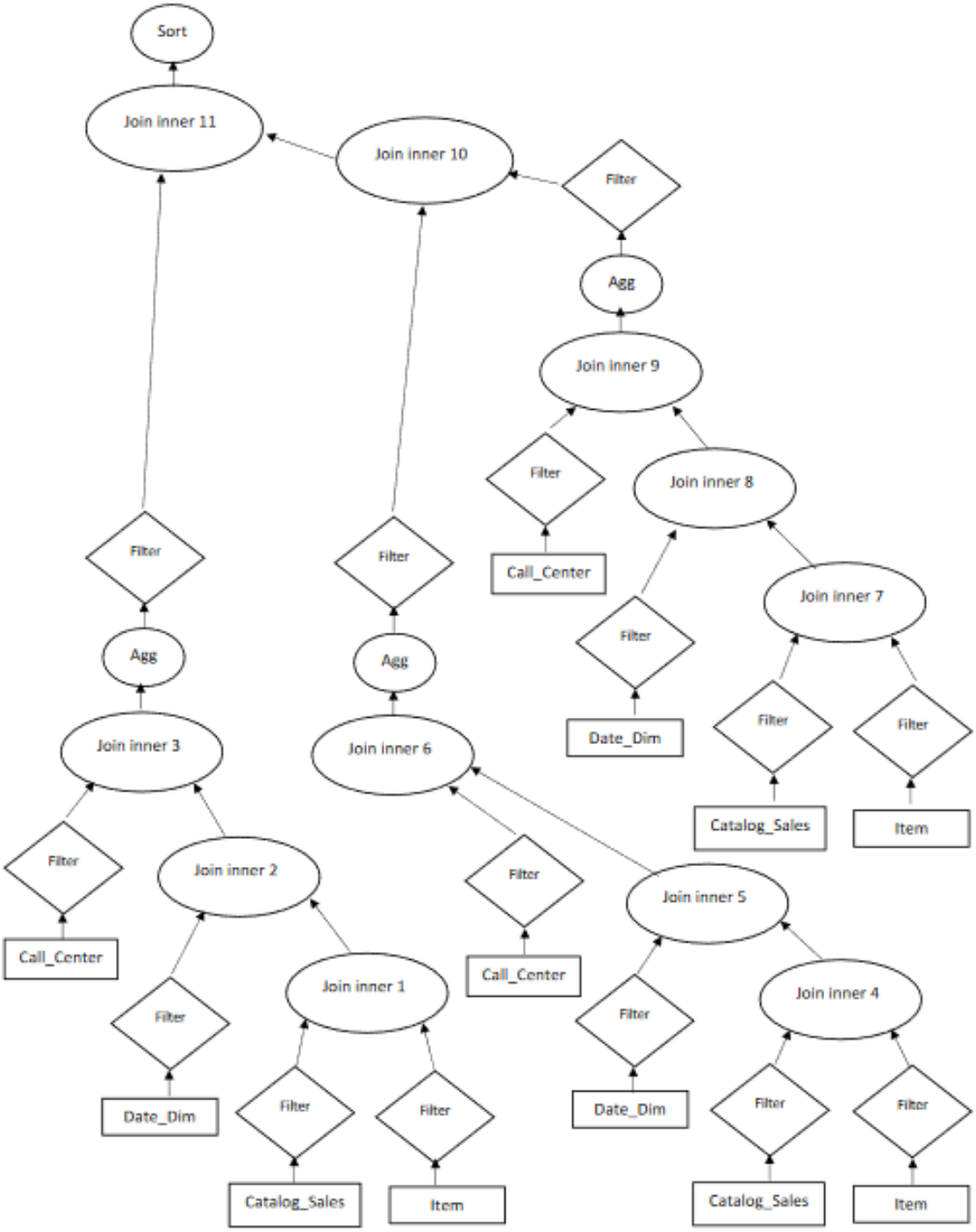

Figure 25 Φυσικό πλάνο εκτέλεσης του ερωτήματος 69 που παράγει ο βελτιστοποιητής ερωτημάτων του Apache Spark

Όπως βλέπουμε από την Εικόνα 25 του φυσικού πλάνου που παράγει η μηχανή Apache Spark όλα τα Joins είναι τύπου Inner Join.Το πρώτο Join δέχεται ως είσοδο τον fact πίνακα Catalog_Sales και τον dimension πίνακα Item με την σχέση cs_item_sk = i_item_sk. Το δεύτερο Join δέχεται ως είσοδο τον dimension πίνακα Date_Dim και το πρώτο Join με την σχέση cs_sold_date_sk = d_date_sk. Το τρίτο Join δέχεται ως είσοδο τον dimension πίνακα Call_Center και το δεύτερο Join με την σχέση cc_call_center_sk = cs_call_center_sk. Έπειτα τα δεδομένα υποβάλλονται σε μια συνάρτηση συνάθροισης και φιλτράρονται. Το τέταρτο Join δέχεται ως είσοδο τον fact πίνακα Catalog_Sales και τον dimension πίνακα Item με την σχέση cs_item_sk = i_item_sk. Το πέμπτο Join δέχεται ως είσοδο τον dimension πίνακα Date_Dim και το τέταρτο Join με την σχέση cs_sold_date_sk = d_date_sk. Το έκτο Join δέχεται ως είσοδο τον dimension πίνακα Call_Center και το πέμπτο Join με την σχέση cc_call_center_sk = cs_call_center_sk. Έπειτα τα δεδομένα υποβάλλονται σε μια συνάρτηση συνάθροισης και φιλτράρονται. Το έβδομο Join δέχεται ως είσοδο τον fact πίνακα Catalog_Sales και τον dimension πίνακα Item με την σχέση cs_item_sk = i_item_sk. Το όγδοο Join δέχεται ως είσοδο τον dimension πίνακα Date_Dim και το έβδομο Join με την σχέση cs_sold_date_sk = d_date_sk. Το ένατο Join δέχεται ως είσοδο τον dimension πίνακα Call_Center και το όγδοο Join με την σχέση cc_call_center_sk = cs_call_center_sk. Έπειτα τα δεδομένα υποβάλλονται σε μια συνάρτηση συνάθροισης και φιλτράρονται. Το δέκατο Join δέχεται ως είσοδο το έκτο και το ένατο Join με την σχέση ((((i_category = i_category) AND (i_brand= i_brand)) AND (cc_name = cc_name)) AND (rn= (rn+ 1))). Το εντέκατο Join δέχεται το τρίτο Join και το εντέκατο Join με την σχέση ((((i_category= i_category) AND (i_brand = i_brand)) AND (cc_name= cc_name)) AND (rn = (rn- 1))). Έπειτα τα τελικά δεδομένα ταξινομούνται.

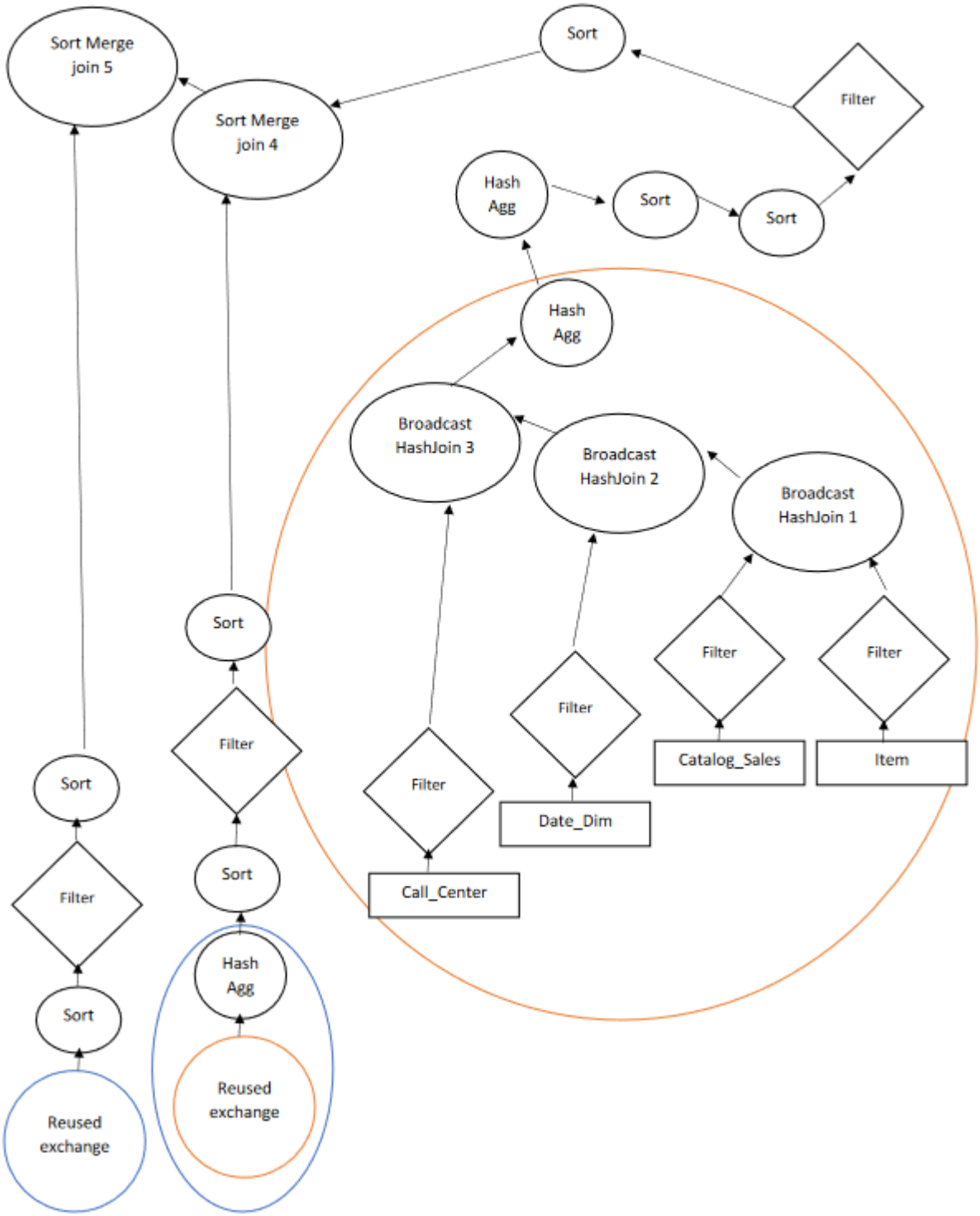

Figure 26 Εκτέλεση του ερωτήματος 69 από την μηχανή Apache Spark

Όπως βλέπουμε στην Εικόνα 26 της εκτέλεσης του ερωτήματος από την μηχανή Apache Spark το πρώτο Join ακολουθεί τον αλγόριθμο Broadcast Join στους πίνακες Catalog_Sales και Item. Το δεύτερο Join ακολουθεί και αυτό τον αλγόριθμο Broadcast Join έχοντας ως είσοδο τον πίνακα Date_Dim και το πρώτο Join. Το τρίτο Join ακολουθεί και αυτό τον αλγόριθμο Broadcast Join και παίρνει ως είσοδο τον πίνακα Call_Center και το δεύτερο Join. Έπειτα τα δεδομένα υποβάλλονται σε μια συνάρτηση συνάθροισης με όνομα "partial sum" που ουσιαστικά εφαρμόζει την συνάρτηση sum σε ένα υποσύνολο των δεδομένων . Όπως φαίνεται και στο σχήμα αποφασίζει να επαναχρησιμοποιήσει όλες τις εργασίες μέχρι τώρα οπότε τις διαμοιράζεται με τους άλλους κόμβους. Έπειτα τα δεδομένα υποβάλλονται σε ακόμα μια συνάρτηση συνάθροισης "sum" που εκτελεί την εργασία στο σύνολο των δεδομένων μετά τα δεδομένα ταξινομούνται. Στην συνέχεια τα δεδομένα φιλτράρονται και ξανά ταξινομούνται. Το τέταρτο Join ακολουθεί τον αλγόριθμο Sort Merge Join, με είσοδο την επαναχρησιμοποίηση της πρώτης σχέσης που διαμοίρασε η Spark και υποβάλει αυτά τα δεδομένα σε μια συνάρτηση συνάθροισης με όνομα "sum" που ο Spark αποφασίζει να διαμοιράσει για να ξανά χρησιμοποιηθεί. Έπειτα τα δεδομένα ταξινομούνται, φιλτράρονται και ξανά ταξινομούνται, το δεύτερο κομμάτι του Join είναι τα δεδομένα του τρίτου Join μετά της επεξεργασίας που υποβλήθηκαν. Το τελευταίο Join ακολουθεί και αυτό τον αλγόριθμο Sort Merge Join με είσοδο την επαναχρησιμοποίησή της δεύτερης σχέσης που διαμοιράζεται η Spark και το προηγούμενο Join.

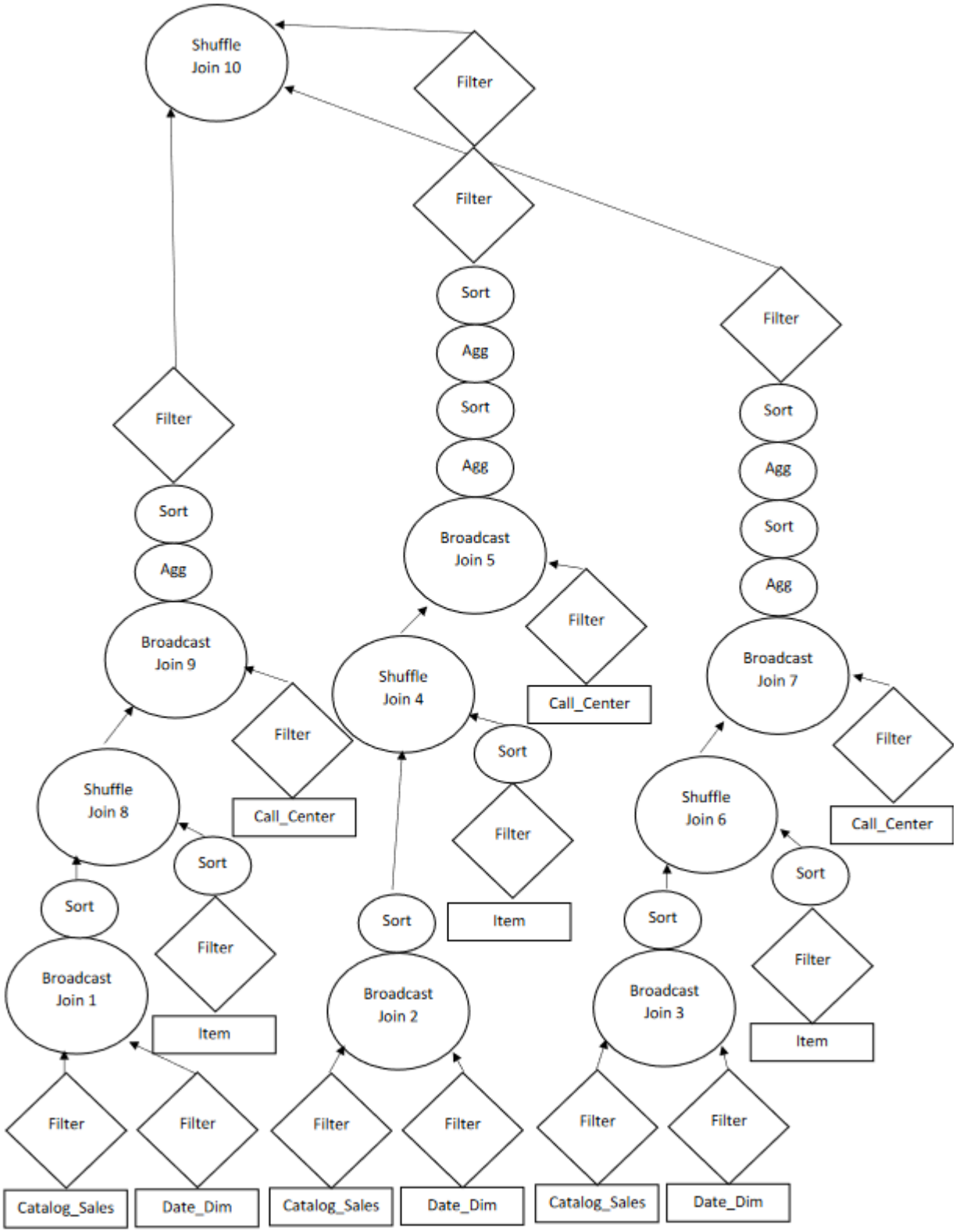

Figure 27 Εκτέλεση του ερωτήματος 69 από την μηχανή Apache Hive

Απο την εκτέλεση του ερωτήματος από την μηχανή Apache Hive όπως φαίνεται στην Εικόνα 27 το πρώτο Join ακολουθεί τον αλγόρθμο Broadcast Join στους πίνακες Catalog_Sales και Date_Dim , διαμοιράζεται ο μικρότερος από τους δυο ο Date_Dim, έπειτα τα δεδομένα ταξινομούνται. Το δεύτερο και το τρίτο Join ακολουθεί την ίδια λογική με το πρώτο.Το τέταρτο Join δέχεται ως είσοδο το δεύτερο Join και τα ταξινομημένα δεδομένα του πίνακα Item ακολουθόντας τον αλγόριθμο Shuffle Join ή αλλιώς Repartition Join .Ο αλγόριθμος Shuffle Join επιλέγεται όταν δεν μπορεί να επιλεχθεί ο αλγόριθμος Broadcast Join ,λόγο του ότι δεν υπάρχει αρκετά μικρός πίνακας που να μπορεί να φορτωθεί στην μνήμη ,έτσι διαβάζονται τα δεδομένα, ταξινομούνται και έπειτα όλες οι εγγραφές με το ίδιο κλειδί απόστέλονται μέσα από το δίκτυο στον ίδιο Reducer ώστε να ενωθούν και να παραχθεί το τελικό αποτέλεσμα. Το πέμπτο Join δέχεται ως είσοδο το τέταρτο Join και τα δεδομένα του πίνακα Call_Center ακολουθόντας τον αλγόριθμο Broadcast Join. Έπειτα τα δεδομένα υποβάλονται σε δυο συναρτήσεις συσσωμάτωσεις “sum” και μετά ταξινομούνται και φιλτράρονται. Τα επόμενα Join ακολουθούν την ίδια λογική με αυτά του τέταρτου και του πέμπτου Join με κάποιες διαφορές στις συναρτήσεις συσσωμάτωσεις και στην διαδικασία φιλτραρίσματος. Τέλος ειδιαίτερο ενδιαφέρον έχει το δέκατο Join που αποτελέι ένα Multi-Way Join που ακλουθεί τον αλγόριθμο Shuffle Join. Λόγο της κοινής σχέσης στην οποία γίνεται Join οι τρείς πινακες η μηχανή Hive αποφασίζει να εκτελέσει αυτά τα δυο Join ως ένα. Ο αλγόριθμος Shuffle Join επιλέγεται λόγο του μεγέθους των δεδομένων που είναι μεγάλος και έτσι δενε μπορεί να επιλεχθεί για αυτά ο Broadcast Join.

Εκτέλεση ερωτήματος της κατηγορίας Reporting ερωτήματα 5.5

Το Ερώτημα 85 5.5.α

Το ερώτημα 85 ανάγεται στην κατηγορία των reporting ερωτημάτων και σκοπό έχει να απαντήσει σε καθορισμένα επιχειρηματικά ερωτήματα έτσι χρησιμοποιούνται οι πίνακες όπως ο Warehouse,item, Catalog_Returns, Catalog_Sales κλπ. , το ερώτημα φαίνεται στην εικόνα 28

```
select
   w_state
  ,i_item_id
  ,sum(case when (cast(d_date as date) < cast ('2002-02-27' as date))
        then cs_sales_price - coalesce(cr_refunded_cash,0) else 0 end) as sales_before
  ,sum(case when (cast(d_date as date) >= cast ('2002-02-27' as date))
        then cs_sales_price - coalesce(cr_refunded_cash,0) else 0 end) as sales_after
 from
   catalog_sales left outer join catalog_returns on
      (cs_order_number = cr_order_number
       and cs_item_sk = cr_item_sk)
  ,warehouse
  ,item
  ,date_dim
 where
     i_current_price between 0.99 and 1.49
 and i_item_sk          = cs_item_sk
 and cs_warehouse_sk    = w_warehouse_sk
 and cs_sold_date_sk    = d_date_sk
 and d_date between (cast ('2002-02-27' as date) - 30 days)
                and (cast ('2002-02-27' as date) + 30 days)
 group by
    w_state,i_item_id
 order by w_state,i_item_id
limit 100;
```

Figure 28 Ερώτημα 85 του benchmark TPC-DS της κατηγορίας reporting query

Εκτέλεση του ερωτήματος 85 από το υπολογιστικό πλαίσιο Spark 5.5.β

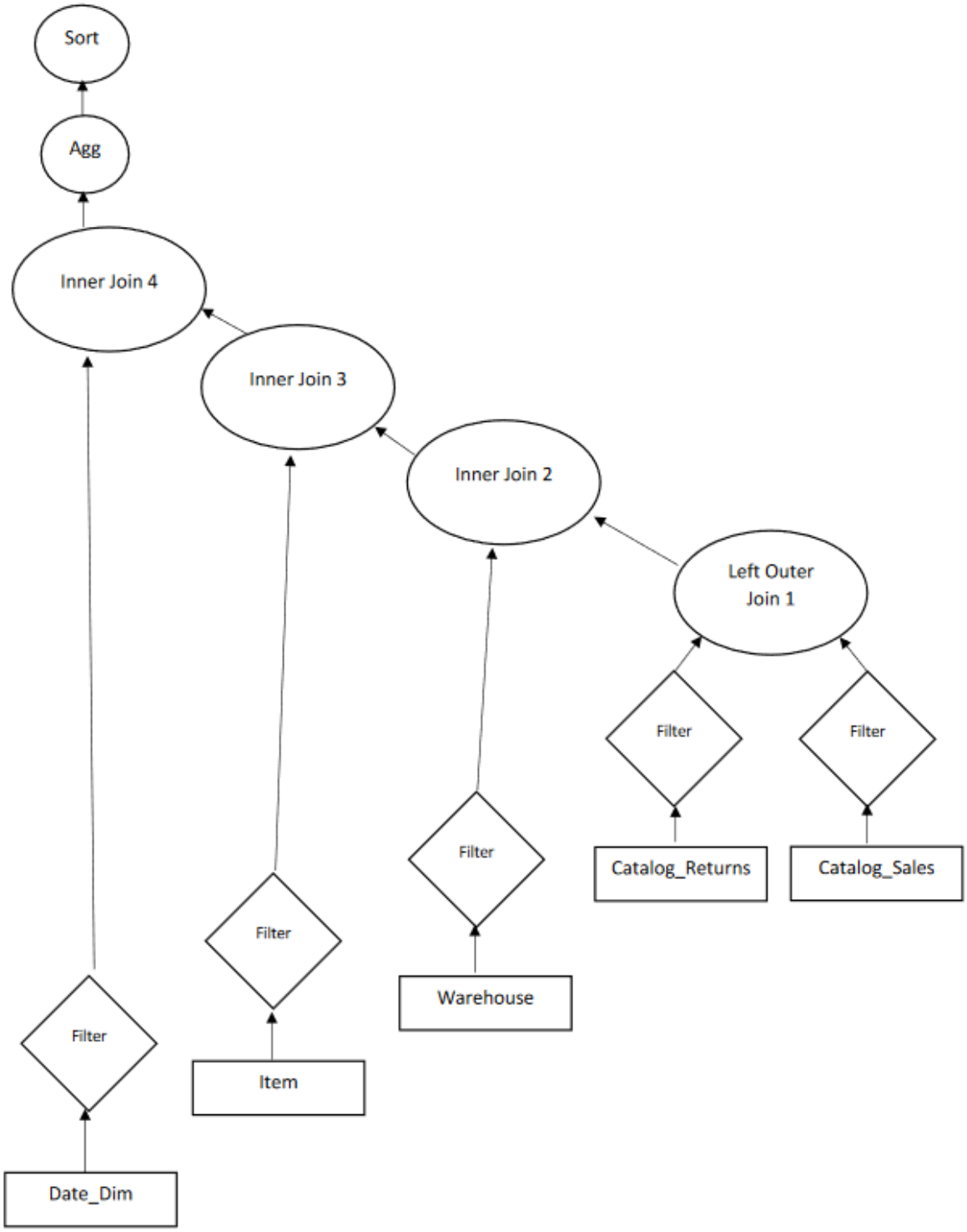

Figure 29 Φυσικό πλάνο εκτέλεσης του ερωτήματος 85 που παράγει ο βελτιστοποιητής ερωτημάτων του Apache Spark

Από την Εικόνα 29 του φυσικού πλάνου που παράγει η μηχανή Apache Spark για αυτό το ερώτημα παρατηρούμε ότι το πρώτο Join είναι τύπου Left Outer Join με είσοδο τους δυο fact πίνακες Catalog_Returns και Catalog_Sales με σχέση (cs_order_number= cr_order_number) AND (cs_item_sk = cr_item_sk). Έπειτα το δεύτερο Join είναι τύπου Inner Join με είσοδο τον dimension πίνακα Warehouse και το πρώτο Join με σχέση cs_warehouse_sk = w_warehouse_sk. Το τρίτο Join είναι και αυτό τύπου Inner Join και δέχεται ως είσοδο τον dimension πίνακα Item και το δεύτερο Join με σχέση i_item_sk = cs_item_sk. Το τελευταίο Join είναι και αυτό του ίδιου τύπου με το προηγούμενο Join και δέχεται ως είσοδο τον dimension πίνακα Date_Dim και το προηγούμενο Join με σχέση cs_sold_date_sk= d_date_sk. Τα δεδομένα από το τελευταίο Join υποβάλλονται σε μια συνάρτηση συνάθροισης και ταξινομούνται.

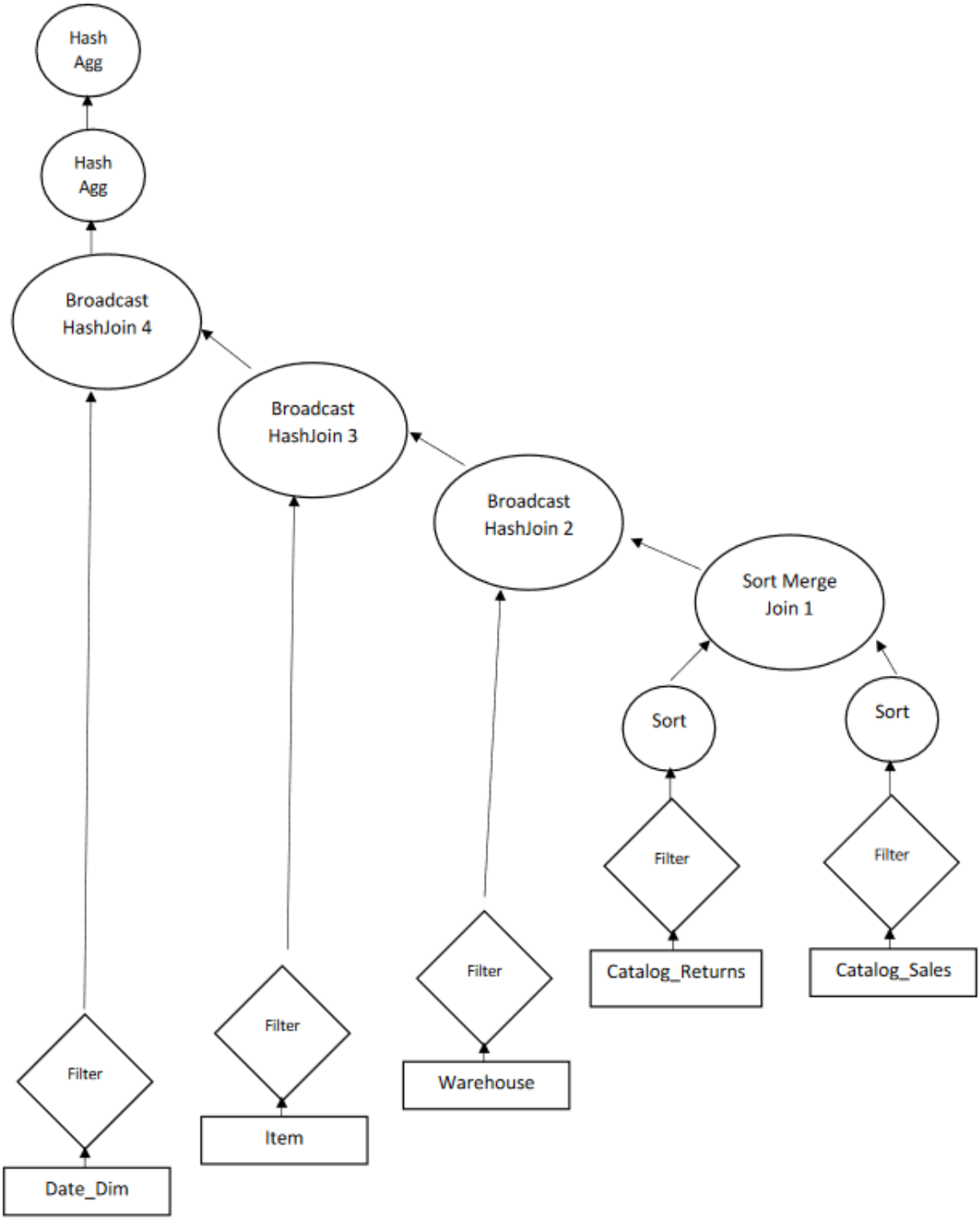

Figure 30 Εκτέλεση ερωτήματος 85 από την μηχανή Apache Spark

Για την εκτέλεση του ερωτήματος από την μηχανή Apache Spark στην Εικόνα 30 βλέπουμε ότι προτού γίνει η διεργασία του πρώτου Join τα δεδομένα από τους πίνακες Catalog_Returns και Catalog_Sales ταξινομούνται . Ο αλγόριθμος που ακολουθεί το πρώτο Join είναι ο Sort Merge Join πάνω στους δυο fact πίνακες . Το επόμενο Join ακολουθεί τον αλγόριθμο του Broadcast Join με είσοδο τον dimension πίνακα Warehouse, που είναι ο μικρότερος και αποστέλλεται μέσα από το δίκτυο , και το πρώτο Join. Το τρίτο ακολουθεί και αυτό τον αλγόριθμο Broadcast Join και παίρνει ως είσοδο τον dimension πίνακα Item , που είναι ο μικρότερος και αποστέλλεται μέσα από το δίκτυο, και το δεύτερο Join. Το τελευταίο Join χρησιμοποιεί και αυτό τον αλγόριθμο Broadcast Join με είσοδο τον dimension πίνακα Date_dim ,που διαμοιράζεται στο δίκτυο, και το τρίτο Join. Τέλος τα τελικά δεδομένα υποβάλλονται σε δυο συναρτήσεις συνάθροισης , με την πρώτη να έχει όνομα “partial_sum” και την δεύτερη να έχει όνομα “sum” που ακολουθάει την ίδια λογική που αναλύσαμε για προηγούμενο ερώτημα.

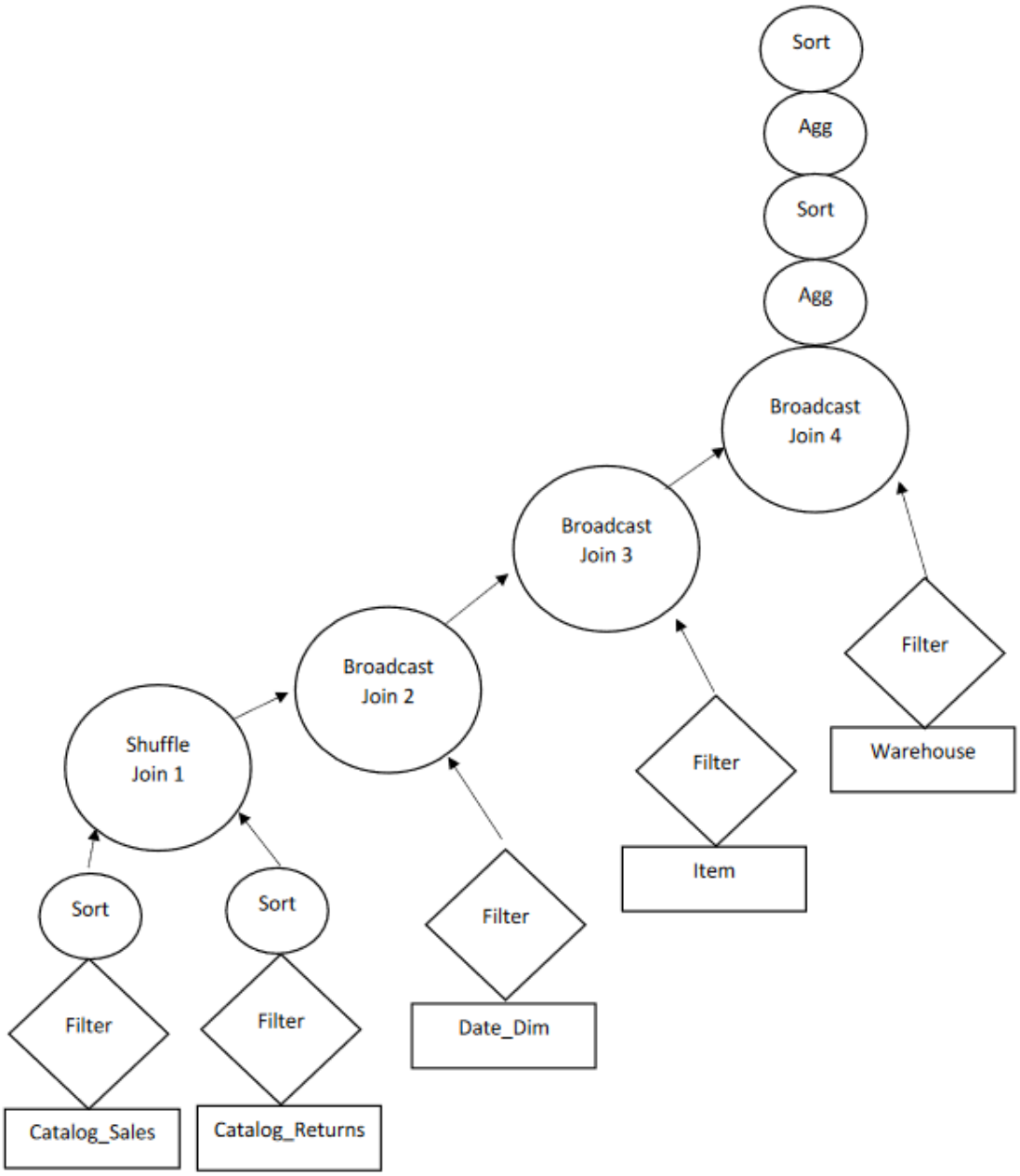

Figure 31 Εκτέλεση ερωτήματος 85 από την μηχανή Apache Hive

Από την εκτέλεση του ερωτήματος από την μηχανή Apache Hive που φαίνεται στην Εικόνα 31 παρατηρούμε ότι εφαρμόζεται Shuffle Join σε δυο fact πίνακες αφού αυτοί είναι μεγάλοι για να χρησιμοποιηθεί ο αλγόριθμος Broadcast Join , οι πίνακες Catalog_Sales και Catalog_Returns ταξινομούνται προτού εφαρμοστεί το Shuffle Join. Έπειτα τα δεδομένα υποβάλλονται ως είσοδο στο δεύτερο Join μαζί με τον μικρότερο πίνακα Date_Dim που διαμοιράζεται με την χρήση του δικτύου. Το τρίτο Join ακολουθεί και αυτό τον αλγόριθμο Broadcast Join και δέχεται ως είσοδο το δεύτερο Join και τον πίνακα Item που διαμοιράζεται με την χρήση του δικτύου. Το τέταρτο Join ακολουθεί τον ίδιο αλγόριθμο με το τρίτο Join και δέχεται ως είσοδο το τρίτο Join και τον πίνακα Warehouse που διαμοιράζεται με την χρήση του δικτύου έπειτα τα δεδομένα υποβάλλονται σε δυο συναρτήσεις συνάθροισης “sum” και ταξινομούνται.

Αποτίμηση των ερωτημάτων για τα δυο υπολογιστικά πλαίσια 5.6

Χρόνοι εκτέλεσης 5.6.α

Ερωτήματα	Apache Spark	Apache Hive
Ερώτημα 27	66.09	258.149
Ερώτημα 33	130.94	247.26
Ερώτημα 69	101.3	809.297
Ερώτημα 85	133.18	669.779

Table 1 Ο χρόνος εκτέλεσης κάθε ερωτήματος απο τις δυο μηχανές σε δευτερόλεπτα

Όπως βλέπουμε από τον πίνακα 1 η μηχανή Spark εκτέλεσε το ερώτημα 27 περίπου 4 φορές γρηγορότερα από την μηχανή Hive. Το ερώτημα 33 εκτελείται περίπου 2 φορές γρηγορότερα από την μηχανή Spark σε σχέση με την εκτέλεση της μηχανής Hive. Το ερώτημα 69 εκτελείται από την μηχανή Spark γρηγορότερα κατά περίπου 8 φορές σε σχέση με την εκτέλεση της μηχανής Hive. Τέλος το ερώτημα 85 εκτελείται κατά περίπου 5 φορές γρηγορότερα από την μηχανή Spark σε σχέση με την μηχανή Hive.

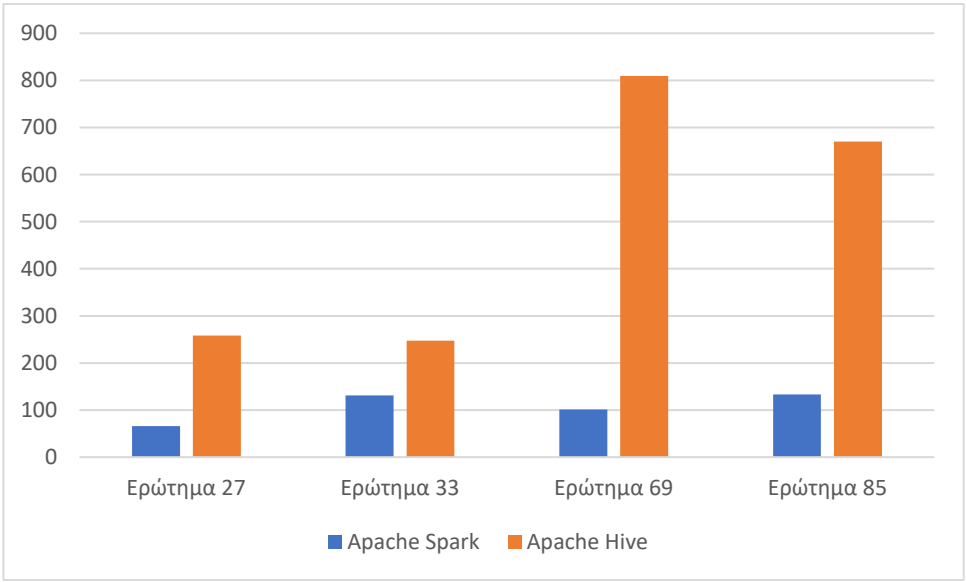

Figure 32 Εικονική απεικόνιση των χρόνων εκτέλεσης των ερωτημάτων για τα δυο υπολογιστικά πλαίσια

Στην εικόνα 32 φαίνεται μια πιο εύπεπτη απεικόνιση των χρονικών διαφορών στην εκτέλεση των ερωτημάτων από τα δυο υπολογιστικά πλαίσια.

ΚΕΦΑΛΑΙΟ 6 Συμπεράσματα

Παρατηρώντας τα αποτελέσματα τον επιλεγμένο ερωτημάτων από το κεφάλαιο 5, παρατηρούμε ότι χρησιμοποιώντας την μηχανή Apache Spark και την Spark-SQL έχουμε καλύτερα αποτελέσματα από ότι εάν χρησιμοποιήσουμε το υπολογιστικό πλαίσιο του Hive με μηχανή εκτέλεσης το Spark (Hive on Spark) αφού χρειάζεται περισσότερο χρόνο για την εκτέλεση των ίδιων ερωτημάτων. Η συστοιχία υπολογιστών στον οποίο έγιναν οι εκτελέσεις των ερωτημάτων μπορεί να θεωρηθεί μικρός αφού περιέχει 3 κόμβους από τους οποίους όπως αναλύσαμε στο κεφάλαιο 5 έχουν τον ίδιο αριθμό πόρων. Κάθε κόμβος λοιπόν περιέχει 8 GB κύριας μνήμης που μπορούν να θεωρηθούν λίγα για συστήματα τέτοιου σκοπού. Ενδεικτικά οι κατασκευαστές του Apache Hive στο wiki για την ρύθμιση του υπολογιστικού πλαίσιού έτσι ώστε αυτό να χρησιμοποιεί ως μηχανή εκτέλεσης το Apache Spark, δίνουν ως παράδειγμα στην κατανομή των πόρων του συστήματος [12] μια συστοιχία υπολογιστών με 10 κόμβους και 64GB κύρια μνήμη για κάθε κόμβο. Η δικιά μας πειραματική συστοιχία υπολογιστών είναι πολύ μικρότερη από το προτεινόμενο. Έτσι μπορούμε να καταλάβουμε ότι η Spark-SQL έχει καλύτερες επιδόσεις σε μικρότερα συστοιχίες υπολογιστών από ότι το Hive on Spark.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Floratou, U. F. Minhas, and F. Özcan, "SQL-on-Hadoop: full circle back to shared-nothing database architectures," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1295–1306, Aug. 2014, doi: 10.14778/2732977.2733002.

[2] E. Kassela, N. Provatas, I. Konstantinou, A. Floratou, and N. Koziris, "General-Purpose vs. Specialized Data Analytics Systems: A Game of ML SQL Thrones," in *2019 IEEE International Conference on Big Data (Big Data)*, Dec. 2019, pp. 317–326, doi: 10.1109/BigData47090.2019.9006412.

[3] "TPC-DS Homepage." http://www.tpc.org/tpcds/ (accessed Jan. 08, 2021).

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.

[5] "A comparison of join algorithms for log processing in MaPreduce | Proceedings of the 2010 ACM SIGMOD International Conference on Management of data." https://dl.acm.org/doi/abs/10.1145/1807167.1807273 (accessed Jan. 06, 2021).

[6] "Cluster Mode Overview - Spark 3.0.1 Documentation." https://spark.apache.org/docs/latest/cluster-overview.html (accessed Jan. 09, 2021).

[7] M. Armbrust *et al.*, "Spark SQL: Relational Data Processing in Spark," p. 12.

[8] "Hive - a petabyte scale data warehouse using Hadoop - IEEE Conference Publication." https://ieeexplore.ieee.org/document/5447738 (accessed Jan. 09, 2021).

[9] E. Begoli, J. C. Rodríguez, J. Hyde, M. J. Mior, and D. Lemire, "Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources," *Proceedings of the 2018 International Conference on Management of Data*, pp. 221–230, May 2018, doi: 10.1145/3183713.3190662.

[10] "Cost-based optimization in Hive - Apache Hive - Apache Software Foundation." https://cwiki.apache.org/confluence/display/Hive/Cost-based+optimization+in+Hive (accessed Jan. 09, 2021).

[11] "Home | ~okeanos IAAS." https://okeanos.grnet.gr/home/ (accessed Jan. 09, 2021).

[12] "Hive on Spark: Getting Started - Apache Hive - Apache Software Foundation." https://cwiki.apache.org/confluence/display/Hive/Hive+on+Spark%3A+Getting+Started# (accessed Jan. 09, 2021).