Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση

Delta Lake, Apache Iceberg και Apache Hudi: Συγκριτική Ανάλυση και Πειραματική Αποτίμηση Συστημάτων Αποθήκευσης Δεδομένων

Διπλωματική Εργασία

του

ΙΩΑΝΝΗ Γ. ΒΟΓΚΑ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ

Αθήνα, Ιούνιος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση

Delta Lake, Apache Iceberg και Apache Hudi: Συγκριτική Ανάλυση και Πειραματική Αποτίμηση Συστημάτων Αποθήκευσης Δεδομένων

Διπλωματική Εργασία

του

ΙΩΑΝΝΗ Γ. ΒΟΓΚΑ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Ιουλίου 2025.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
Νεκτάριος Κοζύρης	Ιωάννης Κωνσταντίνου	Τσουμάκος Δημήτριος
Καθηγητής ΕΜΠ	Αναπληρωτής Καθηγητής Παν. Θεσσαλίας	Αναπληρωτής Καθηγητής ΕΜΠ

Αθήνα, Ιούνιος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

. . . . . .
Ιωάννης Βόγκας

4η Ιουλίου 2025

Περίληψη

Καθώς οι ανάγκες αποθήκευσης και επεξεργασίας δεδομένων αυξάνονται, έχουν εμφανιστεί νέες τεχνολογίες που επιχειρούν να ξεπεράσουν τους περιορισμούς των παραδοσιακών αποθηκών δεδομένων, ενσωματώνοντας πιο αποδοτικά μοντέλα αποθήκευσης και προηγμένες λειτουργίες διαχείρισης. Η παρούσα διπλωματική εργασία επικεντρώνεται στη μελέτη και συγκριτική αξιολόγηση τριών σύγχρονων συστημάτων αποθήκευσης δεδομένων, τα οποία στοχεύουν να συνδυάσουν την αποδοτικότητα της αποθήκευσης σε στήλες με δυνατότητες τροποποίησης των δεδομένων και διαχείρισης ιστορικότητας. Πιο συγκεκριμένα, εξετάζονται τα συστήματα Delta Lake, Apache Iceberg και Apache Hudi, τα οποία σχεδιάστηκαν με σκοπό να ξεπεράσουν τους περιορισμούς τόσο των παραδοσιακών αποθηκών δεδομένων όσο και των μορφών αποθήκευσης ανά στήλη. Αρχικά, πραγματοποιήθηκε θεωρητική ανάλυση της αρχιτεκτονικής και των βασικών χαρακτηριστικών του κάθε συστήματος. Στη συνέχεια, υλοποιήθηκε μία πειραματική αποτίμηση της απόδοσης των συστημάτων, με τη χρήση του εργαλείου LST-Bench, σε διαφορετικά workloads που προσομοιώνουν ρεαλιστικά σενάρια χρήσης. Τα πειράματα υλοποιήθηκαν σε ένα κατανεμημένο περιβάλλον, στο οποίο χρησιμοποιήθηκαν οι τεχνολογίες Apache Spark για την επεξεργασία και Apache Hadoop για την αποθήκευση των δεδομένων. Τα αποτελέσματα προσφέρουν χρήσιμα συμπεράσματα σχετικά με τις επιδόσεις, τα πλεονεκτήματα και τους περιορισμούς του κάθε συστήματος, αναδεικνύοντας σε ποιες περιπτώσεις το καθένα αποδίδει καλύτερα. Έτσι, η εργασία προσφέρει πρακτική καθοδήγηση για την επιλογή του κατάλληλου συστήματος, ανάλογα με τις ανάγκες και τον τύπο της εφαρμογής.

Λέξεις Κλειδιά

Συστήματα Αποθήκευσης Δεδομένων, Delta Lake, Apache Iceberg, Apache Hudi, Log-Structured Tables, Apache Spark, Apache Hadoop, LST-Bench, Time-Travel ερωτήματα

Abstract

As the need for efficient data storage and processing continues to grow, new technologies have emerged to address the limitations of traditional data warehouses by introducing more effective storage models and advanced data management capabilities. This thesis focuses on the analysis and comparative evaluation of three modern data storage systems, Delta Lake, Apache Iceberg, and Apache Hudi, which aim to combine the performance advantages of columnar storage with features for data updates and version tracking. These systems are designed to overcome the drawbacks of both traditional data warehouses and columnar storage formats. The study begins with a theoretical overview of the architecture and main characteristics of each system. This is followed by a performance evaluation using the LST-Bench tool, with a series of workloads that simulate real-world usage scenarios. The experiments were executed in a distributed environment where Apache Spark was used for data processing and Apache Hadoop for data storage. The results offer useful insights into the performance, strengths, and limitations of each solution, showing under which conditions each performs best. Overall, this thesis provides practical guidance for choosing the most suitable system depending on specific needs and workload patterns.

Λέξεις Κλειδιά

Data Storage Systems, Delta Lake, Apache Iceberg, Apache Hudi, Log-Structured Tables, Apache Spark, Apache Hadoop, LST-Bench, Time-Travel Queries

στους γονείς μου

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. Ιωάννη Κωνσταντίνου για την καθοδήγηση και την πολύτιμη υποστήριξή του κατά την εκπόνηση της διπλωματικής μου εργασίας, καθώς και για την ιδέα και την έμπνευση που μου έδωσε με την πρόταση του θέματος. Επίσης, ευχαριστώ ιδιαίτερα τον επιβλέποντα καθηγητή της διπλωματικής μου, κ. Νεκτάριο Κοζύρη. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για την αδιάκοπη υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Αθήνα, Ιούνιος 2025

Ιωάννης Βόγκας

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του Διατμηματικού Προγράμματος Μεταπτυχιακών Σπουδών «Επιστήμη Δεδομένων και Μηχανική Μάθηση» της Σχολής ΗΜΜΥ του Εθνικού Μετσόβιου Πολυτεχνείου. Η εργασία πραγματοποιήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων (Computing Systems Laboratory) του ΕΜΠ.

Κεφάλαιο 1

Εισαγωγή

Καθώς τα σύγχρονα σύνολα δεδομένων αυξάνονται τόσο σε μέγεθος όσο και σε πολυπλοκότητα, η ανάγκη για αποδοτική διαχείριση μεγάλων όγκων δεδομένων γίνεται ολοένα και πιο επιτακτική. Ένας καθοριστικός παράγοντας για την απόδοση μιας αποθήκης δεδομένων είναι η μορφή αποθήκευσης που επιλέγεται, καθώς αυτή επηρεάζει τον τρόπο που τα δεδομένα οργανώνονται, ανακτώνται και υποβάλλονται σε επεξεργασία [2]. Οι τεχνολογικές εξελίξεις στον χώρο της αποθήκευσης και επεξεργασίας δεδομένων έχουν οδηγήσει στην εμφάνιση νέων μοντέλων που επιχειρούν να συνδυάσουν την απόδοση, την ευελιξία, την κλιμακωσιμότητα και τη δυνατότητα παρακολούθησης αλλαγών στα δεδομένα (Change Data Capture - CDC) [3].

1.1 Αντικείμενο της διπλωματικής

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η μελέτη και η πειραματική αξιολόγηση της απόδοσης σύγχρονων και καινοτόμων συστημάτων αποθήκευσης τύπου Log-Structured Tables (LSTs), και συγκεκριμένα των Delta Lake, Apache Iceberg και Apache Hudi, τα οποία επιχειρούν να ξεπεράσουν βασικούς περιορισμούς των παραδοσιακών προσεγγίσεων αποθήκευσης δεδομένων [1]. Πιο συγκεκριμένα, αναλύονται οι διαφορετικές σχεδιαστικές υλοποιήσεις των συστημάτων αυτών και συγκρίνονται οι επιδόσεις τους μέσω της εκτέλεσης διαφορετικών πειραμάτων, τα οποία προσομοιώνουν ρεαλιστικά σενάρια χρήσης. Η συγκριτική αυτή μελέτη στοχεύει στην ανάδειξη τόσο των πλεονεκτημάτων όσο και των περιορισμών κάθε συστήματος, παρέχοντας χρήσιμα συμπεράσματα για την καταλληλότητά τους ανάλογα με το σενάριο εφαρμογής.

1.2 Οργάνωση του τόμου

Η εργασία διακριτοποιείται και παρουσιάζεται σε έξι κεφάλαια. Μετά το παρόν εισαγωγικό κεφάλαιο, ακολουθεί το θεωρητικό υπόβαθρο της μελέτης, στο οποίο αναφέρονται διαφορετικές μορφές αποθήκευσης δεδομένων και αναλύονται τα τρία LST συστήματα που μελετώνται στην παρούσα εργασία. Στο τρίτο κεφάλαιο περιγράφεται αναλυτικά η μεθοδολογία της έρευνας. Παρουσιάζονται το benchmark πάνω στο οποίο βασίστηκαν τα πειράματα, τα δεδομένα που χρησιμοποιήθηκαν, οι μετρικές αξιολόγησης, καθώς και το υπολογιστικό περιβάλλον στο οποίο εκτελέστηκαν τα πειράματα. Στο τέταρτο κεφάλαιο παρουσιάζονται τα αποτελέσματα

των πειραμάτων και η ερμηνεία τους. Στο πέμπτο πραγματοποιείται μία συνολική σύνοψη των κύριων ευρημάτων, καθώς και συζήτηση αυτών. Το έκτο και τελευταίο κεφάλαιο περιλαμβάνει τα γενικά συμπεράσματα της μελέτης και πιθανές μελλοντικές επεκτάσεις της παρούσας εργασίας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Στο παρόν κεφάλαιο εξετάζονται οι βασικές έννοιες που αποτελούν το θεωρητικό υπόβαθρο της μελέτης. Αρχικά, γίνεται αναφορά σε παραδοσιακές μορφές αποθήκευσης (storage formats), στα πλεονεκτήματά τους, αλλά και στους περιορισμούς που παρουσιάζουν. Στη συνέχεια, περιγράφονται τα Log-Structured Tables (LSTs) ως μια σύγχρονη προσέγγιση που επιδιώκει να αντιμετωπίσει αυτά τα ζητήματα. Αναλύονται τρεις βασικές υλοποιήσεις τους, το Delta Lake, το Apache Iceberg και το Apache Hudi, με στόχο τη συγκριτική αξιολόγηση των χαρακτηριστικών και των δυνατοτήτων τους. Τέλος, γίνεται αναφορά σε σχετικές έρευνες και προσδιορίζεται η συμβολή της παρούσας εργασίας στο πεδίο.

2.1 Εξέλιξη μορφών αποθήκευσης δεδομένων

Τα παραδοσιακά συστήματα σχεσιακών βάσεων δεδομένων, τα οποία χρησιμοποιούν row-based αποθήκευση, συναντούν όλο και περισσότερες δυσκολίες στο να ανταποκριθούν στις απαιτήσεις των σύγχρονων αποθηκών δεδομένων (data warehouses). Αντίθετα, οι μορφές αποθήκευσης με βάση τις στήλες (columnar storage) έχουν κερδίσει έδαφος [4]. Σε αυτό το μοντέλο, τα δεδομένα οργανώνονται κατά στήλη αντί για σειρά, επιτρέποντας πιο αποδοτική ανάκτηση πληροφοριών και βελτιώνοντας σημαντικά την απόδοση, κυρίως, των read-heavy ερωτημάτων. Δύο από τις πιο διαδεδομένες μορφές αποθήκευσης με βάση τις στήλες είναι το Parquet [5] και το ORC (Optimized Row Columnar) [6], τα οποία έχουν καθιερωθεί ως πρότυπα σε σύγχρονες αποθήκες δεδομένων [2]. Αυτές οι μορφές αποθήκευσης είναι σχεδιασμένες ώστε, μόλις δημιουργηθούν, να παραμένουν αμετάβλητες και να επιτρέπουν μόνο λειτουργίες ανάγνωσης, διασφαλίζοντας αποδοτική αποθήκευση και ανάκτηση δεδομένων σε δομή στηλών. Αν και είναι ιδιαίτερα αποδοτικές για αναλυτικές εργασίες (OLAP), δεν είναι εξίσου κατάλληλες για OLTP εργασίες, όπου χρείαζεται γρήγορη εισαγωγή, ενημέρωση και διαγραφή δεδομένων [1]. Σε αυτό το πλαίσιο, τα Log-Structured Tables (LSTs) προσφέρουν μια πρακτική λύση, καθώς υποστηρίζουν τροποποιήσεις στα δεδομένα όπως ενημερώσεις και διαγραφές, χωρίς να χάνουν τα πλεονεκτήματα της αποθήκευσης ανά στήλη.

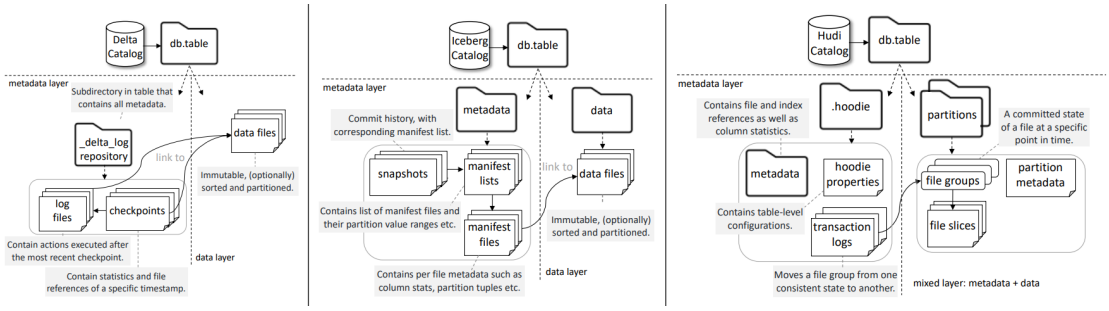
2.2 Ανάλυση των συστημάτων Log-Structured Tables (LSTs)

Τα Log-Structured Tables (LSTs) έχουν αναδειχθεί ως μια καινοτόμος προσέγγιση στην αποθήκευση και διαχείριση δεδομένων, και χρησιμοποιούνται ευρέως σε εφαρμογές μεγάλων δε-

δομένων. Όπως έχει αναφερθεί και παραπάνω, οι τρείς πιο γνωστές υλοποιήσεις τους είναι το Delta Lake [7], το Apache Iceberg [8] και το Apache Hudi [9]. Η κύρια καινοτομία τους είναι η προσθήκη ενός επιπέδου μεταδεδομένων πάνω από τα αμετάβλητα αρχεία στηλών, επιτρέποντας την οργάνωση πολλαπλών εκδόσεων των πινάκων και τον καθορισμό του τρόπου με τον οποίο οι μηχανές επεξεργασίας δεδομένων αλληλεπιδρούν με αυτά. Δηλαδή, τα LSTs χρησιμοποιούν αμετάβλητα αρχεία δεδομένων που οργανώνονται σε στήλες (π.χ. Parquet) και διατηρούν ένα αρχείο καταγραφής αλλαγών, αντί να τροποποιούν τα δεδομένα στη θέση τους. Κάθε φορά που γίνεται μια τροποποίηση στα δεδομένα, δεν αντικαθίστανται οι υπάρχουσες εγγραφές, αλλά δημιουργείται μια νέα έκδοση του πίνακα και οι αλλαγές αυτές καταγράφονται σε ειδικά αρχεία στο επίπεδο των μεταδεδομένων [1]. Αντίθετα, στα παραδοσιακά data warehouses τα δεδομένα αποθηκεύονται σε σειρές και ενημερώνονται απευθείας.

Ένα από τα σημαντικότερα πλεονεκτήματα των Log-Structured Tables (LSTs) είναι η υποστήριξη του Multi-Version Concurrency Control (MVCC). Το MVCC επιτρέπει την ταυτόχρονη ανάγνωση και καταχώριση δεδομένων στους πίνακες, χωρίς συγκρούσεις. Παράλληλα, επειδή τα αρχεία με τις παλαιότερες εκδόσεις των πινάκων διατηρούνται, είναι δυνατή η εκτέλεση time-travel λειτουργιών, επιτρέπουν στους χρήστες να έχουν την κατάσταση των πινάκων όπως αυτή ήταν σε μια συγκεκριμένη χρονική στιγμή στο παρελθόν [10]. Παρόλο που το MVCC προσφέρει σημαντικά πλεονεκτήματα, συνοδεύεται και από ορισμένα μειονεκτήματα. Ένα από τα βασικότερα είναι η αύξηση του χρόνου εκτέλεσης των queries, κυρίως των read queries, λόγω των πολλών αρχείων δεδομένων που δημιουργούνται με την πάροδο του χρόνου [11].

Οι τρεις βασικές υλοποιήσεις των LSTs (Delta, Iceberg, Hudi) διαφέρουν μεταξύ τους σε συγκεκριμένα χαρακτηριστικά, όπως στον τρόπο αποθήκευσης και διαχείρισης των μεταδεδομένων, στους μηχανισμούς ελέγχου (controls) και τις βελτιστοποιήσεις (optimizations) που εφαρμόζουν για τη βελτίωση της απόδοσης, καθώς και στον τρόπο αλληλεπίδρασής τους με τις μηχανές επεξεργασίας δεδομένων. Στις Ενότητες 2.2.1-2.2.3 περιγράφεται αναλυτικά ο τρόπος λειτουργίας τους, ενώ στον Πίνακα 2.1 συνοψίζονται οι βασικές διαφορές τους. Τέλος, στο Σχήμα 2.1 αποτυπώνονται διαγραμματικά οι κύριες αρχές τους.



Delta Iceberg Hudi

Σχήμα 2.1: *Δομή των LSTs (πηγή: [1])*

2.2.1 Delta Lake

Στο Delta Lake, κάθε τροποποίηση του πίνακα καταγράφεται σε ένα αρχείο καταγραφής (log file) με ένα μοναδικό αναγνωριστικό (ID). Αυτό το αρχείο περιλαμβάνει μια σειρά ενερ-

γειών που εφαρμόστηκαν στην προηγούμενη έκδοση του πίνακα, καθώς και στατιστικά στοιχεία, όπως οι ελάχιστες και μέγιστες τιμές για κάθε στήλη. Επιπλέον, το Delta Lake δημιουργεί checkpoint files, κάθε 10 συναλλαγές, τα οποία αποθηκεύουν τις ενέργειες που έγιναν μέχρι τότε στον πίνακα, ενώ έχουν και τα μεταδεδομένα του. Έτσι, επιτυγχάνεται γρήγορη πρόσβαση στο τελευταίο checkpoint, χωρίς να χρειάζεται να διαβαστούν όλα τα αρχεία με τις αλλαγές που έγιναν μέχρι τότε.

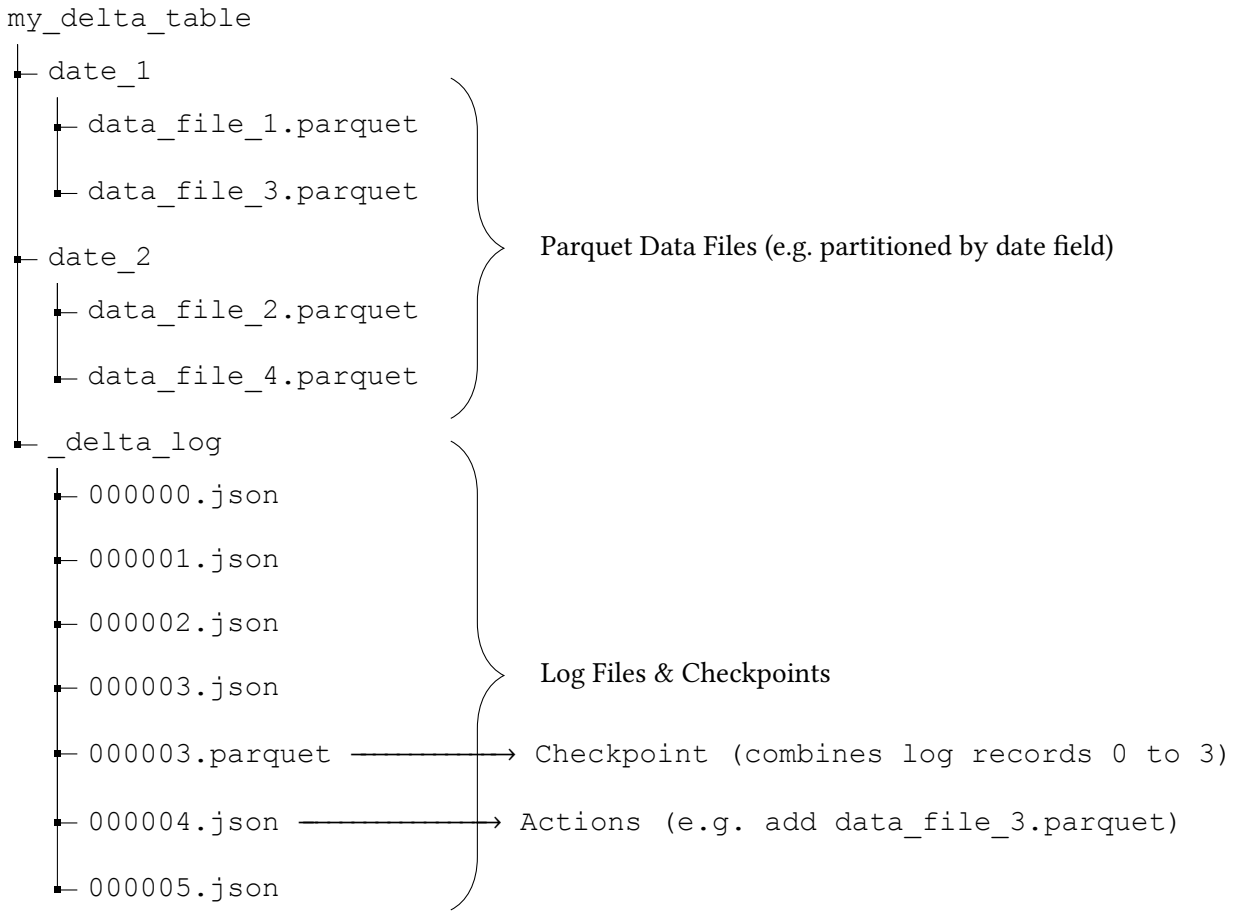
Το Delta Lake υιοθετεί έναν αυστηρό μηχανισμό ελέγχου ταυτόχρονης πρόσβασης, διασφαλίζοντας ότι οι τροποποιήσεις σε έναν πίνακα πραγματοποιούνται με σειριακό τρόπο, βάσει των αναγνωριστικών IDs που καταγράφονται στα logs. Δηλαδή, σε περίπτωση που δύο χρήστες επιχειρήσουν να τροποποιήσουν την ίδια εγγραφή ταυτόχρονα, μόνο η συναλλαγή που ολοκληρώνει πρώτη το commit θα καταχωρηθεί επιτυχώς, ενώ η δεύτερη θα απορριφθεί και θα απαιτηθεί η εκ νέου εκτέλεσή της με βάση την πιο πρόσφατη έκδοση των δεδομένων. Αντίθετα, κατά την ανάγνωση δεδομένων, το σύστημα εφαρμόζει το μοντέλο Snapshot Isolation, επιτρέποντας στους χρήστες να προσπελάζουν ένα σταθερό στιγμιότυπο του πίνακα όπως αυτό ήταν κατά την έναρξη της διεργασίας, ακόμη και αν άλλοι χρήστες πραγματοποιούν εγγραφές ταυτόχρονα. Με αυτό το τρόπο, επιτυγχάνεται υψηλή απόδοση στις αναγνώσεις, διατηρώντας παράλληλα την ακεραιότητα των εγγραφών μέσω αυστηρού ελέγχου σύγκρουσης κατά τη διαδικασία επικύρωσης των αλλαγών [12].

Το Delta Lake υποστηρίζει αποκλειστικά τη στρατηγική Copy-on-Write (CoW), η οποία είναι προτιμότερη για read-heavy εργασίες. Σε αυτήν την προσέγγιση, κάθε τροποποίηση δημιουργεί νέα αντίγραφα των αρχείων δεδομένων. Αντίθετα, στην Merge-on-Read (MoR), η οποία είναι προτιμότερη για write-heavy εργασίες, οι αλλαγές καταγράφονται σε ξεχωριστά αρχεία (delta files), τα οποία συγχωνεύονται κατά την ανάγνωση [1].

Για τη διαγραφή δεδομένων, το Delta λειτουργεί με τον ίδιο τρόπο όπως και για τις νέες εγγραφές, φτιάχνοντας ένα νέο αρχείο δεδομένων, το οποίο δεν περιέχει τις εγγραφές που επιλέχθηκαν να διαγραφούν από την προηγούμενη έκδοση του αρχείου. Από την έκδοση 2.4 και μετά, εισάγονται οι Deletion Vectors, επιτρέποντας την υλοποίηση διαγραφών χωρίς να τροποποιούνται τα αρχεία Parquet. Αντί για επανεγγραφή των αρχείων, οι διαγραμμένες γραμμές κωδικοποιούνται σε μορφή bitmap σε ξεχωριστές δομές, οι οποίες εφαρμόζονται κατά την ανάγνωση [13].

Το Delta Lake παρέχει μια σειρά από λειτουργίες συντήρησης πινάκων (Table Maintenance Operations), οι οποίες βελτιστοποιούν τη διαχείριση δεδομένων και την απόδοση των ερωτημάτων. Η συμπίεση αρχείων (Compaction), που εκτελείται μέσω της εντολής OPTIMIZE, συγχωνεύει μικρά αρχεία σε μεγαλύτερα, μειώνοντας την επιβάρυνση των μεταδεδομένων και βελτιώνοντας την αποδοτικότητα των αναζητήσεων. Από την έκδοση 3.1.0 και μετά, υποστηρίζεται αυτόματη συμπίεση (Auto Compaction), η οποία εκτελείται μετά από κάθε εγγραφή δεδομένων, εξαλείφοντας την ανάγκη για χειροκίνητη παρέμβαση. Επιπλέον, η λειτουργία Optimized Writes περιορίζει τη δημιουργία μικρών αρχείων κατά τη διαδικασία εγγραφής, ενώ το Log Compaction (διαθέσιμο από την έκδοση 3.0.0) μειώνει το μέγεθος του transaction log, μειώνοντας την ανάγκη για συχνά checkpoints [14].

Στο Σχήμα 2.2, φαίνεται ένα παράδειγμα αποθήκευσης δεδομένων σε Delta Lake, παρουσιάζοντας την οργάνωση των σχετικών αρχείων.



Σχήμα 2.2: *Παράδειγμα αποθήκευσης σε Delta Lake*

2.2.2 Apache Iceberg

Το Apache Iceberg υιοθετεί μια ιεραρχική οργάνωση των μεταδεδομένων. Για κάθε τροποποίηση του πίνακα, δημιουργείται ένα νέο metadata.json αρχείο, το οποίο αντικαθιστά ατομικά το προηγούμενο, διατηρώντας πληροφορίες για την τρέχουσα κατάσταση του πίνακα, το σχήμα του και το ιστορικό των snapshots. Κάθε snapshot, αναπαριστά μια συγκεκριμένη χρονική κατάσταση του πίνακα και αναφέρεται σε ένα Manifest List File, το οποίο περιέχει συνδέσμους προς τα αντίστοιχα Manifest Files. Τα Manifest Files καταγράφουν τις τοποθεσίες, τα χαρακτηριστικά και διάφορα άλλα μεταδεδομένα των data files που ανήκουν στο συγκεκριμένο snapshot. Επιπλέον, τα Manifest Files επαναχρησιμοποιούνται σε πολλαπλά snapshots, όταν δεν υπάρχει αλλαγή στα data files [8].

Ως προς την ταυτόχρονη πρόσβαση σε κάποιο πίνακα από πολλού χρήστες, το Iceberg έχει παρόμοιο μηχανισμό με αυτό του Delta Lake. Για την ανάγνωση των δεδομένων, οι χρήστες χρησιμοποιούν το snapshot που ήταν ενεργό τη στιγμή που φόρτωσαν τον πίνακα και δεν επηρεάζονται από ενδιάμεσες αλλαγές. Για την τροποποίηση των δεδομένων, το Iceberg υποστηρίζει παράλληλες εγγραφές μέσω του Optimistic Concurrency Control (OCC). Δηλαδή, κάθε χρήστης φορτώνει το τρέχον metadata.json, το οποίο περιέχει το ενεργό snapshot, και προετοιμάζει τις αλλαγές του υποθέτοντας ότι το metadata αρχείο δεν θα αλλάξει πριν από το commit. Αν ένας άλλος χρήστης έχει ήδη ενημερώσει το metadata αρχείο πριν από το commit και υπάρ-

χουν συγκρούσεις, η συναλλαγή αποτυγχάνει, και ο χρήστης πρέπει να διαβάσει ξανά το πιο πρόσφατο snapshot, να εφαρμόσει ξανά τις αλλαγές του και να επαναλάβει τη διαδικασία [15].

Το Iceberg υποστηρίζει τόσο τη στρατηγική Copy-on-Write (CoW) όσο και την Merge-on-Read (MoR), όπως αυτές περιγράφηκαν στην ενότητα 2.2.1.

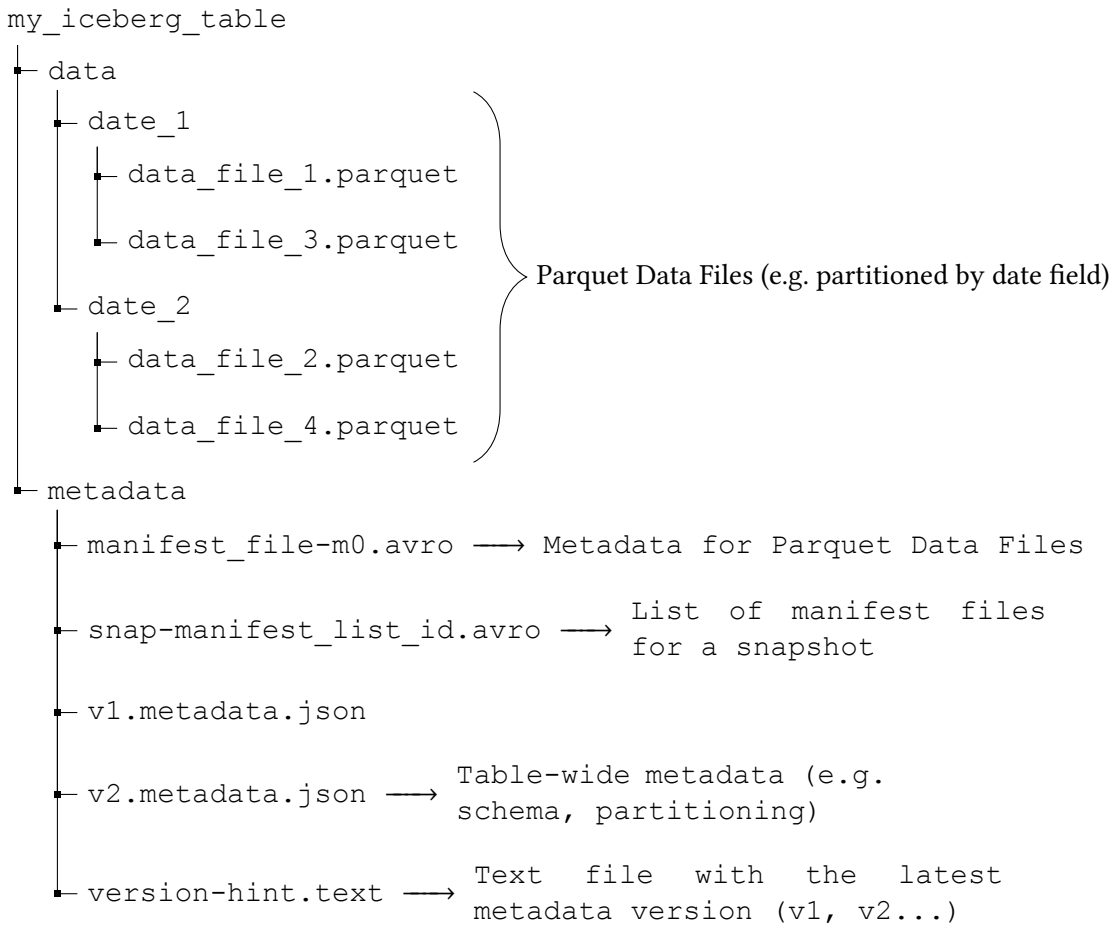
Για τη διαχείριση των διαγραφών δεδομένων, το Iceberg χρησιμοποιεί τις στρατηγικές Position Delete Files και Equality Delete Files. Η στρατηγική Position Delete Files βασίζεται στην καταγραφή της τοποθεσίας του αρχείου με τα δεδομένα και της θέσης των διαγραμμένων γραμμών μέσα σε αυτό. Από την άλλη, η στρατηγική Equality Delete Files βασίζεται στις τιμές των δεδομένων σε συγκεκριμένες στήλες, επιτρέποντας τη διαγραφή γραμμών με βάση την αντιστοιχία των τιμών τους με τα δεδομένα που αναφέρονται στο αρχείο διαγραφής. Από την Έκδοση 3 και μετά, προστέθηκε η στρατηγική Deletion Vectors, η οποία αποτελεί μια πιο αποδοτική μέθοδο διαγραφής γραμμών, καθώς επιτρέπει την κωδικοποίηση των διαγραμμένων θέσεων σε έναν πίνακα δυαδικών τιμών (bitmap). Στη συγκεκριμένη μέθοδο, κάθε σειρά που διαγράφεται αναπαρίσταται από ένα set bit στον πίνακα, κάτι που κάνει τις διαγραφές πιο αποδοτικές κατά την εκτέλεση [16].

Τέλος, υπάρχουν και εδώ λειτουργίες για τη διαχείριση και συντήρηση των πινάκων. Το compaction (RewriteDataFiles), όπως περιγράφηκε και στο Delta Lake, συγχωνεύει μικρά αρχεία δεδομένων σε μεγαλύτερα, μειώνοντας το metadata overhead και επιταχύνοντας τα queries. Παράλληλα, η διαγραφή παλαιών snapshots μέσω του μηχανισμού expireSnapshots αποτρέπει τη συσσώρευση περιττών εκδόσεων και διατηρεί το μέγεθος των μεταδεδομένων μικρό. Επιπλέον, η αφαίρεση ορφανών αρχείων (deleteOrphanFiles) εξασφαλίζει την εκκαθάριση μη χρησιμοποιούμενων δεδομένων που μπορεί να έχουν μείνει από αποτυχημένες διεργασίες. Τέλος, η διαγραφή παλιών metadata files μπορεί να αυτοματοποιηθεί, αποτρέποντας την ανεξέλεγκτη συσσώρευση αρχείων [17].

Στο Σχήμα 2.3, φαίνεται ένα παράδειγμα αποθήκευσης δεδομένων σε Apache Iceberg, παρουσιάζοντας την οργάνωση των σχετικών αρχείων.

2.2.3 Apache Hudi

Η αρχιτεκτονική του Apache Hudi βασίζεται σε τρία κύρια στοιχεία: ένα Timeline, ένα Metadata Table και κάποια File Groups & File Slices. Με το timeline καταγράφονται όλες τις ενέργειες που έχουν εκτελεστεί σε ένα πίνακα, σε αυστηρά χρονολογική σειρά. Χρησιμοποιώντας transaction logs, επιτυγχάνεται η ανακατασκευή προηγούμενων εκδόσεων των δεδομένων. Τα File Groups και File Slices είναι οι βασικές δομές οργάνωσης των αρχείων. Μέσα σε κάθε partition, τα δεδομένα αποθηκεύονται σε Base Files και Log Files (μόνο για την MoR στρατηγική), τα οποία οργανώνονται σε File Groups. Ένα File Group αποτελείται από πολλαπλές εκδόσεις των δεδομένων, που ονομάζονται File Slices. Κάθε File Slice περιλαμβάνει ένα Base File (Parquet) που περιέχει μια έκδοση των δεδομένων και ένα ή περισσότερα Log Files (μόνο για την MoR στρατηγική) που αποθηκεύουν τις διαδοχικές αλλαγές. Ο Metadata Table αποτελεί ένα ξεχωριστό Merge-on-Read Hudi πίνακα, σχεδιασμένο για να επιταχύνει τις λειτουργίες ανάκτησης μεταδεδομένων. Ουσιαστικά, λειτουργεί ως ένα αποδοτικό ευρετήριο αρχείων, αφού περιέχει πληροφορίες όπως διαδρομές αρχείων και ευρετηριασμένα αρχεία (indexed files), επιτρέποντας στο σύστημα να γνωρίζει πού βρίσκονται τα δεδομένα χωρίς να χρειάζεται να κάνει



Σχήμα 2.3: *Παράδειγμα αποθήκευσης σε Apache Iceberg*

πλήρη σάρωση όλων των αρχείων [9].

Ως προς την διαχείριση της ταυτόχρονης πρόσβασης, το Hudi εφαρμόζει snapshot isolation, διασφαλίζοντας ότι κάθε συναλλαγή διαβάζει μια συνεπή εκδοχή των δεδομένων όπως αυτά υπήρχαν κατά την έναρξή της, ανεξάρτητα από άλλες ταυτόχρονες τροποποιήσεις. Επιπλέον, επιτρέπει παράλληλες εγγραφές χωρίς αποκλεισμούς (locks), εφαρμόζοντας έλεγχο συγκρούσεων μόνο κατά τη πραγματοποίηση των αλλαγών (Optimistic Concurrency Control). Το Hudi μπορεί να υποστηρίξει μία διαφορετική προσέγγιση, αν υπάρξει σύγκρουση δύο εγγραφών, με την οποία προσπαθεί να συγχωνεύσει τις αλλαγές αντί να απορρίψει κάποιο commit, μέσω του Non-Blocking Concurrency Control (NBCC) [18]. Τέλος, υποστηρίζει τόσο τη στρατηγική Copy-on-Write (CoW) όσο και την Merge-on-Read (MoR), όπως αυτές περιγράφηκαν στην ενότητα 2.2.1.

Για τη διαγραφή δεδομένων, το Hudi υποστηρίζει δύο μεθόδους διαγραφών, τις soft deletes και τις hard deletes. Στην πρώτη περίπτωση, τα πεδία που επιλέγονται να διαγραφούν δεν αφαιρούνται πλήρως από τον πίνακα, αλλά παίρνουν την τιμή null, ενώ διατηρείται το κλειδί της συγκεκριμένης εγγραφής. Αντίθετα, στις hard διαγραφές, οι εγγραφές αφαιρούνται πλήρως από τα δεδομένα, στο νέο αρχείο που δημιουργείται. Από προεπιλογή, το Hudi υλοποιεί τη στρατηγική soft για τις διαγραφές [19].

Το Hudi παρέχει μια σειρά από λειτουργίες συντήρησης των πινάκων, οι οποίες διασφαλίζουν τη βέλτιστη απόδοση, τη διατήρηση του ιστορικού δεδομένων και τον έλεγχο του αποθηκευτικού κόστους. Το Compaction συγχωνεύει τα Log Files με το Base File (Parquet), μειώνοντας την καθυστέρηση των αναγνώσεων (Compaction γίνεται μόνο στους MoR πίνακες). Επιπλέον, η λειτουργία Clustering εστιάζει στην αναδιοργάνωση των δεδομένων για να βελτιώσει την τοπθεσία των δεδομένων και να επιτρέψει πιο αποδοτική εκτέλεση ερωτημάτων, συγχωνεύοντας μικρά αρχεία σε μεγαλύτερα. Η λειτουργία καθαρισμού (Cleaning) διαγράφει παλαιότερες εκδόσεις αρχείων που δεν χρειάζονται, βοηθώντας στη διαχείριση του χώρου αποθήκευσης. Επίσης, διαθέτει επίσης έναν μηχανισμό Rollback, ο οποίος εντοπίζει και διαγράφει μερικώς αποτυχημένες εγγραφές, αποτρέποντας την ανάγνωσή τους από τους χρήστες [20].

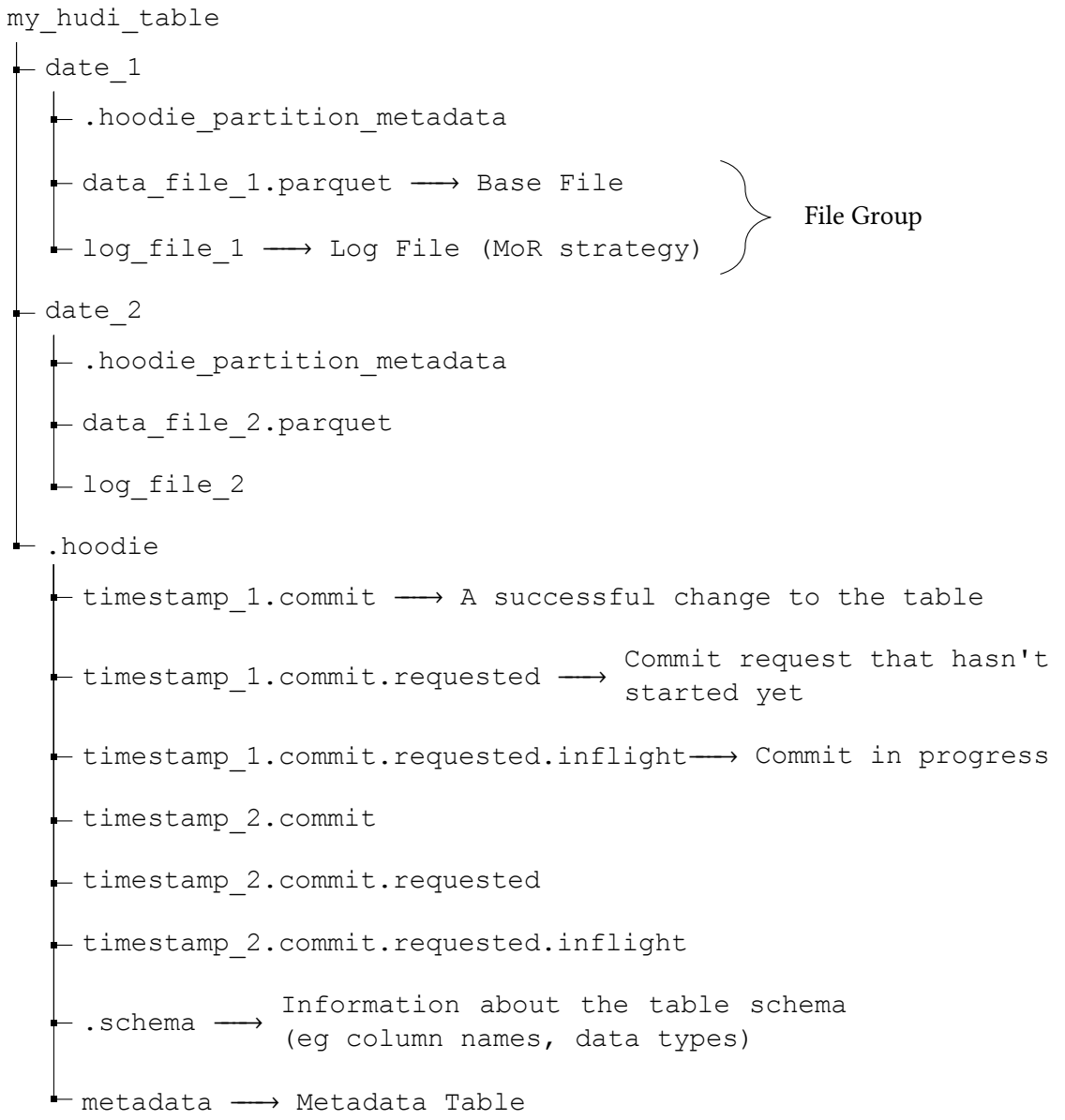
Στο Σχήμα 2.4, φαίνεται ένα παράδειγμα αποθήκευσης δεδομένων σε Apache Hudi, παρουσιάζοντας την οργάνωση των σχετικών αρχείων.

Χαρακτηριστικό	**Delta Lake**	**Apache Iceberg**	**Apache Hudi**
Διαχείριση Μεταδεδομένων	Αρχεία καταγραφής συναλλαγών (log files) & Checkpoints κάθε 10 συναλλαγές	Ιεραρχική διαχείριση μεταδεδομένων (metadata.json, Manifest Lists, Manifest Files)	Timeline (χρονολόγιο ενεργειών) & Metadata Table (ευρετήριο)
Έλεγχος ταυτόχρονης πρόσβασης	Optimistic Concurrency Control (OCC) & Σειριακές Εγγραφές & Snapshot Isolation στις αναγνώσεις	Optimistic Concurrency Control (OCC) & Σειριακές Εγγραφές & Snapshot Isolation στις αναγνώσεις	Optimistic Concurrency Control (OCC) & Non-Blocking Concurrency Control (NBCC) & Snapshot Isolation στις αναγνώσεις
Στρατηγικές εγγραφών	Copy-on-Write (CoW)	Copy-on-Write (CoW) & Merge-on-Read (MoR)	Copy-on-Write (CoW) & Merge-on-Read (MoR)
Στρατηγικές διαγραφών	Copy-on-Write (CoW), Deletion Vectors (v2.4 και μετά)	Position Delete Files, Equality Delete Files, Deletion Vectors (v3 και μετά)	Soft Deletes & Hard Deletes
Λειτουργίες συντήρησης πινάκων	Compaction (OPTIMIZE), Log Compaction, Optimized Writes κ.ά.	Compaction (RewriteDataFiles), Expiring Snapshots, Deleting Orphan Files κ.ά.	Compaction (μόνο σε MoR), Clustering, Cleaning, Rollback κ.ά.

Πίνακας 2.1: *Σύνοψη διαφορών των τριών LSTs*

2.3 Σχετικές έρευνες και συμβολή της παρούσας μελέτης

Η ανάλυση και η σύγκριση των Log-Structured Tables (LSTs) αποτελεί ένα ενεργό πεδίο έρευνας, με συνεχώς αυξανόμενο όγκο μελετών που εξετάζουν τις διαφορές τους τόσο σε θεω-



Σχήμα 2.4: *Παράδειγμα αποθήκευσης σε Apache Hudi*

ρητικό όσο και σε πειραματικό επίπεδο. Οι προσεγγίσεις διαφέρουν σημαντικά ανάλογα με τα benchmarks που χρησιμοποιούνται, το μέγεθος και τη σύνθεση των δεδομένων, μετρικές αξιολόγησης και τις υπολογιστικές υποδομές στις οποίες διεξάγονται τα πειράματα. Στην ενότητα αυτή, αναφέρονται σημαντικές μελέτες που έχουν διεξαχθεί για τη σύγκριση των LSTs, αναλύοντας τις μεθοδολογίες που χρησιμοποιήθηκαν και τους παράγοντες που επηρεάζουν τα αποτελέσματά τους, με στόχο μια πιο ολοκληρωμένη κατανόηση του συγκεκριμένου πεδίου έρευνας. Επιπλέον, εξετάζεται η σχέση της παρούσας διπλωματικής εργασίας με την υπάρχουσα βιβλιογραφία, αναδεικνύοντας τη συμβολή της στην κατανόηση και εξέλιξη της έρευνας γύρω από τα LSTs.

Σε μία έρευνα των Jain et al. [21], μελετήθηκαν και συγκρίθηκαν τα LSTs εστιάζοντας στις αρχιτεκτονικές τους διαφορές, καθώς και στην απόδοσή τους σε κάποια σενάρια χρήσης. Οι

πειραματικές δοκιμές πραγματοποιήθηκαν σε Apache Spark και χρησιμιποήθηκε το LHBench benchmark. Τα αποτελέσματα της μελέτης έδειξαν ότι το Delta Lake είναι το ταχύτερο στην εκτέλεση ερωτημάτων, ενώ το Hudi είναι σημαντικά πιο αργό στη φόρτωση δεδομένων. Ακόμα, φάνηκε ότι το Hudi-MoR είναι ταχύτερο από το Hudi-CoW κατά τη διάρκεια των ενημερώσεων των πινάκων, ωστόσο το Hudi-MoR είναι πιο αργό από το Hudi-CoW στα read-queries. Αντίστοιχα συμπεράσματα έβγαλαν και για την σύγκριση μεταξύ των Iceberg-MoR και Iceberg-CoW. Το Delta Lake παρουσιάζει ανταγωνιστικές επιδόσεις τόσο στις ενημερώσεις των πινάκων όσο και στα read-queries.

Πολλές τεχνικές αναφορές σε blogs εξετάζουν πειραματικές αξιολογήσεις και συγκρίσεις μεταξύ των Delta Lake, Apache Iceberg και Apache Hudi. Το benchmark που χρησιμοποιείται κυρίως για αυτές τις συγκρίσεις είναι το TPC-DS. Συνοψίζοντας τα αποτελέσματα των σχετικών αναφορών, διαπιστώνεται ότι το Delta Lake υπερέχει τόσο στη φόρτωση δεδομένων όσο και στην εκτέλεση queries, το Hudi έχει με διαφορά τη χαμηλότερη απόδοση στη φόρτωση δεδομένων, ενώ έχουν διερευνηθεί και τρόποι για την μείωση του απαιτούμενου χρόνου [22] [23]. Ως προς τις θεωρτικες προσεγγισεις, το Delta Lake προτιμάται για τη διαχείριση δεδομένων σε data lakes, data pipelines και για την επεξεργασία δεδομένων σε μεγάλες παρτίδες (Batch Processing), ιδιαίτερα σε περιβάλλοντα όπου χρησιμοποιείται το Spark. Από την άλλη, το Iceberg θεωρείται καταλληλότερο για αναλύσεις μεγάλης κλίμακας δεδομένων (large-scale analytics), σε περιπτώσεις όπου τα δεδομένα αποθηκεύονται σε ένα data warehouse και όπου υπάρχει ανάγκη για συχνές αλλαγές στη δομή των δεδομένων (schema evolution). Τέλος, το Hudi ενδείκνυται για σενάρια που απαιτούν συχνές ενημερώσεις και διαγραφές δεδομένων, καθώς και για εφαρμογές που βασίζονται σε real-time επεξεργασία δεδομένων [24] [25].

Σε πρόσφατη έρευνα, οι Jesús Camacho-Rodríguez et al. [1] υποστηρίζουν ότι τα παραδοσιακά benchmarks που χρησιμοποιούνται, όπως το TPC-DS, δεν λαμβάνουν υπόψη σημαντικά χαρακτηριστικά για την αξιολόγηση των LSTs (π.χ. επίδραση της συσσώρευσης αρχείων, συντήρηση πινάκων, ταυτόχρονη πρόσβαση στους πίνακες) και για αυτό ανέπτυξαν ένα νέο benchmark, το LST-Bench, το οποίο είναι ειδικά σχεδιασμένο για την αξιολόγηση των LSTs σε πραγματικά workloads. Τα κύρια ευρήματα της έρευνάς τους δείχνουν ότι η συσσώρευση αρχείων δεδομένων υποβαθμίζει την απόδοση των LSTs, ο τρόπος αποθήκευσης (Copy-on-Write και Merge-on-Read) έχει σημαντικές επιπτώσεις στην απόδοση των αναγνώσεων και των εγγραφών σε έναν πίνακα και οι λειτουργίες συντήρησης είναι κρίσιμες για τη διατήρηση σταθερού χρόνου απόκρισης των queries. Τέλος, εκτελώντας τα πειράματα τόσο σε Spark όσο και σε Trino, διαπίστωσαν ότι η απόδοση των LSTs εξαρτάται σημαντικά από τη μηχανή επεξεργασίας δεδομένων που χρησιμοποιείται.

Όπως προκύπτει από τη σχετική βιβλιογραφία, για τη σύγκριση των LSTs χρησιμοποιούνται διάφορα benchmarks, με κυριότερο το TPC-DS, τα οποία, ωστόσο, παρουσιάζουν σημαντικούς περιορισμούς στην αξιολόγηση των LSTs. Το πιο πρόσφατο LST-Bench φαίνεται να υπερβαίνει αυτούς τους περιορισμούς, καθώς αξιολογεί τα LSTs σε περισσότερες διαστάσεις, με πιο εκτεταμένες μετρικές και βαθύτερη σύγκριση των συστημάτων. Παρότι το LST-Bench αποτελεί μια πολλά υποσχόμενη προσέγγιση, δεν έχει ακόμη χρησιμοποιηθεί ευρέως στην έρευνα. Για τον λόγο αυτό, η παρούσα μελέτη αξιοποιεί το LST-Bench προκειμένου να διεξάγει μια πολυδιάστατη ανάλυση και σύγκριση των LSTs, καλύπτοντας τα κενά των προηγούμενων προσεγγίσεων.

Κεφάλαιο 3

Σχεδιασμός και Μεθοδολογία

Στο παρόν κεφάλαιο, παρουσιάζεται αναλυτικά ο σχεδιασμός και η μεθοδολογία των πειραμάτων που πραγματοποιήθηκαν για την αξιολόγηση της απόδοσης των Log-Structured Tables (LSTs). Για τον σκοπό της αξιολόγησης, χρησιμοποιήθηκε το εξειδικευμένο LST-Benchmark, το οποίο και περιγράφεται λεπτομερώς. Στην αρχή, εξετάζονται τα tasks και τα workloads που προτείνονται από το LST-Bench, τα οποία αποτέλεσαν τη βάση για τη διεξαγωγή των πειραμάτων της παρούσας μελέτης. Επίσης, γίνεται αναφορά στα δεδομένα που χρησιμοποιήθηκαν κατά την εκτέλεση των πειραμάτων. Ακολουθεί μια παρουσίαση των μετρικών που χρησιμοποιήθηκαν για τη σύγκριση των τριών LSTs, καθώς και ο τρόπος καταγραφής και ανάλυσης αυτών των μετρήσεων. Το κεφάλαιο ολοκληρώνεται με την περιγραφή του κατανεμημένου περιβάλλοντος και των τεχνολογιών που χρησιμοποιήθηκαν για τα πειράματα.

3.1 LST-Benchmark

Το LST-Bench [26] είναι ένα εξειδικευμένο benchmarking framework που επιτρέπει την αξιολόγηση της απόδοσης των Log-Structured Tables (LSTs), τα οποία χρησιμοποιούνται ως μορφές αποθήκευσης δεδομένων σε data lakes. Ενώ τα περισσότερα υπάρχοντα benchmarks, όπως το TPC-DS, επικεντρώνονται σε γενικές αξιολογήσεις OLAP συστημάτων, το LST-Bench έχει σχεδιαστεί ειδικά για να αναδείξει κρίσιμες πτυχές των LSTs που δεν αποτυπώνονται σε προηγούμενα benchmarks. Πιο συγκεκριμένα, μέσω του LST-Bench, μπορούν να εξαχθούν συμπεράσματα σχετικά με: (i) τη σταθερότητα της απόδοσης των LSTs σε βάθος χρόνου, (ii) την επίδραση που έχουν οι συνεχείς ενημερώσεις και διαγραφές δεδομένων, (iii) τη σημαντικότητα λειτουργιών συντήρησης και βελτιστοποίησης των πινάκων, όπως η συμπίεση (compaction) αρχείων, καθώς και (iv) τη δυνατότητα ιστορικής ανάκτησης δεδομένων (time travel), έτσι όπως αυτά βρίσκονται σε κάποια συγκεκριμένη χρονική στιγμή στο παρελθόν.

Το LST-Bench χρησιμοποιεί το TPC-DS [27] ως βάση, διατηρώντας τον μηχανισμό δημιουργίας δεδομένων (data generator) και το προκαθορισμένο σύνολο ερωτημάτων του (query set). Παράλληλα, εισάγει νέες εργασίες (tasks) ειδικά σχεδιασμένες για την αξιολόγηση των LSTs, ενώ ακόμα προτείνει συγκεκριμένα πρότυπα φόρτου εργασίας (workloads) που έχουν αναπτυχθεί με σκοπό να αναδείξουν τις κρίσιμες πτυχές της απόδοσης και λειτουργικότητας των LSTs, όπως αυτές που συζητήθηκαν προηγουμένως. Επιπλέον, εκτός από τα προτεινόμενα workloads, το LST-Bench παρέχει στους χρήστες τη δυνατότητα να δημιουργήσουν προσαρμοσμένα workloads, σύμφωνα με τις εκάστοτε απαιτήσεις και τις ιδιαιτερότητες του συστήματος

που αξιολογούν [1].

3.1.1 Περιγραφή των εργασιών (tasks) και των σεναρίων χρήσης (workloads)

Η κατανόηση της μεθοδολογίας που ακολουθήθηκε για την εκτέλεση των πειραμάτων με το LST-Bench προϋποθέτει τη λεπτομερή αναφορά και περιγραφή των tasks που χρησιμοποιήθηκαν. Στην παρούσα ενότητα, παρουσιάζονται αναλυτικά τα tasks, ο ρόλος τους στη διαδικασία αξιολόγησης των LSTs και οι βασικές λειτουργίες που επιτελούν.

Τα tasks που χρησιμοποιήθηκαν στα πειράματα:

- **Load Task (L)**: Φόρτωση των αρχικών δεδομένων στους πίνακες για την εκτέλεση των πειραμάτων.
- **Single User Task (SU)**: Εκτέλεση μιας σειράς από 99 πολύπλοκα ερωτήματα ανάγνωσης δεδομένων (read-queries) σε τυχαία σειρά.
- **Data Maintenance Task (DM)**: Εκτέλεση λειτουργιών εισαγωγής και διαγραφής δεδομένων στους πίνακες.
- **Optimize Task (O)**: Εκτέλεση διαδικασιών συντήρησης στους πίνακες, όπως αναλύθηκαν στην Ενότητα 2. Πιο συγκεκριμένα, για το Delta Lake εκτελείται συγχνώνευση αρχείων μέσω της λειτουργίας OPTIMIZE, για το Apache Iceberg εφαρμόζεται η λειουργία RewriteDataFiles και για το Hudi υλοποιείται το Clustering.
- **Time Travel Task**: Εκτέλεση ερωτημάτων σε προηγούμενες χρονικές εκδόσεις των πινάκων. Εκτελούνται τα ίδια queries με το Single User Task (SU), αλλά αναφέρονται σε μια συγκεκριμένη χρονική στιγμή στο παρελθόν.

Τα τελευταία δύο tasks, το Optimize Task και το Time Travel Task, αποτελούν νέες προσθήκες στο LST-Bench, ενώ τα υπόλοιπα προέρχονται από το TPC-DS, στο οποίο βασίζεται το LST-Bench [1].

Το LST-Bench έχει σχεδιαστεί με ένα σύνολο από προτεινόμενα workloads, τα οποία προσομοιώνουν ρεαλιστικά σενάρια χρήσης και αποτελούνται από τα tasks που αναφέρθηκαν. Για την παρούσα μελέτη, χρησιμοποιήθηκαν όλα τα προτεινόμενα, από το benchmark, workloads, τα οποία αναλύονται στη συνέχεια και απεικονίζονται διαγραμματικά στο Σχήμα 3.1.

Workload 1 (W1): Μακροχρόνια Χρήση. Η διαδικασία του πειράματος περιλαμβάνει έξι διαδοχικές φάσεις Single User, κατά τις οποίες εκτελούνται τα 99 ερωτήματα ανάγνωσης (read queries), με κάθε φάση Single User να ακολουθείται από μια φάση συντήρησης δεδομένων (Data Maintenance), στην οποία πραγματοποιούνται λειτουργίες εισαγωγής και διαγραφής δεδομένων στους πίνακες. Η συγκεκριμένη διαμόρφωση καθιστά το workload κατάλληλο για την ανάλυση των LSTs σε συνθήκες μακροχρόνιας χρήσης, επιτρέποντας την εξαγωγή συμπερασμάτων σχετικά με την απόδοσή τους υπό διαρκώς μεταβαλλόμενα δεδομένα (Σχήμα 3.1α΄).

Workload 2 (W2): Συνεχείς τροποποιήσεις δεδομένων και λειτουργίες συντήρησης. Αυτό το workload χρησιμοποιείται για την αξιολόγηση των LSTs υπό συνθήκες συνεχών τροποποιήσεων των δεδομένων, οι οποίες διαφέρουν σε όγκο κάθε φορά, καθώς και για τη μέτρηση

της απόδοσης των λειτουργιών συντήρησης, όπως η συμπίεση (compaction). Περιλαμβάνει μια σειρά φάσεων Single User (SU) και Data Maintenance (DM), με την προσθήκη φάσεων Optimize (O) ανάμεσα στις Single User (SU) φάσεις (Σχήμα 3.1β΄).

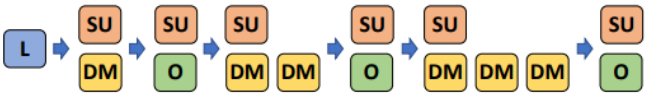
Workload 3 (W3): Παράλληλες εργασίες. Το W3 αξιολογεί την επίδραση της ταυτόχρονης εκτέλεσης αναγνώσεων, εγγραφών και συμπίεσης στους πίνακες δεδομένων. Συγκεκριμένα, η διαδικασία εκτέλεσης σε αυτό το workload ακολουθεί την ίδια μεθοδολογία που χρησιμοποιείται στο W2, με τη διαφορά ότι οι φάσεις SU εκτελούνται ταυτόχρονα με τις φάσεις DM και O (Σχήμα 3.1γ΄).

Workload 4 (W4): Time Travel ερωτήματα. Το W4 αποσκοπεί στην αξιολόγηση της δυνατότητας των LSTs να χειρίζονται ερωτήματα για προηγούμενες εκδόσεις δεδομένων. Στο πλαίσιο αυτού του workload, εκτελούνται πολλαπλές φάσεις Data Maintenance (DM) πάνω στα αρχικά δεδομένα. Μετά από κάθε φάση DM, ακολουθούν time-travel ερωτήματα, τα οποία εκτελούνται για κάθε προγενέστερη κατάσταση των δεδομένων (Σχήμα 3.1δ΄).

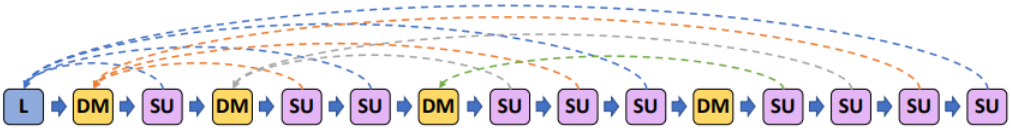
Workload 5 (W5): Ταυτόχρονες αναγνώσεις χρηστών. Το συγκεκριμένο workload εξετάζει την ταυτόχρονη εκτέλεση αναγνώσεων από πολλούς χρήστες. Στο πλαίσιο της παρούσας μελέτης, επιλέχθηκε η χρήση δύο χρηστών για την εκτέλεση των ερωτημάτων (Σχήμα 3.1ε΄).

(α΄) ***W1: Μακροχρόνια Χρήση***

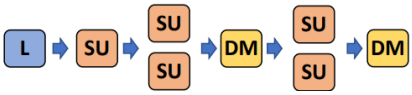
(β΄) ***W2: Συνεχείς τροποποιήσεις δεδομένων και λειτουργίες συντήρησης***



(γ΄) ***W3: Παράλληλες εργασίες***



(δ΄) ***W4: Time Travel ερωτήματα***



(ε΄) ***W5: Ταυτόχρονες αναγνώσεις***

Σχήμα 3.1: *Workloads για την Πειραματική Αξιολόγηση των LSTs (πηγή: [1])*

3.1.2 Περιγραφή συνθετικών δεδομένων των πειραμάτων

Το LST-Bench δημιουργεί δεδομένα δοκιμών χρησιμοποιώντας το εργαλείο dsdgen, που αποτελεί τη γεννήτρια συνθετικών δεδομένων του TPC-DS Benchmark. Στην παρούσα ενότητα παρουσιάζεται μια σύντομη αναφορά και περιγραφή των δεδομένων αυτών, τα οποία χρησιμοποιήθηκαν για τα πειράματα της παρούσας μελέτης.

Τα δεδομένα του TPC-DS προσομοιώνουν τη βάση δεδομένων μιας μεγάλης επιχείρησης λιανικής πώλησης που δραστηριοποιείται μέσω τριών καναλιών πωλήσεων (φυσικό κατάστημα, ηλεκτρονικό κατάστημα και καταλόγους). Τα δεδομένα αυτά οργανώνονται σε ένα snowflake σχήμα, που περιλαμβάνει επτά fact tables και δεκαεπτά dimension tables. Οι fact tables περιλαμβάνουν έναν πίνακα πωλήσεων (Sales) και έναν πίνακα επιστροφών (Returns) για κάθε ένα από τα κανάλια διανομής, καθώς και έναν πίνακα αποθεμάτων (Inventory), ο οποίος παρακολουθεί τα αποθέματα των προϊόντων. Οι dimension tables συνδέονται με αυτούς τους πίνακες και περιλαμβάνουν δεδομένα όπως στοιχεία πελατών, τοποθεσίες καταστημάτων, πληροφορίες προϊόντων και άλλες σχετικές πληροφορίες [27].

Για την παρούσα μελέτη, χρησιμοποιήθηκε ένα σύνολο δεδομένων συνολικού μεγέθους 10 GB. Από αυτά, 4 GB χρησιμοποιήθηκαν για το αρχικό φόρτωμα των δεδομένων στους πίνακες (Load Task) και 1 GB επιπλέον αξιοποιήθηκε σε κάθε Data Maintenance Task.

3.2 Μετρικές αξιολόγησης των πειραμάτων

Για την πλήρη και πολυδιάστατη σύγκριση των LSTs, είναι απαραίτητο να ληφθούν υπόψη διάφορες μετρικές, οι οποίες καλύπτουν διαφορετικές πτυχές της απόδοσης και της λειτουργίας τους. Στην παρούσα ενότητα, παρουσιάζονται οι σχετικές μετρικές και αναλύεται ο τρόπος με τον οποίο μετρήθηκε η κάθε μία.

Αρχικά, ο χρόνος εκτέλεσης των ερωτημάτων αποτελεί έναν από τους πιο σημαντικούς παράγοντες για την αξιολόγηση των LSTs, καθώς αποτυπώνει την ικανότητα του συστήματος να επεξεργάζεται τα δεδομένα εντός ενός ικανοποιητικού χρονικού διαστήματος. Για την εκτίμηση αυτής της παραμέτρου, χρησιμοποιήθηκε η μετρική latency, η οποία μετρά τον χρόνο που απαιτείται για την εκτέλεση του κάθε task. Το benchmark που χρησιμοποιήθηκε παρέχει έναν μηχανισμό με τον οποίο αποθηκεύονται οι ακριβείς χρόνοι έναρξης και τερματισμού του κάθε task. Τα δεδομένα αυτά οπτικοποιήθηκαν μέσω βιβλιοθηκών της Python.

Η σταθερότητα (Stability) αποτελεί μια νέα μετρική που προτείνεται από το LST-Bench και έχει σχεδιαστεί για να μετράει την ικανότητα των LSTs να διατηρούν σταθερή την απόδοσή τους με την πάροδο του χρόνου, παρά τις συνεχείς ενημερώσεις δεδομένων και την πιθανή συσσώρευση πολλών αρχείων. Στην παρούσα διπλωματική εργασία, χρησιμοποιήθηκε για να αξιολογηθεί το κατά πόσο επηρεάζεται ο χρόνος εκτέλεσης των Single User εργασιών (δηλαδή των 99 read-queries) από τις συνεχείς ενημερώσεις των πινάκων, ενώ γενικά μπορεί να εφαρμοστεί και σε άλλες παραμέτρους (π.χ. χρήση CPU ή χρήση δίσκου). Για τον υπολογισμό της σταθερότητας, χρησιμοποιείται μία μετρική ρυθμού υποβάθμισης (Degradation Rate - S_{DR}) η οποία υπολογίζεται ως εξής:

$$S_{DR} = \frac{1}{n} \sum_{i=1}^{n} \frac{M_i - M_{i-1}}{M_{i-1}}$$

όπου,

- M_i είναι ο χρόνος εκτέλεσης της i-ής SU φάσης σε ένα workload,
- n είναι ο συνολικός αριθμός των SU φάσεων στο συγκεκριμένο workload, και
- S_{DR} είναι ο ρυθμός υποβάθμισης της απόδοσης.

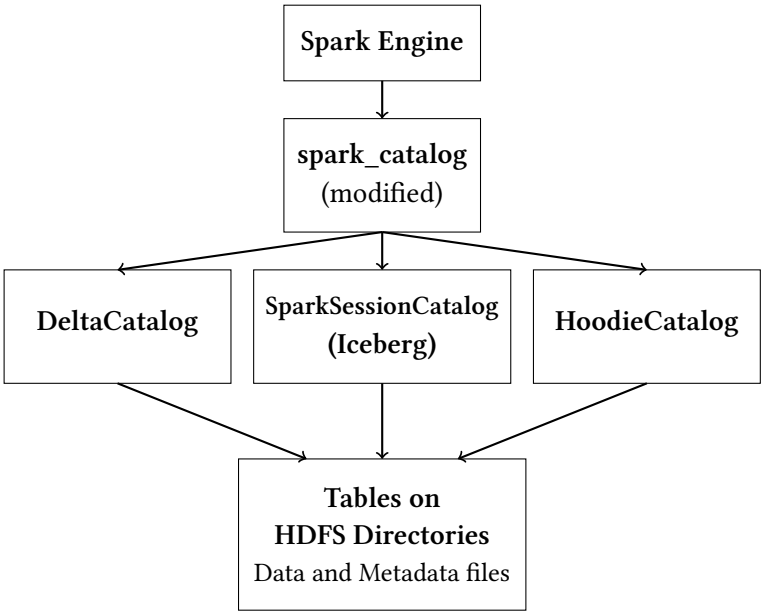
Στην παρούσα μελέτη, η αξιολόγηση της απόδοσης και της αποδοτικότητας του συστήματος κατά την εκτέλεση των εργασιών βασίστηκε σε ένα σύνολο μετρικών, οι οποίες περιλαμβάνουν: τη χρήση της CPU (CPU % Usage), τη χρήση του δίσκου (Disk I/O % Utilization), τις λειτουργίες I/O στο δίσκο (Disk I/O Operations), και τη μεταφορά δεδομένων (shuffle data) μεταξύ των Virtual Machines (Network Data Transfer). Για την καταγραφή αυτών των μετρικών, αξιοποιήθηκε ένα Azure Log Analytics Workspace [28], το οποίο είναι ένας αποθηκευτικός χώρος δεδομένων που επιτρέπει τη συλλογή και τη διαχείριση καταγραφών (logs) από εφαρμογές τόσο εντός όσο και εκτός της πλατφόρμας Azure. Η συγκεκριμένη υπηρεσία παρέχει ισχυρά εργαλεία για ανάλυση, παρακολούθηση και έλεγχο των δεδομένων αυτών σε πραγματικό χρόνο. Για να επιτευχθεί αυτό, τα τρία VMs που χρησιμοποιήθηκαν για την εκτέλεση των πειραμάτων συνδέθηκαν σε ένα Azure Log Analytics Workspace. Σε κάθε VM εγκαταστάθηκε ένας Azure Monitor Agent, ο οποίος, όταν ενεργοποιούνταν, έστελνε τις καταγραφές (logs) των σχετικών μετρικών σε πραγματικό χρόνο στο Log Analytics Workspace. Τα δεδομένα αυτά οπτικοποιήθηκαν μέσω βιβλιοθηκών της Python.

3.3 Υποδομή εκτέλεσης πειραμάτων

Τα πειράματα πραγματοποιήθηκαν σε ένα κατανεμημένο περιβάλλον (cluster) που βασίστηκε στις τεχνολογίες Apache Spark 3.3.1 [29] και Apache Hadoop 3.3.6 [30], οι οποίες χρησιμοποιήθηκαν για την επεξεργασία και την αποθήκευση των δεδομένων αντίστοιχα. Το cluster αποτελούνταν από τρεις εικονικές μηχανές (VMs), οι οποίες λειτουργούσαν ως κόμβοι του συστήματος: ένας master κόμβος και δύο worker κόμβοι. Κάθε κόμβος διέθετε 4 εικονικούς πυρήνες επεξεργασίας (vCPUs) και 8 GB μνήμης RAM, ενώ οι υπολογιστικοί πόροι προέρχονταν από επεξεργαστές Intel Skylake με συχνότητα λειτουργίας 2.2 GHz. Για τα πειράματα χρησιμοποιήθηκαν οι εξής εκδόσεις των LSTs: Delta Lake v2.2.0, Apache Iceberg v1.1.0 και Apache Hudi v0.12.2.

Στο πλαίσιο της παρούσας διπλωματικής, η αποθήκευση των δεδομένων πραγματοποιείται εντός του Spark/Hadoop cluster μέσω HDFS, ενώ οι πίνακες που χρησιμοποιούνται στα πειράματα έχουν οριστεί πάνω σε HDFS directories. Για την υποστήριξη των συστημάτων αποθήκευσης Delta Lake, Apache Iceberg και Apache Hudi, αξιοποιείται ο μηχανισμός των Spark Catalogs. Συγκεκριμένα, γίνεται τροποποίηση του προκαθορισμένου καταλόγου spark_catalog μέσω των αντίστοιχων Spark extensions και των υλοποιήσεων καταλόγου (DeltaCatalog, SparkCatalog, HoodieCatalog), ανάλογα με το σύστημα που εξετάζεται κάθε φορά. Έτσι, σε κάθε πειραματικό σενάριο τροποποιούνται δυναμικά οι ρυθμίσεις εκκίνησης του Spark (spark.sql.catalog και

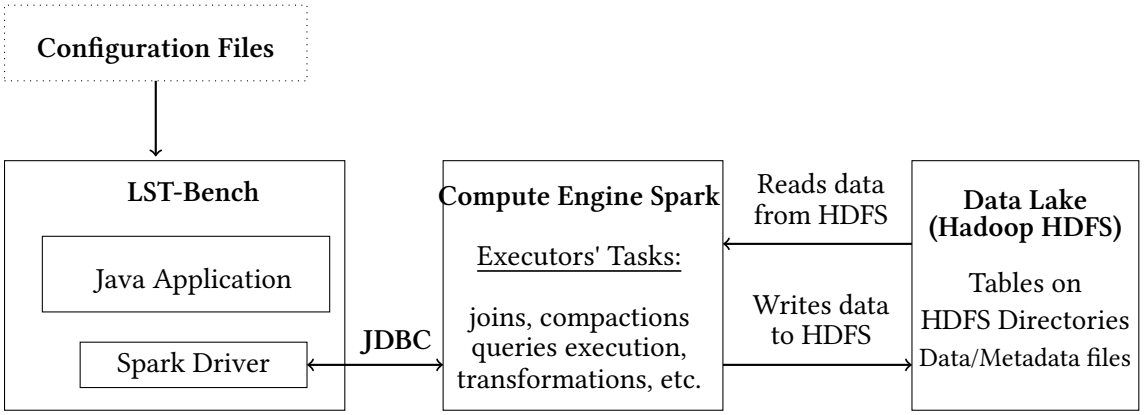
spark.sql.extensions), ώστε να ενεργοποιείται η υποστήριξη για το αντίστοιχο storage system. Τα μεταδεδομένα και τα δεδομένα αποθηκεύονται σε καταλόγους του HDFS. (Σχήμα 3.2)



Σχήμα 3.2: *Spark Catalogs και πίνακες στο HDFS*

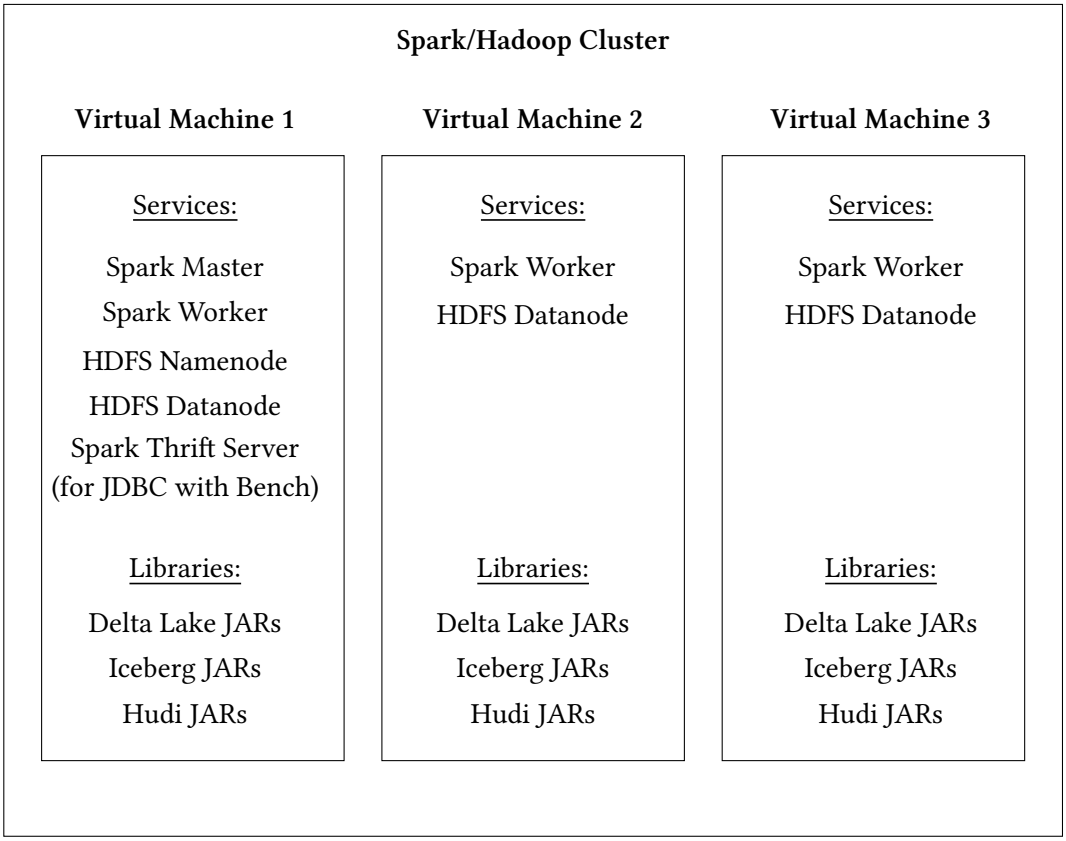
Το LST-Bench συνδέεται με τον Spark μέσω JDBC και χρησιμοποιεί τον κατάλληλο driver, για να κάνει αυτή τη σύνδεση δυνατή. Η εκτέλεση των πειραμάτων πραγματοποιείται μέσω μιας αυτόνομης Java εφαρμογής, η οποία αποτελεί τον βασικό μηχανισμό λειτουργίας του benchmark. Η εφαρμογή αυτή είναι υπεύθυνη για τη διαχείριση και εκτέλεση των SQL workloads, καθώς και για τον χειρισμό της αλληλεπίδρασης με το LST υπό δοκιμή. Η λειτουργία της εφαρμογής καθορίζεται από αρχεία ρυθμίσεων (configuration files), τα οποία διαμορφώνονται από τον χρήστη και περιέχουν πληροφορίες για τη σύνδεση με το cluster, τα πειραματικά σενάρια, τις φάσεις εκτέλεσης, καθώς και τις παραμέτρους για κάθε task. Η υλοποίηση αυτή απεικονίζεται διαγραμματικά στο Σχήμα 3.3. Για την αποδοτική διαχείριση των συνεδριών, δημιουργείται ένα JDBC connection pool, το οποίο εξυπηρετεί τις ταυτόχρονες εκτελέσεις που απαιτούνται κατά τη διάρκεια των διαδοχικών φάσεων των πειραμάτων.

Το Σχήμα 3.3 παρουσιάζει την αρχιτεκτονική εκτέλεσης των εργασιών του συστήματος με βασικά μέρη τον Compute Engine Spark και το Data Lake που βασίζεται στο Hadoop HDFS. Ο Compute Engine αναλαμβάνει την εκτέλεση των εργασιών μέσα από τους executors, οι οποίοι εκτελούν λειτουργίες όπως joins, compactions, μετασχηματισμούς και queries. Τα δεδομένα και τα metadata αποθηκεύονται στο Data Lake, που σε αυτήν την περίπτωση είναι ένα σύστημα αποθήκευσης βασισμένο στο Hadoop HDFS. Τα δεδομένα οργανώνονται σε αρχεία σε directories τα οποία οι executors διαβάζουν και γράφουν κατά την εκτέλεση των εργασιών. Οι executors διαβάζουν τα δεδομένα από το HDFS, εκτελούν τις εργασίες και επανατοποθετούν τα αποτελέσματα ξανά στο HDFS. Η ανάγνωση/εγγραφή δεδομένων γίνεται μέσω ροών προς και από το HDFS. Μέσω του catalog, όπως περιγράφηκε παραπάνω, ο Spark κατανοεί τη δομή των δεδομένων και μπορεί να δουλεύει σωστά με αυτά στο σύστημα αποθήκευσης.



Σχήμα 3.3: *Αρχιτεκτονική Εκτέλεσης πειραμάτων μέσω του LST-Bench*

Το Σχήμα 3.4 απεικονίζει την δομή του Spark/Hadoop cluster που χρησιμοποιείται για την εκτέλεση πειραμάτων. Το cluster αποτελείται από τρεις εικονικές μηχανές, καθεμία από τις οποίες φιλοξενεί συγκεκριμένες υπηρεσίες και βιβλιοθήκες. Το Virtual Machine 1 λειτουργεί ως κόμβος ελέγχου και περιλαμβάνει βασικές υπηρεσίες όπως Spark Master, Spark Worker, HDFS NameNode/DataNode και Spark Thrift Server για τη σύνδεση του LST-Bench μέσω JDBC. Οι VM2 και VM3 λειτουργούν ως worker κόμβοι με Spark Worker και HDFS DataNode Services. Όλες οι μηχανές διαθέτουν τις απαραίτητες βιβλιοθήκες (JARs) για τα storage formats (Delta Lake, Iceberg και Hudi), οι οποίες χρησιμοποιούνται κατά την εκτέλεση των Spark jobs.



Σχήμα 3.4: *Deployment Diagram*

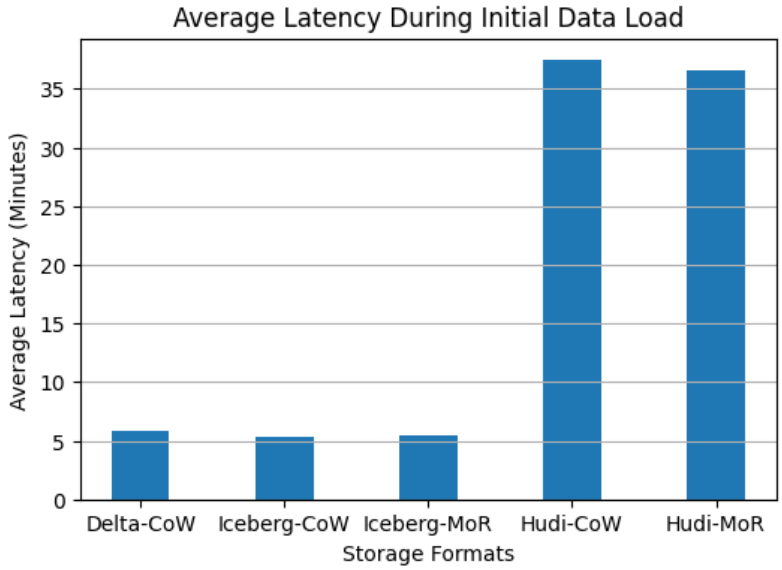
Κεφάλαιο 4

Αποτελέσματα Πειραμάτων

Στην παρούσα ενότητα, παρουσιάζονται τα αποτελέσματα από τα πειράματα που πραγματοποιήθηκαν, όπως αυτά περιγράφηκαν στην Ενότητα 3. Τα αποτελέσματα οργανώνονται και παρουσιάζονται ανά Workload, ενώ αναλύεται και ο χρόνος που απαιτήθηκε για το αρχικό φόρτωμα των δεδομένων.

4.1 Αρχική φόρτωση πινάκων

Στην παρούσα ενότητα αναλύεται ο χρόνος που απαιτήθηκε για το αρχικό φόρτωμα των δεδομένων, τη φάση Load, η οποία αποτελεί το πρώτο βήμα σε όλα τα πειράματα. Το Σχήμα 4.1 απεικονίζει τον μέσο χρόνο που απαιτήθηκε για το φόρτωμα των δεδομένων σε όλα τα πειράματα που εκτελέστηκαν, ανά LST.



Σχήμα 4.1: *Μέσος Χρόνος Φόρτωσης Αρχικών Δεδομένων*

Το Hudi παρουσιάζει σημαντικά μεγαλύτερο χρόνο φόρτωσης δεδομένων σε σχέση με τα Delta και Iceberg. Αυτό οφείλεται στο γεγονός ότι το Hudi κατά τη διαδικασία φόρτωσης εκτελεί κάποιους έλεγχους μοναδικότητας κλειδιών και κάνει κατανομή των εγγραφών στα File Groups, λειτουργίες που απαιτούν περισσότερο χρόνο επεξεργασίας [21]. Στο πλαίσιο της έρευ-

νας, δοκιμάστηκαν προσαρμογές στη διαδικασία δημιουργίας των πινάκων Hudi [22], προκειμένου να αποφευχθούν οι έλεγχοι μοναδικότητας και να επιτευχθεί ταχύτερη φόρτωση των δεδομένων. Ως αποτέλεσμα, παρατηρήθηκε σημαντική μείωση του χρόνου εισαγωγής των δεδομένων, φτάνοντας σε επίπεδα που συγκρίνονται με αυτά των συστημάτων Delta και Iceberg. Ωστόσο, η αποφυγή αυτών των ελέγχων οδήγησε στην αδυναμία των πινάκων Hudi να εκτελούν ενημερώσεις και διαγραφές δεδομένων, καθώς και να υποστηρίζουν time-travel queries, τα οποία είναι απαραίτητα για τα πειράματα της παρούσας μελέτης. Η υλοποίηση αυτή θα ήταν ιδιαίτερα σημαντική σε σενάρια χρήσης που αφορούν μόνο την εισαγωγή και ανάγνωση των δεδομένων.

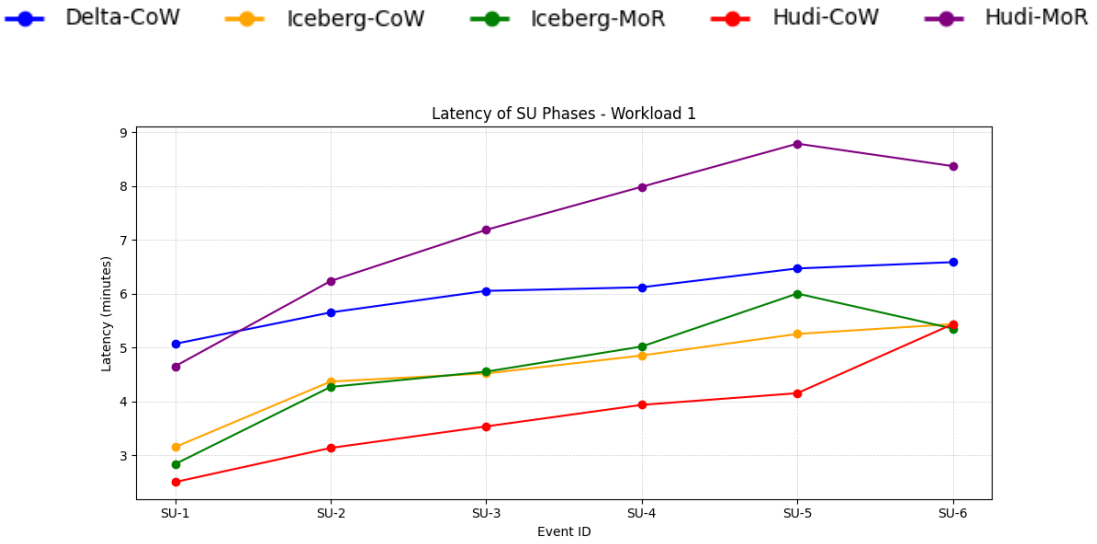
4.2 Workload 1: Μακροχρόνια Χρήση

Στην παρούσα ενότητα παρατίθενται τα αποτελέσματα του Workload 1 (3.1α΄), το οποίο αποτελείται από μια αλληλουχία έξι SU και πέντε DM φάσεων, με σκοπό να αξιολογηθεί η αποδοτικότητα των LSTs σε σενάρια με συνεχείς ενημερώσεις των δεδομένων.

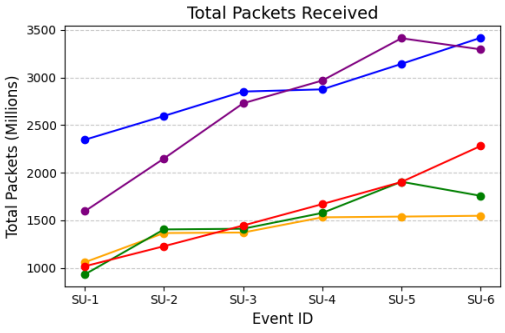
Όπως προκύπτει από το Σχήμα 4.2α΄, υπάρχει μία σταδιακή αύξηση του χρόνου εκτέλεσης των φάσεων SU, καθώς τα δεδομένα ενημερώνονται και δημιουργούνται επιπλέον αρχεία. Η στρατηγική που χρησιμοποιείται για την υλοποίηση των LSTs (CoW και MoR) φαίνεται ότι επηρεάζει τον χρόνο που απαιτείται για την ανάγνωση των δεδομένων. Πιο συγκεκριμένα, με το Hudi-MoR απαιτείται σημαντικά περισσότερος χρόνος σε σύγκριση με το Hudi-CoW για την εκτέλεση των ερωτημάτων. Το γεγονός αυτό επιβεβαιώνει τη θεωρία, όπως έχει αναφερθεί παραπάνω, ότι η στρατηγική CoW είναι προτιμότερη για ανάγνωση δεδομένων, αφού δεν χρειάζεται να συγχωνευτούν πολλά αρχεία για την ανάγνωση, όπως συμβαίνει στην στρατηγική MoR. Η παρατήρηση αυτή σχετικά με τις στρατηγικές είναι σύμφωνη και με άλλες μελέτες [1] [21]. Επιπλέον, αν και το φαινόμενο αυτό είναι λιγότερο έντονο στο Iceberg, παρατηρείται και εκεί ότι, καθώς αυξάνεται ο αριθμός των φάσεων DM, και συνεπώς ο αριθμός των δημιουργούμενων αρχείων, η στρατηγική CoW συνεχίζει να αποδίδει καλύτερα στις διαδικασίες ανάγνωσης δεδομένων.

Όσον αφορά τη μετρική της σταθερότητας, το Delta φαίνεται να είναι το πιο σταθερό σε σύγκριση με τα υπόλοιπα, όπως προκύπτει από το Σχήμα 4.3. Συγκεκριμένα, ο χρόνος εκτέλεσης των διαφορετικών SU φάσεων αυξάνεται με τον μικρότερο ρυθμό, υποδεικνύοντας ότι το Delta διατηρεί πιο σταθερή απόδοση καθώς πραγματοποιούνται ενημερώσεις. Αυτό οφείλεται στο γεγονός ότι, όπως αναλύθηκε σε προηγούμενη ενότητα, το Delta διατηρεί checkpoints που συνοψίζουν τις αλλαγές που έχουν γίνει στους πίνακες, και συνεπώς δεν απαιτεί την ανάγνωση όλων των JSON αρχείων που περιέχουν τις τροποποιήσεις που έχουν γίνει στα δεδομένα.

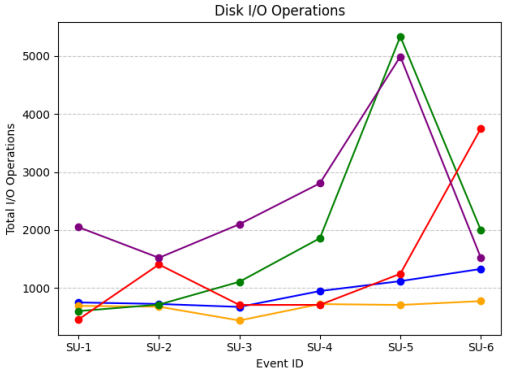
Στα Σχήματα 4.2β΄ και 4.2γ΄ παρατίθενται δεδομένα σχετικά με τη λειτουργία του συστήματος κατά τη διάρκεια εκτέλεσης του πειράματος W1. Από τα αποτελέσματα που αφορούν τη μεταφορά δεδομένων (shuffle data) μεταξύ των τριών VMs, προκύπτει ότι τα συστήματα που χρησιμοποιούν τη στρατηγική CoW επιδεικνύουν μεγαλύτερη αποδοτικότητα, καθώς ο συνολικός όγκος των δεδομένων που μετακινούνται εντός των VMs κατά τις φάσεις SU είναι χαμηλότερος σε σύγκριση με τα αντίστοιχα LSTs που υιοθετούν τη στρατηγική MoR. Το συμπέρασμα αυτό ενισχύει την υπόθεση ότι η στρατηγική CoW αποτελεί την πιο κατάλληλη επιλογή για εργασίες που απαιτούν ανάγνωση δεδομένων. Παρόμοια συμπεράσματα ισχύουν και για τις λειτουργίες



(α΄) *Χρόνος Εκτέλεσης*



(β΄) *Μεταφορά Δεδομένων μεταξύ των VMs*

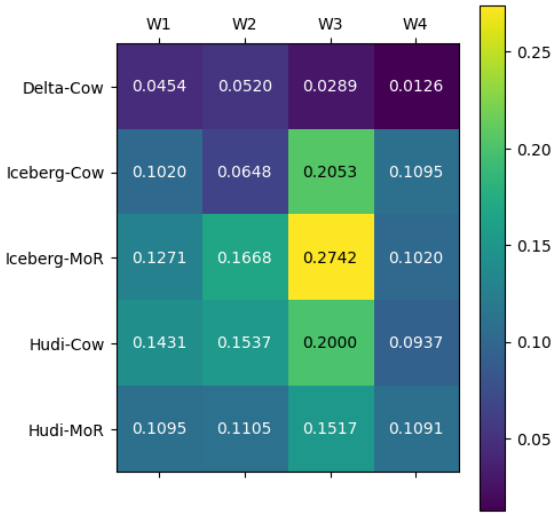


(γ΄) *Λειτουργίες IO στο Δίσκο*

Σχήμα 4.2: *Μετρικές Απόδοσης για τις Φάσεις SU στο W1*

I/O στους δίσκους των τριών VMs, καθώς το Hudi-CoW και το Iceberg-CoW απαιτούν λιγότερες λειτουργίες I/O σε σύγκριση με το Hudi-MoR και το Iceberg-MoR αντίστοιχα. Επιπλέον, η στρατηγική MoR δημιουργεί συνεχώς αυξημένο φόρτο στον δίσκο, εξαιτίας της διαδικασίας συγχώνευσης ολοένα και περισσότερων αρχείων κατά την ανάγνωση, ενώ με τη στρατηγική CoW, ο φόρτος στον δίσκο παραμένει σχετικά σταθερός. Για τη στρατηγική MoR, παρατηρείται ότι τόσο στο Iceberg όσο και στο Hudi, υπάρχει μια σημαντική μείωση των λειτουργιών I/O στον δίσκο κατά την τελευταία φάση Single User. Στο Iceberg, υποστηρίζεται μια λειτουργία κατά την οποία τα αρχεία manifest συμπιέζονται αυτόματα [17], ενώ στο Hudi εκτελείται από προεπιλογή μια διαδικασία συγχώνευσης των αρχείων που περιέχουν τα δεδομένα [20]. Αυτές οι λειτουργίες είναι πιθανό να ευθύνονται για τη μείωση των I/O λειτουργιών στον δίσκο, ενώ η χρησιμότητά τους φαίνεται και από τη μείωση του χρόνου που απαιτείται για την ανάγνωση των δεδομένων στην τελευταία φάση SU. Πιο αναλυτικά εξετάζονται οι βελτιστοποιήσεις που παρέχουν τα LSTs στο πλαίσιο του W2.

Στον Πίνακα 4.1 και στο Σχήμα 4.4 περιγράφεται και απεικονίζεται το ποσοστό χρήσης της



Σχήμα 4.3: *Stability: Ρυθμός Υποβάθμισης για SU Φάσεις*

CPU (μέση χρήση μεταξύ των VMs), ενώ στον Πίνακα 4.2 και στο Σχήμα 4.5 παρουσιάζονται οι αντίστοιχες πληροφορίες για τη χρήση του δίσκου (μέση χρήση μεταξύ των VMs). Η παραμετροποίηση του περιβάλλοντος Spark που υλοποιήθηκε φαίνεται να μην επιτρέπει στο cluster να αξιοποιεί αποδοτικά όλους τους διαθέσιμους υπολογιστικούς πόρους, γεγονός που οδηγεί σε υπο-αξιοποίηση (underutilization) του συστήματος, διατηρώντας όμως τη σταθερότητα της εκτέλεσης και διασφαλίζοντας την επιτυχή ολοκλήρωση όλων των πειραμάτων. Η κατάσταση αυτή αποτυπώνεται στο ποσοστό χρήσης της CPU, το οποίο διατηρείται γενικά σε χαμηλά επίπεδα καθ' όλη τη διάρκεια των πειραμάτων. Όλα τα πειράματα εκτελέστηκαν με την ίδια παραμετροποίηση, γεγονός που διασφαλίζει τη συγκρισιμότητα των αποτελεσμάτων μεταξύ των διαφορετικών LSTs.

Όσον αφορά την αρχική φόρτωση των δεδομένων, το Hudi εμφανίζει μεγαλύτερα ποσοστά χρήσης CPU και Disk I/O, σε σύγκριση με τα Delta και Iceberg. Το γεγονός αυτό συμβαίνει επειδή, όπως αναφέρθηκε και στην Ενότητα 4.1, το Hudi υλοποιεί μια πιο σύνθετη λογική κατά την εγγραφή δεδομένων, η οποία περιλαμβάνει ελέγχους μοναδικότητας κλειδιών και κατανομή των εγγραφών στα File Groups [21], λειτουργίες που εντείνουν τη χρήση υπολογιστικών πόρων και τον φόρτο στον δίσκο.

Κατά τις φάσεις Single User, παρατηρείται αυξημένος φόρτος στον δίσκο για τις στρατηγικές MoR, συγκριτικά με τις αντίστοιχες CoW υλοποιήσεις των LSTs. Η αυξημένη χρήση οφείλεται στο γεγονός ότι οι MoR στρατηγικές απαιτούν συγχώνευση πολλών αρχείων κατά την ανάγνωση, γεγονός που συνεπάγεται επιπλέον λειτουργίες I/O. Αντίθετα, στη στρατηγική CoW όλα τα δεδομένα του κάθε πίνακα βρίσκονται ήδη σε ενιαία μορφή εντός ενός αρχείου.

Κατά τη διαδικασία ενημέρωσης των πινάκων (DM φάσεις), για όλα τα LSTs παρατηρείται σημαντικά χαμηλή χρήση της CPU και, συγκριτικά με άλλες φάσεις, υψηλότερος φόρτος στον δίσκο. Ο συνδυασμός αυτών των δύο παρατηρήσεων μπορεί να ερμηνευθεί ως εξής: η CPU παραμένει σε μεγάλο βαθμό ανενεργή, επειδή οι διεργασίες αναμένουν την ολοκλήρωση λειτουργιών εγγραφής στον δίσκο (I/O-bound). Όπως αναφέρθηκε και προηγουμένως, η παραμετροποίηση των εργασιών Spark φαίνεται να περιορίζει την πρόσβαση σε όλους τους διαθέσιμους υπολογιστικούς πόρους του συστήματος, και η τροποποίησή της πιθανώς θα αύ-

ξανε την απόδοση του συστήματος.

	Load	Single User	Data Maintenance
Delta-CoW	Μέτρια (25-55%)	Χαμηλή (25-40%)	Χαμηλή (20-35%)
Iceberg-CoW	Μέτρια (30-55%)	Χαμηλή (25-45%)	Χαμηλή (20-35%)
Iceberg-MoR	Μέτρια (30-55%)	Χαμηλή (20-45%)	Χαμηλή (20-45%)
Hudi-CoW	Υψηλή (40-70%)	Μέτρια (30-55%)	Χαμηλή (20-35%)
Hudi-MoR	Υψηλή (40-70%)	Μέτρια (30-50%)	Χαμηλή (10-30%)

Πίνακας 4.1: *Χρήση CPU ανά LST και Task στο W1*

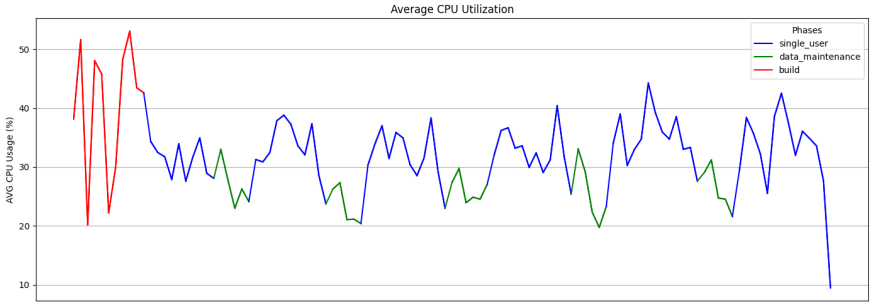
	Load	Single User	Data Maintenance
Delta-CoW	Χαμηλή	Χαμηλή	Μέτρια
Iceberg-CoW	Χαμηλή	Χαμηλή	Μέτρια
Iceberg-MoR	Χαμηλή	Μέτρια	Μέτρια
Hudi-CoW	Υψηλή	Χαμηλή	Μέτρια
Hudi-MoR	Μέτρια	Υψηλή	Μέτρια

Πίνακας 4.2: *Χρήση Δίσκου ανά LST και Task στο W1*

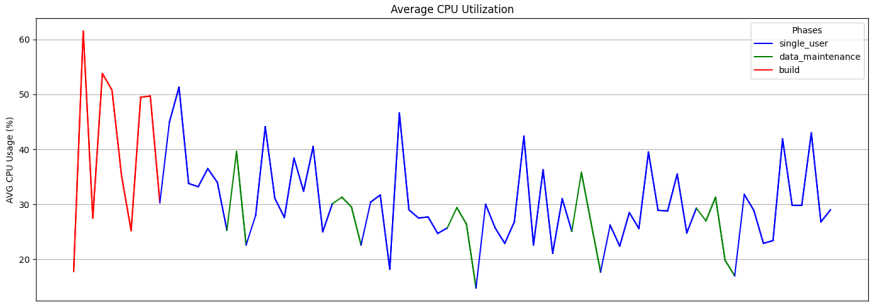
4.3 Workload 2: Συνεχείς τροποποιήσεις δεδομένων και λειτουργίες συντήρησης

Στην παρούσα ενότητα παρατίθενται τα αποτελέσματα του Workload 2 (3.1β΄), το οποίο περιλαμβάνει μια σειρά φάσεων SU, DM και O. Σκοπός του πειράματος είναι να αξιολογηθεί η αποδοτικότητα των LSTs σε σενάρια με συνεχείς ενημερώσεις δεδομένων διαφορετικού όγκου, καθώς και η επίδραση των λειτουργιών βελτιστοποίησης στην απόδοση των συστημάτων κατά την ανάγνωση των δεδομένων.

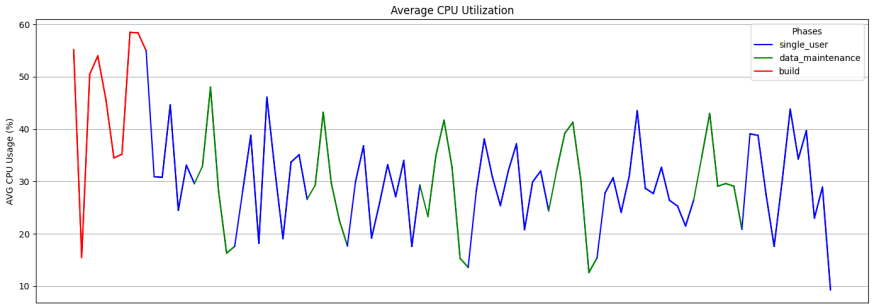
Το Σχήμα 4.6 απεικονίζει την χρονική καθυστέρηση στις φάσεις Single User (SU) του Workload 2 (W2). Για κάθε ι-στή φάση SU, εκτελούνται τα ερωτήματα ανάγνωσης δεδομένων πριν και μετά την εφαρμογή της διαδικασίας Optimize (SU-ι και SU-ι-O αντίστοιχα), όπως αναλύθηκε στην Ενότητα 3. Από το διάγραμμα προκύπτει ότι το Hudi δεν επωφελείται από την διαδικασία του Clustering, ενώ η λειτουργία συγχώνευσης αρχείων στο Iceberg έχει σημαντική επίδραση στην απόδοση του. Οι παρατηρήσεις αυτές είναι πανομοιότυπες με αντίστοιχες έρευνες [1]. Για το Delta Lake παρατηρείται μία πολύ μικρή, σχεδόν αμελητέα, βελτίωση στις φάσεις 3 και 4, δείχνοντας ότι η λειτουργία συγχώνευσης (OPTIMIZE) είναι χρήσιμη όταν πραγματοποιούνται περισσότερες ενημερώσεις στους πίνακες. Ακόμα, χρόνος εκτέλεσης των



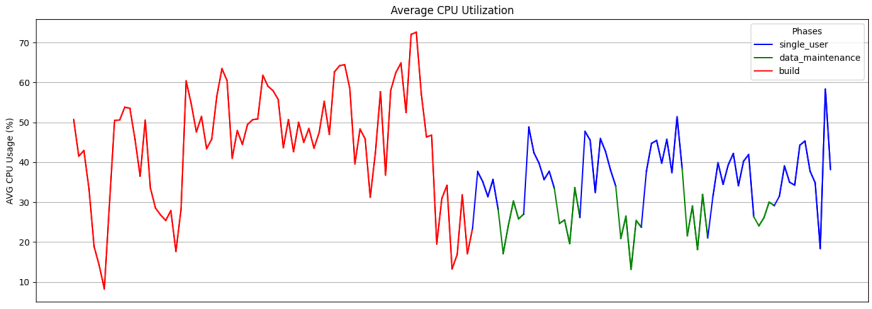
(α΄) *Delta-CoW: Χρήση CPU στο W1*



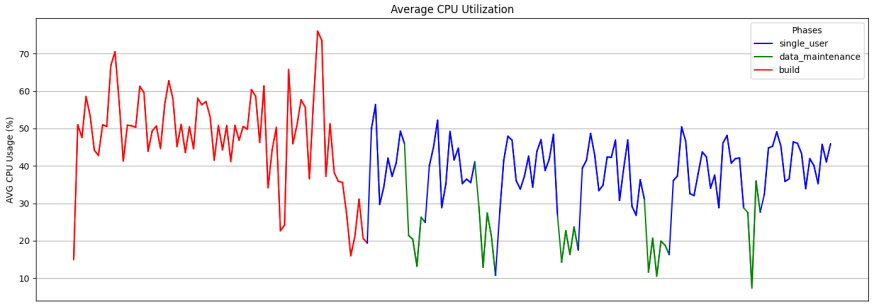
(β΄) *Iceberg-CoW: Χρήση CPU στο W1*



(γ΄) *Iceberg-MoR: Χρήση CPU στο W1*

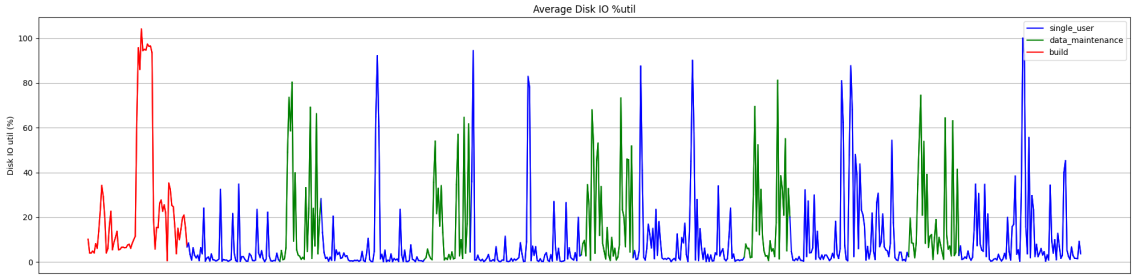


(δ΄) *Hudi-CoW: Χρήση CPU στο W1*

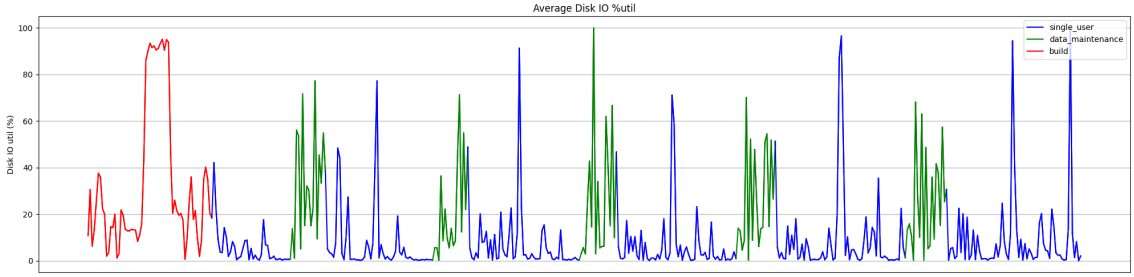


(ε΄) *Hudi-MoR: Χρήση CPU στο W1*

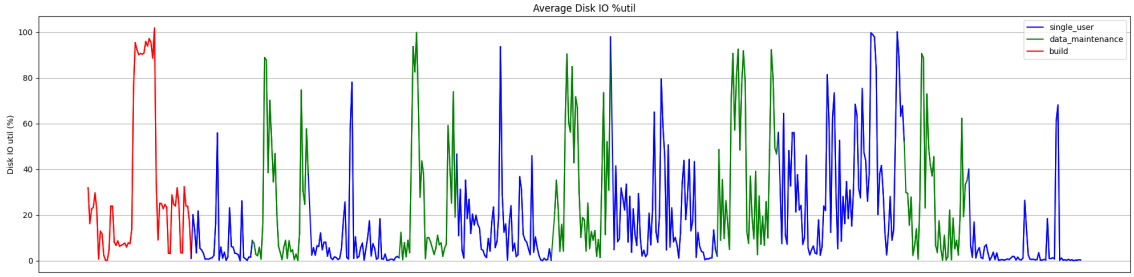
Σχήμα 4.4: *Μέση Χρήση CPU στα 3 VMs στο W1*



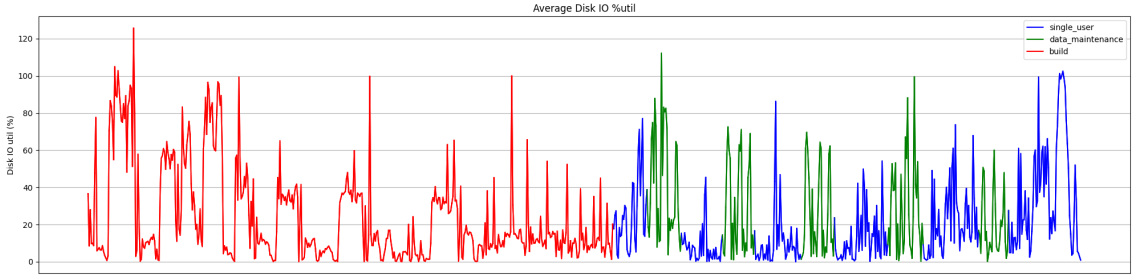
(α΄) *Delta-CoW: Χρήση Δίσκου στο W1*



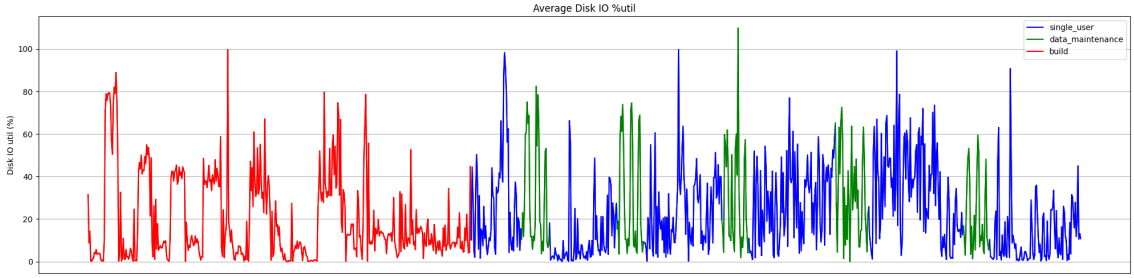
(β΄) *Iceberg-CoW: Χρήση Δίσκου στο W1*



(γ΄) *Iceberg-MoR: Χρήση Δίσκου στο W1*



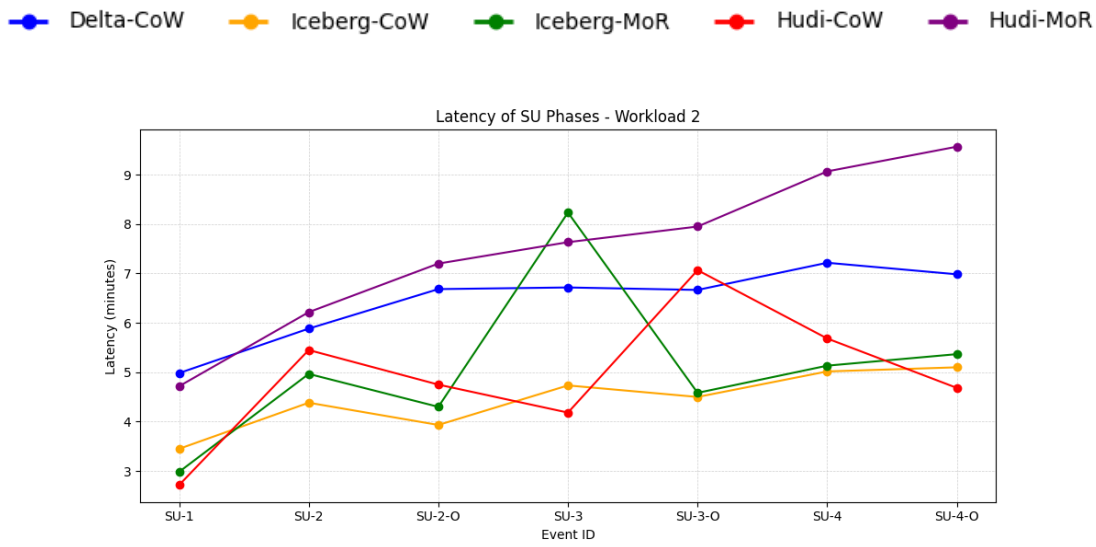
(δ΄) *Hudi-CoW: Χρήση Δίσκου στο W1*



(ε΄) *Hudi-MoR: Χρήση Δίσκου στο W1*

Σχήμα 4.5: *Μέση Χρήση Δίσκου στα 3 VMs στο W1*

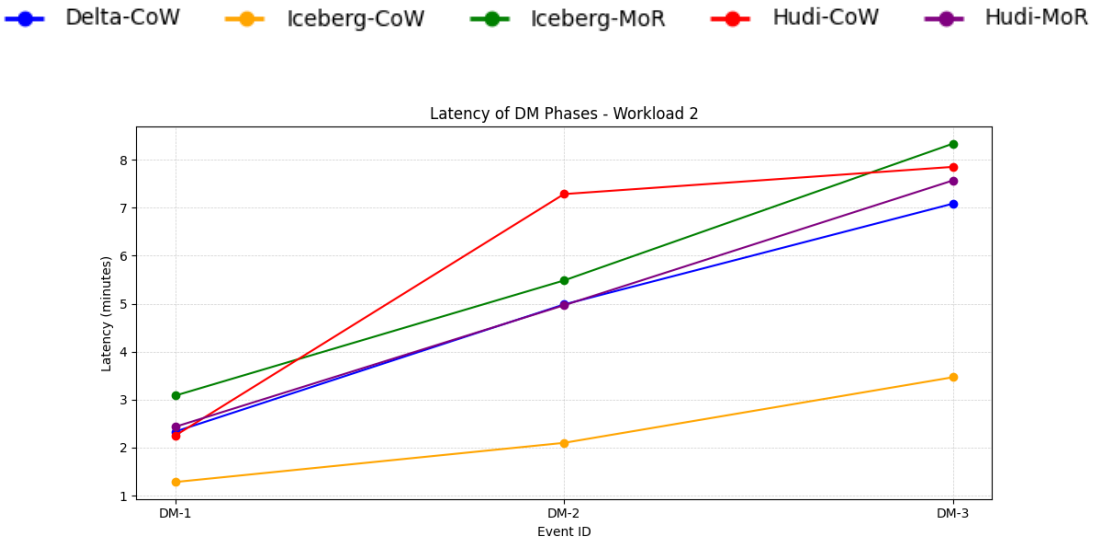
ερωτημάτων παραμένει και εδώ πιο σταθερός σε σχέση με τα άλλα δύο LSTs, κατά την εξέλιξη του πειράματος. Τα checkpoints που χρησιμοποιεί το Delta Lake, όπως αναλύθηκε στην Ενότητα 2, φαίνεται ότι είναι ιδιαίτερα σημαντικά για τη διαχείριση των μεταδεδομένων, αφού δεν συσσωρεύονται πολλά αρχεία καθώς γίνονται ενημερώσεις στα δεδομένα. Έτσι, το Delta Lake καταγράφει και εδώ τη χαμηλότερη τιμή στη μετρική της σταθερότητας (Σχήμα 4.3).



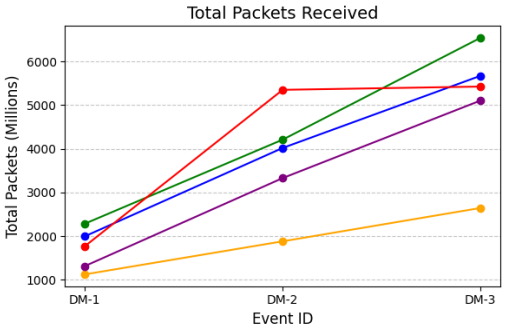
Σχήμα 4.6: *Χρόνος Εκτέλεσης SU Φάσεων στο W2*

Όσον αφορά τις λειτουργίες ενημέρωσης και διαγραφής δεδομένων στους πίνακες (φάσεις DM), παρατηρείται ότι το σύστημα Iceberg-CoW ολοκληρώνει αυτές τις λειτουργίες σε σημαντικά μικρότερο χρόνο σε σχέση με τα υπόλοιπα LSTs (Σχήμα 4.7α΄). Από το γεγονός αυτό μπορεί να εξαχθεί το συμπέρασμα ότι οι στρατηγικές Position Delete Files και Equality Delete Files είναι πολύ αποδοτικές για την διαγραφή εγγραφών των πινάκων. Για τα υπόλοιπα συστήματα LSTs, δεν παρατηρούνται σημαντικές διαφορές στο χρόνο εκτέλεσης αυτών των λειτουργιών. Σχετικά με τις λειτουργίες του δίσκου, διαπιστώνεται μια απότομη αύξηση κατά τη φάση DM-2 για το Hudi-CoW στο συγκεκριμένο πείραμα. Ωστόσο, παρατηρώντας τη ίδια μετρική σε άλλες φάσεις και πειράματα, φαίνεται ότι αυτή η αύξηση είναι μια εξαίρεση και δεν παρατηρείται παρόμοια συμπεριφορά σε κανένα άλλο πείραμα. Συνεπώς, μπορεί να θεωρηθεί ότι αυτή η απότομη αύξηση των λειτουργιών στον δίσκο δεν αντικατοπτρίζει τη συνήθη συμπεριφορά του Hudi-CoW. Γενικά, τα τρία LSTs, για τις δύο στρατηγικές, δεν παρουσιάζουν σημαντικές αποκλίσεις στις απαιτούμενες λειτουργίες στον δίσκο, με τα Delta-CoW και Iceberg-CoW να εμφανίζουν ελαφρώς λιγότερες λειτουργίες.

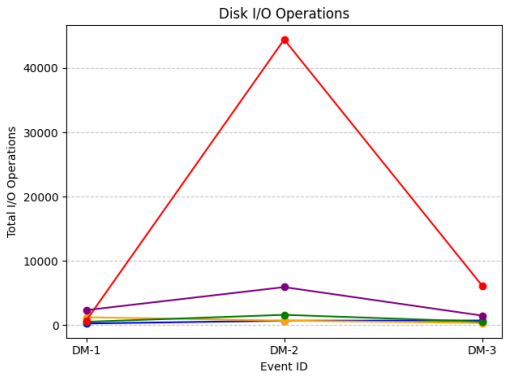
Σημαντικές διαφορές παρατηρούνται στις φάσεις Optimize των τριών LSTs. Τα Iceberg (CoW και MoR) καθυστερούν σημαντικά περισσότερο στην O-1 φάση σε σχέση με τα άλλα δύο LSTs (Σχήμα 4.8α΄). Ο λόγος για αυτήν την καθυστέρηση είναι ότι το Iceberg εκτελεί τις λειτουργίες συμπίεσης ανά ομάδες αρχείων (file group-by-group), και όχι με παράλληλη επεξεργασία πολλών ομάδων, ενώ αυτή η διαδικασία μπορεί να βελτιωθεί αν οι παράμετροι της λειτουργίας ρυθμιστούν ώστε να τρέχουν παράλληλα για πολλές ομάδες [31]. Στις δύο επόμενες φάσεις



(α΄) *Χρόνος Εκτέλεσης*



(β΄) *Μεταφορά Δεδομένων μεταξύ των VMs*

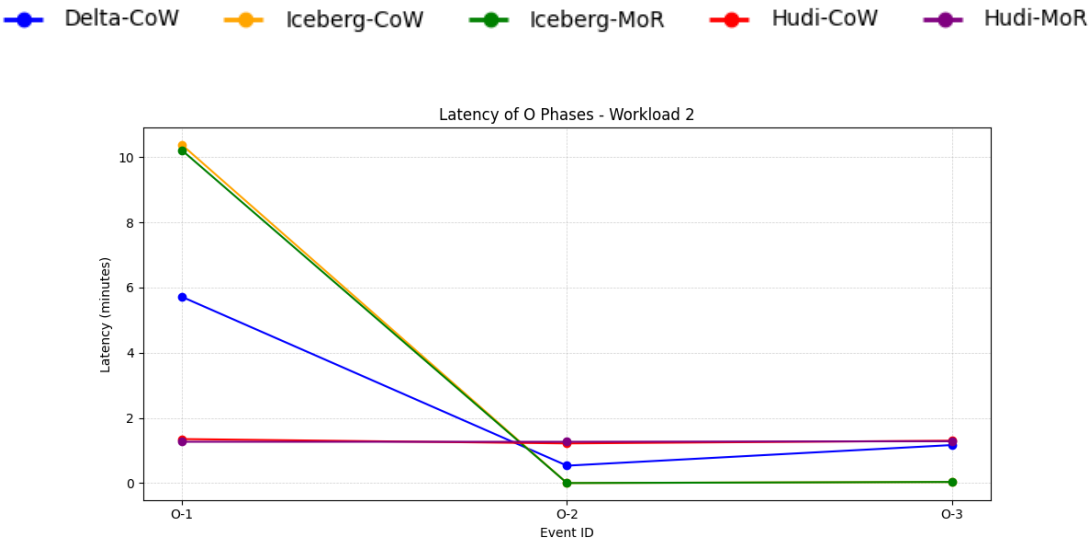


(γ΄) *Λειτουργίες IO στο Δίσκο*

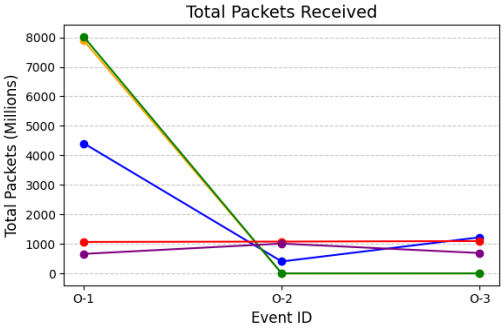
Σχήμα 4.7: *Μετρικές Απόδοσης για τις Φάσεις DM στο W2*

Optimize, παρότι ο χρόνος σχεδόν μηδενίζεται, φαίνεται ότι η λειτουργία συμπίεσης παραμένει σημαντική, καθώς συνεχίζει να μειώνεται η καθυστέρηση στην ανάγνωση των δεδομένων μετά από κάθε εκτέλεση της διαδικασίας αυτής. Η διαδικασία της συμπίεσης δεν οδηγεί σε πολλές λειτουργίες στον δίσκο, ακόμα και αν είναι αρκετά χρονοβόρα. Όσον αφορά τον χρόνο, παρόμοια αποτελέσματα παρατηρούνται και στο Delta Lake, με αρκετή καθυστέρηση στην πρώτη φάση Optimize σε σχέση με τις επόμενες δύο. Για τα Hudi (CoW και MoR), η λειτουργία του Clustering, όπως αυτή αναλύθηκε σε προηγούμενη ενότητα, κρατάει τόσο τον χρόνο όσο και τις μετρικές του συστήματος σταθερές και στις τρεις φάσεις Optimize. Αυτό υποδεικνύει ότι προσπαθεί να αναδιοργανώσει τα αρχεία κάθε φορά, ανεξάρτητα από την τρέχουσα τοποθεσία τους. Στην φάση O-1 παρατηρείται μια αύξηση στις λειτουργίες I/O του δίσκου, και αυτό υποδηλώνει ότι γίνονται σημαντικές αλλαγές στη διάταξη των δεδομένων στον αρχικό πίνακα. Συνοψίζοντας, για τις φάσεις Optimize, δεν παρατηρούνται σημαντικές διαφορές μεταξύ των στρατηγικών CoW και MoR, αλλά μόνο μεταξύ των LSTs. Για το Iceberg, οι λειτουργίες συντήρησης επηρεάζουν σημαντικά τον χρόνο που απαιτείται για την ανάγνωση των δεδομένων.

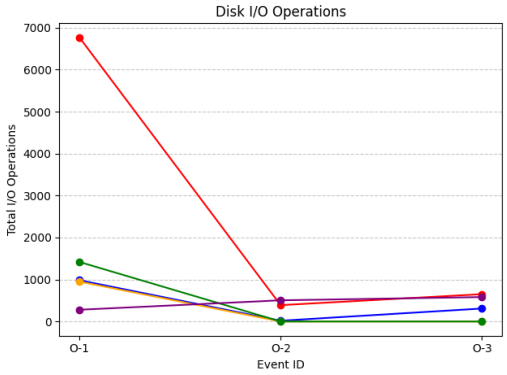
Τέλος, για τα Delta και Iceberg, οι φάσεις Optimize εκτελούνται σημαντικά ταχύτερα όταν έχει προηγηθεί πρόσφατα μία αντίστοιχη φάση βελτιστοποίησης. Το γεγονός αυτό υποδηλώνει ότι η ύπαρξη ήδη οργανωμένων ή συγχωνευμένων αρχείων μειώνει το κόστος της βελτιστοποίησης.



(α΄) *Χρόνος Εκτέλεσης*



(β΄) *Μεταφορά Δεδομένων μεταξύ των VMs*



(γ΄) *Λειτουργίες IO στο Δίσκο*

Σχήμα 4.8: *Μετρικές Απόδοσης για τις Φάσεις Optimize στο W2*

4.4 Workload 3: Παράλληλες εργασίες

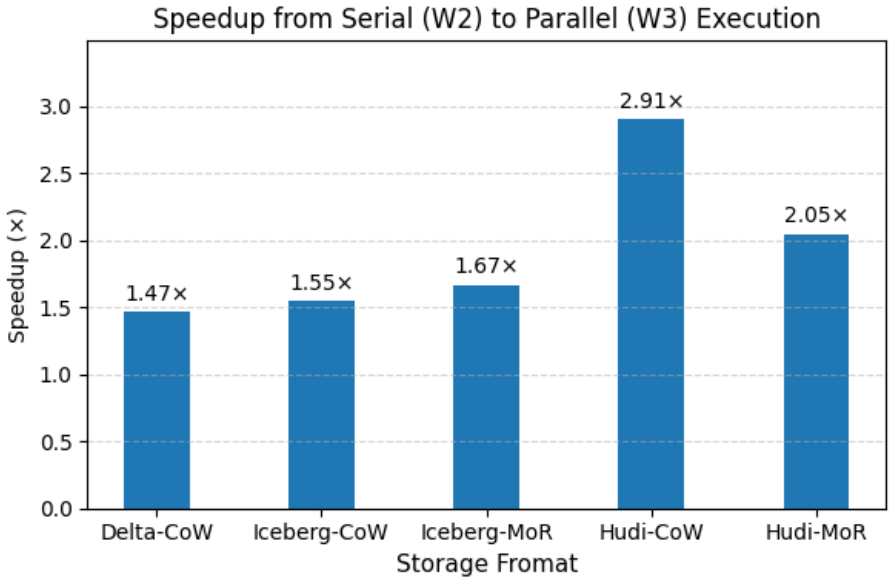
Στην παρούσα ενότητα παρουσιάζονται τα αποτελέσματα του Workload 3 (3.1γ΄). Το Workload 3 περιλαμβάνει τα ίδια στάδια με αυτά του Workload 2, με την διαφορά ότι οι φάσεις SU εκτελούνται ταυτόχρονα με τις φάσεις DM και O. Ο κύριος στόχος αυτού του πειράματος είναι να αξιολογηθεί η επίδραση της ταυτόχρονης εκτέλεσης των λειτουργιών ανάγνωσης, εγγραφής και συντήρησης στην απόδοση των συστημάτων.

Στο Σχήμα 4.9 παρουσιάζεται η επιτάχυνση (speedup) που επιτυγχάνεται μέσω της παράλληλης εκτέλεσης των εργασιών στο W3, σε σύγκριση με τη σειριακή τους εκδοχή στο W2, για

κάθε LST. Ο δείκτης speedup ορίζεται ως:

$$\text{Speedup} = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

όπου T_{serial} είναι ο συνολικός χρόνος εκτέλεσης του πειράματος όταν οι φάσεις εκτελούνται σειριακά, και T_{parallel} ο αντίστοιχος χρόνος όταν οι φάσεις εκτελούνται ταυτόχρονα. Τιμές μεγαλύτερες της μονάδας υποδεικνύουν ότι η παράλληλη εκτέλεση ολοκληρώνει το πείραμα γρηγορότερα, ενώ τιμές μικρότερες του 1 φανερώνουν μείωση της απόδοσης.



Σχήμα 4.9: *Speedup από σειριακή σε παράλληλη εκτέλεση*

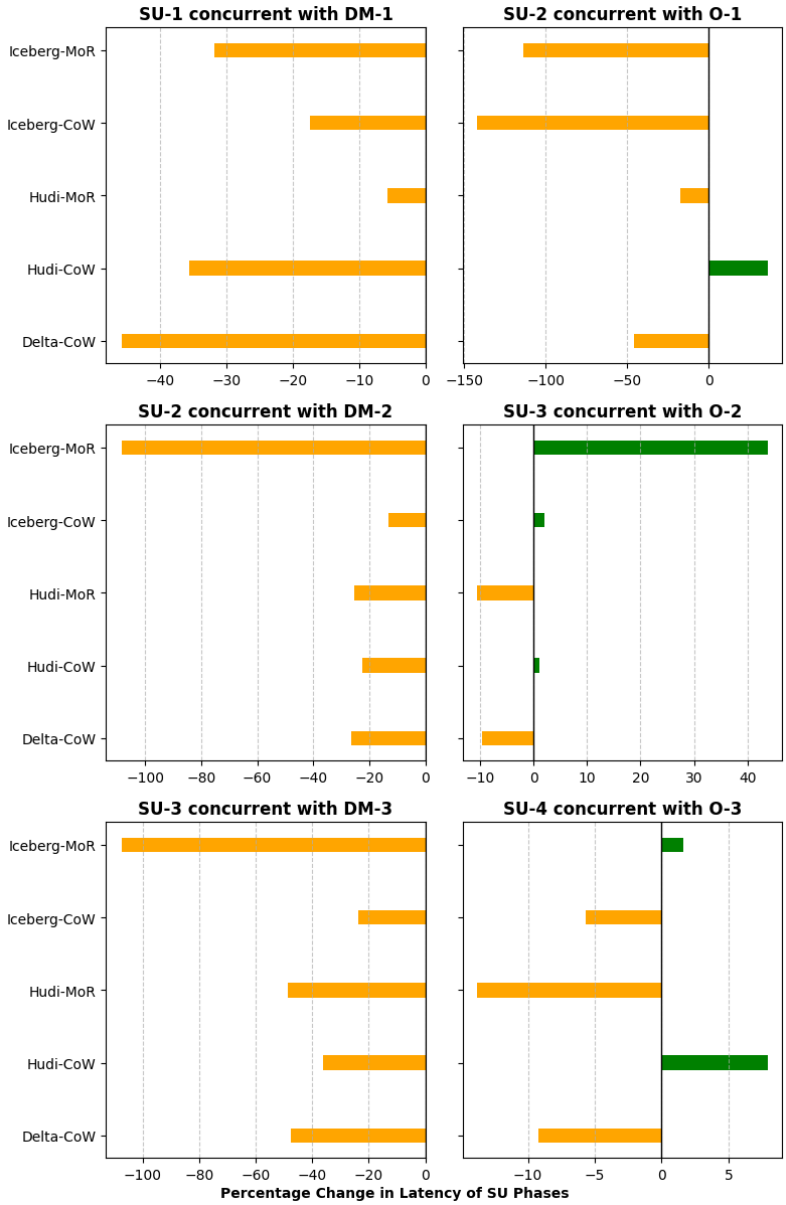
Όπως παρατηρείται, όλα τα συστήματα παρουσιάζουν κάποιο βαθμό επιτάχυνσης, γεγονός που υποδεικνύει ότι η παράλληλη εκτέλεση συνεισφέρει στη μείωση του συνολικού χρόνου. Το Hudi, και κυρίως για την στρατηγική CoW, εμφανίζει τη μεγαλύτερη βελτίωση, με δείκτη speedup μεγαλύτερο του 2, κάτι που υποδηλώνει ότι το σύστημα ολοκληρώνει το workload σε λιγότερο από το μισό χρόνο σε σχέση με τη σειριακή εκτέλεση. Για τα υπόλοιπα LSTs, υπάρχει και πάλι επιτάχυνση αλλά σε μικρότερο βαθμό. Θεωρήθηκε σημαντικό να εξεταστεί πως προκύπτει η κάθε επιτάχυνση, και για τον σκοπό αυτό αναλύθηκαν οι επιμέρους φάσεις του πειράματος ξεχωριστά. Ειδικότερα, εξετάστηκε ο χρόνος εκτέλεσης των Single User (SU) φάσεων, τόσο όταν αυτές εκτελούνται μεμονωμένα, όσο και όταν συνυπάρχουν χρονικά με άλλες εργασίες, δηλαδή τις Data Maintenance (DM) και Optimize (O) φάσεις. Στο Σχήμα 4.10 φαίνεται η ποσοστιαία μεταβολή στον χρόνο εκτέλεσης των SU φάσεων μεταξύ W2 και W3, η οποία υπολογίζεται με τον εξής τύπο

$$\text{Μεταβολή (\%)} = \left(\frac{T_{\text{SU,W2}} - T_{\text{SU,W3}}}{T_{\text{SU,W2}}} \right) \times 100$$

όπου,

- $T_{\text{SU,W2}}$ είναι ο χρόνος εκτέλεσης της SU φάσης στο Workload 2, όπου εκτελείται μόνη της.

- $T_{SU,W3}$ είναι ο χρόνος εκτέλεσης της SU φάσης στο Workload 3, όπου εκτελείται παράλληλα με άλλη εργασία.



Σχήμα 4.10: *Επίδραση Παράλληλης Εκτέλεσης στην Απόδοση SU Εργασιών*

Από την ανάλυση προκύπτει ότι, στις περισσότερες περιπτώσεις, η παράλληλη εκτέλεση επιδρά αρνητικά στις SU φάσεις, οδηγώντας σε αύξηση της καθυστέρησης. Ορισμένες εξαιρέσεις παρατηρούνται όταν οι αναγνώσεις των δεδομένων πραγματοποιούνται ταυτόχρονα με τις λειτουργίες συντήρησης πινάκων (φάσεις Optimize) που χρησιμοποιούνται στα συγκεκριμένα πειράματα. Πιο συγκεκριμένα, για το Hudi-CoW, οι SU φάσεις δεν επιβραδύνονται όταν παράλληλα υλοποιείται η λειτουργία Clustering, και σε αυτό αποδίδεται το γεγονός ότι εμφανίζει τον μεγαλύτερο speedup. Συμπεραίνοντας, παρά το γεγονός ότι σχεδόν όλες οι εργασίες επιβραδύνονται όταν εκτελούνται παράλληλα, η συνολική εκτέλεση του workload επιταχύνεται.

Για τη μετρική της σταθερότητας (Σχήμα 4.3), το Delta Lake παρουσιάζει και εδώ τη χαμηλότερη τιμή. Αντιθέτως, το Iceberg φαίνεται να εμφανίζει μεγάλες διακυμάνσεις στον χρόνο που απαιτείται για την εκτέλεση κάθε φάσης SU.

Είναι σημαντικό να σημειωθεί ότι τα ποσοστά χρήσης της CPU και του δίσκου, αυξάνονται σε σχέση με τις αντίστοιχες σειριακές εκτελέσεις των εργασιών, κάτι αναμενόμενο σε περιπτώσεις ανταγωνισμού των διαθέσιμων πόρων. Παρόλα αυτα δεν φαίνεται να αξιοποιούν στο μέγιστο όλους τους διαθέσιμους πόρους του συστήματος, λόγω της παραμετροποίησης του Spark όπως αναλύθηκε στην Ενότητα 4.2.

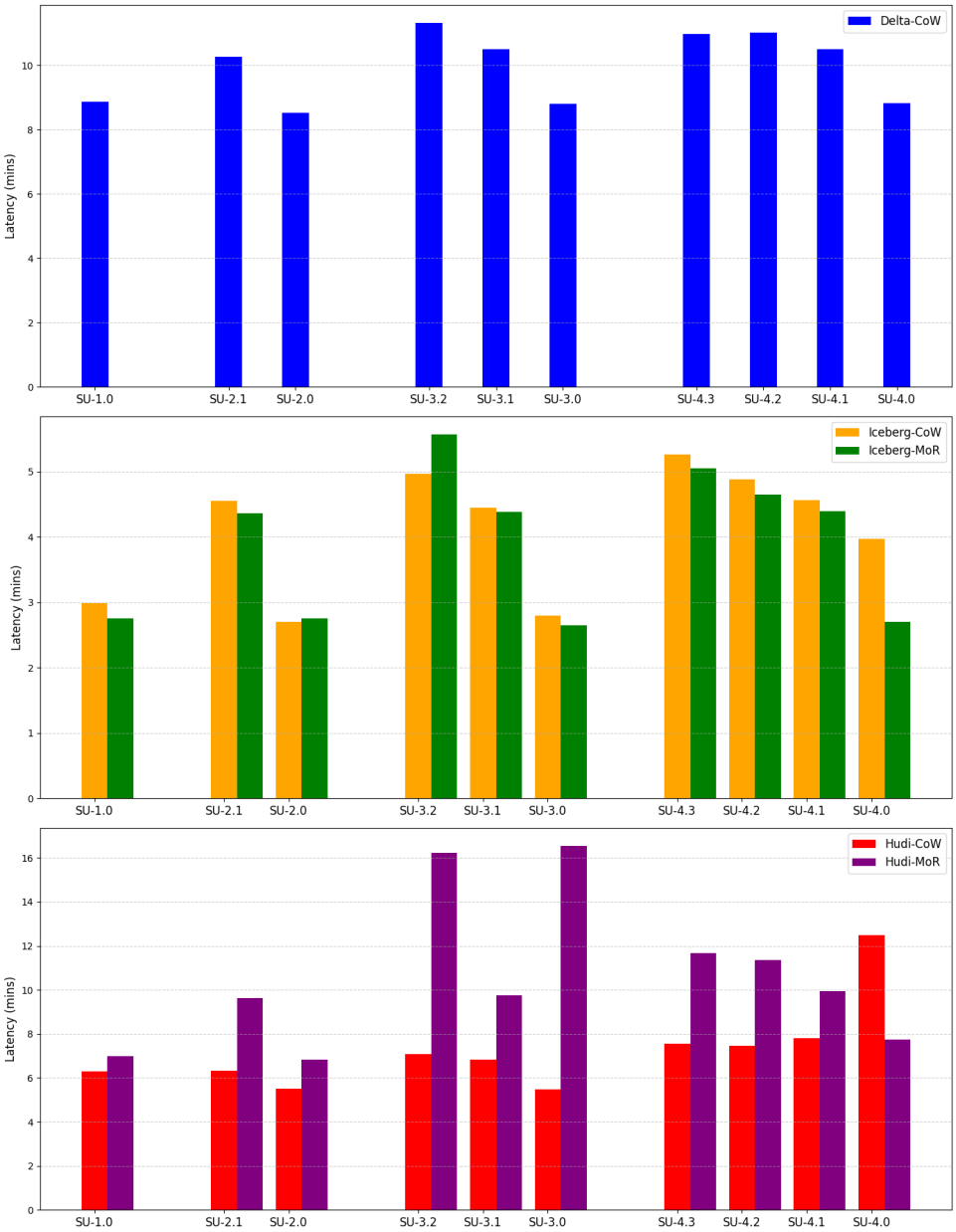
Σε μελέτη των Jain et al. [1], παρατηρήθηκε ότι στο Spark, οι φάσεις Single User παρουσιάζουν ταχύτερους χρόνους εκτέλεσης σε σχέση με τη σειριακή εκδοχή, όταν οι αντίστοιχες φάσεις Data Maintenance και Optimize εκτελούνται παράλληλα αλλά σε διαφορετικό cluster. Έτσι, υποδηλώνεται ότι η απομόνωση των εργασιών ανά cluster μπορεί να μειώσει την αλληλεπίδραση στους πόρους και να βελτιώσει την αποδοτικότητα των επιμέρους φάσεων.

4.5 Workload 4: Ερωτήματα time-travel

Στην παρούσα ενότητα παρουσιάζονται τα αποτελέσματα του Workload 4 (Σχήμα 3.1δ΄), το οποίο έχει σχεδιαστεί με στόχο την αξιολόγηση της απόδοσης των συστημάτων σε σενάρια εκτέλεσης ερωτημάτων Time Travel, δηλαδή στην ανάκτηση δεδομένων από προηγούμενες εκδόσεις των πινάκων. Το Σχήμα 4.11 απεικονίζει τους χρόνους εκτέλεσης των time-travel ερωτημάτων, με κάθε φάση Single User (SU) να συμβολίζεται ως SU-i.j, όπου το i δηλώνει τον αριθμό των ενημερώσεων (DM) που έχουν εφαρμοστεί στους πίνακες μέχρι εκείνη τη στιγμή, ενώ το j αντιστοιχεί στην έκδοση των δεδομένων στην οποία εκτελούνται τα ερωτήματα. Για παράδειγμα, στη φάση SU-2.0, οι πίνακες έχουν υποστεί δύο ενημερώσεις, και τα ερωτήματα αναφέρονται στην αρχική τους κατάσταση.

Όσον αφορά τα Delta και Iceberg, διαπιστώνεται ότι όταν εκτελείται ένα ερώτημα σε μία παλαιότερη έκδοση των πινάκων, τα νέα αρχεία που έχουν δημιουργηθεί λόγω των μεταγενέστερων ενημερώσεων δεν επηρεάζουν την καθυστέρηση του query. Για κάθε έκδοση των δεδομένων, ο χρόνος που απαιτείται για την εκτέλεση μίας SU φάσης παραμένει σταθερός, ανεξαρτήτως της χρονικής στιγμής κατά την οποία εκτελείται. Παρατηρώντας μετρικές όπως οι λειτουργίες I/O στον δίσκο και τα ποσοστά χρήσης του, είναι φανερό ότι δεν προκύπτουν διαφορές μεταξύ των SU φάσεων που εκτελούνται για την ίδια έκδοση των δεδομένων. Όπως ήταν αναμενόμενο, το Delta Lake και σε αυτό το workload αναδεικνύεται το πιο σταθερό σύστημα (με βάση τη μετρική stability) ως προς τον χρόνο εκτέλεσης των SU, με σημαντική διαφορά από τα υπόλοιπα. Συνεπώς, τα Delta και Iceberg αποδεικνύονται ιδιαίτερα αποδοτικά στην εκτέλεση time-travel ερωτημάτων, γεγονός που επιβεβαιώνεται και από την έρευνα των Jain et al. [1].

Το Hudi φαίνεται να ευθυγραμμίζεται σε γενικές γραμμές με τα ευρήματα που αναλύθηκαν προηγουμένως. Εξαιρουμένων ορισμένων περιπτώσεων, όπως οι φάσεις SU-3.2 και SU-3.0 για τη στρατηγική MoR, καθώς και η SU-4.0 για τη στρατηγική CoW, όπου παρατηρούνται μη αναμενόμενα αποτελέσματα, στις υπόλοιπες φάσεις οι καθυστερήσεις δεν παρουσιάζουν σημαντικές αποκλίσεις μεταξύ των εκτελέσεων που αναφέρονται στην ίδια έκδοση δεδομένων. Αυτό υποδηλώνει μία σχετικά καλή συμπεριφορά του συστήματος Hudi κατά την εκτέλεση time-travel ερωτημάτων, χωρίς ωστόσο να φαίνεται ότι είναι τόσο αποδιτικό όσο τα άλλα δύο.



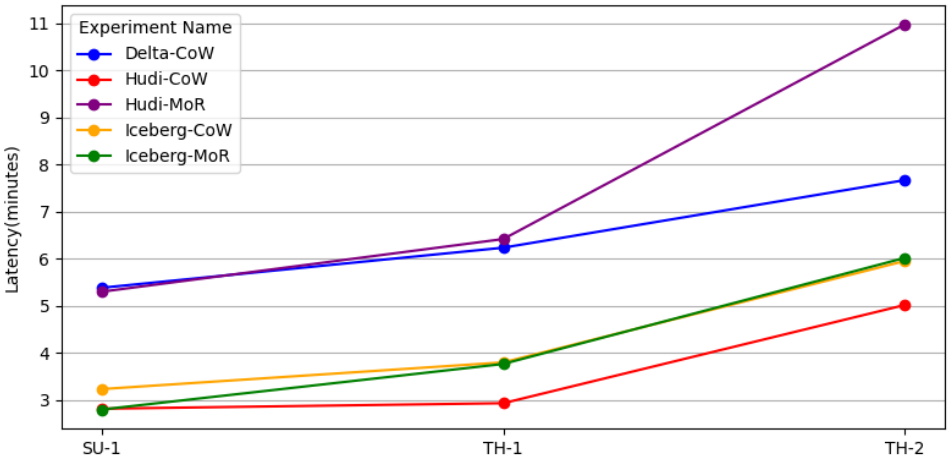
Σχήμα 4.11: *Καθυστέρηση Ερωτημάτων Time Travel*

4.6 Workload 5: Ταυτόχρονες αναγνώσεις χρηστών

Στην παρούσα ενότητα παρουσιάζονται τα αποτελέσματα του Workload 5 (Σχήμα 3.1ε΄), το οποίο έχει ως στόχο τη διερεύνηση της συμπεριφοράς των συστημάτων όταν ταυτόχρονα διαφορετικοί χρήστες εκτελούν ερωτήματα ανάγνωσης δεδομένων από τους πίνακες.

Στο Σχήμα 4.12 αποτυπώνονται οι χρόνοι απόκρισης των ερωτημάτων, όταν αυτά εκτελούνται από έναν χρήστη σε σύγκριση με την ύπαρξη περισσότερων χρηστών. Οι φάσεις SU-1 και TH-1 (Throughput) αναφέρονται στην αρχική μορφή των δεδομένων, χωρίς κάποια ενημέρωση ενδιάμεσα, ενώ πριν την TH-1 εκτελείται μία DM φάση. Βάσει των παράλληλων υλοποιήσεων που αναλύθηκαν στο Workload 3, τα αποτελέσματα είναι αναμενόμενα, καθώς παρατηρείται μία μικρή υποβάθμιση της απόδοσης όταν υπάρχουν ταυτόχρονοι χρήστες. Αξίζει να σημειω-

θεί ότι αρχικά επιχειρήθηκε η ταυτόχρονη εκτέλεση ερωτημάτων με περισσότερους από δύο παράλληλους χρήστες, αλλά κατά τις δοκιμές με το Hudi παρουσιάστηκαν σφάλματα εκτέλεσης, τα οποία εμπόδισαν την ολοκλήρωση των ερωτημάτων. Το πρόβλημα αυτό οδήγησε στον περιορισμό των ταυτόχρονων χρηστών σε δύο, ώστε τα αποτελέσματα να είναι συγκρίσιμα. Τα Delta και Iceberg ανταποκρίθηκαν κανονικά και σε περισσότερους χρήστες. Το γεγονός αυτό αναδεικνύει πιθανούς περιορισμούς της υλοποίησης Hudi σε σενάρια με αυξημένο φόρτο ταυτόχρονων αναγνώσεων.



Σχήμα 4.12: *Καθυστέρηση Συστημάτων σε Σενάριο Πολλαπλών Χρηστών*

Κεφάλαιο 5

Σύνοψη Αποτελεσμάτων

5.1 Σύνοψη και σχολιασμός αποτελεσμάτων

Στην παρούσα ενότητα παρουσιάζονται συνοπτικά τα βασικά αποτελέσματα των πειραμάτων, με στόχο την ανάδειξη των σημαντικότερων παρατηρήσεων που προέκυψαν από την ανάλυση. Τα ευρήματα που ακολουθούν συνοψίζουν την πειραματική αποτίμηση των Delta Lake, Apache Iceberg και Apache Hudi:

- Το Hudi εμφανίζει σημαντικά μεγαλύτερο χρόνο φόρτωσης δεδομένων σε σχέση με τα άλλα δύο, λόγω των ελέγχων μοναδικότητας κλειδιών και της κατανομής εγγραφών στα File Groups.
- Η στρατηγική CoW αποδείχθηκε πιο αποδοτική για αναγνώσεις δεδομένων, καθώς δεν απαιτεί συγχώνευση αρχείων όπως η MoR, γεγονός που οδήγησε σε χαμηλότερους χρόνους εκτέλεσης των ερωτημάτων και χαμηλότερο φόρτο στον δίσκο.
- Το Delta Lake εμφάνισε τη μεγαλύτερη σταθερότητα απόδοσης κατά τη διάρκεια των φάσεων SU, σε όλα τα πειράματα. Τα επιπλέον αρχεία που δημιουργούνται κατά τις ενημερώσεις φαίνεται να επηρεάζουν λιγότερο την καθυστέρηση εκτέλεσης των ερωτημάτων, σε σύγκριση με τα άλλα δύο συστήματα αποθήκευσης.
- Για τη διαδικασία Optimize, παρατηρείται ότι το Iceberg επωφελείται σημαντικά μέσω της συγχώνευσης αρχείων δεδομένων, σε αντίθεση με το Hudi, όπου το Clustering δεν βελτιώνει σημαντικά την απόδοση. Ωστόσο, το Iceberg εμφανίζει τη μεγαλύτερη καθυστέρηση κατά την εκτέλεση της διαδικασίας, ιδιαίτερα όταν δεν έχει προηγηθεί πρόσφατη φάση βελτιστοποίησης.
- Οι παράλληλες εκτελέσεις διαδικασιών οδήγησαν σε συνολική επιτάχυνση των πειραμάτων, αν και οι επιμέρους φάσεις, όπως οι Single User, παρουσίασαν κατά κανόνα επιβράδυνση όταν εκτελούνταν ταυτόχρονα με άλλες εργασίες, λόγω ανταγωνισμού στους πόρους.
- Κατά την εκτέλεση ερωτημάτων Time Travel, τα Delta Lake και Iceberg διατήρησαν σταθερή απόδοση για κάθε έκδοση των δεδομένων, επιδεικνύοντας υψηλή αποδοτικότητα και σταθερότητα, ενώ το Hudi παρουσίασε γενικά καλή συμπεριφορά, αλλά με ορισμένες αποκλίσεις και μικρότερη συνολική συνέπεια.

- Παρατηρήθηκε μικρή υποβάθμιση της απόδοσης με την παρουσία ταυτόχρονων χρηστών, ενώ το Hudi εμφάνισε σφάλματα σε περιπτώσεις με πολλούς χρήστες, σε αντίθεση με τα Delta και Iceberg, που διατήρησαν σταθερή συμπεριφορά.

Αξίζει να σημειωθει ότι στο πλαίσιο των πειραμάτων, χρησιμοποιήθηκαν οι προκαθορισμένες (default) ρυθμίσεις παραμετροποίησης για όλα τα εξεταζόμενα συστήματα αποθήκευσης. Τα συστήματα Delta Lake, Apache Iceberg και Apache Hudi προσφέρουν ευρύ σύνολο διαθέσιμων παραμέτρων και επιλογών ρύθμισης, οι οποίες εφόσον αξιοποιηθούν κατάλληλα, μπορούν να επιδράσουν σημαντικά στην απόδοση και τη λειτουργικότητα τους. Η περαιτέρω διερεύνηση των επιλογών αυτών θα μπορούσε να αποκαλύψει πρόσθετα πλεονεκτήματα ή ιδιαιτερότητες του κάθε συστήματος και να διαφοροποιήσει τα αποτελέσματα.

Ακόμα, η επιλογή και η υλοποίηση του υπολογιστικού περιβάλλοντος αποτελεί έναν παράγοντα που επηρεάζει σημαντικά την απόδοση των συστημάτων αποθήκευσης [1]. Στόχος της παρούσας μελέτης ήταν η πειραματική αποτίμηση και η συγκριτική αξιολόγηση των διαφορετικών LSTs υπό κοινές συνθήκες εκτέλεσης, ώστε τα αποτελέσματα να είναι συγκρίσιμα. Η υλοποίηση του περιβάλλοντος, η επιλογή του Apache Spark ως υπολογιστική μηχανή και η παραμετροποίησή του, καθώς και η χρήση του Apache Hadoop για την αποθήκευση των δεδομένων, είναι παράγοντες που επηρεάζουν σε σημαντικό βαθμό την έκβαση των πειραμάτων. Περαιτέρω παραμετροποίηση ή η χρήση εναλλακτικών τεχνολογιών είναι πιθανό να μπορούσαν να προσφέρουν επιπλέον πληροφορίες ως προς την αξιολόγηση των συστημάτων.

Συνολικά, τα αποτελέσματα της μελέτης αναδεικνύουν ότι η επιλογή του κατάλληλου συστήματος αποθήκευσης εξαρτάται σε μεγάλο βαθμό από τις απαιτήσεις του εκάστοτε σεναρίου χρήσης. Κανένα από τα εξεταζόμενα συστήματα δεν υπερέχει καθολικά έναντι των υπολοίπων, και η επιλογή του κατάλληλου συστήματος αποθήκευσης, καθώς και η βέλτιστη αξιοποίησή του, συνιστούν μια σύνθετη και περίπλοκη διαδικασία.

Κεφάλαιο 6

Επίλογος

6.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία ανέδειξε σημαντικές πτυχές της συμπεριφοράς και της απόδοσης των σύγχρονων συστημάτων αποθήκευσης δεδομένων Delta Lake, Apache Iceberg και Apache Hudi. Η μελέτη προσέφερε μια συγκριτική αποτίμηση των συστημάτων αποθήκευσης, σε διαφορετικά σενάρια χρήσης που αντανακλούν τις πραγματικές απαιτήσεις ενός σύγχρονου συστήματος διαχείρισης δεδομένων. Σημαντικό μέρος της εργασίας αποτέλεσε και η θεωρητική ανάλυση των συγκεκριμένων συστημάτων, με στόχο την κατανόηση των βασικών αρχών σχεδιασμού και λειτουργίας τους.

Μέσα από την εκτέλεση των πειραμάτων, αξιολογήθηκαν τα εξεταζόμενα συστήματα σε ποικίλες εργασίες, όπως αναγνώσεις και ενημερώσεις δεδομένων, ταυτόχρονες εκτελέσεις εργασιών, εκτέλεση ερωτημάτων σε παλαιότερες εκδόσεις των δεδομένων (time travel), καθώς και λειτουργίες συντήρησης των πινάκων με στόχο τη βελτιστοποίηση της απόδοσής τους. Η ανάλυση ανέδειξε τα πλεονεκτήματα και τους περιορισμούς κάθε συστήματος, προσφέροντας κατευθύνσεις για το πού είναι προτιμότερη η χρήση τους, ανάλογα με τις ανάγκες της εκάστοτε εφαρμογής.

Η εργασία συμβάλλει επίσης στην αξιολόγηση της απόδοσης των συστημάτων αποθήκευσης στο πλαίσιο ενός συγκεκριμένου υπολογιστικού περιβάλλοντος, βασισμένου στις τεχνολογίες Apache Spark και Apache Hadoop. Όπως έχει ήδη επισημανθεί, η υπολογιστική υποδομή αποτελεί έναν καθοριστικό παράγοντα που μπορεί να επηρεάσει σημαντικά την αποδοτικότητα και τη συμπεριφορά των εν λόγω συστημάτων [1].

Τα πειράματα που υλοποιήθηκαν στο πλαίσιο της παρούσας μελέτης είναι προσανατολισμένα στην ανάδειξη και αξιολόγηση πολλαπλών πτυχών της λειτουργικότητας των LSTs, καλύπτοντας ένα ευρύ φάσμα χαρακτηριστικών και επιδόσεων. Για τον λόγο αυτό, μπορούν να αποτελέσουν σημείο αναφοράς και αφετηρία για οποιοδήποτε εξειδικευμένο σενάριο χρήσης.

6.2 Μελλοντικές Επεκτάσεις

Το σύνολο της πειραματικής διαδικασίας και της ανάλυσης που πραγματοποιήθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας μπορεί να αποτελέσει αφετηρία για περαιτέρω διερεύνηση και εξέλιξη. Πιο συγκεκριμένα, προτείνονται οι εξής δυνατότητες μελλοντικής επέκτασης:

- Η χρήση του Apache Spark αποτέλεσε τη βάση για την παρούσα μελέτη. Ωστόσο, η αξιολόγηση των LST συστημάτων με άλλες μηχανές επεξεργασίας δεδομένων (π.χ. Trino) ενδέχεται να αναδείξει σημαντικές διαφορές στην απόδοση τους και στον τρόπο αλληλεπίδρασης μεταξύ τους. Επιπλέον, η αξιοποίηση διαφορετικών υποδομών αποθήκευσης, εκτός του Hadoop (π.χ. Azure Data Lake Storage), μπορεί να φέρει στην επιφάνεια κρίσιμες πτυχές της συμπεριφοράς των LSTs.
- Τα συστήματα Delta Lake, Apache Iceberg και Apache Hudi προσφέρουν πληθώρα επιλογών παραμετροποίησης και βελτιστοποίησης, και η παρούσα εργασία βασίστηκε κυρίως στις προκαθορισμένες (default) ρυθμίσεις τους. Μελλοντικές επεκτάσεις που θα εξετάζουν την απόδοση αυτών των συστημάτων δίνοντας ιδιαίτερη έμφαση στις παραμετροποιήσεις μπορούν να αποκαλύψουν νέες δυνατότητες των συστημάτων και να ενισχύσουν τα αποτελέσματα.
- Η εκτέλεση παρόμοιων πειραμάτων σε μεγαλύτερα clusters, με περισσότερους κόμβους και αυξημένα δεδομένα, θα μπορούσε να προσεγγίσει πιο ρεαλιστικά σενάρια χρήσης υψηλής κλίμακας και να ενισχύσει την αξιοπιστία της αξιολόγησης.

Βιβλιογραφία

[1] J. Camacho-Rodríguez, A. Agrawal, A. Gruenheid, A. Gosalia, C. Petculescu, J. Aguilar-Saborit, A. Floratou, C. Curino kai R. Ramakrishnan. *LST-Bench: Benchmarking Log-Structured Tables in the Cloud*. *Proceedings of the ACM on Management of Data*, 2(1):1--26, 2024.

[2] S.R. Julakanti, N.S.K. Sattiraju kai R.Julakanti. *Optimizing Storage Formats for Data Warehousing Efficiency*. *International Journal on Recent and Innovation Trends in Computing and Communication*, 9(5):71--78, 2021.

[3] *Change Data Capture (CDC)*. https://docs.databricks.com/aws/en/dlt/what-is-change-data-capture. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[4] H. Jianping, W. Yongyi, S. Fan kai X. Chengxi. *Compared Analysis of Row-based Storage and Column-based Storage*. *2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, Harbin, China, 2018.

[5] *Apache Parquet*. https://parquet.apache.org/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[6] *Apache ORC*. https://orc.apache.org/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[7] *Delta Lake*. https://delta.io/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[8] *Apache Iceberg*. https://iceberg.apache.org/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[9] *Apache Hudi*. https://hudi.apache.org/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[10] Y. Wu, J. Arulraj, J. Lin, R. Xian kai A. Pavlo. *An Empirical Evaluation of In-Memory Multi-Version Concurrency Control*. *Proceedings of the VLDB Endowment*, 10(7):781--792, 2017.

[11] J. Kim, K. Kim, H. Cho, J. Yu, S. Kang kai H. Jung. *Rethink the Scan in MVCC Databases*. *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*, New York, NY, USA, 2021.

[12] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H.van Hovell, A. Ionescu, A. Łuszczak, M. Świtakowski, M. Szafrański, X. Li, T. Ueshin, M. Mokhtar, P. Boncz, A. Ghodsi, S. Paranjpye, P. Senster, R. Xin kai M. Zaharia. *Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores*. *Proceedings of the VLDB Endowment*, 13(12):3411--3424, 2020.

[13] *Delta Lake Deletes*. https://delta.io/blog/2022-12-07-delete-rows-from-delta-lake-table/. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[14] *Delta Lake Optimizations*. https://docs.delta.io/latest/optimizations-oss.html. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[15] *Apache Iceberg Concurrency Control*. https://iceberg.apache.org/docs/1.6.0/reliability/. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[16] *Apache Iceberg Deletes*. https://iceberg.apache.org/spec/#delete-formats. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[17] *Apache Iceberg Maintenance*. https://iceberg.apache.org/docs/nightly/maintenance/. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[18] *Apache Hudi Concurrency Control*. https://hudi.apache.org/docs/concurrency_control/. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[19] *Apache Hudi Deletes*. https://hudi.apache.org/docs/next/write_operations/. Ημερομηνία πρόσβασης: Απρίλιος 2025.

[20] *Apache Hudi Compaction*. https://hudi.apache.org/docs/compaction/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[21] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica kai M. Zaharia. *Analyzing and Comparing Lakehouse Storage Systems*. *Proceedings of the 13th Annual Conference on Innovative Data Systems Research (CIDR '23)*, Amsterdam, The Netherlands, 2023.

[22] A. Kudinkin. *Apache Hudi vs Delta Lake - Transparent TPC-DS Data Lakehouse Performance Benchmarks*. https://www.onehouse.ai/blog/apache-hudi-vs-delta-lake-transparent-tpc-ds-lakehouse-performance-benchmarks. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[23] Data Beans. *Delta vs Iceberg vs hudi: Reassessing Performance*. https://databeans-blogs.medium.com/delta-vs-iceberg-vs-hudi-reassessing-performance-cb8157005eb0. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[24] R. Ganesh. *Apache Hudi vs Delta Lake vs Apache Iceberg!* https://medium.com/@rganesh0203/apache-hudi-vs-delta-lake-vs-apache-iceberg-87025e60b7e5. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[25] A. Banerjee. *Open Table Formats for Efficient Data Processing: Delta Lake vs Iceberg vs Hudi*. https://medium.com/starschema-blog/open-table-formats-for-efficient-data-processing-delta-lake-vs-iceberg-vs-hudi-b1107141e9a6. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[26] *LST-Bench*. https://github.com/microsoft/lst-bench. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[27] Transaction Processing Performance Council. *TPC Benchmark™ DS: Standard Specification, Version 4.0.0*. Technical Report, Transaction Processing Performance Council (TPC), 2024.

[28] *Azure Log Analytics Workspace*. https://learn.microsoft.com/en-us/azure/azure-monitor/logs/log-analytics-workspace-overview. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[29] *Apache Spark*. https://spark.apache.org/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[30] *Apache Hadoop*. https://hadoop.apache.org/. Ημερομηνία πρόσβασης: Μάρτιος 2025.

[31] *Apache Iceberg Rewrite*. https://iceberg.apache.org/javadoc/1.1.0/org/apache/iceberg/actions/RewriteDataFiles.html#MAX_CONCURRENT_FILE_GROUP_REWRITES. Ημερομηνία πρόσβασης: Μάρτιος 2025.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλπ	βλέπε
κ.ά.	και άλλα
π.χ.	παραδείγματος χάριν
LST	Log-Structured Table
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
MVCC	Multi-Version Concurrency Control
CoW	Copy on Write
MoR	Merge on Read
NBCC	Non Blocking Concurrency Control
CPU	Central Processing Unit
W{1,2,3,4,5}	Workload {1,2,3,4,5}
SU	Single User
DM	Data Maintenance
O	Optimize
I/O	Input/Output
VM	Virtual Machine
GB	Gigabytes

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
μορφές αποθήκευσης	storage formats
δεδομένα	data
ανά γραμμή	row-based
αποθήκη δεδομένων	data warehouse
αποθήκευση ανά στήλη	columnar storage
παρακολούθηση αλλαγών δεδομένων	change data capture
φόρτος εργασίας	workload
ερώτημα	query
πρόσβαση σε παρελθοντικά δεδομένα	time travel
καταγραφή	log
σημείο ελέγχου	checkpoints
αντιγραφλη κατά την εγγραφή	copy on write
συγχώνευση κατά την ανάγνωση	merge on read
αρχείο	file
δίσκος	disk
συγχώνευση	compaction
συντήρηση	maintenance
βελτιστοποίηση	optimize
στιγμιότυπο	snapshot
χρονική ακολουθία	timeline
μεταδεδομένα	metadata
διαγραφή	delete
συσταδοποίηση	clustering
παράλληλα	concurrent
καθυστέρηση	latency
σταθερότητα	stability
εικονική μηχανή	virtual machine
υπο-αξιοποίηση	underutilization
επιτάχυνση	speedup
συστάδα	cluster