ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑ ΔΙΔΑΚΤΙΚΗΣ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΚΑΤΕΥΘΥΝΣΗ ΗΛΕΚΤΡΟΝΙΚΗΣ ΜΑΘΗΣΗΣ

Επεξεργασία Skyline Επερωτήσεων σε Πολυδιάστατα Δεδομένα στο Υπολογιστικό Νέφος

Μεταπτυχιακή Διπλωματική Εργασία

Μπισιώτη Δ. Αρετή

Επιβλέπων: Δουλκερίδης Χρήστος
Λέκτορας Πανεπιστημίου Πειραιώς

Πειραιάς, 2013

Περίληψη

Η εργασία αναφέρεται και στοχεύει στην «Επεξεργασία Skyline Επερωτήσεων σε Πολυδιάστατα Δεδομένα στο Υπολογιστικό Νέφος».

Αρχικά, παρουσιάζεται το Υπολογιστικό Νέφος και η χρησιμότητα των Skyline Επερωτήσεων όσο αφορά στην επεξεργασία πολυδιάστατων δεδομένων.

Στη συνέχεια, παρουσιάζεται το framework του Hadoop-MapReduce στο οποίο βασίζεται και η υλοποίηση της παρούσας εργασίας και περιγράφονται οι φάσεις της παράλληλης επεξεργασίας πολυδιάστατων δεδομένων από τις Skyline Επερωτήσεις.

Κατόπιν, θέτονται οι στόχοι της εργασίας και περιγράφεται αναλυτικά η σχεδίαση του κώδικα σε επίπεδο αρχιτεκτονικής αλλά και κλάσεων. Συγκεκριμένα περιγράφονται οι τρείς μέθοδοι διαμερισμού του χώρου, δηλαδή Διαμερισμός Χώρου με βάση Grid τεχνική, Διαμοιρασμός Χώρου με βάση Γωνιακές Συντεταγμένες και Διαμοιρασμός Χώρου μέσω Hyperplane Προβολών, αλλά και ο τρόπος υπολογισμού των skyline σημείων.

Ακολούθως, περιγράφεται το περιβάλλον στο οποίο έγινε η πειραματική μελέτη. Η πειραματική μελέτη αποτελείται από τρία σενάρια, τα οποία διαφέρουν μεταξύ τους στον αριθμό των δεδομένων προς επεξεργασία, την κατανομή τους και στον αριθμό των διαμερίσεων που πρέπει να δημιουργηθούν. Τα αποτελέσματα των πειραμάτων αναπαρίστανται και γραφικά.

Τέλος, καταγράφονται τα συμπεράσματα τα οποία προέκυψαν από την πειραματική μελέτη και ακολουθούν προτάσεις για μελλοντική έρευνα.

Λέξεις κλειδιά: skyline επερωτήσεις, hadoop, map-reduce, διαμοιρασμός χώρου, γωνιακές συντεταγμένες, grid τεχνική, hyperplane προβολή.

Abstract

This thesis refers to and aims at the "Processing of Skyline Queries over Multidimensional Data in Cloud Computing".

In the first part of the thesis, Cloud Computing is introduced along with the necessity of Skyline Query in terms of the processing of Multidimensional Data.

Next, we deal with the presentation of Hadoop-MapReduce framework – on which the current project is based- as well as with the description of the phases of the parallel processing of Multidimensional Data by Skyline Queries.

In the next chapter, the focus is on the goal setting of the thesis and on the designing of the code with regard to architecture and classes. In particular, the three methods of space partitioning that is Grid Partitioning, Angle-based Partitioning, Partitioning via Hyperplane Projection, are presented as well as the way of Skyline Computation.

In the following chapter, a description of the environment, where the experimental analysis was carried out, takes place. This analysis consists of three scenarios, which are different when it comes to the number of the data to be processed and their distribution and the number of partitions to be created. There is also, a graphic representation of the experiment results.

Finally, conclusions of the study are made as well as suggestions for future research.

Key words: skyline queries, hadoop, map-reduce, space partitioning, grid partitioning, angle-based partitioning, hyperplane projection

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον καθηγητή μου, κύριο **Δουλκερίδη Χρήστο**, για την ανάθεση της παρούσας εργασίας, για την αμέριστη και συνεχή συμπαράσταση και βοήθεια που μου παρείχε για την ολοκλήρωση της εργασίας καθώς και για την υπομονή του.

Επίσης, θα ήθελα να ευχαριστήσω τη συμφοιτήτριά μου και φίλη μου Γεωργάτου Χρυσούλα για την ηθική συμπαράσταση που μου πρόσφερε μέχρι το πέρας την εργασίας.

Τέλος, θερμές ευχαριστίες θα ήθελα να εκφράσω στην οικογένειά μου για την αγάπη και τη στήριξη που μου προσέφεραν στα χρόνια φοίτησής μου στο πανεπιστήμιο.

Περιεχόμενα

Κατάλογος σχημάτων

1. ΕΙΣΑΓΩΓΗ

Η ηλεκτρονική μάθηση μέχρι τώρα έχει κερδίσει σημαντικό έδαφος στην εξ αποστάσεως εκπαίδευση. Πέρα όμως από το εκπαιδευτικό της κομμάτι, μεγάλο μέρος της βασίζεται στις νέες τεχνολογίες είτε γιατί διευκολύνουν την εκπαιδευτική διαδικασία είτε γιατί χρησιμοποιούνται ως τεχνολογικό υπόβαθρο για την καλή λειτουργία των εφαρμογών που εξυπηρετούν το ηλεκτρονική μάθηση.

Τα υπάρχοντα μοντέλα ηλεκτρονικής μάθησης υστερούν σε υποδομές, οι οποίες παρέχουν δυναμικά τις απαιτούμενες δυνατότητες αποθήκευσης και υπολογισμού. Η τεχνολογική υποδομή αποτελεί ένα από τα σημαντικότερα ‘συστατικά’ ενός συστήματος ηλεκτρονικής μάθησης και έχει άμεσο αντίκτυπο στη βελτίωση και την αειφορία του [1].

Κατά τη διάρκεια της εκπαιδευτικής διαδικασίας χρησιμοποιούνται εκπαιδευτικά αντικείμενα, όπως βίντεο, κείμενα, blogs, websites και άλλα τα οποία πρέπει να είναι διαθέσιμα στους εμπλεκόμενους ανά πάσα στιγμή. Τα εκπαιδευτικά αντικείμενα μπορεί να έχουν πολλά χαρακτηριστικά όπως ο αριθμός των χρηστών που τα έχουν χρησιμοποιήσει, η αξιολόγησή τους, η ημερομηνία δημιουργίας τους, η ταυτότητα του χρήστη που τα δημιούργησε κλπ. Επίσης, τα εκπαιδευτικά αυτά αντικείμενα είναι αποθηκευμένα, μαζί με τις παραμέτρους που τα χαρακτηρίζουν, σε διάφορες κατανεμημένες βάσεις δεδομένων ή ‘αποθήκες’ μαθησιακών αντικειμένων όπως λέγονται στη γλώσσα της ηλεκτρονικής μάθησης. Επομένως, πρέπει να διασφαλιστεί η ποιότητα και η σωστή λειτουργία του τεχνικού μέρους που υποστηρίζει όλη αυτή τη διαδικασία έτσι ώστε κάθε χρήστης να μπορεί να επιλέξει τα καλύτερα γι’ αυτόν αντικείμενα, να τα ανασύρει και να τα ταξινομήσει με βάση τα χαρακτηριστικά τους.

Το Υπολογιστικό Νέφος είναι μια πολλά υποσχόμενη υποδομή, η οποία μπορεί να προσφέρει τεράστια οφέλη σ’ ένα σύστημα ηλεκτρονικής μάθησης εξαιτίας της δυνατότητας να προσφέρει υπολογιστικούς και αποθηκευτικούς πόρους ως υπηρεσίες [1]. Επίσης, επιτρέπει σ’ ένα σύστημα ηλεκτρονικής μάθησης να έχει μια τεχνολογική υποδομή, η οποία είναι αξιόπιστη, ευέλικτη, αποδοτική από πλευράς

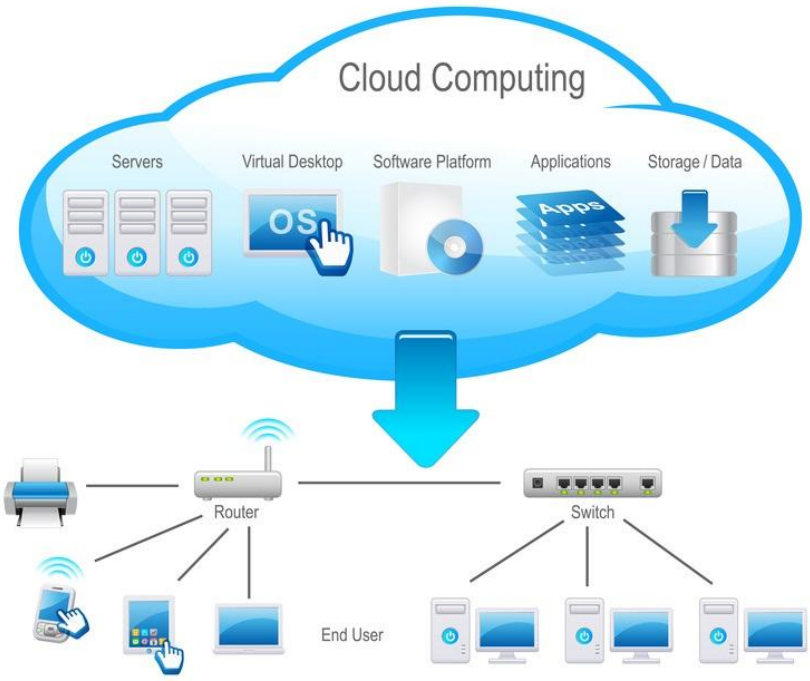
κόστους, αυτορυθμιζόμενη και διασφαλιζόμενη από πλευράς ποιότητας υπηρεσιών [1].

Στην παρούσα εργασία, η τεχνολογία η οποία θα μελετηθεί και ίσως θα μπορούσε να βελτιώσει την τεχνολογική υποδομή των συστημάτων ηλεκτρονικής μάθησης είναι το υπολογιστικό νέφος και συγκεκριμένα, η αποδοτική εκτέλεση skyline επερωτήσεων σε πολυδιάστατα δεδομένα σε αυτό.

1.1 Υπολογιστικό Νέφος

Η ανάπτυξη της υποδομής των δικτύων καθιστά ικανή την παροχή διάφορων υπηρεσιών στο Internet, η οποία αποκαλείται 'Cloud Computing' [2]. Το υπολογιστικό νέφος ή αλλιώς Cloud Computing είναι ένα μοντέλο που επιτρέπει στους χρήστες να έχουν εύκολη πρόσβαση και να διαχειρίζονται μεγάλο όγκο απομακρυσμένων υπολογιστικών πόρων. Αυτό το μοντέλο έχει κερδίσει έδαφος κυρίως εξαιτίας της ευκολίας χρήσης του και της δυνατότητάς του να προσαρμόζεται ανάλογα με τη ζήτηση [3]. Στο περιβάλλον του Υπολογιστικού Νέφους κάθε πάροχος υπηρεσιών βρίσκεται σε διαφορετικές τοποθεσίες και συλλέγει δεδομένα στη δική του βάση δεδομένων. Μάλιστα, κάποιες υπηρεσίες συλλέγουν πληροφορίες σε κατανεμημένες βάσεις, κι έτσι, οι πληροφορίες των χρηστών αποθηκεύονται ξεχωριστά στο Internet [2]. Τελευταία, μάλιστα το υπολογιστικό νέφος χρησιμοποιείται, εκτός από τους υπόλοιπους τομείς και στην εκπαίδευση. Εξαιτίας των ευκολιών που παρέχει το υπολογιστικό νέφος, προσφέρεται ήδη από αρκετούς παρόχους υπηρεσιών όπως οι Amazon, IBM, Microsoft και Yahoo! [4].

Στην Εικόνα 1, παρακάτω, απεικονίζεται σε γενικές γραμμές η αρχιτεκτονική του υπολογιστικού νέφους.



Εικόνα 1: Αρχιτεκτονική του Υπολογιστικού Νέφους [5].

1.2 Skyline Επερωτήσεις και Πολυδιάστατα Δεδομένα

Τα δεδομένα, τελευταία, έπαψαν να είναι μονοδιάστατα και περιγράφονται από πολλαπλά χαρακτηριστικά. Ας σκεφτούμε μια βάση δεδομένων, η οποία περιέχει πληροφορίες για ξενοδοχεία. Κάθε εγγραφή της βάσης δεδομένων αναπαρίσταται σαν ένα σημείο στο χώρο που αποτελείται από πολλές διαστάσεις. Αν υποθέσουμε ότι έχουμε 2 άξονες x και y, τότε στη διάσταση y αναπαρίσταται η τιμή του ξενοδοχείου και στη x διάσταση, η απόστασή του από ένα σημείο πχ, τη θάλασσα [6]. Τέτοια δεδομένα λέγονται πολυδιάστατα και αυτά θα είναι ένα από τα αντικείμενα μελέτης της εργασίας.

Η επεξεργασία τέτοιων δεδομένων είναι αρκετά δύσκολη και πολύπλοκη καθώς δεν υπάρχει περιορισμός στις διαστάσεις των σημείων. Όπως θα δούμε και σε επόμενα κεφάλαια, οι Skyline Επερωτήσεις, ενώ είναι οι πλέον κατάλληλες για την επεξεργασία τέτοιων δεδομένων είναι ακριβές υπολογιστικά, γι' αυτό είναι αναγκαία η ύπαρξη αποδοτικών και κλιμακώσιμων υποδομών για την επεξεργασία τους, όπως το υπολογιστικό νέφος.

1.3 Ορισμός προβλήματος και στόχοι

Από τη βιβλιογραφική έρευνα, η οποία παρουσιάζεται σε επόμενο κεφάλαιο, είναι προφανή τα πλεονεκτήματα του Υπολογιστικού Νέφους σε διάφορες υπηρεσίες αλλά και της παράλληλης επεξεργασίας πολυδιάστατων δεδομένων μέσω των Skyline Επερωτήσεων. Στην πορεία αναπτύχθηκαν διάφορες τεχνικές, οι οποίες αφορούν στη βελτίωση των Skyline Επερωτήσεων και κυρίως στο διαμοιρασμό του χώρου. Σύντομα, λοιπόν, προέκυψε η ιδέα του συνδυασμού αυτών των δυο τεχνολογιών, δηλαδή του Υπολογιστικού Νέφους και των Skyline Επερωτήσεων. Κεντρική ιδέα της εργασίας είναι να επεξεργαστούν παράλληλα διάφορα πολυδιάστατα δεδομένα με τη χρήση Skyline Επερωτήσεων αλλά αυτή τη φορά σε περιβάλλον Υπολογιστικού Νέφους.

Εν ολίγοις, ο βασικός στόχος που πρέπει να έχει επιτευχθεί μετά το πέρας της παρούσας εργασίας είναι:

- Να εκτελεστούν αποδοτικά Skyline Επερωτήσεις πολυδιάστατων δεδομένων στο Υπολογιστικό Νέφος.

Τεχνικοί στόχοι που οδηγούν στην επίτευξη του βασικού στόχου είναι:

- Να αναπτυχθεί κώδικας με συγκεκριμένη δομή, κατάλληλη έτσι ώστε να μπορέσει να εκτελεστεί στο περιβάλλον του Υπολογιστικού Νέφους.
- Να προσαρμοστεί ο ήδη υπάρχων κώδικας έτσι ώστε να μπορέσει να εκτελεστεί στο περιβάλλον του Υπολογιστικού Νέφους.

1.4 Διάρθρωση εργασίας

Η εργασία είναι οργανωμένη ως εξής:

Κεφάλαιο 1: Στο πρώτο κεφάλαιο περιγράφεται σε γενικές γραμμές το Υπολογιστικό Νέφος και οι Skyline Επερωτήσεις σε Πολυδιάστατα Δεδομένα. Επίσης, διατυπώνεται το πρόβλημα που καλείται να αντιμετωπίσει η εργασία καθώς και ο στόχος της.

Κεφάλαιο 2: Εδώ, περιγράφονται τα προαπαιτούμενα που πρέπει να γνωρίζει κανείς σε σχέση με τη λειτουργία του Hadoop – MapReduce και των Skyline Επερωτήσεων σε πολυδιάστατα δεδομένα. Περιγράφονται αναλυτικά οι τεχνικές που χρησιμοποιούνται για το διαμοιρασμό του χώρου και την παράλληλη επεξεργασία των Skyline Επερωτήσεων.

Κεφάλαιο 3: Στο τρίτο κεφάλαιο, διατυπώνονται ο βασικός και οι επιμέρους στόχοι της εργασίας.

Κεφάλαιο 4: Στο τέταρτο κεφάλαιο, περιγράφεται η γενική σχεδίαση και η αρχιτεκτονική του κώδικα, δηλαδή πώς αλληλεπιδρούν μεταξύ τους τα κομμάτια του κώδικα αλλά και με το σύστημα αρχείων του Hadoop.

Κεφάλαιο 5: Στο πέμπτο κεφάλαιο, περιγράφεται λεπτομερώς σε επίπεδο κλάσεων, η υλοποίηση του κώδικα. Περιγράφεται ακριβώς πώς υλοποιείται η κάθε κλάση και για ποιο σκοπό χρησιμοποιείται.

Κεφάλαιο 6: Στο έκτο κεφάλαιο, περιγράφεται το περιβάλλον στο οποίο πραγματοποιήθηκε η πειραματική μελέτη, τα σενάρια εκτέλεσης του κώδικα καθώς και τα αποτελέσματα που προέκυψαν.

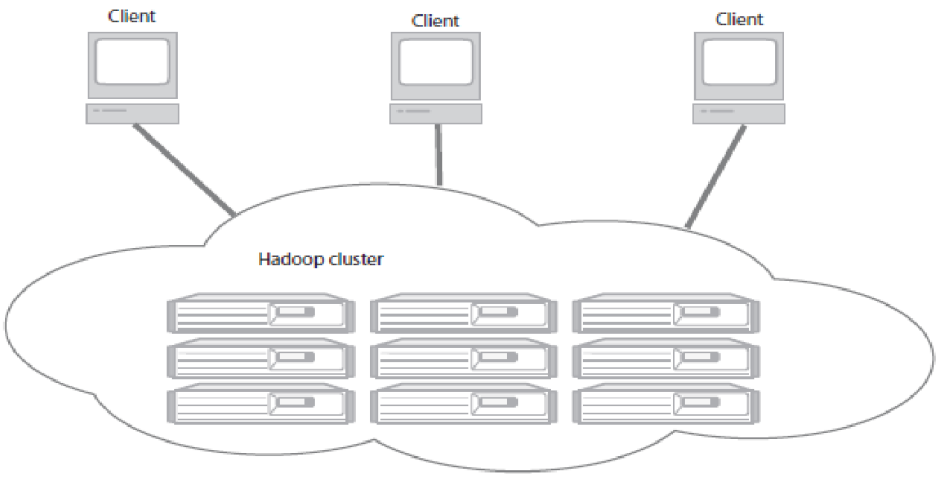
Κεφάλαιο 7: Στο τελευταίο κεφάλαιο, συνοψίζεται η εργασία, περιγράφονται συγκεντρωτικά τα αποτελέσματα των πειραμάτων και ακολουθούν προτάσεις για μελλοντική έρευνα.

2. ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ ΚΑΙ ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

2.1 MapReduce και Hadoop

Καθώς οι ολοένα αυξανόμενοι χρήστες 'παράγουν' συνεχώς δεδομένα, ανεβάζοντας στο διαδίκτυο βίντεο, φωτογραφίες, ενημερώνοντας τα προφίλ τους στις σελίδες κοινωνικής δικτύωσης, τα υπάρχοντα εργαλεία καθίστανται όλο και πιο ανεπαρκή να επεξεργαστούν τόσο μεγάλο όγκο δεδομένων. Αυτή την ανεπάρκεια, έρχεται να καλύψει το MapReduce [7].

Το MapReduce, είναι ένα προγραμματιστικό μοντέλο, το όποιο προορίζεται για την παράλληλη επεξεργασία τεράστιου όγκου δεδομένων, του οποίου η κύρια ιδέα είναι να κρύψει τις λεπτομέρειες της παράλληλης εκτέλεσης και να επιτρέψει στους χρήστες να επικεντρωθούν στις στρατηγικές επεξεργασίας δεδομένων [8].



Εικόνα 2: Hadoop cluster [7].

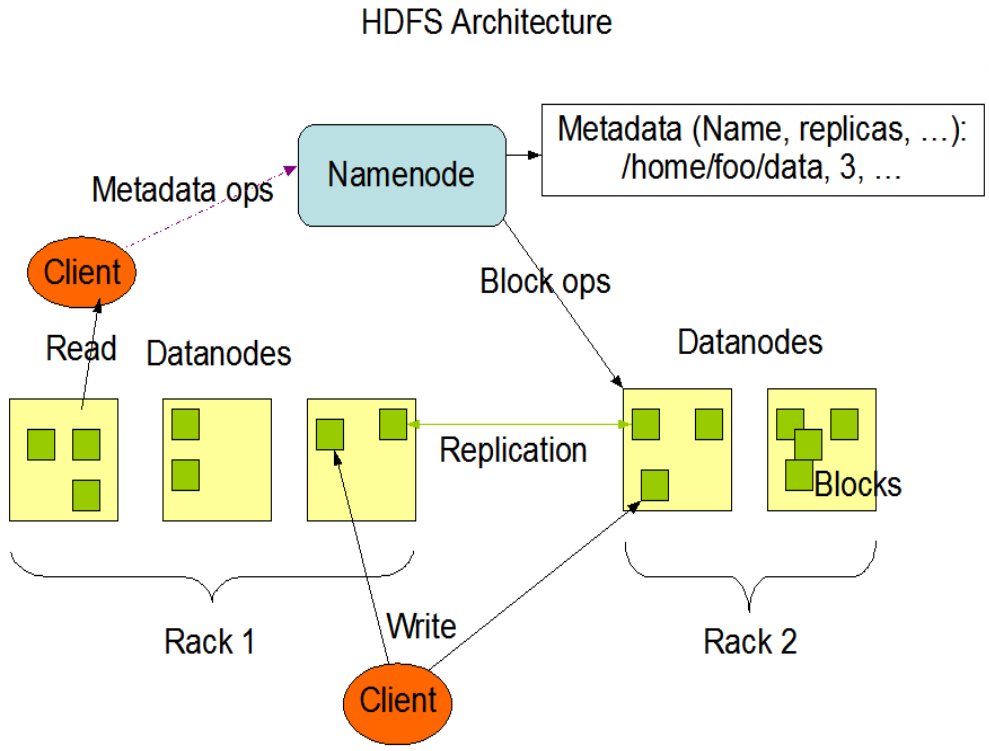
Το Hadoop, στο οποίο βασίζεται η εργασία, είναι μια open source υλοποίηση του MapReduce, η οποία έχει χρησιμοποιηθεί ευρέως από ένα πλήθος οργανισμών [8]. Πιο συγκεκριμένα, το Hadoop είναι ένα framework που διευκολύνει την ανάπτυξη εφαρμογών οι οποίες επεξεργάζονται τεράστιο όγκο δεδομένων παράλληλα σε μεγάλες συστάδες (χιλιάδες κόμβους) με αξιοπιστία και ανοχή σε σφάλματα.

Το Hadoop αποτελείται από δύο επίπεδα: το επίπεδο αποθήκευσης δεδομένων που λέγεται Hadoop DFS (HDFS) και το επίπεδο επεξεργασίας δεδομένων που λέγεται Hadoop MapReduce Framework. Το HDFS είναι ένα σύστημα αρχείων το οποίο χρησιμοποιεί το Hadoop για να διαβάζει τα αρχεία προς επεξεργασία καθώς και για να 'γράφει' εκεί τα αποτελέσματα.

2.1.1 Σύστημα Αρχείων του Hadoop

Το HDFS έχει master/slave αρχιτεκτονική. Μια συστάδα HDFS αποτελείται από έναν μόνο NameNode, έναν master server που διαχειρίζεται το namespace του συστήματος αρχείων και ρυθμίζει την πρόσβαση των πελατών στα αρχεία. Συμπληρωματικά υπάρχει ένας αριθμός DataNodes, συνήθως ένας ανά κάθε κόμβο της συστάδας, που ελέγχουν την αποθήκευση στον κόμβο πάνω στον οποίο τρέχουν. Το HDFS εκθέτει ένα file system namespace και επιτρέπει στα δεδομένα των χρηστών να αποθηκευτούν σε αρχεία. Εσωτερικά κάθε αρχείο σπάει σε ένα ή περισσότερα μπλοκς τα οποία με τη σειρά τους αποθηκεύονται στους datanodes. Ο NameNode εκτελεί εργασίες του namespace του συστήματος αρχείων όπως άνοιγμα, κλείσιμο και μετονομασία αρχείων και φακέλων. Επιπρόσθετα καθορίζει την αντιστοίχιση των μπλοκς στους datanodes. Οι DataNodes είναι υπεύθυνοι για την εξυπηρέτηση αιτημάτων ανάγνωσης και εγγραφής από τους πελάτες του συστήματος αρχείων. Επιπλέον δημιουργούν, διαγράφουν και αντιγράφουν μπλοκς μετά από οδηγίες του NameNode.

Η ύπαρξη ενός και μοναδικού NameNode στη συστάδα απλοποιεί σημαντικά την αρχιτεκτονική του συστήματος. Ο NameNode κρατά και ελέγχει όλα τα μεταδεδομένα του HDFS. Με τον τρόπο αυτό τα μεταδεδομένα είναι αποκομμένα από τα δεδομένα του συστήματος το οποίο έχει σχεδιαστεί με τέτοιο τρόπο ώστε τα δεδομένα των χρηστών να μην περνάνε πότε μέσα από το NameNode.



Εικόνα 3: Αρχιτεκτονική του HDFS [37].

Μια πλήρως διαμορφωμένη HDFS συστάδα τρέχει ένα σετ από daemons που περιλαμβάνουν:

1. **NameNode** : Είναι ο master του HDFS ο οποίος διευθύνει τους σκλάβους DataNode daemons.

2. **DataNode** : είναι ο εργάτης του κατανεμημένου συστήματος αρχείων ο οποίος γράφει και διαβάζει HDFS μπλοκς σε πραγματικά αρχεία στο τοπικό σύστημα αρχείων.

3. **SecondaryNameNode** : Είναι ένας βοηθητικός daemon για την παρακολούθηση της κατάστασης της HDFS συστάδας.

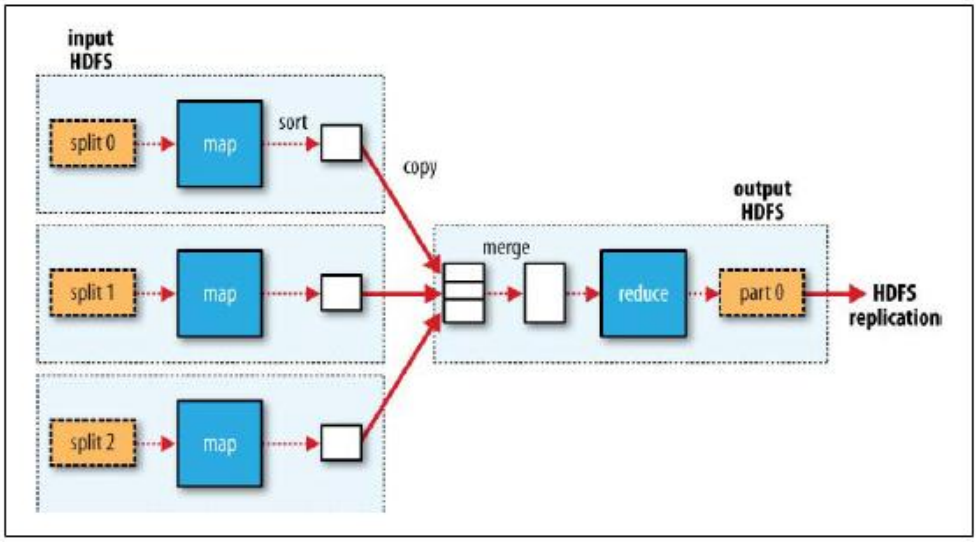
4. **JobTracker** : Είναι ο σύνδεσμος ανάμεσα στις εφαρμογές-πελάτες και στο Hadoop, αποφασίζει το πλάνο εκτέλεσης για τις καταχωρημένες εργασίες, αναθέτει διαφορετικές εργασίες στους κόμβους και παρακολουθεί όλες τις εργασίες που τρέχουν.

5. **TaskTracker** : Είναι υπεύθυνος για την εκτέλεση των μεμονωμένων εργασιών που αναθέτει ο JobTracker.

2.1.2 MapReduce Framework

Ένα MapReduce job εκτελείται σε δύο φάσεις: την map και τη reduce. Επίσης μπορεί να χωριστεί σε πολλά map tasks και reduce tasks, ανάλογα με τον αριθμό των Mappers και Reducers που θα ορίσει ο χρήστης. Σημαντικό ρόλο παίζει ο master node, ο οποίος διαλέγει κενούς «εργάτες» και τους αναθέτει ένα map ή reduce task ανάλογα με το στάδιο. Πριν ξεκινήσει ένα Map task, ένα αρχείο εισόδου φορτώνεται στο σύστημα αρχείων του Hadoop. Κατά τη διάρκεια του φορτώματος, το αρχείο χωρίζεται σε πολλαπλά blocks δεδομένων τα οποία έχουν το ίδιο μέγεθος και κάθε block αντιγράφεται τρείς φορές ώστε να είναι εγγυημένη η ανοχή σφαλμάτων. Στη συνέχεια, κάθε block ανατίθεται σ' ένα mapper και αυτός εφαρμόζει την Map() σε κάθε εγγραφή του block δεδομένων. Τα ενδιάμεσα αποτελέσματα που παράγονται από τους mappers, ταξινομούνται τοπικά σε ζεύγη κλειδιού-τιμής με βάση το κλειδί και στη συνέχεια, τα αποτελέσματα αποθηκεύονται στους τοπικούς δίσκους των mappers όπου χωρίζονται σε R κομμάτια, όπου R είναι ο αριθμός των Reducers που χρησιμοποιούνται.

Όταν όλα τα Map tasks ολοκληρωθούν, θα ξεκινήσει η reduce φάση όπου θα ανατεθούν στους «εργάτες» τα Reduce tasks. Ο Reducer διαβάζει τα ενδιάμεσα αποτελέσματα και τα συγχωνεύει με βάση τα ενδιάμεσα κλειδιά έτσι ώστε τιμές του ίδιου κλειδιού να ομαδοποιούνται με βάση αυτό. Στη συνέχεια, ο Reducer εφαρμόζει την μέθοδο Reduce() στις ενδιάμεσες τιμές για κάθε κλειδί που συναντά και τα τελικά αποτελέσματα αποθηκεύονται και αντιγράφονται τρείς φορές στο HDFS.



Εικόνα 4: Διάγραμμα ροής ενός Map-Reduce job με έναν reducer [9].

Σημαντικό πλεονέκτημα του MapReduce είναι η ανοχή σφαλμάτων και αυτό επιτυγχάνεται ανιχνεύοντας κόμβους που δε λειτουργούν πια και αναθέτοντας ξανά τα tasks τους σε άλλους υγιείς. Επίσης, κόμβοι που έχουν ολοκληρώσει τα tasks τους μπορούν να δεχτούν νέα κόμβοι δεδομένων και να ξεκινήσουν νέα tasks. Ακόμα, tasks που εκτελούνται σε αργούς κόμβους μετατίθενται σε άλλους κενούς που έχουν ολοκληρώσει τα προηγούμενα tasks τους, χωρίς βέβαια αυτό να σημαίνει ότι θα εκτελεστούν γρηγορότερα [3].

2.2 Βελτιώσεις του MapReduce

Για όλα τα παραπάνω χαρακτηριστικά, το MapReduce και γενικότερα τα υπολογιστικά νέφη έχουν κερδίσει έδαφος και έχουν γίνει διάφορες μελέτες και πειράματα σε σχέση με την απόδοσή τους συγκριτικά με άλλα συστήματα. Παρακάτω θα δούμε κάποιες από αυτές τις έρευνες.

Οι C.Doulkeridis et al. [10] παρουσιάζουν μια έρευνα στην οποία συγκεντρώνουν όλες τις προσπάθειες βελτίωσης της απόδοσης της παράλληλης επεξεργασίας των επερωτήσεων χρησιμοποιώντας το MapReduce. Πιο συγκεκριμένα, καταγράφουν τις πιο σημαντικές αδυναμίες και περιορισμούς του MapReduce, προτείνοντας παράλληλα τεχνικές επίλυσής τους. Τέλος, παρουσιάζουν μια

ταξινόμηση των ήδη υπαρχουσών βελτιώσεων με βάση το πρόβλημα στου οποίου την επίλυση στοχεύουν.

Οι J.Schad et al [4] έχουν κάνει runtime μετρήσεις σε υπολογιστικό νέφος και συγκεκριμένα στο Amazon Elastic Computing Cloud (EC2), που αφορούν στο instance startup, CPU απόδοση, ταχύτητα μνήμης, disk I/O, network bandwidth και S3 access. Ο λόγος αυτής της έρευνας είναι η απρόβλεπτη απόδοση στο Cloud computing η οποία επηρεάζει τους ερευνητές βάσεων δεδομένων, οι οποίοι εκτελούν wall clock πειράματα, και τις εφαρμογές βάσεων δεδομένων που παρέχουν service level agreements. Για την έρευνα χρησιμοποίησαν μια multimode MapReduce εφαρμογή για να ποσοτικοποιήσουν τον αντίκτυπο σε πραγματικές εφαρμογές ευαίσθητων δεδομένων. Συγκέντρωσαν δεδομένα για ένα ολόκληρο μήνα και μετά τα συνέκριναν με τα αποτελέσματα που απέκτησαν σε ένα τοπικό cluster. Η ανάλυσή τους έδειξε ότι μικρά και μεγάλα instances πλήττονται από μεγάλη διακύμανση στην επίδοση. Επίσης παρατηρούν ότι ένας από τους λόγους αυτής της διακύμανσης είναι οι διαφορετικοί τύποι συστημάτων που χρησιμοποιούνται σε virtual nodes, π.χ: τα Xeon-based system έχουν καλύτερη επίδοση από τα Opteon-based systems. Επίσης, συνέκριναν τη διακύμανση σε υπολογιστικό νέφος με τη διακύμανση σε φυσική συστάδα και τα αποτελέσματα έδειξαν ότι ένα MapReduce job είχε πολύ περισσότερη διακύμανση στο EC2. Ένα σημαντικό συμπέρασμα αυτής της εργασίας είναι ότι η διακύμανση στο EC2 είναι προς το παρόν τόσο μεγάλη που τα wall clock πειράματα μπορούν να εκτελεστούν μόνο με πολύ προσοχή.

Οι J.Dean et al. [11] και οι M.Stonebraker et al. [12] παρουσιάζουν το MapReduce, περιγράφουν την αρχιτεκτονική του και κυρίως το συγκρίνουν με παράλληλες βάσεις δεδομένων παραθέτοντας τα πλεονεκτήματα και τα μειονεκτήματά του. Στις έρευνές τους αναφέρουν ότι έχουν παρατηρηθεί διάφορες παρανοήσεις για το MapReduce σε σχετική βιβλιογραφία. Κάποιες από αυτές αναφέρουν ότι το MapReduce δεν μπορεί να χρησιμοποιήσει δείκτες και εφαρμόζει πλήρες σκανάρισμα σε όλα τα δεδομένα εισόδου, ότι η είσοδος και η έξοδος είναι πάντα απλά αρχεία σ' ένα σύστημα αρχείων και ότι απαιτεί τη χρήση ανεπαρκών textual data formats. Επίσης ασχολούνται και με άλλα σημαντικά ζητήματα όπως το ότι το MapReduce είναι ανεξάρτητο από συστήματα αποθήκευσης και μπορεί να επεξεργαστεί τα δεδομένα χωρίς προηγουμένως να απαιτεί να φορτώνονται σε βάση δεδομένων. Ακόμα αναφέρουν ότι περίπλοκοι μετασχηματισμοί είναι συχνά

ευκολότερο να εκφραστούν σε MapReduce απ' ότι σε SQL. Τέλος, καταλήγουν στα εξής συμπεράσματα για το MapReduce. Πρώτον, είναι ένα μοντέλο εύκολο να χρησιμοποιηθεί. Δεύτερον, μια ευρεία γκάμα προβλημάτων εκφράζονται εύκολα με υπολογισμούς MapReduce. Τρίτον, η υλοποίηση του MapReduce είναι εύρωστη σε μεγάλες συστάδες που αποτελούνται από χιλιάδες μηχανήματα.

Στη συνέχεια, οι J.Dean et al. [8] παρατηρούν ότι τα ζητήματα του πώς να παραλληλιστούν οι υπολογισμοί, να κατανεμηθούν τα δεδομένα και να διαχειρίζονται οι αποτυχίες συντελούν στο να υποβαθμιστεί ο αρχικός απλός υπολογισμός χρησιμοποιώντας μεγάλο πολύπλοκο κώδικα για να τα αντιμετωπίσουν. Εξαιτίας αυτής της πολυπλοκότητας σχεδίασαν ένα abstraction που τους επιτρέπει να εκφράζουν απλούς υπολογισμούς που προσπαθούν να εφαρμόσουν κρύβοντας όμως τις λεπτομέρειες του παραλληλισμού, της ανοχής σφαλμάτων, της κατανομής των δεδομένων και την ισορροπία φορτίου σε μια βιβλιοθήκη. Το abstraction είναι εμπνευσμένο από την map και την reduce μέθοδο σε Lisp αλλά και από άλλες λειτουργικές γλώσσες. Η χρήση ενός λειτουργικού μοντέλου με map και reduce λειτουργίες καθορισμένες από το χρήστη, τους επιτρέπει να παραλληλίζουν εύκολα μεγάλους υπολογισμούς και να χρησιμοποιούν την επανεκτέλεση ως τον βασικό μηχανισμό για την ανοχή σφαλμάτων. Οι σημαντικότερες προσφορές αυτής της εργασίας είναι ένα απλό και δυνατό interface που επιτρέπει τον αυτόματο παραλληλισμό και την κατανομή υπολογισμών, συνδυασμένο με μια υλοποίηση του interface που επιτυγχάνει υψηλή απόδοση σε μεγάλα clusters. Από αυτή την εργασία έφτασαν στο συμπέρασμα ότι ο περιορισμός του προγραμματιστικού μοντέλου κάνει ευκολότερο τον παραλληλισμό και την κατανομή υπολογισμών καθώς επίσης κάνει τους υπολογισμούς ανεκτικούς σε σφάλματα. Αρκετές βελτιώσεις στην εργασία τους είχαν στόχο να μειώσουν τα δεδομένα που στέλνονται στο δίκτυο. Η τοπική βελτιστοποίηση τούς επέτρεψε να διαβάζουν δεδομένα από τοπικούς δίσκους και το να γράφουν ένα αντίγραφο από τα ενδιάμεσα δεδομένα σε τοπικούς δίσκους, εξοικονομεί εύρος ζώνης του δικτύου. Ακόμα, οι περιττές εκτελέσεις μπορούν να χρησιμοποιηθούν για να μειώσουν το φαινόμενο της ύπαρξης αργών μηχανημάτων και για να διαχειρίζονται τις αποτυχίες και την απώλεια δεδομένων. Μετά τις μετρήσεις που έκαναν έφτασαν στο συμπέρασμα ότι το MapReduce είναι ένα άκρως αποτελεσματικό και αποδοτικό εργαλείο για εύρωστη και ανεκτική σε σφάλματα ανάλυση δεδομένων. Επίσης, παρέχει ανοχή σε σφάλματα για μεγάλες εργασίες και η

αποτυχία στην μέση μιας πολύωρης εκτέλεσης δεν απαιτεί επανεκκίνηση της εργασίας από την αρχή.

Οι K.Lee et al. [3] κάνουν στην ουσία μια επισκόπηση και παραθέτουν τα πλεονεκτήματα και μειονεκτήματα του MapReduce, ταξινομούν τις βελτιώσεις που έχουν γίνει και καταλήγουν στο συμπέρασμα ότι το MapReduce είναι απλό παρέχοντας επεκτασιμότητα και ανοχή σε σφάλματα για την επεξεργασία μεγάλου όγκου δεδομένων. Ωστόσο, το MapReduce είναι απίθανο να αντικαταστήσει τα ΣΔΒΔ αλλά αναμένεται να λειτουργήσουν συμπληρωματικά με εύρωστη και ευέλικτη, παράλληλη επεξεργασία για δεδομένα διαφόρων τύπων.

Οι A.Abouzeid et. al [13] εξετάζουν τη δυνατότητα δημιουργίας ενός υβριδικού συστήματος, το οποίο παίρνει τα καλύτερα χαρακτηριστικά των παράλληλων βάσεων δεδομένων και του MapReduce. Αφορμή γι' αυτή την έρευνα ήταν οι δύο απόψεις που υπήρχαν σχετικά με τι είδους τεχνολογία πρέπει να χρησιμοποιηθεί για την ανάλυση μεγάλου όγκου δεδομένων. Έτσι, δημιούργησαν το HadoopDB, το οποίο προσεγγίζει την επίδοση και την αποδοτικότητα των παράλληλων ΣΔΒΔ, επιτυγχάνοντας παράλληλα την ανοχή σφαλμάτων, την ευρωστία και την ευελιξία του MapReduce. Η βασική ιδέα του HadoopDB είναι να συνδέσει πολλά συστήματα βάσεως δεδομένων ενός κόμβου χρησιμοποιώντας το Hadoop ως task coordinator και ως στρώμα δικτύου επικοινωνίας. Οι επερωτήσεις επεξεργάζονται παράλληλα στους κόμβους (nodes) χρησιμοποιώντας το framework του MapReduce. Το HadoopDB επιτυγχάνει ανοχή σφαλμάτων και έχει την ικανότητα να λειτουργεί σε ετερογενή περιβάλλοντα κληρονομώντας την scheduling και job tracking υλοποίηση του Hadoop. Επίσης αποκτά την επίδοση των παράλληλων βάσεων δεδομένων εκτελώντας την επεξεργασία των queries μέσα στην μηχανή της βάσης δεδομένων. Το HadoopDB επεκτείνει το Hadoop προσθέτοντας κάποια στοιχεία όπως: Database Connector, Catalog, Data Loader και SQL to MapReduce to SQL (SMS) Planner. Στο πείραμα συνέκριναν το Hadoop, το HadoopDB (με PostgreSQL) και δύο παράλληλα ΣΔΒΔ εκτελώντας διάφορα tasks όπως: Join Task, UDF Aggregation Task, Aggregation Task, Selection Task, Grep Task κλπ. Οι μετρήσεις των πειραμάτων έδειξαν ότι το HadoopDB έχει όντως την ικανότητα να πλησιάσει την επίδοση των παράλληλων ΣΔΒΔ, να λειτουργήσει σε ετερογενή περιβάλλοντα επιτυγχάνοντας παράλληλα ανοχή σφαλμάτων.

Οι M.Eltabakh et al. [14] παρουσιάζουν το CoHadoop, το οποίο προσπαθεί να λύσει ένα πρόβλημα του Hadoop το οποίο έχει να κάνει με την αδυναμία του να τοποθετήσει στο ίδιο set κόμβων δεδομένα σχετικά μεταξύ τους. Το CoHadoop είναι ένα ελαφρύ extension του Hadoop που επιτρέπει στις εφαρμογές να ελέγχουν πού αποθηκεύονται τα δεδομένα. Στόχος είναι ν' αποθηκεύονται κοντά, αρχεία στο HDFS που είναι σχετικά μεταξύ τους. Το CoHadoop διατηρεί την ευελιξία του Hadoop στο ότι οι χρήστες δεν είναι υποχρεωμένοι να μετατρέψουν τα δεδομένα τους σε συγκεκριμένη μορφή. Αντίθετα, οι εφαρμογές δίνουν hints στο CoHadoop έτσι ώστε κάποια σετ αρχείων που σχετίζονται μεταξύ τους να μπορούν να επεξεργάζονται μαζί και μετά το CoHadoop προσπαθεί να τα τοποθετήσει μαζί για να βελτιώσει την αποδοτικότητα. Η τοποθέτηση αρχείων κοντά μπορεί να βελτιώσει την αποδοτικότητα λειτουργιών όπως indexing, grouping, aggregation, columnar storage, joins και sessionization. Οι M.Eltabakh et al. έκαναν μια λεπτομερή μελέτη με joins και sessionization στο πλαίσιο του log processing και πρότειναν αποδοτικούς map-only αλγόριθμους, οι οποίοι εκμεταλλεύονται collocated κομμάτια δεδομένων. Πιο συγκεκριμένα, η προσέγγισή τους όχι μόνο αποδίδει καλύτερα από τους repartition-based αλγόριθμους αλλά υπερτερεί των map-only αλγορίθμων που εκμεταλλεύονται το διαχωρισμό δεδομένων (data partitioning) αλλά όχι το collocation. Επέκτειναν το HDFS έτσι ώστε τα αρχεία που είναι σχετικά μεταξύ τους να μπορούν να τοποθετηθούν μαζί στο σύστημα αρχείων. Οι επεκτάσεις τους απαιτούν ελάχιστες αλλαγές στο HDFS: εισάγουν μια νέα ιδιότητα στα αρχεία για ν' αναγνωρίζουν τα αρχεία που είναι σχετικά μεταξύ τους και τροποποιούν την πολιτική της τοποθέτησης δεδομένων στο HDFS για να τοποθετηθούν μαζί όλα τ' αντίγραφα από τα συσχετιζόμενα αρχεία. Αυτές οι αλλαγές διατηρούν τα πλεονεκτήματα του Hadoop, συμπεριλαμβανομένου του load balancing και της ανοχής σφαλμάτων.

Οι A.Thusso et al. [15] παρουσιάζουν το Hive, μια λύση ανοιχτού κώδικα για την αποθήκευση δεδομένων πάνω από το Hadoop. Μπορεί το Hadoop να είναι μια δημοφιλής εναλλακτική για την επεξεργασία και την αποθήκευση εξαιρετικά μεγάλου όγκου δεδομένων, ωστόσο το προγραμματιστικό του μοντέλο είναι low-level και απαιτεί από τους προγραμματιστές να γράφουν custom προγράμματα, τα οποία είναι δύσκολο να διατηρηθούν και να επαναχρησιμοποιηθούν. Το Hive υποστηρίζει επερωτήσεις, εκφρασμένα σε μια γλώσσα παρόμοια με την SQL, την HiveQL. Οι επερωτήσεις μεταγλωττίζονται μέσα στα map-reduce jods που εκτελούνται στο

Hadoop. Η γλώσσα αυτή περιλαμβάνει ένα σύστημα τύπων με υποστήριξη πινάκων που περιέχουν primitive types, collections όπως arrays και maps και εμφωλευμένες συνθέσεις των ίδιων τύπων. Οι IO βιβλιοθήκες μπορούν να επεκταθούν ώστε να αναζητούν τα δεδομένα σε διάφορους τύπους. Το Hive επίσης περιλαμβάνει ένα σύστημα καταλόγου, το Hive-Metastore που περιέχει schemas και statistics, τα οποία είναι χρήσιμα στην εκμετάλλευση των δεδομένων και την βελτιστοποίηση των queries. Το Hive είναι το πρώτο βήμα δημιουργίας open-source warehouse πάνω στο Hadoop. Έτσι, τα χαρακτηριστικά της αποθήκευσης και της εκτέλεσης, τους αναγκάζουν να ξαναεπισκεφτούν τις τεχνικές για την επεξεργασία επερωτήσεων. Ανακάλυψαν ότι θα πρέπει να ξαναγράψουν ή να τροποποιήσουν κάποιους αλγόριθμους επεξεργασίας επερωτήσεων έτσι ώστε να γίνουν πιο αποδοτικοί. Επίσης αναφέρουν ότι το Hive είναι ένα sub-project της Apache, του οποίου το instance στο Facebook περιέχει πάνω από 700 terabytes χρησιμοποιήσιμων δεδομένων και υποστηρίζει πάνω από 5000 επερωτήσεις σε καθημερινή βάση.

Οι J.Dittrich et al. [16] υποστηρίζουν ότι το MapReduce επεξεργάζεται τα tasks σε scan-oriented fashion και αυτό έχει σαν αποτέλεσμα η επίδοσή του να μην φτάνει αυτή ενός καλά ρυθμισμένου παράλληλου ΣΔΒΔ. Έτσι προτείνουν ένα νέο τύπο συστήματος που λέγεται Hadoop++ το οποίο δίνει σημαντική ώθηση στην επίδοση ενός task χωρίς ν' αλλάζει καθόλου το framework του Hadoop. Για να επιτευχθεί αυτό, εισάγουν την τεχνολογία τους στα κατάλληλα σημεία με UDFs επηρεάζοντας έτσι το Hadoop από μέσα. Αυτό έχει σαν αποτέλεσμα, πρώτον το Hadoop++ να υπερτερεί σημαντικά του Hadoop. Δεύτερον, οποιεσδήποτε μελλοντικές αλλαγές του Hadoop μπορούν να χρησιμοποιηθούν κατευθείαν στο Hadoop++ χωρίς να ξαναγραφτεί κώδικας. Τρίτον, το Hadoop++ δεν απαιτεί την αλλαγή του Hadoop interface. Επίσης, είδαν ότι όσον αφορά στην επεξεργασία επερωτήσεων το Hadoop++ φτάνει και καμιά φορά βελτιώνει τα runtimes των επερωτήσεων του HadoopDB και αλλάζει το εσωτερικό layout ενός split — ένα μεγάλο partition δεδομένων — και/ή τροφοδοτεί το Hadoop με τα κατάλληλα UDFs χωρίς ν' αλλάξει καθόλου το framework του Hadoop. Για τις μετρήσεις τους υλοποίησαν εργασίες όπως data loading, selection task και join task. Επίσης υλοποίησαν fault tolerance πειράματα (node failures και straggler nodes πειράματα). Τα πειραματικά αποτελέσματα έδειξαν ότι σε γενικές γραμμές το Hadoop++ υπερτερεί του Hadoop. Ακόμα σε tasks που σχετίζονται με indexing και join

processing το Hadoop++ υπερτερεί και του HadoopDB – χωρίς να απαιτείται ένα ΣΔΒΔ ή μεγάλες αλλαγές στο framework της εκτέλεσης ή το interface του Hadoop. Επίσης, παρατήρησαν ότι όσο αυξάνεται το μέγεθος του split, τόσο το Hadoop++ βελτιώνεται στα selection και join tasks ενώ η επίδοση στην ανοχή σφαλμάτων μειώνεται. Αυτό δείχνει ότι υπάρχει μια σύνδεση μεταξύ των runtime και fault tolerance jobs του MapReduce.

Οι E.Jahani et al. [17] υποστηρίζουν ότι υπάρχουν αποδείξεις ότι οι υπάρχουσες υλοποιήσεις του MapReduce είναι ανεπαρκείς και απαιτούν περισσότερο hardware από μια παραδοσιακή σχεσιακή βάση δεδομένων για να ολοκληρώσουν παρόμοιες εργασίες. Έτσι, παρουσιάζουν το MANIMAL, ένα σύστημα για τη βελτιστοποίηση του Hadoop. Το MANIMAL ανιχνεύει αυτόματα ευκαιρίες για βελτιστοποίηση. Ακόμα επιτρέπει την εκτέλεση MapReduce προγραμμάτων με ουσιώδη speedups σε σχέση με τα συμβατικά συστήματα, χρησιμοποιώντας το ίδιο hardware και χωρίς να χρειάζεται ο προγραμματιστής να κάνει τροποποιήσεις στο πρόγραμμα. Παρατήρησαν επιταχύνσεις από 296% ως 1,121% σε πραγματικό κώδικα. Ο μηχανισμός του MANIMAL εντοπίζει κώδικα που μπορεί να βελτιστοποιηθεί σε ήδη compiled προγράμματα. Δεν μπορεί να εγγυηθεί ότι θα βρει όλες τις πιθανές βελτιστοποιήσεις αλλά υποδεικνύει μια βελτιστοποίηση όταν είναι εντελώς ασφαλής για να γίνει. Το MANIMAL είναι σχεδιασμένο να θυσιάζει πιθανές βελτιστοποιήσεις αν υπάρχει πιθανότητα να μην πετύχει και αποτελείται από τρία στοιχεία που του επιτρέπουν την αυτόματη βελτιστοποίηση και αυτά είναι ο analyzer, ο optimizer και το execution fabric. Ο χρήστης δεν βλέπει σχεδόν τίποτα από το MANIMAL. Το υποβαλλόμενο πρόγραμμα δεν χρειάζεται αν τροποποιηθεί από τον προγραμματιστή και το τελικό πρόγραμμα θα πρέπει να είναι το ίδιο με αυτό που θα παραγόταν από ένα συμβατικό MapReduce σύστημα. Το MANIMAL ψάχνει για τρία διαφορετικά είδη βελτιστοποίησης: selection, projection και data compression. Τα πειράματα που έκανα έδειξαν ότι και στα τρία επιτεύχθηκαν σημαντικές επιταχύνσεις.

2.3 Skyline Επερωτήσεις

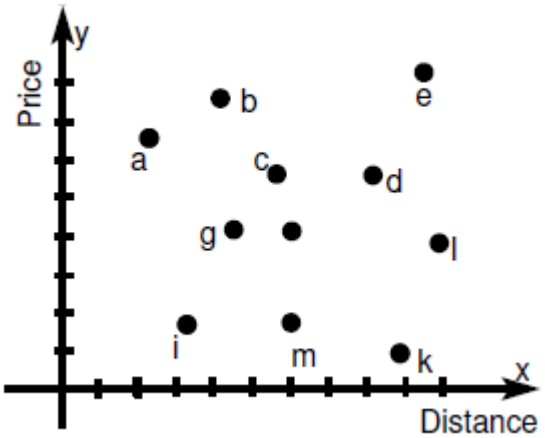
Τα τελευταία χρόνια, η διαχείριση και η αποθήκευση των δεδομένων έχει γίνει εξαιρετικά κατανεμημένη. Αυτό έχει σαν αποτέλεσμα, να είναι απαραίτητη η ύπαρξη περισσότερο προηγμένων διαχειριστών ερωτημάτων σε πολυδιάστατα δεδομένα, όπως οι skyline επερωτήσεις, οι οποίες θα διευκολύνουν τους χρήστες στη διαχείριση αυτού του τεράστιου όγκου δεδομένων [18].

Ο εκάστοτε χρήστης πρέπει, μέσα από ένα σύνολο πολυδιάστατων αντικειμένων, να επιλέξει εκείνα τα αντικείμενα τα οποία είναι καλύτερα από κάποια άλλα, όμως, δεν ξέρει με βάση ποιό κριτήριο ή ποια διάσταση να τα κατατάξει και να τα αξιολογήσει ώστε να καταλήξει στα καλύτερα. Το Skyline του διασφαλίζει ότι θα βρει τα αντικείμενα τα οποία ικανοποιούν καλύτερα τις συνθήκες που έχει ορίσει ο χρήστης και δεν υπάρχουν άλλα αντικείμενα καλύτερα από αυτά. Για τον υπολογισμό των Skyline σημείων πραγματοποιείται μια σειρά από συγκρίσεις μεταξύ των σημείων με τη βοήθεια των Skyline Επερωτήσεων [19].

Πιο συγκεκριμένα, για ένα δεδομένο σετ από σημεία p_1, p_2,.. p_N, η skyline επερώτηση επιστρέφει ένα σύνολο σημείων P, τέτοια ώστε να μην υπάρχει κανένα άλλο σημείο στο σύνολο των αρχικών δεδομένων που να επικρατεί έναντι των σημείων αυτών, δηλαδή των σημείων που ανήκουν στο σύνολο P.

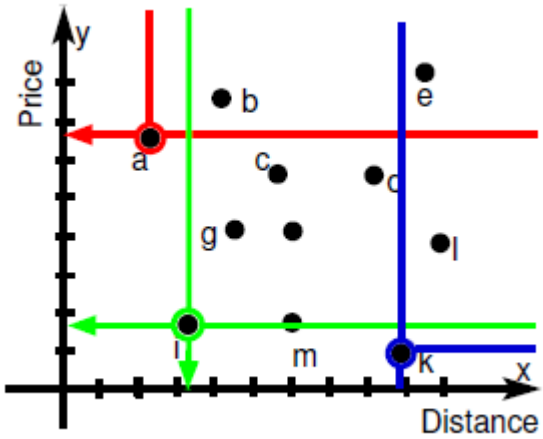
Επικράτηση Σημείου σημαίνει το εξής: ένα σημείο p_i επικρατεί ενός άλλου σημείου p_j αν και μόνο αν η τιμή της συντεταγμένης του p_i σε οποιοδήποτε άξονα δεν είναι μεγαλύτερη από την αντίστοιχη τιμή της συντεταγμένης του σημείου p_j ή η τιμή μιας τουλάχιστον συντεταγμένης του p_i να είναι μικρότερη από την τιμή της αντίστοιχης συντεταγμένης του σημείου p_j.

Για παράδειγμα, έστω ότι υπάρχει ένα σύνολο σημείων, όπως φαίνεται στο παρακάτω σχήμα, το οποίο περιέχει πληροφορίες για ξενοδοχεία και κάθε τελεία αντιστοιχεί σ' ένα ξενοδοχείο. Δηλαδή, απεικονίζεται η απόσταση από την παραλία και η τιμή για κάθε σημείο.



Εικόνα 3: Ξενοδοχεία που αναπαρίστανται σε σημεία στο χώρο μεταξύ των αξόνων x και y [20] [6].

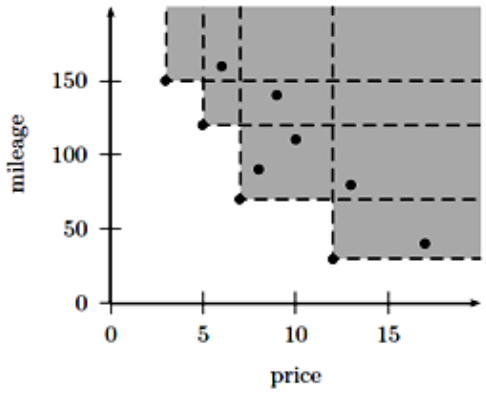
Στόχος της αναζήτησης είναι να βρεθούν ξενοδοχεία των οποίων η απόσταση από τη θάλασσα και η τιμή να είναι η καλύτερη δυνατή. Άρα, δεν πρέπει να υπάρχουν άλλα ξενοδοχεία που να διαθέτουν καλύτερη τιμή και μικρότερη απόσταση από τη θάλασσα, από αυτά που θα επιστρέψει η skyline επερώτηση.



Εικόνα 4: Skyline σημεία [20] [6].

Όπως φαίνεται στο παραπάνω σχήμα, τα σημεία που μας ενδιαφέρουν είναι τα {a, i, k}. Το a έχει τη λιγότερη απόσταση από την παραλία, το k έχει τη χαμηλότερη τιμή και το i δεν έχει ούτε την μικρότερη τιμή, ούτε την μικρότερη απόσταση. Όλα τα υπόλοιπα σημεία του σετ γίνονται dominated από τα σημεία {a, i, k}, δηλαδή οι τιμές της απόστασης και την τιμής είναι μεγαλύτερες από ένα ή περισσότερα skyline σημεία [20].

Ένα αντίστοιχο παράδειγμα θα μπορούσε να είναι η αναζήτηση μεταχειρισμένων αυτοκινήτων που έχουν τα λιγότερα χιλιόμετρα και την μικρότερη τιμή. Με τη βοήθεια μιας Skyline Επερώτησης καταλήγουμε στα skyline σημεία όπως φαίνεται και στο σχήμα παρακάτω [19]. Τα σημεία που βρίσκονται στην γκρίζα περιοχή είναι σημεία που γίνονται dominated από τα skyline σημεία.



Εικόνα 5: Skyline αποτελέσματα στο παράδειγμα των μεταχειρισμένων αυτοκινήτων [19].

Στη βιβλιογραφία υπάρχουν αρκετές αναφορές και μελέτες σχετικά με τα Skyline Queries και την απόδοσή τους. Οι D.Papadias et al. [20] παρουσιάζουν διάφορες τεχνικές για την αξιολόγηση των skyline επερωτήσεων όπως Block Nested Loop, Divide and Conquer, Plane-sweep, Nearest Neighbor Search and Branch and Bound Skyline.

Οι D.Papadias et al. [21] ανέπτυξαν τον BBS, έναν αλγόριθμο που βασίζεται στην αναζήτηση του κοντινότερου ‘γείτονα’ και είναι I/O optimal, δηλαδή αποκτά μια πρόσβαση μόνο σε εκείνους τους κόμβους που μπορεί να περιέχουν skyline σημεία. Ο BBS είναι απλός στην υλοποίηση και υποστηρίζει όλους τους τύπους της βαθμιαίας επεξεργασίας.

Οι J.Chomicki et al. [22] προτείνουν τον SFS skyline αλγόριθμο που βασίζεται στο presorting και μπορεί να χρησιμοποιηθεί αποτελεσματικά σε οποιαδήποτε skyline επερώτηση.

Οι K.Hose et al. [18] κάνουν μια έρευνα σχετικά με την επεξεργασία των skyline επερωτήσεων σε κατανεμημένο περιβάλλον. Αρχικά καταγράφουν τους στόχους και τις βασικές αρχές στις οποίες κάθε τεχνική που ασχολείται με το συγκεκριμένο τομέα, πρέπει να υπακούει. Στη συνέχεια, παρουσιάζουν λεπτομερώς τις ήδη υπάρχουσες τεχνικές και προσεγγίσεις για την επεξεργασία των skyline επερωτήσεων σε κατανεμημένα περιβάλλοντα και τις συγκρίνουν μεταξύ τους για να προτείνουν στο τέλος ερευνητικά θέματα που αφορούν σε αυτό το πεδίο και δεν έχουν ακόμη ερευνηθεί.

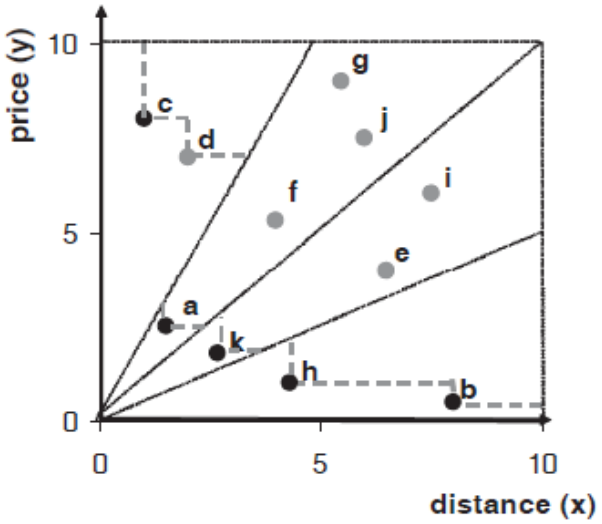
2.4 Διαμοιρασμός χώρου για παράλληλη επεξεργασία των Skyline Επερωτήσεων

Ο διαμερισμός του χώρου είναι η διαδικασία κατά την οποία ένας χώρος (συνήθως ευκλείδειος χώρος) χωρίζεται σε μικρότερα υποσύνολα έτσι ώστε κάθε σημείο να υπάρχει μόνο σ' ένα υποσύνολο [23].

Στην παράλληλη επεξεργασία των skyline επερωτήσεων, ο διαμοιρασμός του χώρου παίζει έναν ιδιαίτερα σημαντικό ρόλο καθώς σκοπός του είναι να χωρίζει το χώρο ή τα σημεία σε κατάλληλα κομμάτια, αλλιώς partitions ή διαμερίσεις , τα οποία είτε θα 'χωράνε' στην κύρια μνήμη είτε θα επεξεργάζονται σε διαφορετικά μηχανήματα μοιράζοντας έτσι το φόρτο εργασίας [6].

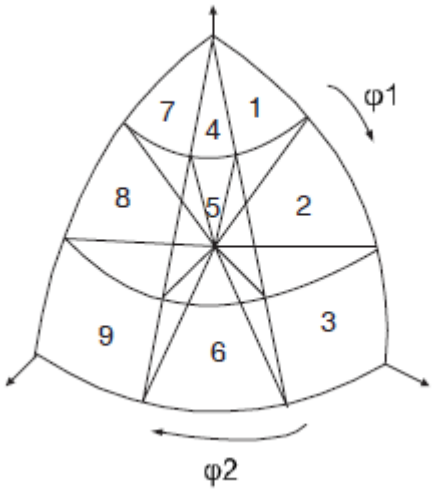
Παρακάτω παρουσιάζονται οι τρεις μέθοδοι διαμοιρασμού του χώρου για παράλληλη επεξεργασία των skyline επερωτήσεων, οι οποίες υλοποιούνται στην παρούσα εργασία.

2.4.1 Διαμοιρασμός χώρου με βάση τις γωνιακές συντεταγμένες για αποδοτική παράλληλη επεξεργασία Skyline επερωτήσεων



Εικόνα 6: Διαμοιρασμός χώρου με βάση τις γωνιακές συντεταγμένες [6].

Οι A.Vlachou et al. [6] πρώτοι προτείνουν διαμοιρασμό του χώρου βασισμένο σε γωνιακές συντεταγμένες για παράλληλο skyline υπολογισμό με σκοπό να επιλύσουν το πρόβλημα της αποδοτικής παράλληλης επεξεργασίας των skyline επερωτήσεων σ' ένα σύνολο από N servers, μειώνοντας το χρόνο εκτέλεσης και μοιράζοντας το υπολογιστικό φορτίο ισόποσα σε όλους τους κόμβους. Σε αυτή την προσέγγιση, οι επερωτήσεις φτάνουν σ' ένα κεντρικό κόμβο, ο οποίος ονομάζεται και coordinator server, ο οποίος κατανέμει την επεξεργασία σε όλους τους συμμετέχοντες κόμβους και συλλέγει από αυτούς τα αποτελέσματα. Η επερώτηση εκτελείται ταυτόχρονα σε όλους τους κόμβους.



Εικόνα 7:Διαμοιρασμός χώρου τριών διαστάσεων με βάση γωνιακές συντεταγμένες [6].

Αυτή η τεχνική πρώτη αντιστοιχίζει το καρτεσιανό επίπεδο σε επίπεδο υπερσφαιρικών συντεταγμένων και στη συνέχεια χωρίζει το χώρο των δεδομένων σε N κομμάτια, βασισμένη στις γωνιακές συντεταγμένες. Σκοπός αυτής της τεχνικής είναι να δημιουργηθούν κομμάτια ίσου όγκου και να κατανεμηθούν σε αυτά ισόποσα τα δεδομένα έτσι ώστε κάθε κομμάτι να περιέχει σχεδόν τον ίδιο αριθμό σημείων, υποθέτοντας ότι έχουμε ομοιόμορφη κατανομή. Αναλυτικότερα, οι καρτεσιανές συντεταγμένες ενός σημείου x = [x_1, x_2, …, x_d], όπου d είναι ο αριθμός των διαστάσεων του σημείου, αντιστοιχίζονται σε υπερσφαιρικές συντεταγμένες, που αποτελούνται από μια ακτινωτή συνιστώσα r και d-1 γωνιακές Συντεταγμένες φ_1, φ_2, φ_{d-1}, χρησιμοποιώντας γνωστές εξισώσεις:

$$r = \sqrt{x_n^2 + x_{n-1}^2 + \ldots + x_1^2}$$

$$\tan(\phi_1) = \frac{\sqrt{x_n^2 + x_{n-1}^2 + \ldots + x_2^2}}{x_1}$$

$$\ldots$$

$$\tan(\phi_{d-2}) = \frac{\sqrt{x_n^2 + x_{n-1}^2}}{x_{n-2}}$$

$$\tan(\phi_{d-1}) = \frac{x_n}{x_{n-1}}$$

Γενικά, αυτή είναι μια νέα προσέγγιση στο διαμοιρασμό ενός συνόλου δεδομένων σε δεδομένο αριθμό servers έτσι ώστε να υποστηριχθεί αποδοτική

παράλληλη επεξεργασία των skyline επερωτήσεων. Από πειραματική έρευνα, έχει αποδειχθεί ότι ο διαμοιρασμός χώρου βασισμένος σε γωνιακές συντεταγμένες εξομαλύνει πολλά από τα προβλήματα των παραδοσιακών grid τεχνικών, καταφέρνοντας έτσι να μειώσει το χρόνο απόκρισης 10 φορές και να μοιράσει το υπολογιστικό φόρτο εργασίας περισσότερο δίκαια.

Δυναμικός διαμοιρασμός χώρου με βάση τις γωνιακές συντεταγμένες

Εκτός από το στατικό διαμοιρασμό του χώρου που υλοποιείται στην παρούσα εργασία, οι A. Vlachou et al. [6], προτείνουν και ένα δυναμικό διαμοιρασμό του χώρου σε περίπτωση που τα δεδομένα δεν ακολουθούν ομοιόμορφη κατανομή. Στην περισσότερη κοινή περίπτωση, κατά την οποία τα δεδομένα ακολουθούν μια μη – ομοιόμορφη κατανομή, οι στατικές μέθοδοι διαμοιρασμού του χώρου αδυνατούν να επιτύχουν ισορροπημένη κατανομή των δεδομένων στις διαμερίσεις. Αυτό συμβαίνει γιατί τα δεδομένα είναι συνήθως συγκεντρωμένα σε συγκεκριμένα σημεία του χώρου.

Ο δυναμικός διαμοιρασμός του χώρου εξομαλύνει αυτό το πρόβλημα μοιράζοντας το χώρο ανάλογα με την κατανομή των σημείων. Στόχος του δυναμικού διαμοιρασμού του χώρου είναι να κατανεμηθούν ισομερώς τα δεδομένα στις διαμερίσεις ανεξάρτητα από την κατανομή τους. Σε γενικές γραμμές, ο δυναμικός διαμοιρασμός του χώρου επιτυγχάνεται θέτοντας έναν μέγιστο αριθμό σημείων για κάθε διαμερισμό. Μόλις τα σημεία μιας διαμέρισης φτάσουν στο μέγιστο δυνατό πλήθος, τότε η διαμέριση χωρίζεται εκ νέου.

2.4.2 Διαμοιρασμός χώρου με βάση Grid τεχνική

Πρόκειται για μια παραδοσιακή τεχνική που χρησιμοποιείται για το διαμοιρασμό του χώρου. Πιο συγκεκριμένα, η τεχνική του grid διαμερίζει το χώρο σε κελιά, χωρίζοντας κάποιες διαστάσεις του χώρου σε δύο μέρη [6].

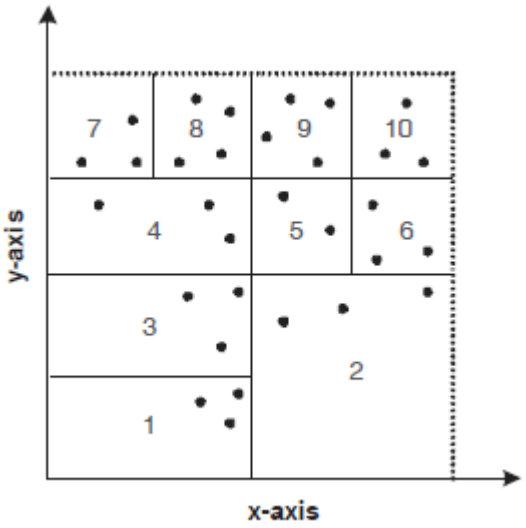
Εικόνα 8: Διαμοιρασμός χώρου με βάση Grid τεχνική [6].

Το βασικό πλεονέκτημα αυτής της τεχνικής είναι ότι κάποιες διαμερίσεις μπορεί να μην εξεταστούν όταν η επερώτηση δεν επεξεργάζεται με παράλληλο τρόπο. Μέσω της μερικώς σειριακής επικοινωνίας μεταξύ των servers, διασφαλίζεται ότι επιστρέφονται τα σωστά αποτελέσματα, αποφεύγοντας την επικοινωνία με όλους τους servers, ειδικά αυτούς που δε συνεισφέρουν στο σύνολο των αποτελεσμάτων [6].

Ωστόσο, στην περίπτωση που η επεξεργασία είναι παράλληλη, όπως στην προκείμενη περίπτωση, και οι διαμερίσεις επεξεργάζονται ταυτόχρονα, η απόδοση αυτής της τεχνικής μειώνεται αισθητά. Αυτό συμβαίνει κυρίως διότι πολλές διαμερίσεις δεν περιέχουν skyline σημεία με αποτέλεσμα να έχουμε άσκοπη επεξεργασία. Αυτό φαίνεται και στο παραπάνω σχήμα, όπου ο πάνω δεξιά διαμερισμός επεξεργάζεται χωρίς λόγο αφού δεν περιέχει κανένα σημείο το οποίο να ανήκει στα τελικά skyline σημεία [6].

Δυναμικός διαμοιρασμός χώρου με βάση Grid τεχνική

Οι A.Vlachou et al. [6] προτείνουν και γι' αυτή την τεχνική έναν τρόπο δυναμικού διαμοιρασμού του χώρου στην περίπτωση κατά την οποία τα δεδομένα δεν ακολουθούν ομοιόμορφη κατανομή.

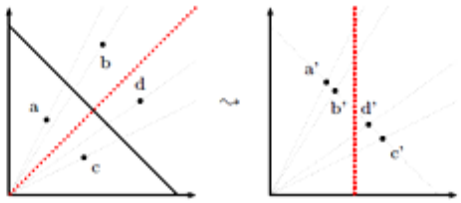


Εικόνα 9: Δυναμικός διαμοιρασμός χώρου δύο διαστάσεων [6].

Ο στατικός grid διαμοιρασμός του χώρου μπορεί να επεκταθεί και να χωρίζει δυναμικά το χώρο έτσι ώστε να μπορούν να διαχειριστεί αυθαίρετες κατανομές δεδομένων. Για να το επιτύχουν αυτό, ορίζουν ένα όριο n_{max} στο πλήθος των σημείων το οποίο μπορεί να διαχειριστεί μια διαμέριση. Δεδομένης της πληθικότητας n ενός συνόλου δεδομένων και του αριθμού των διαμερίσεων N, ορίζουν $n_{max} = 2* n/N$.

Αρχικά, υπάρχει μόνο μια διαμέριση και τα σημεία υπάγονται σε αυτήν. Μόλις, το πλήθος των σημείων της διαμέρισης γίνει ίσο με το μέγιστο, δηλαδή το n_{max} , η διαμέριση χωρίζεται στα δυο με βάση κάποια διάστασή της και τα όριά της καθορίζονται με τέτοιο τρόπο έτσι ώστε ο αριθμός των σημείων να χωρίζεται ισόποσα στις δυο νέες διαμερίσεις. Ο αλγόριθμος συνεχίζει μέχρι όλα τα σημεία να αντιστοιχηθούν σε κάποια διαμέριση. Αν ο αλγόριθμος δημιουργήσει λιγότερες από N διαμερίσεις, τότε πραγματοποιείται ένα επιπλέον στάδιο όπου η διαμέριση με το μεγαλύτερο πλήθος σημείων χωρίζεται μέχρι να δημιουργηθούν ακριβώς N διαμερίσεις.

2.4.3 Διαμερισμός Χώρου μέσω Hyperplane Προβολής



Εικόνα 10: Διαμερισμός Χώρου μέσω Hyperplane προβολής [19].

Οι H.Kohler et al. [19] ανέπτυξαν έναν παράλληλο αλγόριθμο, ο οποίος επικεντρώνεται στον αρχικό διαμερισμό των δεδομένων. Η βασική ιδέα είναι να ομαδοποιηθούν τα σημεία με βάση την κατεύθυνσή τους από την αρχή των αξόνων αυξάνοντας έτσι την πιθανότητα να επικρατούν το ένα πάνω στο άλλο. Αυτό επιτυγχάνεται προβάλλοντας τα σημεία σε μια ευθεία (hyperplane) διασφαλίζοντας έτσι ότι τα τοπικά skyline sets είναι μικρά.

Η τεχνική αυτή είναι παρόμοια με αυτή των A.Vlachou et al. [6], όμως εδώ χρησιμοποιούν, όπως αναφέρθηκε νωρίτερα, την προβολή στην ευθεία για την οποία ισχύει $x_1 + \ldots + x_d = 1$. Οι προβολές δε γίνονται ορθογώνια αλλά εισάγοντας τη γραμμή από την αρχή των αξόνων. Στη συνέχεια, τα σημεία κατανέμονται στις διαμερίσεις με βάση τις προβολές τους. Είναι σημαντικό να αναφερθεί ότι σημεία που μπορεί να επικρατούν το ένα στο άλλο, έχουν μεγάλη πιθανότητα να έχουν παρόμοιες προβολές στην ευθεία κι έτσι να βρίσκονται στην ίδια διαμέριση για τον τοπικό υπολογισμό των skyline σημείων. Οι προβολές στην ευθεία είναι απλές και φθηνές στον υπολογισμό, όπως φαίνεται παρακάτω:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \cdot \frac{1}{x_1 + \ldots + x_d}$$

Εικόνα 11: Υπολογισμός προβολών στην ευθεία [19].

Για παράδειγμα, αν θεωρήσουμε ότι έχουμε ένα σύνολο σημείων P τριών διαστάσεων όπως φαίνεται παρακάτω:

$$P = \left\{ \begin{pmatrix} 9 \\ 6 \\ 15 \end{pmatrix}, \begin{pmatrix} 13 \\ 3 \\ 4 \end{pmatrix}, \begin{pmatrix} 15 \\ 5 \\ 5 \end{pmatrix}, \begin{pmatrix} 12 \\ 8 \\ 20 \end{pmatrix} \right\}$$

Η προβολή στην ευθεία για την οποία ισχύει $x + y + z = 1$, έχει σαν αποτέλεσμα την μετατροπή των σημείων στα ακόλουθα σημεία:

$$\mathcal{H}(P) = \left\{ \begin{pmatrix} 0.3 \\ 0.2 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0.65 \\ 0.15 \\ 0.2 \end{pmatrix}, \begin{pmatrix} 0.6 \\ 0.2 \\ 0.2 \end{pmatrix}, \begin{pmatrix} 0.3 \\ 0.2 \\ 0.5 \end{pmatrix} \right\}$$

Αν υποθέσουμε ότι η συνθήκη διαμοιρασμού είναι $x_1 \leq 0.4$, δηλαδή το όριο της πρώτης διάστασης των σημείων πρέπει να είναι μικρότερο ή ίσο του 0.4, τότε δημιουργούνται οι ακόλουθοι διαμερισμοί:

$$P_1 = \left\{ \begin{pmatrix} 9 \\ 6 \\ 15 \end{pmatrix}, \begin{pmatrix} 12 \\ 8 \\ 20 \end{pmatrix} \right\} \quad \text{και} \quad P_2 = \left\{ \begin{pmatrix} 13 \\ 3 \\ 4 \end{pmatrix}, \begin{pmatrix} 15 \\ 5 \\ 5 \end{pmatrix} \right\}$$

2.5 Skyline επεξεργασία στο MapReduce

Σύμφωνα με τους K.Mullesgaard et al. [24] προηγούμενες εργασίες έχουν παρουσιάσει αρκετούς τρόπους να εκτελεστούν οι skyline επερωτήσεις με τη χρήση του MapReduce, χωρίς όμως να γίνεται χρήση των πλεονεκτημάτων της παράλληλης επεξεργασίας αφού ένα σημαντικό μέρος της επερώτησης τρέχει πάντα σειριακά και χρησιμοποιείται ένας μόνο reducer για τον υπολογισμό των τελικών skyline σημείων. Έτσι, προτείνουν ένα νέο αλγόριθμο, τον MapReduce – Grid partitioning Multiple Reducers Skyline (MR - GPMRS), ο οποίος τρέχει όλη την επερώτηση παράλληλα. Η βάση αυτού του αλγόριθμου είναι ο διαμοιρασμός του αρχικού συνόλου δεδομένων χρησιμοποιώντας μια grid τεχνική. Χρησιμοποιείται ένα bitstring, το οποίο καταγράφει ποιοι διαμερισμοί δεν είναι κενοί, πράγμα το οποίο καθιστά ικανή τη λήψη αποφάσεων βασισμένων στην κατανομή ολόκληρου του συνόλου δεδομένων. Ο τρόπος που συνδυάζεται η λειτουργία της επικράτειας σημείων έναντι άλλων στις skyline επερωτήσεις με το διαμοιρασμό χώρου με βάση grid τεχνικές, επιτρέπει το διαμοιρασμό των δεδομένων σε μέρη τα οποία μπορούν να επεξεργαστούν ανεξάρτητα το ένα από το άλλο. Αυτό σημαίνει ότι αυτά τα μέρη δεδομένων μπορούν να επεξεργαστούν από διαφορετικούς reducers. Αυτό σημαίνει ότι ο MR-GPMRS λειτουργεί πολύ καλά για μεγάλα σετ δεδομένων και μεγάλες συστάδες. Με τα πειράματά τους, δείχνουν ότι ο MR-GPMRS τρέχει αρκετές φορές καλύτερα από τους αντίστοιχους αλγόριθμους σε μεγάλα σετ δεδομένων. Ωστόσο, το γεγονός ότι οι reducers πρέπει να λαμβάνουν αντίγραφα των διαμερίσεις αυξάνει σημαντικά το κόστος επικοινωνίας. Η πειραματική μελέτη τους, δείχνει ότι η χρήση πολλαπλών reducers δεν είναι η καλύτερη επιλογή όταν το skyline ποσοστό είναι χαμηλό. Έτσι, για τη βελτιστοποίηση του αλγόριθμου για οποιοδήποτε σύνολο δεδομένων, είναι απαραίτητο να βρεθεί τρόπος, ο MR-GPMRS να επιλέγει έξυπνα τον αριθμό των reducers που θα χρησιμοποιήσει.

Οι L.Chen et al. [25] παρουσιάζουν μια νέα MapReduce Skyline μέθοδο για παράλληλη επεξεργασία των skyline επερωτήσεων. Η νέα αυτή μέθοδος μειώνει σημαντικά το χρόνο της φάσης του Reduce εξαιτίας του περιορισμού των περιττών υπολογισμών για την ύπαρξη dominated σημείων. Τα πειράματα που διεξήγαν στο Hadoop έδειξαν ότι η μέθοδός τους λειτουργεί το ίδιο καλά με την αύξηση των διαστάσεων και την πληθικότητα των δεδομένων. Επίσης, έδειξαν ότι η νέα angular

partitioned MapReduce μέθοδος τρέχει 1.7 και 2.3 φορές γρηγορότερα από τις dimensional και grid partitioning μεθόδους.

Οι B.Zhang et al. [26] ορίζουν το πρόβλημα της επεξεργασίας skyline επερωτήσεων στο MapReduce framework. Βασιζόμενοι σε διάφορες στρατηγικές διαμερισμού των δεδομένων, αναπτύσσουν τρείς αλγόριθμους για τον υπολογισμό skyline στο MapReduce: MapReduce based BNL (MR-BNL), MapReduce based SFS (MR-SFS) και MapReduce based Bitmap (MR-Bitmap). Εκτεταμένα πειράματα έλαβαν χώρα για την αξιολόγηση και τη σύγκριση των αλγορίθμων αυτών υπό διαφορετικές κάθε φορά συνθήκες, όπως κατανομή των δεδομένων, διαστάσεις, μέγεθος buffer και μέγεθος συστάδας. Τα πειράματα έδειξαν ότι οι αλγόριθμοι MR-BNL και MR-SFS λειτουργούν καλά στις περισσότερες περιπτώσεις ενώ αντιμετωπίζουν προβλήματα με τις διαστάσεις σε παράλληλα περιβάλλοντα. Ο αλγόριθμος MR-Bitmap λειτουργεί το ίδιο καλά ανεξάρτητα από τις κατανομές των δεδομένων, ειδικά όταν το bitmap ταιριάζει ακριβώς στην μνήμη ενός μόνο κόμβου.

3. ΣΤΟΧΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Στην παρούσα εργασία θα παρουσιαστεί η φιλοσοφία του Cloud Computing και του Hadoop – MapReduce καθώς επίσης και η χρησιμότητα των Skyline Επερωτήσεων. Στα προηγούμενα κεφάλαια περιγράφηκε αναλυτικά ο τρόπος λειτουργίας των παραπάνω τεχνολογιών. Μεγάλο βάρος δόθηκε στην περιγραφή των τριών τεχνικών που χρησιμοποιούνται για το διαμοιρασμό του χώρου και της μεθόδου που χρησιμοποιείται για τον υπολογισμό των skyline σημείων.

Στη συνέχεια, θα παρουσιαστεί και θα αξιολογηθεί ο συνδυασμός αυτών των δυο τεχνολογιών, υλοποιώντας στο Hadoop τις τρείς διαφορετικές τεχνικές διαμοιρασμού ενός χώρου (διαμοιρασμό χώρου με βάση grid τεχνική, διαμοιρασμός χώρου με βάση γωνιακές συντεταγμένες, διαμοιρασμός χώρου μέσω hyperplane προβολής) και τον αλγόριθμο παράλληλης επεξεργασίας, ο οποίος υπολογίζει τα skyline σημεία από ένα σετ δεδομένων. Ο συνδυασμός αυτών των δυο τεχνολογιών θα οδηγήσει στην επίτευξη του βασικού στόχου, ο οποίος είναι να εκτελεστούν αποδοτικά skyline επερωτήσεις στο Hadoop. Μετά την υλοποίησή τους, οι τεχνικές αυτές θα συγκριθούν μεταξύ τους, μέσω πειραματικής μελέτης, ώστε να προκύψουν συγκεντρωτικά αποτελέσματα για την απόδοσή τους.

Υπάρχουν πάρα πολλές εφαρμογές στις οποίες ο χρήστης θέλει να επιλέξει κάποια ‘αντικείμενα’ ανάμεσα από πολλά άλλα, βάσει πολλαπλών κριτηρίων. Τέτοια παραδείγματα θα μπορούσε να είναι η επιλογή ξενοδοχείου με βάση την απόσταση από τη θάλασσα και την τιμή ή η επιλογή ενός μεταχειρισμένου αυτοκινήτου με βάση τα χιλιόμετρα και την τιμή [6], [19]. Ένα άλλο παράδειγμα θα μπορούσε να είναι η αναζήτηση εστιατορίων με βάση την ποιότητα που προσφέρουν και την τιμή [27]. Όπως φαίνεται από τα παραδείγματα μια skyline επερώτηση αποτελεί μια αποτελεσματική τεχνική για την επεξεργασία επερωτήσεων που επαφίενται στις προτιμήσεις του χρήστη. Ωστόσο, με την τρομακτική ταχύτητα που οι χρήστες δημιουργούν και χρησιμοποιούν δεδομένα, οι skyline επερωτήσεις αντιμετωπίζουν προβλήματα με τη διαχείριση ενός τόσο μεγάλου όγκου δεδομένων. Αυτή την αδυναμία έρχεται να καλύψει το Υπολογιστικό Νέφος. Η επεξεργασία των επερωτήσεων μεγάλου όγκου δεδομένων σε υπολογιστικό νέφος, παρέχει μεγάλη

υπολογιστική δύναμη και δυνατότητες αποθήκευσης λύνοντας αποδοτικά το πρόβλημα του μεγάλου όγκου δεδομένων [28]. Ακριβώς, λόγω αυτού του μεγάλου όγκου δεδομένων, ο υπολογισμός των skyline επερωτήσεων μπορεί να είναι ιδιαίτερα ακριβός και η ύπαρξη αποδοτικών αλγόριθμων είναι ζωτικής σημασίας για εφαρμογές που απαιτούν μικρό χρόνο απόκρισης. Με τη χρήση προηγμένων πολυπύρηνων αρχιτεκτονικών και άλλων πλατφορμών παράλληλης επεξεργασίας, οι παράλληλοι skyline αλγόριθμοι είναι ένας τρόπος να βελτιωθεί η απόδοσή τους [19]. Ένα τέτοιο εργαλείο παράλληλης επεξεργασίας είναι και το MapReduce, το οποίο έχει κερδίσει σημαντικό έδαφος στην επεξεργασία μεγάλου όγκου δεδομένων [3].

Κατά την παράλληλη επεξεργασία οποιωνδήποτε δεδομένων, ένας παράλληλος αλγόριθμος τρέχει σ' ένα cluster με P servers που συνδέονται μεταξύ τους με ένα γρήγορο δίκτυο. Τα δεδομένα αρχικά μοιράζονται στους servers, με τέτοιο τρόπο έτσι ώστε κάθε server να έχει O (n/P) δεδομένα, όπου n ο αριθμός των δεδομένων. Ο υπολογισμός διεξάγεται σε «κύκλους», όπου κάθε κύκλος αποτελείται από τοπικό υπολογισμό ακολουθούμενο από global ανταλλαγή δεδομένων. Ωστόσο, υπάρχουν δυο σημαντικοί παράμετροι πολυπλοκότητας. Η πρώτη παράμετρος είναι ο αριθμός των φορών που πρέπει οι servers να επικοινωνήσουν μεταξύ τους ώστε να συγχρονιστούν. Κάθε φορά που επικοινωνούν μεταξύ τους, αυξάνεται σημαντικά ο χρόνος εκτέλεσης του αλγόριθμου, αφού όλοι οι servers πρέπει να περιμένουν τον πιο αργό server να ολοκληρώσει την εργασία του. Η δεύτερη παράμετρος είναι ο μέγιστος φόρτος εργασίας για κάθε server. Αν κάποιος server πρέπει να επεξεργαστεί περισσότερα δεδομένα από τους άλλους servers, θα καθυστερήσει σημαντικά ολόκληρη τη διαδικασία επεξεργασίας των δεδομένων. Επίσης, θα αναγκαστεί να χρησιμοποιήσει και το δίσκο εκτός από την κύρια μνήμη. Άρα για να επιτευχθεί χαμηλός φόρτος εργασίας ανά server, θα πρέπει να μοιραστούν ισόποσα τα δεδομένα στους servers και να αποφευχθούν οι αντιγραφές των ίδιων δεδομένων σε πολλαπλούς servers [27]. Για την αποφυγή της καθυστέρησης της επεξεργασίας των δεδομένων, το μοντέλο επεξεργασίας το οποίο θα αναπτυχθεί στην παρούσα εργασία δε στηρίζεται σε πολλαπλές φάσεις επικοινωνίας μεταξύ των servers.

Η παράλληλη επεξεργασία μιας skyline επερώτησης χωρίζεται σε τρεις φάσεις. Στην πρώτη φάση εκτελείται ο διαμοιρασμός του χώρου σε N διαμερίσεις με κάποια από τις μεθόδους διαμοιρασμού χώρου που έχουν ήδη αναφερθεί. Στη δεύτερη φάση, εκτελείται η skyline επερώτηση σε κάθε διαμέριση ξεχωριστά και παράλληλα κι έτσι

παράγονται τα τοπικά skyline σημεία για κάθε διαμέριση. Στην τρίτη και τελευταία φάση, συγχωνεύονται τα τοπικά skyline σημεία των Ν διαμερίσεων, συγκρίνονται μεταξύ τους και προκύπτουν τα τελικά skyline σημεία [6].

Ο υπάρχων κώδικας ήδη επεξεργάζεται παράλληλα τις skyline επερωτήσεις εκτός Hadoop. Επομένως, επιμέρους στόχος και πρόκληση για την ολοκλήρωση της εργασίας αποτελεί η κατάλληλη προσαρμογή των ήδη υπαρχουσών μεθόδων που εκτελούν το διαμερισμό χώρου και τον υπολογισμό των skyline σημείων, έτσι ώστε να υποστούν παράλληλη επεξεργασία και στο περιβάλλον του Hadoop-MapReduce εξάγοντας τα ίδια αποτελέσματα με αυτά που εξάγουν όταν εκτελούνται εκτός Hadoop, με την προϋπόθεση ότι διαβάζουν το ίδιο αρχείο εισόδου. Πιο συγκεκριμένα, χρειάστηκε να βρεθεί τρόπος οι τρεις βασικές φάσεις του παράλληλου skyline υπολογισμού, δηλαδή διαμερισμός χώρου, τοπικά skyline σημεία και συγχώνευση, να χωριστούν σε jobs και μεθόδους map() και reduce() οι οποίες θα τρέξουν στο MapReduce.

Κάποιοι από τους τεχνικούς στόχους που πρέπει να επιτευχθούν είναι η ελαχιστοποίηση των MapReduce jobs καθώς και η ελαχιστοποίηση των δεδομένων που γίνονται shuffle έτσι ώστε να μειωθεί ο χρόνος εκτέλεσης των επερωτήσεων και να χρησιμοποιηθεί μόνο η κύρια μνήμη αποφεύγοντας τη χρήση του δίσκου. Ωστόσο, γενικός και απώτερος στόχος της εργασίας είναι να εξεταστεί κατά πόσο η παράλληλη επεξεργασία των Skyline Επερωτήσεων στο Hadoop είναι αποδοτική έτσι ώστε να μπορέσει αυτό να χρησιμοποιηθεί ως τεχνολογικό υπόβαθρο στις διαδικασίες, εφαρμογές και πλατφόρμες της ηλεκτρονικής μάθησης.

Για την αξιολόγηση και την εξαγωγή συμπερασμάτων σχετικά με την απόδοση της παράλληλης εκτέλεσης των επερωτήσεων στο περιβάλλον του Hadoop, θα πραγματοποιηθεί πειραματική μελέτη κατά τη διάρκεια της οποίας θα εκτελεστεί ο κώδικας με διαφορετικές κάθε φορά συνθήκες. Κάθε φορά θα αλλάζει το αρχείο εισόδου, η κατανομή των σημείων καθώς και ο αριθμός των διαμερίσεων. Από την πειραματική μελέτη, αναμένονται αποτελέσματα τα οποία θα δείξουν ποιά μέθοδος είναι αποδοτικότερη. Για να γίνει περισσότερο σαφές, θα εξεταστεί ποια μέθοδος χρειάζεται το λιγότερο χρόνο για να ολοκληρώσει το διαμοιρασμό του χώρου και τον υπολογισμό των skyline σημείων σε παράλληλη επεξεργασία και ποιά μέθοδος έχει

τα λιγότερα shuffle δεδομένα κάθε φορά που αλλάζει ο όγκος των δεδομένων προς επεξεργασία, η κατανομή τους και ο αριθμός των διαμερίσεων.

Άρα, συνοπτικά βασικός στόχος της εργασίας είναι:

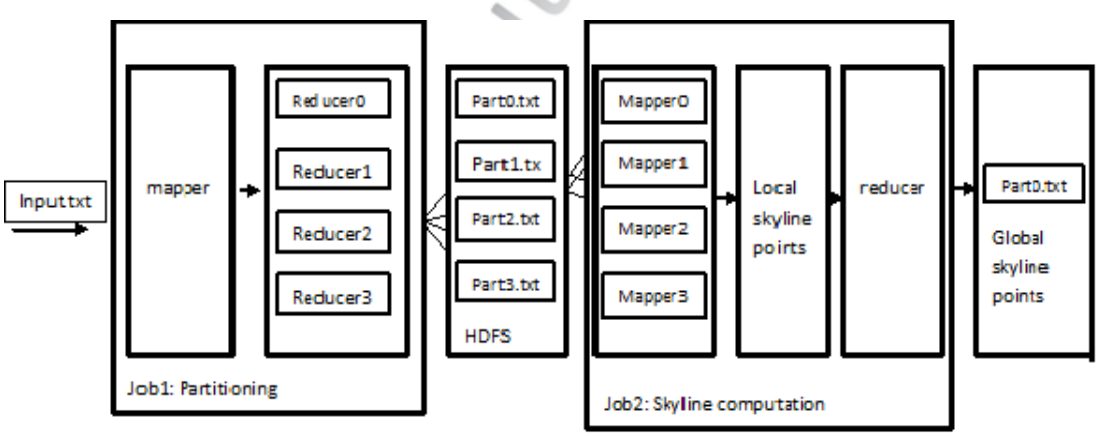
- Να εκτελεστούν αποδοτικά και παράλληλα οι Skyline Επερωτήσεις στο Hadoop συνδυάζοντας τα πλεονεκτήματα και των δύο τεχνολογιών.

Τεχνικοί στόχοι της εργασίας, οι οποίοι θα συμβάλλουν στην επίτευξη του βασικού στόχου, είναι οι εξής:

- Να προσαρμοστούν κατάλληλα οι ήδη υπάρχουσες μέθοδοι ώστε να εκτελεστούν στο περιβάλλον του Hadoop.
- Να χωριστούν σε jobs και σε map() και reduce() μεθόδους, οι φάσεις του διαμοιρασμού χώρου, τοπικού υπολογισμού των skyline σημείων και συγχώνευση αυτών.
- Να ελαχιστοποιηθούν οι MapReduce jobs ώστε να μειωθεί ο χρόνος εκτέλεσης.
- Να ελαχιστοποιηθούν τα shuffle data ώστε να χρησιμοποιηθεί μόνο η κύρια μνήμη.

4. ΣΧΕΔΙΑΣΗ

Για την επίτευξη των στόχων και σκοπών της παρούσας εργασίας χρειάστηκε να γίνει μια σχεδίαση σε επίπεδο αρχιτεκτονικής αλλά και κώδικα, έτσι ώστε οι τρεις φάσεις της παράλληλης επεξεργασίας των skyline επερωτήσεων, δηλαδή διαμοιρασμός του χώρου, τοπικός υπολογισμός των skyline σημείων και συγχώνευση αυτών, να χωριστούν σε MapReduce jobs. Έτσι λοιπόν, ο κώδικας σχεδιάστηκε έτσι ώστε να εκτελείται σε δυο jobs, όπως φαίνεται στην Εικόνα 12, με σκοπό να εκμεταλλευτούν οι δυνατότητες που προσφέρει έτσι κι αλλιώς το MapReduce, όπως είναι η παραγωγή διαμερίσεων και η συγχώνευση των σημείων στην μέθοδο reduce(). Έτσι, χρειάστηκε μόνο να οριστεί ο τρόπος που θα εκτελεστεί ο διαμερισμός του χώρου και να καθοριστούν τα κριτήρια σύμφωνα με τα οποία τα σημεία θα πάνε στις αντίστοιχες διαμερίσεις. Με αυτό τον τρόπο διευκολύνθηκε σε σημαντικό βαθμό η ανάπτυξη του κώδικα.



Εικόνα 12: MapReduce διάγραμμα ροής.

Ωστόσο, πριν από την εκτέλεση του κώδικα θα πρέπει να δημιουργηθούν τα αρχικά αρχεία ή το αρχείο προς επεξεργασία με τη βοήθεια γεννητριών παραγωγής τυχαίων αριθμών. Επίσης, για τη σωστή εκτέλεση του κώδικα χρειάζεται να γίνουν κάποιες ρυθμίσεις σε μεταβλητές που χρησιμοποιούνται από τον κώδικα. Οι

μεταβλητές αυτές μπορεί να αφορούν στο framework του Hadoop, στον κώδικα ή και στο σύστημα αρχείων του Hadoop (HDFS).

Σε επίπεδο κλάσεων, για τη φάση του διαμερισμού του χώρου δημιουργήθηκαν τρείς κλάσεις, δηλαδή μια για κάθε μέθοδο που εκτελεί το διαμοιρασμό ενώ για τη φάση του υπολογισμού των skyline σημείων δημιουργήθηκε μια κλάση. Κάθε μια κλάση που υλοποιεί το διαμοιρασμό του χώρου μετά την ολοκλήρωσή της και πριν την έξοδο από το σύστημα, καλεί την κλάση που υπολογίζει τα skyline σημεία, χρησιμοποιώντας την μέθοδο του δεσίματος των MapReduce jobs (chaining MR jobs), με την οποία τα jobs φαίνεται να τρέχουν σαν ένα job αποκρύπτοντας από το χρήστη τις τεχνικές λεπτομέρειες. Άρα, η έξοδος της πρώτης κλάσης είναι η είσοδος της δεύτερης.

Όλες οι κλάσεις του κώδικα είναι δομημένες σύμφωνα με το MapReduce όπως έχει περιγραφεί σε προηγούμενο κεφάλαιο. Πιο συγκεκριμένα, όλες οι κλάσεις περιλαμβάνουν κλάσεις Mapper, Partitioner και Reducer οι οποίες εκτελούνται η μια μετά την άλλη με τη σειρά που αναφέρονται. Μέσα στις κλάσεις Mapper και Reducer υλοποιούνται οι μέθοδοι map() και reduce(), οι οποίες παίρνουν ορίσματα συγκεκριμένου τύπου. Όπως έχει αναφερθεί και νωρίτερα η είσοδος για το MapReduce είναι μια λίστα από ζεύγη (key1, value1) συγκεκριμένου τύπου, οι οποίοι θα αναλυθούν στο επόμενο κεφάλαιο.

4.1 Παραγωγή αρχείων τυχαίων αριθμών

Τα δεδομένα τα οποία διαχειρίζονται από τον κώδικα είναι πολυδιάστατα, όπως περιγράφεται και στο κεφάλαιο 1. Οι βάσεις δεδομένων περιέχουν εγγραφές οι οποίες αποτελούνται από πολλές διαστάσεις. Στο παράδειγμα των ξενοδοχείων, το οποίο είναι παράδειγμα με δεδομένα δυο διαστάσεων, η τιμή του δωματίου αποτελεί τη μια διάσταση και η απόστασή του από τη θάλασσα την άλλη. Σ' ένα σύστημα αξόνων, κάθε διάσταση αναπαρίσταται σ' έναν άξονα, όπως φαίνεται και στην εικόνα 4. Στη συγκεκριμένη περίπτωση, δημιουργούνται και επεξεργάζονται δεδομένα τριών διαστάσεων, άρα και στην απεικόνισή τους στο χώρο υπάρχουν τρείς άξονες x,y και z.

Στην παρούσα εργασία, η παραγωγή του αρχικού αρχείου προς επεξεργασία γίνεται από γεννήτριες τυχαίων αριθμών, στις οποίες ο χρήστης ορίζει τον αριθμό των

διαστάσεων των σημείων, τον αριθμό των σημείων καθώς και την ανώτατη τιμή που μπορεί να πάρουν οι διαστάσεις. Επίσης, στην παραγωγή του αρχικού αρχείου μπορεί να καθοριστεί και η κατανομή των σημείων. Εδώ, για τις ανάγκες της πειραματικής μελέτης, όπως θα δούμε και σε επόμενο κεφάλαιο, έχουν δημιουργηθεί αρχεία με uniform, correlated και anticorrelated κατανομή, με σημεία τριών διαστάσεων και με διαφορετικό αριθμό σημείων κάθε φορά. Κάθε γραμμή των αρχείων συμβολίζει ένα σημείο και αποτελείται από το id του σημείου και τις συντεταγμένες του, δηλαδή είναι της μορφής (id, double1 double2 double3). Ενδεικτικά, παρατίθεται μέρος του αρχικού αρχείου προς επεξεργασία:

```
.....
13 562.3367543543488 6408.050210416008 9291.057191908905
14 5083.328248721817 7173.588900683193 2696.513295970286
15 846.1908181911843 6395.0458613535475 597.4706344085845
16 3731.8676602636974 3302.1851256857194 5500.365951174946
17 3866.6443144518594 5536.402697346013 5258.445899533972
.....
```

4.2 Σύστημα αρχείων του Hadoop (HDFS) και Ρύθμιση παραμέτρων

Σημαντικό ρόλο στη σωστή εκτέλεση του κώδικα παίζει η ρύθμιση των παραμέτρων που χρησιμοποιούνται είτε για τη ρύθμιση του συστήματος αρχείων είτε για τη ρύθμιση των παραμέτρων που χρησιμοποιούνται από το framework του Hadoop. Μια σωστή και ακριβής ρύθμιση παραμέτρων εξασφαλίζει την εύρυθμη λειτουργία του κώδικα ενώ ταυτόχρονα ελαχιστοποιεί το περιθώριο σφάλματος.

4.2.1 Ρύθμιση παραμέτρων

Στην αρχή της εκτέλεσης του κώδικα και πριν την εκτέλεση του πρώτου Job, πραγματοποιείται η ρύθμιση των παραμέτρων των jobs του MapReduce καθώς και η αρχικοποίηση των παραμέτρων που χρησιμοποιούνται αργότερα στον κώδικα. Η ρύθμιση όλων των παραμέτρων πραγματοποιείται μέσα από ένα instance της κλάσης JobConf, όπως αυτή ορίζεται στις βιβλιοθήκες του hadoop-0.20. Ο χρήστης μπορεί να αρχικοποιήσει τις παραμέτρους του μέσα από ένα συγκεκριμένο αρχείο, το οποίο

διαβάζεται πριν την εκτέλεση του βασικού κώδικα. Στο αρχείο αυτό, ο χρήστης μπορεί να παραμετροποιήσει τον αριθμό των διαστάσεων των σημείων, τον αριθμό των σημείων, την ανώτατη τιμή που μπορούν οι διαστάσεις, τον αριθμό των διαμερίσεων του χώρου, τα partitions δηλαδή, καθώς και οποιαδήποτε άλλη παράμετρο θέλει να χρησιμοποιήσει για την εκτέλεση του κώδικα. Στην ουσία, μέσα από αυτό το αρχείο ο χρήστης θα έχει τη δυνατότητα να αλληλεπιδρά με τον κώδικα και ν' αλλάζει τις συνθήκες εκτέλεσής του εάν το επιθυμεί, χωρίς να επεξεργάζεται τον ίδιο τον κώδικα.

Επίσης, κατά τη διάρκεια της ρύθμισης των παραμέτρων και πριν την εκτέλεση του κώδικα, αρχικοποιούνται και κάποιες μεταβλητές του framework του Hadoop. Πιο αναλυτικά, ορίζεται από ποια κλάση θα δημιουργηθεί το εκτελέσιμο αρχείο, πόσα reduce tasks θα χρησιμοποιηθούν, το όνομα των κλάσεων που θα λειτουργήσουν ως Mapper, Partitioner και Reducer αντίστοιχα. Επίσης, ορίζεται ο τύπος του τελικού key και value, που στην προκειμένη περίπτωση θα είναι τύπου Text.

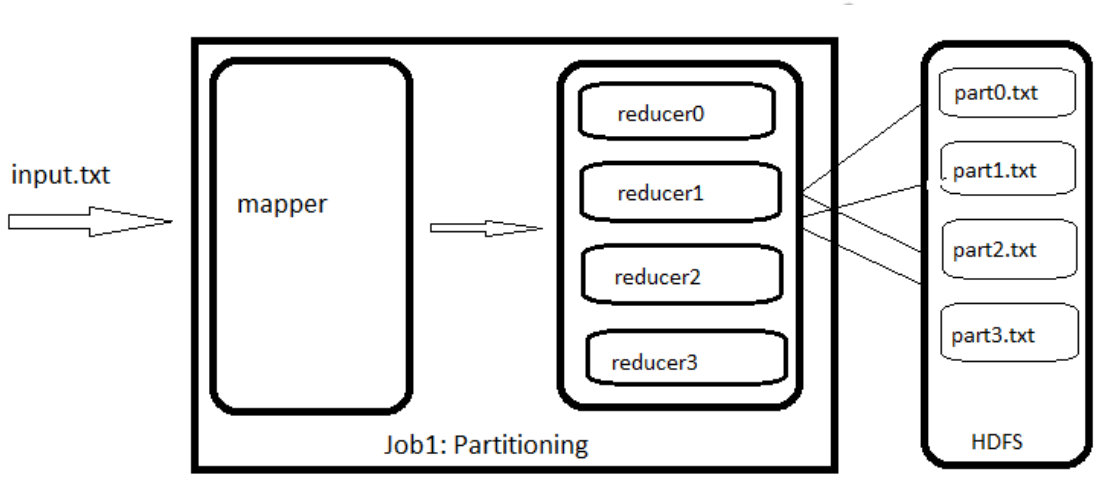
4.2.2 Σύστημα αρχείων του Hadoop

Η κατάλληλη ρύθμιση του συστήματος αρχείων του Hadoop (HDFS) αποτελεί μια πολύ σημαντική διαδικασία καθώς το HDFS είναι ένα κομμάτι με το οποίο αλληλεπιδρά συνεχώς ο κώδικας είτε για να διαβάσει αρχεία είτε για να γράψει αρχεία σε αυτό.

Όπως είναι αναμενόμενο, η ρύθμιση του συστήματος αρχείων πραγματοποιείται πριν την έναρξη της εκτέλεσης του κώδικα. Σε αυτή τη φάση λοιπόν, δημιουργούνται ή ορίζονται στο File System του Hadoop (HDFS) συγκεκριμένα μονοπάτια για κάθε κλάση, η ονομασία των οποίων καθορίζεται από το χρήστη. Συγκεκριμένα δημιουργείται από το ίδιο το framework ένα μονοπάτι εξόδου στο HDFS πριν ξεκινήσει η φάση του mapping, ενώ το μονοπάτι εισόδου δημιουργείται από το χρήστη μέσω command prompt, οπότε κατά τη διάρκεια της ρύθμισης των παραμέτρων ορίζεται μόνο τα ονόματά τους. Επίσης, ο χρήστης θα πρέπει να έχει 'ανεβάσει' στο μονοπάτι εισόδου του HDFS το αρχικό αρχείο προς επεξεργασία πριν την εκτέλεση του κώδικα.

Τέλος, επειδή όπως έχει αναφερθεί νωρίτερα, στην παρούσα εργασία η έξοδος της πρώτης κλάσης είναι η είσοδος της δεύτερης, ο χρήστης πρέπει να τις ορίσει ακριβώς με το ίδιο όνομα. Για παράδειγμα, αν το μονοπάτι εξόδου της κλάσης που εκτελεί το διαμοιρασμό του χώρου είναι το "outputPart" , τότε και το μονοπάτι εισόδου της κλάσης που εκτελεί τον υπολογισμό των skyline σημείων πρέπει να είναι το ίδιο.

4.3 Job 1: Διαμοιρασμός του Χώρου



Εικόνα 13: Job1 – Διαμερισμός του χώρου σε 4 διαμερίσεις με χρήση 4 reducers.

Στην παρούσα εργασία, ο διαμερισμός του χώρου γίνεται με τις τρεις μεθόδους όπως αναφέρθηκαν στο κεφάλαιο 2. Θ' αναπτυχθούν τρείς υλοποιήσεις, μια για κάθε τεχνική διαμοιρασμού του χώρου. Ωστόσο, θα προσαρμοστούν κατάλληλα ώστε να εκτελεστούν μέσω των μεθόδων map() και reduce(), λειτουργώντας με τον ίδιο τρόπο και εξάγοντας τα ίδια αποτελέσματα με αυτά που εξάγουν όταν εκτελούνται εκτός Hadoop.

Το πρώτο job, στο οποίο πραγματοποιείται ο διαμερισμός του χώρου, αποτελείται από μια φάση map και μια φάση reduce και σε αυτό αντιστοιχεί και εκτελείται η φάση του διαμοιρασμού του χώρου από το οποίο παράγονται όσες διαμερίσεις έχει ορίσει ο χρήστης. Οι reducers είναι τόσοι όσες και οι διαμερίσεις που θέλει ο χρήστης να δημιουργηθούν και κάθε reducer δημιουργεί μια διαμέριση στο HDFS. Σε αυτό το σημείο, όπως θα αναλυθεί και παρακάτω, τα σημεία

χωρίζονται σε διαμερίσεις και το μόνο που διαφέρει κάθε φορά είναι ο τρόπος αυτού του διαμοιρασμού και ίσως ο αριθμός των διαμερίσεων.

Κάθε κλάση, η οποία εκτελεί διαμοιρασμό χώρου, περιλαμβάνει μια κλάση που υλοποιεί το interface Mapper, Reducer και Partitioner αντίστοιχα. Το σημαντικότερο ρόλο βέβαια έχει η κλάση Partitioner , η οποία στην ουσία εκτελεί το διαμερισμό του χώρου, χωρίς αυτό να σημαίνει ότι οι υπόλοιπες κλάσεις είναι υποδεέστερες, αφού και αυτές διευκολύνουν το 'διάβασμα' του αρχείου εισόδου αλλά και τη δημιουργία των αρχείων εξόδου. Ο αριθμός των reducers που χρησιμοποιούνται εξαρτάται από τον αριθμό των διαμερίσεων, όπως φαίνεται και στην Εικόνα 13.

4.3.1 Mapper

Η κλάση που υλοποιεί το interface Mapper περιλαμβάνει την μέθοδο map(), η οποία διαβάζει το αρχείο εισόδου. Όπως, έχει αναφερθεί νωρίτερα κάθε γραμμή του αρχείου συμβολίζει ένα σημείο και έχει την μορφή (id double1 double2 double3), όπου id είναι το id του σημείου και οι double αριθμοί είναι οι τρείς συντεταγμένες του, εφόσον αυτό έχει τρείς διαστάσεις.

Η μέθοδος map() δέχεται ένα key τύπου Object και value όλη τη γραμμή του αρχείου εισόδου. Στην μέθοδο δε γίνεται καμιά επεξεργασία της γραμμής απλώς εξάγει σαν key τη γραμμή του αρχείου που προηγουμένως ήταν value και σαν value εξάγει ένα κενό string. Δηλαδή, η είσοδος της map είναι (Key = Object, Value = id double1 double2 double3) και η έξοδος είναι (Key = id double1 double2 double3, Value = String).

4.3.2 Partitioner

Μια από τις μεθόδους που περιλαμβάνει η κλάση που υλοποιεί το interface Partitioner είναι η μέθοδος getPartition(Text key, Text value, int m_nNrPartitions) που δέχεται ως είσοδο την έξοδο της μεθόδου map() και τον αριθμό των διαμερίσεων. Δηλαδή το key είναι ολόκληρη η γραμμή του αρχείου (id double1 double2 double3). Αυτή η μέθοδος μετατρέπει τις συντεταγμένες των σημείων, ανάλογα με την τεχνική του διαμοιρασμού χώρου που χρησιμοποιείται, υπολογίζει τα όρια των διαμερίσεων και στη συνέχεια αντιστοιχίζει τα σημεία με τις διαμερίσεις ώστε να είναι ξεκάθαρο

κάθε σημείο σε ποια διαμέριση θα πάει. Ο διαμερισμός του χώρου που πραγματοποιείται είναι στατικός.

Διαμερισμός χώρου με βάση τις γωνιακές συντεταγμένες

Σε αυτή την μέθοδο, με δεδομένο τον αριθμό των διαμερίσεων που θέλει ο χρήστης να δημιουργήσει, υπολογίζονται τα όριά τους και στη συνέχεια διαβάζεται το αρχείο προς επεξεργασία εγγραφή προς εγγραφή. Κάθε εγγραφή αναπαριστά και ένα σημείο στο χώρο. Στη συνέχεια, οι καρτεσιανές συντεταγμένες του κάθε σημείου θα μετατρέπονται σε υπερσφαιρικές και με βάση αυτές τα σημεία θα αντιστοιχίζονται στις διαμερίσεις.

Διαμερισμός χώρου με βάση την grid τεχνική

Στο στατικό διαμοιρασμό του χώρου με την grid τεχνική, διαβάζεται το αρχείο προς επεξεργασία γραμμή προς γραμμή και στη συνέχεια με βάση τον αριθμό των διαστάσεων και τον αριθμό των διαμερίσεων υπολογίζονται για κάθε διάσταση οι διαμερίσεις που θα δημιουργηθούν. Στο τέλος καθορίζεται σε ποιά διαμέριση θα πάει κάθε σημείο.

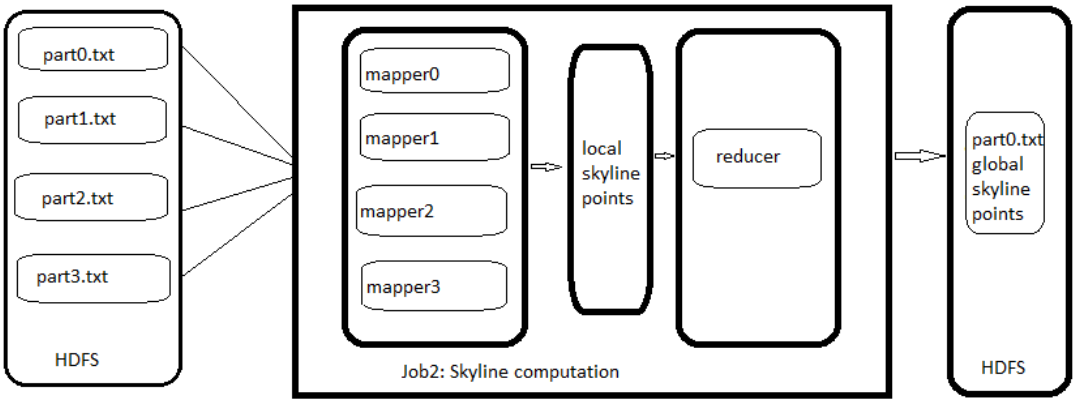
Διαμερισμός χώρου μέσω hyperplane προβολής

Στην μέθοδο της προβολής των σημείων στην ευθεία γραμμή, αρχικά διαβάζεται το αρχείο εγγραφή προς εγγραφή. Κάθε εγγραφή ή γραμμή αναπαριστά ένα σημείο στο χώρο. Στη συνέχεια, τα σημεία προβάλλονται στην ευθεία υπολογίζοντας εκ νέου τις συντεταγμένες τους. Τέλος, αντιστοιχίζονται τα σημεία στις διαμερίσεις, ανάλογα με τα όρια στα οποία χωρίζονται οι διαστάσεις.

4.3.3 Reducer

Η κλάση που υλοποιεί το interface Reducer περιλαμβάνει την μέθοδο reduce(), η οποία στη συγκεκριμένη περίπτωση απλά γράφει σε κάθε διαμέριση ολόκληρη τη γραμμή του αρχείου, γι' αυτό και η έξοδός της έχει ως key ολόκληρη τη γραμμή και value ένα κενό string, δηλαδή (Key = id double1 double2 double3, Value = String).

4.4 Job 2: Skyline Επερώτηση



Εικόνα 14: Job2 - Υπολογισμός skyline σημείων με 4 διαμερίσεις ως αρχεία εισόδου.

Το δεύτερο job, όπου εκτελείται ο υπολογισμός των skyline σημείων, υλοποιείται σε μια μόνο κλάση, η οποία χρησιμοποιείται σε όλα τα σενάρια εκτέλεσης του κώδικα. Σε αυτό το job αντιστοιχούν οι φάσεις του τοπικού υπολογισμού των skyline σημείων και της συγχώνευσης αυτών. Η κλάση που υπολογίζει τα skyline σημεία δέχεται ως αρχεία εισόδου τις διαμερίσεις που έχουν παραχθεί κατά τη διάρκεια του πρώτου job, οι οποίες επεξεργάζονται παράλληλα και περιλαμβάνει και αυτή τις κλάσεις που υλοποιούν τα interface Mapper και Reducer.

Συγκεκριμένα, στη φάση του mapping χρησιμοποιούνται τέσσερις mappers, όσες είναι δηλαδή και οι διαμερίσεις που δίνονται ως είσοδος στην κλάση. Κάθε mapper διαβάζει το αρχείο του και υπολογίζει τα τοπικά skyline σημεία για κάθε διαμέριση. Στη συνέχεια, τα τοπικά skyline σημεία περνάνε ως είσοδος στη φάση του reducing, όπου συγκρίνονται μεταξύ τους με σκοπό να παραμείνουν αυτά που επικρατούν έναντι των υπολοίπων. Στο τέλος, θα έχουν μείνει τα τελικά skyline σημεία για τα οποία ισχύει ότι δεν υπάρχει κανένα άλλο σημείο καλύτερο από αυτά. Στην ουσία, μέσα σ' ένα job υπολογίζονται τα τοπικά skyline σημεία και κατόπιν τα τελικά.

4.4.1 Mapper

Η κλάση που υλοποιεί το interface Mapper περιλαμβάνει τη μέθοδο map(), η οποία θα διαβάσει τις διαμερίσεις. Κάθε γραμμή της διαμέρισης είναι της μορφής (id double1 double2 double3) και αναπαριστά ένα σημείο. Η map() δέχεται ως key έναν αριθμό (byte offset) LongWritable, ο οποίος δε χρησιμοποιείται στον κώδικα, και ως value την κάθε γραμμή του αρχείου. Στη συνέχεια, υπολογίζεται η εντροπία των σημείων που είναι ένας double αριθμός. Όσο μικρότερος είναι αυτός ο αριθμός, τόσο μεγαλύτερη πιθανότητα έχει ένα σημείο να ανήκει στο σύνολο των skyline σημείων. Στη συνέχεια, υπολογίζονται τα skyline σημεία για κάθε διαμέριση συγκρίνοντάς τα και εξετάζοντας ποια σημεία επικρατούν έναντι άλλων. Τα σημεία που επικρατούν έναντι των άλλων αποτελούν τα skyline σημεία και περνάνε ως είσοδος στην επόμενη φάση, ενώ τα υπόλοιπα απορρίπτονται. Συνοπτικά, η είσοδος της μεθόδου map() είναι της μορφής (Key = LongWritable, Value = id double1 double2 double3) και η έξοδος είναι της μορφής (Key = DoubleWritable, Value = id double1 double2 double3).

4.4.2 Reducer

Τέλος, για τη φάση του reducing δημιουργήθηκε μια κλάση που υλοποιεί το interface Reducer και περιλαμβάνει την μέθοδο reduce(), η οποία δέχεται ως είσοδο την έξοδο της map(). Δηλαδή, η είσοδός της είναι της μορφής (Key = DoubleWritable, Value = id double1 double2 double3). Εδώ κάθε σημείο συγκρίνεται με όλα τα υπόλοιπα για να εξεταστεί αν επικρατεί των άλλων ή όχι. Όσα σημεία επικρατούν των άλλων αποθηκεύονται προσωρινά σε μια δομή ενώ τα υπόλοιπα σημεία απορρίπτονται. Μετά τη σύγκριση όλων των τοπικών skyline σημείων όλων των διαμερίσεις και αφού έχουν βρεθεί τα τελικά skyline σημεία, ο reducer δημιουργεί ένα αρχείο στο σύστημα αρχείων του Hadoop και μέσα σε αυτό γράφει τα τελικά skyline σημεία. Η έξοδος της μεθόδου reduce() και οι εγγραφές του τελικού αρχείου είναι της μορφής (Key = DoubleWritable, Value = id double1 double2 double3).

5. ΥΛΟΠΟΙΗΣΗ

Επόμενο βήμα μετά τη σχεδίαση του κώδικα προς εκτέλεση είναι η ανάπτυξή του. Η ανάπτυξη του κώδικα έγινε με τη βοήθεια του Eclipse [29] σε γλώσσα προγραμματισμού Java. Στο project που δημιουργήθηκε στο Eclipse, προστέθηκαν οι βιβλιοθήκες του hadoop-0.20 αλλά και οτιδήποτε ήταν απαραίτητο για τη σωστή λειτουργία του κώδικα.

Στο ίδιο εργαλείο πραγματοποιήθηκε και η παραγωγή των αρχικών αρχείων προς επεξεργασία με τη βοήθεια του κώδικα των γεννητριών παραγωγής τυχαίων αριθμών. Στο συγκεκριμένο κώδικα, ο χρήστης ορίζει το πλήθος των σημείων που πρέπει να δημιουργηθούν, την ανώτατη τιμή που μπορούν να πάρουν καθώς και το όνομα του αρχείου. Μετά την εκτέλεση του κώδικα, τα αρχεία που δημιουργούνται τοποθετούνται στο τοπικό σύστημα αρχείων.

Μετά την ολοκλήρωση της συγγραφής του κώδικα χρειάστηκε να δημιουργηθεί το κατάλληλο περιβάλλον στο οποίο θα εκτελεστεί. Για το λόγο αυτό, χρειάστηκε να εγκατασταθεί virtual machine με λειτουργικό Linux στο οποίο υπάρχει ήδη εγκατεστημένο το περιβάλλον του Hadoop. Το virtual machine παρέχεται δωρεάν από την Cloudera [30] και η εγκατάστασή του είναι εξαιρετικά εύκολη.

Για τη διευκόλυνση της εκπόνησης της εργασίας, ο αρχικός κώδικας που χρησιμοποιούνταν για τον υπολογισμό των skyline σημείων και για το διαμερισμό του χώρου με τις δυο πρώτες τεχνικές, δηλαδή με βάση την grid τεχνική και με βάση τις γωνιακές συντεταγμένες, και εκτελούνταν εκτός Hadoop παραχωρήθηκε από τον επιβλέποντα της παρούσας εργασίας κύριο Δουλκερίδη Χρήστο [6]. Από τον ίδιο παραχωρήθηκε και ο κώδικας των γεννητριών παραγωγής τυχαίων αριθμών για την παραγωγή των αρχικών αρχείων.

Ο προσαρμοσμένος κώδικας για την εκτέλεση των δυο πρώτων τεχνικών στο περιβάλλον του Hadoop παραχωρήθηκε από τους φοιτητές Ναστούλη Δήμητρα, Σταμπολτά Χριστόφορο και Τυροβολά Κωνσταντίνο και αναπτύχθηκε στα πλαίσια εξαμηνιαίας εργασίας με τίτλο “Cloud Data Management Partitioning Methods in Hadoop” για το μάθημα ‘Διαχείριση Δεδομένων’ στο τμήμα Ψηφιακών Συστημάτων του Παν. Πειραιώς.

5.1 Ρυθμίσεις παραμέτρων και Σύστημα αρχείων του Hadoop

Κατά τη συγγραφή του κώδικα, ένας στόχος που χρειάστηκε να επιτευχθεί ήταν η ευελιξία του κατά την εκτέλεσή του σε διαφορετικά σενάρια, όπως θα δούμε στο επόμενο κεφάλαιο, χωρίς να χρειάζεται ο χρήστης να επεξεργάζεται κάθε φορά τον κώδικα. Για το λόγο αυτό, δημιουργήθηκε αρχείο με τίτλο **settings.ini**, στο οποίο ο χρήστης δίνει τιμές στις παραμέτρους που θα χρησιμοποιηθούν. Στη συγκεκριμένη περίπτωση οι μεταβλητές που αρχικοποιούνται είναι η ανώτερη τιμή που μπορεί να πάρει μια συντεταγμένη, ο αριθμός των διαμερίσεων και ο αριθμός των διαστάσεων των σημείων. Έτσι, κάθε φορά που ο χρήστης θέλει ν' αλλάξει κάποια παράμετρο αρκεί να επεξεργαστεί μόνο το συγκεκριμένο αρχείο και όχι ολόκληρο τον κώδικα. Το συγκεκριμένο αρχείο πρέπει να βρίσκεται σε κάποιο τοπικό φάκελο του virtual machine έτσι ώστε να μπορεί να διαβαστεί από τον κώδικα. Για την αξιοποίηση του συγκεκριμένου αρχείου δημιουργήθηκε μια κλάση, η οποία διαβάζει το αρχείο και θα αναλυθεί περισσότερο παρακάτω.

Εκτός από τις ρυθμίσεις των παραμέτρων του κώδικα, πρέπει να ρυθμιστούν και οι παράμετροι που αφορούν στο HDFS. Αρχικά, ο χρήστης πρέπει να δημιουργήσει στο HDFS το μονοπάτι εισόδου, στο οποίο θα ανέβουν τα αρχικά αρχεία προς επεξεργασία. Η δημιουργία του μονοπατιού εισόδου καθώς και η μεταφορά του αρχείου σε αυτό θα γίνει μέσω command prompt.

5.1.1 Τύποι Δεδομένων

Αξίζει να σημειωθεί ότι για τη συγγραφή κώδικα, κατάλληλου να εκτελεστεί στο περιβάλλον του Hadoop, χρησιμοποιήθηκαν συγκεκριμένοι τύποι δεδομένων. Στην παρούσα εργασία, οι αρχικοί τύποι δεδομένων που χρησιμοποιούνται είναι String, Double και Long οι οποίοι θα πρέπει να μετατραπούν σε Text, DoubleWritable και LongWritable αντίστοιχα. Για παράδειγμα, αν τα δεδομένα των αρχείων που έχουν δημιουργηθεί είναι τύπου Double, τότε θα πρέπει να μετατραπούν σε DoubleWritable πριν επεξεργαστούν από τις μεθόδους map() και reduce(). Η αντίστροφη διαδικασία θα πρέπει να ακολουθηθεί όταν τα δεδομένα θα πρέπει να εγγραφούν στο HDFS. Για την μετατροπή των τύπων δεδομένων υπάρχουν διαθέσιμες εντολές στις βιβλιοθήκες του hadoop-0.20.

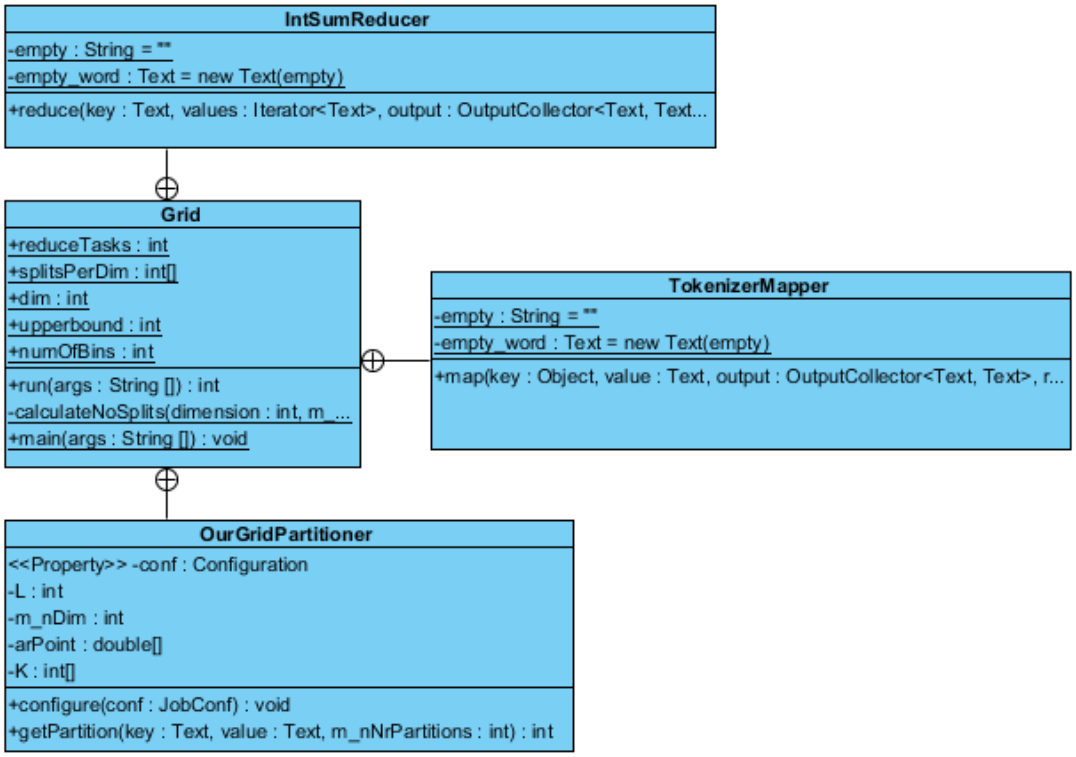
5.1.2 Ρύθμιση παραμέτρων

Για τη ρύθμιση των παραμέτρων δημιουργήθηκε η κλάση Settings.java, η οποία διαβάζει το αρχείο settings.ini και απλά αρχικοποιεί τις παραμέτρους που θα χρησιμοποιηθούν από τις υπόλοιπες κλάσεις για την εκτέλεση του κώδικα. Κάποιες από τις παραμέτρους που αρχικοποιούνται σε αυτήν την κλάση είναι ο αριθμός των διαμερίσεων, ο αριθμός των διαστάσεων και η ανώτατη τιμή που μπορεί να πάρει κάθε διάσταση.

5.2 Job1: Διαμοιρασμός Χώρου

Για κάθε τεχνική που χρησιμοποιείται για το διαμοιρασμό χώρου δημιουργήθηκε μια κλάση με αντίστοιχο όνομα. Παρακάτω παρουσιάζονται αναλυτικά οι τρείς κλάσεις.

5.2.1 Διαμερισμός χώρου με βάση την Grid τεχνική



Εικόνα 15: Διάγραμμα κλάσης Grid.java

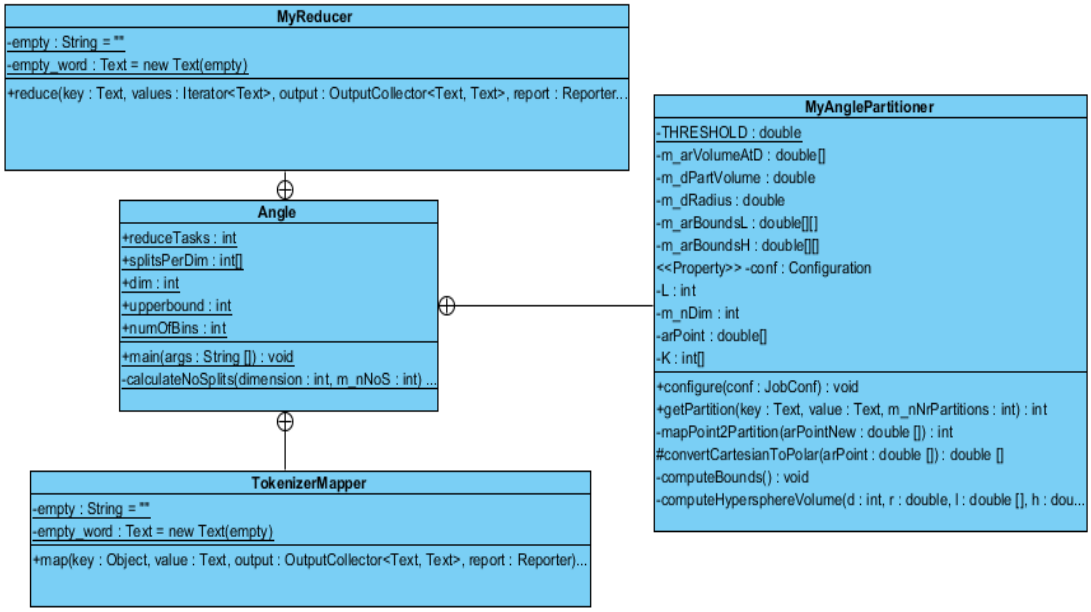
Η κλάση Grid.java εκτελεί το διαμοιρασμό του χώρου με τη Grid μέθοδο και περιέχει τρείς κλάσεις οι οποίες υλοποιούν τα interfaces Mapper, Reducer και Partitioner αντίστοιχα. Στην κλάση που υλοποιεί το mapping βασική μέθοδος είναι η map(). Η map() απλώς διαβάζει γραμμή γραμμή το αρχείο εισόδου και στέλνει στην

κλάση που υλοποιεί το interface Partitioner αυτές τις γραμμές ως κλειδιά (key). Εκεί γίνεται στην ουσία ο διαμερισμός του χώρου.

Αρχικά, με βάση τον αριθμό των διαστάσεων και τον αριθμό των διαμερίσεων που επιθυμεί ο χρήστης, υπολογίζονται οι διαμερίσεις της κάθε διάστασης και στη συνέχεια, καθορίζεται ποιό σημείο θα πάει σε ποιά διαμέριση.

Για την κλάση που υλοποιεί το interface Reducer, δημιουργούνται τόσα reduce tasks όσες είναι και οι διαμερίσεις. Δηλαδή, αν έχουμε τέσσερις διαμερίσεις, τότε θα υπάρχουν και τέσσερα reduce tasks. Κάθε task θα δημιουργήσει μια διαμέριση στο μονοπάτι εξόδου και θα γράψει μέσα τα σημεία που αντιστοιχούν σε αυτό. Το μονοπάτι εξόδου δημιουργείται στο HDFS από το framework, αρκεί ο χρήστης να έχει ορίσει το όνομά του στη φάση της ρύθμισης των παραμέτρων.

5.2.2 Διαμερισμός χώρου με βάση τις γωνιακές συντεταγμένες



Εικόνα 16: Διάγραμμα κλάσης Angle.java

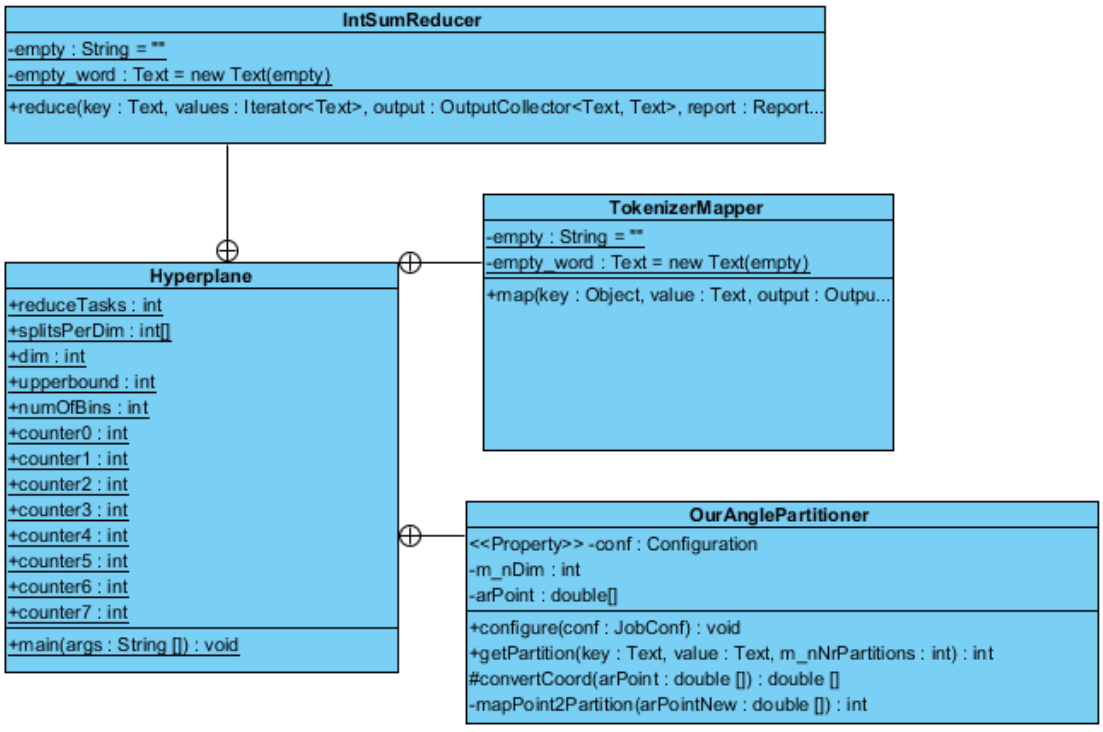
Η κλάση Angle.java είναι δομημένη ακριβώς με τον ίδιο τρόπο με την προηγούμενη κλάση. Οι κλάσεις, οι οποίες υλοποιούν τις φάσεις mapping και reducing, λειτουργούν ακριβώς με τον ίδιο τρόπο. Εδώ η διαφορά με την προηγούμενη κλάση είναι ο τρόπος που γίνεται ο διαμερισμός του χώρου, εφόσον χρησιμοποιείται διαφορετική τεχνική.

Η κλάση που υλοποιεί το interface Mapper και περιέχει την μέθοδο map(), διαβάζει το αρχείο εισόδου γραμμή γραμμή και στέλνει τα σημεία ως κλειδιά στην μέθοδο που εκτελεί το διαμοιρασμό του χώρου.

Η κλάση που υλοποιεί το interface Partitioner δέχεται ως είσοδο από την map() ένα ένα τα σημεία τα οποία είναι της μορφής (id double1 double2 double3). Πριν επεξεργαστούν τα σημεία που δέχεται η μέθοδος ως είσοδο, υπολογίζονται τα όρια των διαμερίσεων με βάση τις διαστάσεις των σημείων και τον αριθμό των διαμερίσεων που επιθυμεί ο χρήστης. Στη συνέχεια, κάθε γραμμή του αρχείου χωρίζεται σε κομμάτια (tokens) με τη βοήθεια ενός StringTokenizer. Κατόπιν, κάθε double αριθμός, δηλαδή κάθε συντεταγμένη, καταχωρίζεται σ' έναν πίνακα και για κάθε σημείο καλείται η μέθοδος convertCartesianToPolar(), η οποία μετατρέπει τις συντεταγμένες σε γωνιακές συντεταγμένες. Παρακάτω, καλείται η μέθοδος mapPoint2Partition(), η οποία με βάση τις τιμές των συντεταγμένων των σημείων και τα όρια των διαμερίσεων αντιστοιχίζει τα σημεία στις ανάλογες διαμερίσεις.

Τέλος, στην κλάση που υλοποιεί το interface Reducer, δημιουργούνται reduce tasks ίσα σε αριθμό με των διαμερίσεων, τα οποία δημιουργούν τις διαμερίσεις στο μονοπάτι εξόδου του HDFS και γράφουν σε αυτά τα αντίστοιχα σημεία.

5.2.3 Διαμοιρασμός χώρου μέσω Hyperplane προβολής



Εικόνα 17:Διάγραμμα κλάσης Hyperplane.java

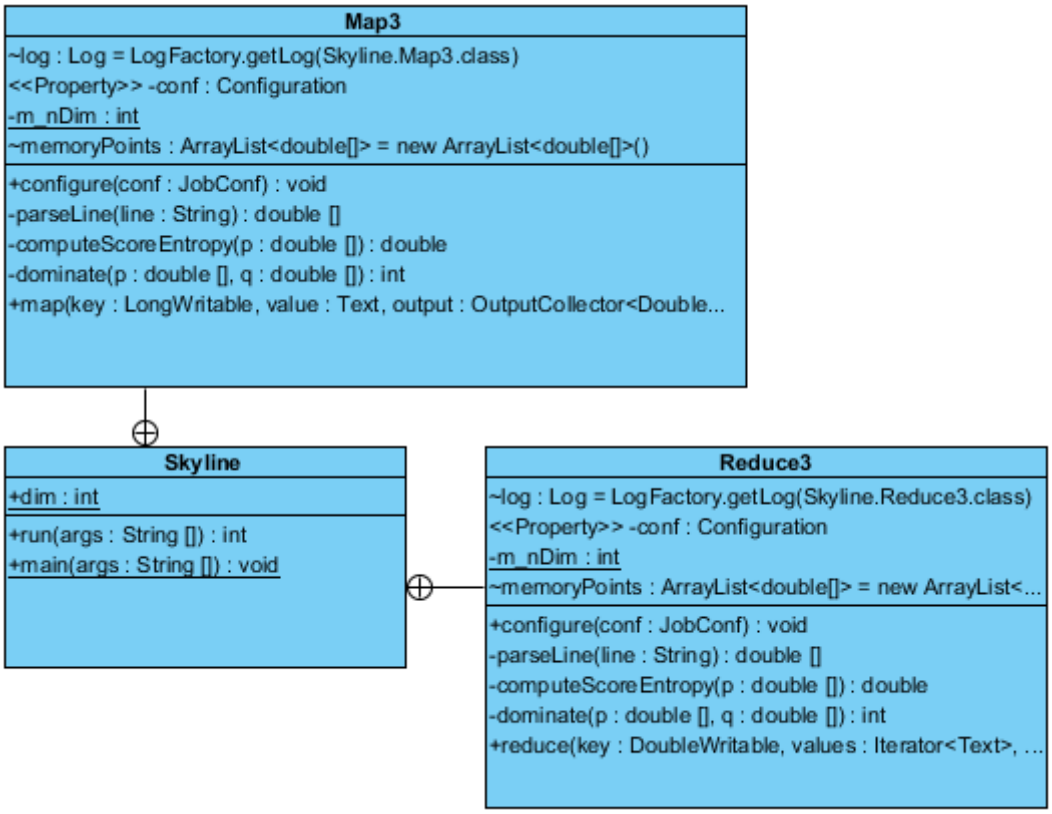
Η κλάση Hyperplane.java αποτελείται κι αυτή από τρεις κλάσεις που υλοποιούν τα interfaces Mapper, Partitioner και Reducer. Η μέθοδος map() της κλάσης που υλοποιεί το interface Mapper, διαβάζει γραμμή γραμμή το αρχείο εισόδου και στέλνει τα σημεία ως κλειδιά στην κλάση που ουσιαστικά υλοποιεί το διαμερισμό του χώρου.

Η βασική μέθοδος που υλοποιεί το διαμοιρασμό του χώρου δέχεται ως κλειδί την κάθε γραμμή του αρχείου που στέλνει η μέθοδος map(). Το κλειδί είναι της μορφής (id double1 double2 double3) και χωρίζεται σε κομμάτια με τη χρήση ενός StringTokenizer. Στη συνέχεια, κάθε συντεταγμένη των σημείων αποθηκεύεται σ' ένα πίνακα ενώ το id των σημείων περνάει σ' ένα string αφού δεν παίζει ρόλο στον υπολογισμό των διαμερίσεων. Έπειτα, για κάθε σημείο του πίνακα καλείται η μέθοδος convertCoord(), η οποία με τη βοήθεια ενός μαθηματικού τύπου όπως περιγράφεται στο κεφάλαιο 2, υπολογίζει τις νέες συντεταγμένες προβάλλοντάς τες σε μια ευθεία. Αφού ολοκληρωθεί η μετατροπή των συντεταγμένων των σημείων, τότε καλείται η μέθοδος mapPoint2Partition(), η οποία συγκρίνει τις τιμές των συντεταγμένων με τα όρια των διαμερίσεων και τοποθετεί τα σημεία στις κατάλληλες διαμερίσεις.

Στην μέθοδο mapPoint2Partition() υπολογίζονται τα όρια των διαμερίσεων και χωρίζεται με στατικό τρόπο ο χώρος. Για οριστούν τα όρια των διαμερίσεων, ο χώρος που σχηματίζεται μεταξύ των τριών αξόνων χωρίστηκε σε 8 διαμερίσεις. Μετά από δοκιμές οι άξονες χωρίστηκαν με στατικό τρόπο σε τέτοια σημεία έτσι ώστε όσες διαμερίσεις περιέχουν σημεία, αυτά να είναι σχεδόν ισόποσα μοιρασμένα. Συγκεκριμένα, στον άξονα x το όριο είναι στο 0.3, στον άξονα y το όριο είναι στο 0.4 και στον άξονα z το όριο είναι 0.3 αφού πρέπει να ισχύει η εξής συνθήκη x+y+z = 1.

Αφού λοιπόν έχουν οριστεί οι διαμερίσεις και τα σημεία που περιέχουν, η μέθοδος reduce() λαμβάνει τα αποτελέσματα της κλάσης που υλοποιεί το interface Partitioner και δημιουργεί το μονοπάτι εξόδου στο HDFS. Κάθε reduce task ή reducer δημιουργεί στο μονοπάτι εξόδου ένα αρχείο και γράφει σε αυτό τα σημεία που του αντιστοιχούν.

5.3 Υπολογισμός skyline σημείων



Εικόνα 18: Διάγραμμα κλάσης Skyline.java

Η κλάση Skyline.java είναι αυτή που υπολογίζει αρχικά τα τοπικά skyline σημεία και μετά τα τελικά skyline σημεία. Η κλάση αυτή περιλαμβάνει δυο κλάσεις που υλοποιούν τα interfaces Mapper και Reducer.

Η μέθοδος map() διαβάζει γραμμή γραμμή το αρχείο εισόδου. Κάθε γραμμή που είναι της μορφής (id double1 double2 double3) σπάει σε κομμάτια με τη χρήση ενός StringTokenizer με σκοπό να περαστούν σ' ένα string μόνο οι double αριθμοί, οι οποίοι συμβολίζουν τις συντεταγμένες. Στη συνέχεια, καλείται η μέθοδος parseLine() η οποία διαβάσει το string με τις συντεταγμένες και τις καταχωρεί μια μια σ' ένα πίνακα. Έπειτα, για κάθε σημείο καλείται η μέθοδος computeScoreEntropy(), η οποία υπολογίζει την εντροπία. Όσο πιο μικρή είναι η τιμή της εντροπίας ενός σημείου, τόσο πιο πιθανό είναι αυτό το σημείο να είναι ένα skyline σημείο. Σε αυτό το σημείο καλείται η μέθοδος dominate(), η οποία εξετάζει ποιά σημεία επικρατούν έναντι άλλων. Πιο αναλυτικά, το πρώτο σημείο του αρχείου τοποθετείται σ' έναν πίνακα και για κάθε επόμενο σημείο που διαβάζεται, καλείται η μέθοδος dominate() και συγκρίνει το τρέχων σημείο με κάθε ένα σημείο του πίνακα. Αν το τρέχων σημείο επικρατεί έναντι κάποιου σημείου του πίνακα, τότε αυτό βγαίνει από τον πίνακα και

τοποθετείται στη θέση του το τρέχων σημείο. Αν κάποιο σημείο του πίνακα επικρατεί του τρέχοντος, τότε αμέσως αυτό απορρίπτεται. Από αυτή τη διαδικασία προκύπτουν τα skyline σημεία για κάθε διαμέριση, τα οποία περνάνε ως είσοδος στην κλάση που υλοποιεί τη φάση reducing.

Η μέθοδος reduce() δέχεται ως είσοδο τα τοπικά skyline σημεία όλων των διαμερίσεων και ακολουθεί την ίδια διαδικασία με την μέθοδο map() για να καταλήξει στα τελικά skyline σημεία. Δηλαδή για κάθε σημείο καλεί την μέθοδο dominate() και συγκρίνει το τρέχων σημείο με όλα τα υπόλοιπα. Αφού τελειώσει η σύγκριση των σημείων και προκύψουν τα τελικά skyline σημεία, τότε αυτά γράφονται σ' ένα αρχείο μέσα στο μονοπάτι εξόδου του HDFS.

6. ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ

Για την αξιολόγηση του κώδικα που αναπτύχθηκε χρησιμοποιήθηκε η μέθοδος της πειραματικής έρευνας. Το ουσιώδες χαρακτηριστικό της πειραματικής έρευνας είναι ότι οι εκάστοτε ερευνητές ηθελημένα ελέγχουν και χειρίζονται τις συνθήκες οι οποίες καθορίζουν τα γεγονότα για τα οποία ενδιαφέρονται. Στην απλούστερη μορφή του ένα πείραμα εμπεριέχει την αλλαγή στην τιμή μιας μεταβλητής – η οποία καλείται ανεξάρτητη μεταβλητή- και την παρατήρηση της επίδρασης αυτής της αλλαγής πάνω σε μια άλλη μεταβλητή – η οποία καλείται εξαρτημένη μεταβλητή [31].

Στην πειραματική έρευνα της παρούσας εργασίας οι ανεξάρτητες μεταβλητές, των οποίων αλλάζουν οι τιμές, είναι ο αριθμός των σημείων, ο αριθμός των διαμερίσεις και η κατανομή των σημείων ενώ οι εξαρτημένες μεταβλητές είναι ο χρόνος εκτέλεσης και ο αριθμός των δεδομένων που γίνονται shuffle.

Αναλυτικότερα, σε κάθε σενάριο και εκτέλεση του κώδικα μετριέται ο χρόνος εκτέλεσής του (σε msec) και τα δεδομένα (σε bytes) που γίνονται shuffle. Πιο συγκεκριμένα, εκτός από το συνολικό χρόνο της κάθε μεθόδου, μετράμε ξεχωριστά το χρόνο που χρειάζεται για να ολοκληρωθεί ο διαμερισμός του χώρου σε κάθε περίπτωση και το χρόνο που χρειάζεται για να υπολογιστούν τα skyline σημεία. Στα διαγράμματα μέτρησης του χρόνου που ακολουθούν κάθε μπάρα συμβολίζει μια τεχνική διαμερισμού του χώρου και υπολογισμού των skyline σημείων και χωρίζεται σε δύο μέρη. Τα δύο διαφορετικά χρώματα αναπαριστούν το χρόνο που χρειάζεται για το διαμερισμό του χώρου και τον υπολογισμό των skyline σημείων αντίστοιχα. Στην περίπτωση κατά την οποία μετριούνται τα δεδομένα που γίνονται shuffle, κάθε μπάρα συμβολίζει την μέθοδο διαμερισμού του χώρου και τον υπολογισμό των skyline σημείων.

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, για την εκτέλεση του κώδικα χρησιμοποιήθηκε Virtual Machine σε Linux, από την Cloudera [30], στο οποίο υπάρχει ήδη εγκατεστημένο το hadoop-0.20. Για τη χρήση του virtual machine είναι αναγκαία η γνώση των βασικών εντολών Linux. Επίσης, κατά τη διάρκεια εκτέλεσης του κώδικα, διαπιστώθηκε ότι ήταν αναγκαία η αύξηση της μνήμης του virtual

machine σε 2GB. Η αύξηση της μνήμης έγινε μέσω των επιλογών που προσφέρει το περιβάλλον του virtual machine.

Μετά τη συγγραφή του κώδικα στο Eclipse, χρειάστηκε να γίνει export σε μορφή εκτελέσιμου αρχείου (jar) και να μεταφερθεί στο virtual machine για να εκτελεστεί. Για την μεταφορά του αρχείου από το περιβάλλον ανάπτυξης του κώδικα στο virtual machine χρησιμοποιήθηκε η εφαρμογή WinSCP. Με την ίδια εφαρμογή μεταφέρθηκαν στο virtual machine όλα τα απαραίτητα αρχεία για την εκτέλεση του κώδικα.

Αρχικά, μεταφέρθηκε σε συγκεκριμένο τοπικό φάκελο του virtual machine, το εκτελέσιμο αρχείο (jar), το αρχείο settings.ini και τα αρχεία εισόδου προς επεξεργασία. Για την εκτέλεση οποιασδήποτε ενέργειας, η οποία έχει να κάνει με το HDFS ή με την εκτέλεση κώδικα, πρέπει πρώτα να μεταφερθούμε στον υποφάκελο hadoop-0.20, όπου βρίσκονται τα αρχεία του hadoop και αυτό γίνεται από τη γραμμή εντολών.

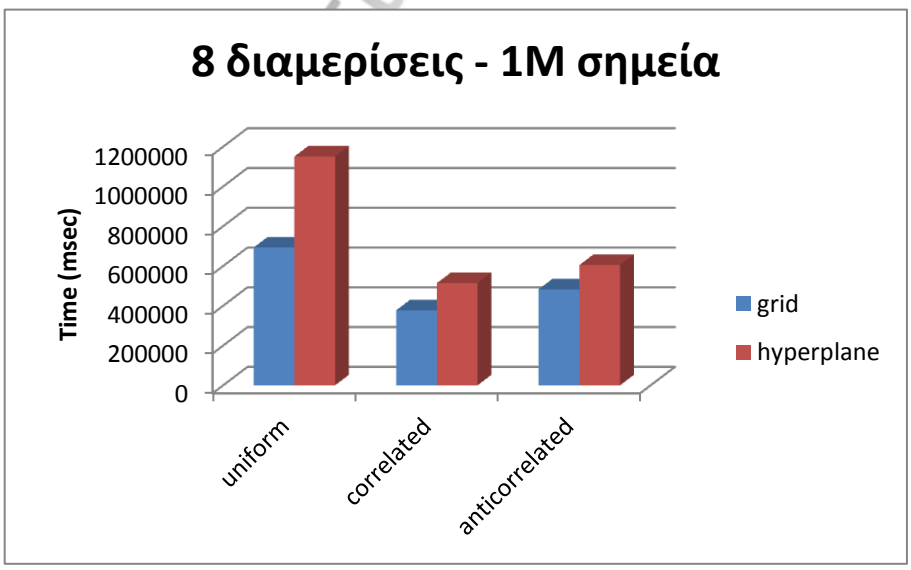
Στη συνέχεια, εφόσον έχουμε μεταφερθεί στον υποφάκελο hadoop-0.20, από τη γραμμή εντολών του virtual machine μεταφέρουμε τα αρχεία εισόδου από τον τοπικό φάκελο στο HDFS ώστε να μπορέσουν να διαβαστούν από τον κώδικα. Για κάθε κατανομή δημιουργήθηκαν αρχεία με 1.000.000 σημεία και 10.000.000 σημεία, δηλαδή συνολικά δημιουργήθηκαν 6 αρχεία με ονόματα uniform.txt, correlated.txt, anticorrelated.txt και uniform10M.txt, correlated10M.txt, anticorrelated10M.txt αντίστοιχα τα οποία θα χρησιμοποιηθούν σε ανάλογα σενάρια εκτέλεσης του κώδικα.

Στον παρακάτω πίνακα, φαίνονται αναλυτικά τα σενάρια εκτέλεσης του κώδικα, με τις συγκρινόμενες μεθόδους που εκτελούνται κάθε φορά, τα αρχεία προς επεξεργασία, τον αριθμό των σημείων που περιέχουν, την κατανομή που ακολουθούν τα σημεία και τον αριθμό των διαμερίσεων που θα δημιουργηθούν.

	# Σημείων	#Διαμερίσεων	Κατανομή	Αρχείο προς επεξεργασία	Συγκρινό-μενες τεχνικές
Σενάριο 1	1.000.000	8	Uniform Correlated Anticorrelated	Uniform.txt Correlated.txt Anticorrelated.txt	Grid Hyperplane
Σενάριο 2	10.000.000	8	Uniform Correlated Anticorrelated	Uniform10M.txt Correlated10M.txt Anticorrelated10M.txt	Grid Hyperplane
Σενάριο 3	1.000.000	4	Uniform Correlated Anticorrelated	Uniform.txt Correlated.txt Anticorrelated.txt	Grid Angle

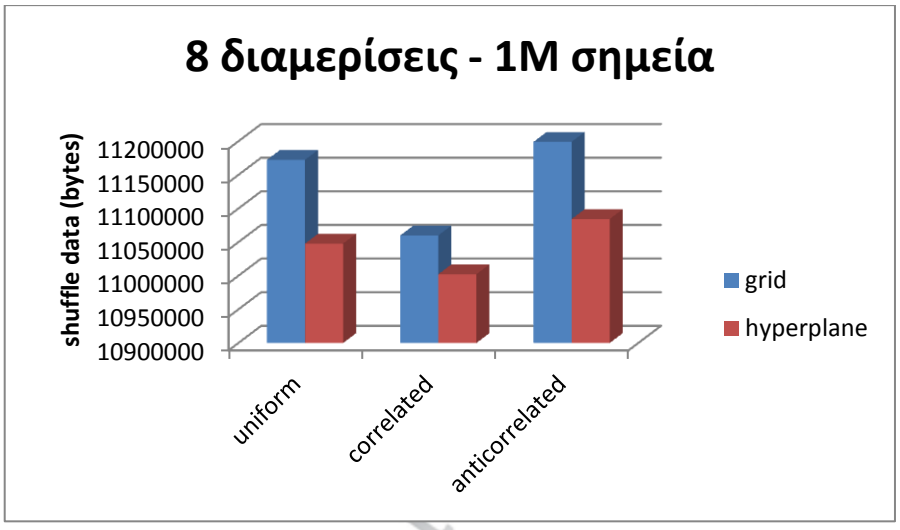
6.1 Σενάριο 1: Αρχείο εισόδου με 1.000.000 σημεία και 8 διαμερίσεις

Στο πρώτο σενάριο ο κώδικας εκτελέστηκε με αρχεία εισόδου με 1.000.000 σημεία τριών διαστάσεων για κάθε κατανομή, τα οποία πρέπει να μοιραστούν σε 8 διαμερίσεις. Στο παρακάτω συγκεντρωτικό γράφημα παρατηρούμε ότι σε όλες τις κατανομές η μέθοδος Hyperplane διαρκεί περισσότερο από την Grid ενώ η μεγαλύτερη διαφορά χρόνου παρατηρείται στη uniform κατανομή.



Εικόνα 19: Χρόνος εκτέλεσης για 8 διαμερίσεις με αρχείο εισόδου με 1M σημεία

Στο επόμενο διάγραμμα αναπαρίστανται τα δεδομένα που γίνονται shuffle σε κάθε μέθοδο και ανά κατανομή. Παρατηρούμε ότι σε όλες τις κατανομές τα δεδομένα που γίνονται shuffle είναι περισσότερα στη Grid μέθοδο απ' ότι στην Hyperplane. Η μεγαλύτερη διαφορά των shuffle data μεταξύ των μεθόδων διαμοιρασμού του χώρου παρατηρείται στη uniform και την anticorrelated κατανομή, ενώ η μικρότερη στην correlated κατανομή.

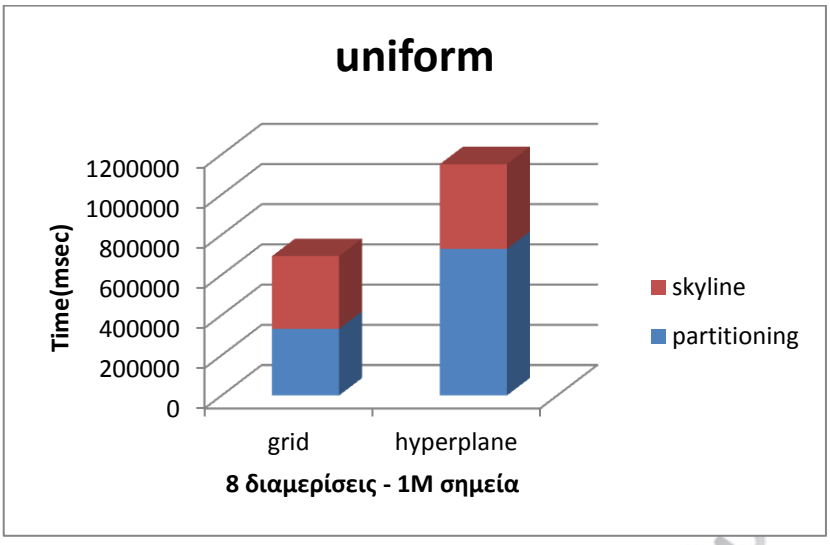


Εικόνα 20:Shuffle data ανά κατανομή και ανά μέθοδο διαμερισμού για 8 διαμερίσεις και 1M σημεία

Ακολουθούν αναλυτικά διαγράμματα για κάθε κατανομή όπου καταγράφεται ξεχωριστά ο χρόνος για την ολοκλήρωση του διαμερισμού του χώρου και τον υπολογισμό των skyline σημείων.

6.1.1 Uniform κατανομή

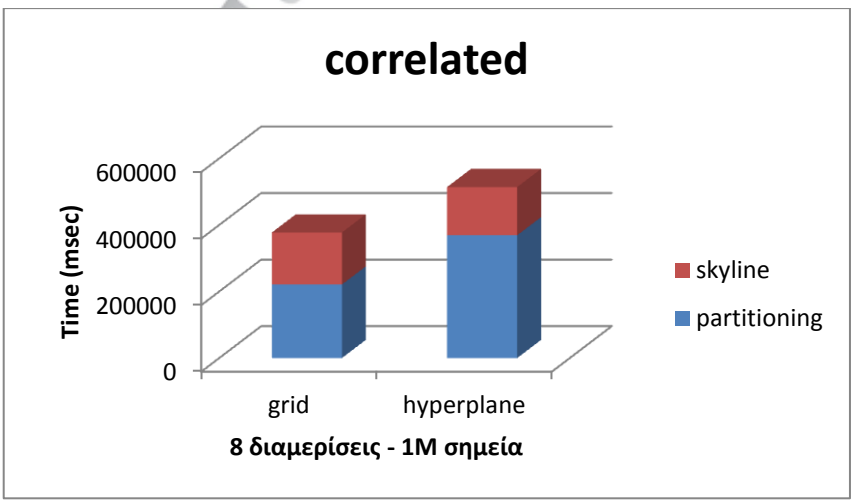
Στη uniform κατανομή, ο κώδικας εκτελέστηκε με αρχείο εισόδου το uniform.txt. Όπως παρατηρούμε και στο διάγραμμα παρακάτω η μεγαλύτερη διαφορά χρόνου στις δυο μεθόδους καταγράφεται στη φάση που πραγματοποιείται ο διαμερισμός του χώρου, ο οποίος διαρκεί περισσότερο στην hyperplane μέθοδο. Η διάρκεια υπολογισμού των skyline σημείων είναι σχεδόν ίδια και στις δύο μεθόδους.



Εικόνα 21:Χρόνος εκτέλεσης σε uniform κατανομή ανά μέθοδο και job

6.1.2 Correlated κατανομή

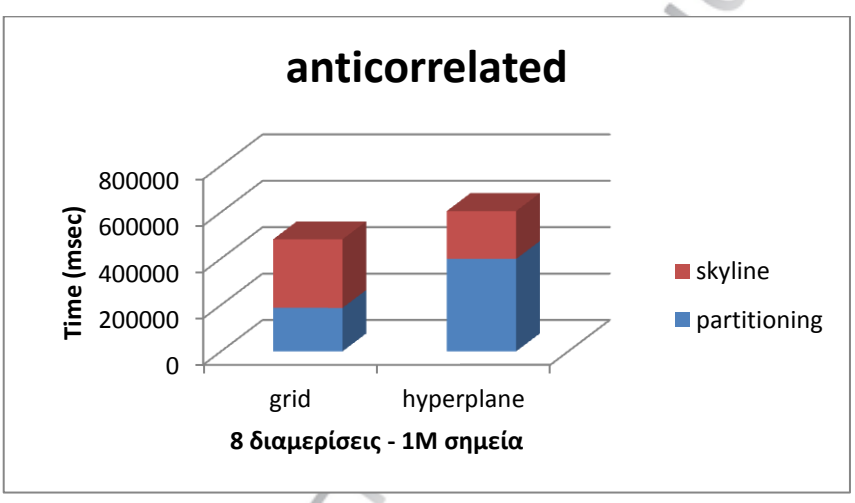
Στην correlated κατανομή, ο κώδικας εκτελέστηκε με αρχείο εισόδου το correlated.txt . Στο διάγραμμα φαίνεται ότι η διαφορά χρόνου καταγράφεται και πάλι στη φάση όπου πραγματοποιείται ο διαμοιρασμός του χώρου στην hyperplane μέθοδο ενώ ο χρόνος υπολογισμού των skyline σημείων είναι σχεδόν ίσος και στις δυο μεθόδους.



Εικόνα 22:Χρόνος εκτέλεσης για correlated κατανομή ανά μέθοδο και job

6.1.3 Anticorrelated κατανομή

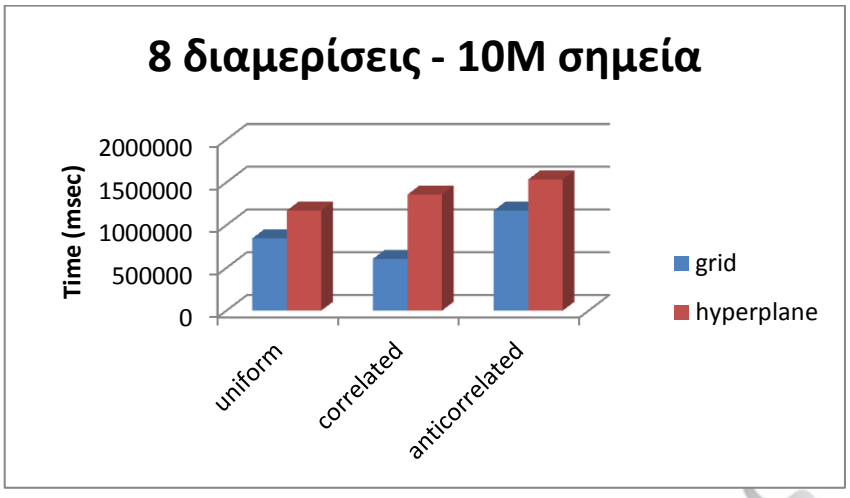
Στην anticorrelated κατανομή, ο κώδικας εκτελέστηκε με αρχείο εισόδου το anticorrelated.txt. Όπως και στις προηγούμενες κατανομές βλέπουμε ότι υπάρχει μια διαφορά στο χρόνο εκτέλεσης του διαμερισμού του χώρου με τη διαφορά ότι εδώ υπάρχει και μια μικρή διαφορά στο χρόνο υπολογισμού των skyline σημείων. Συγκεκριμένα, στη grid μέθοδο είναι μεγαλύτερος ο χρόνος υπολογισμού των skyline σημείων ενώ στη hyperplane μέθοδο είναι μεγαλύτερος ο χρόνος που χρειάζεται για το διαμερισμό του χώρου.



Εικόνα 23:Χρόνος εκτέλεσης σε anticorrelated κατανομή ανά μέθοδο και ανά job

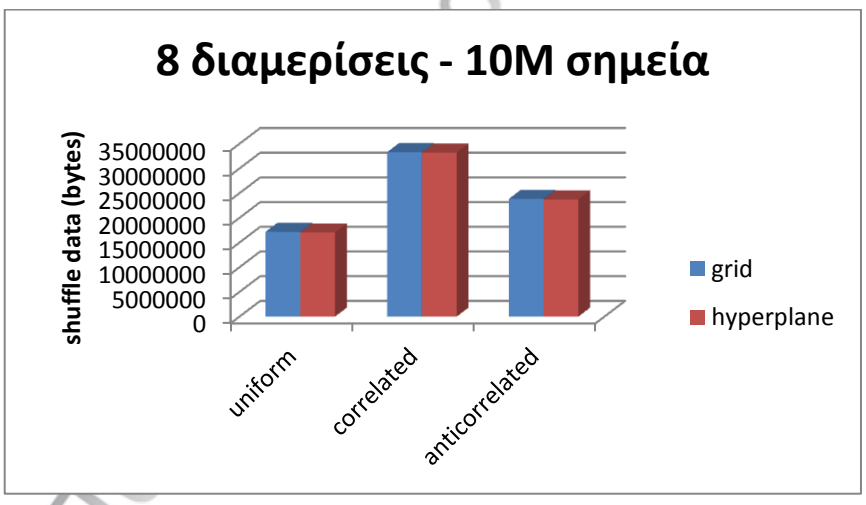
6.2 Σενάριο 2: Αρχείο εισόδου με 10.000.000 σημεία και 8 διαμερίσεις

Στο δεύτερο σενάριο, ο κώδικας εκτελέστηκε με αρχεία εισόδου με 10.000.000 σημεία τριών διαστάσεων για κάθε κατανομή, τα οποία πρέπει να χωριστούν σε 8 διαμερίσεις. Παρατηρείται και εδώ ότι η hyperplane μέθοδος χρειάζεται περισσότερο χρόνο για την ολοκλήρωσή της. Η διαφορά χρόνου μεταξύ των δυο μεθόδων είναι περισσότερο αισθητή στην correlated κατανομή.



Εικόνα 24:Χρόνος εκτέλεσης ανά κατανομή και μέθοδο με 8 διαμερίσεις και αρχείο εισόδου 10M σημεία

Στο παρακάτω διάγραμμα βλέπουμε ότι τα δεδομένα που γίνονται shuffle σε κάθε μέθοδο και ανά κατανομή είναι σχεδόν ισόποσα. Βέβαια, στη σύγκριση μεταξύ των κατανομών παρατηρείται ότι στην correlated κατανομή τα δεδομένα που γίνονται shuffle είναι περισσότερα.

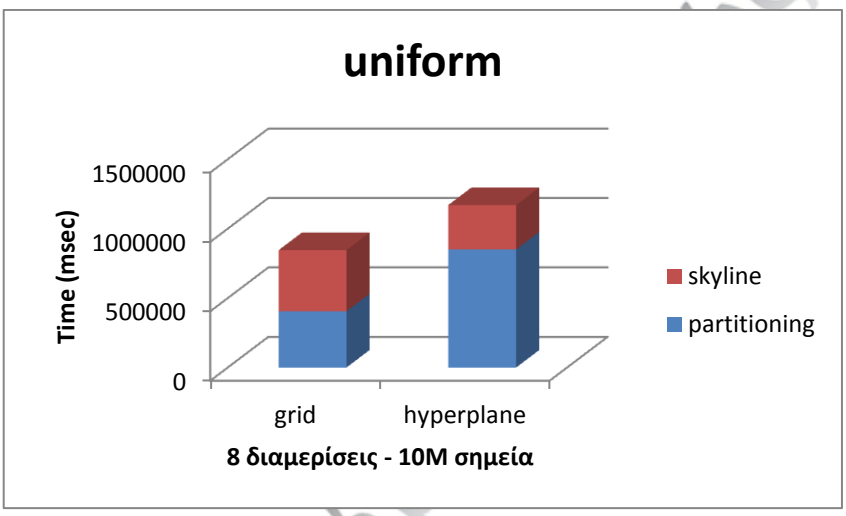


Εικόνα 25:Shuffle data ανά κατανομή και μέθοδο διαμερισμού με 8 διαμερίσεις και 10M σημεία

Παρακάτω ακολουθούν αναλυτικά γραφήματα για κάθε κατανομή όσο αφορά στην μέτρηση του χρόνου.

6.2.1 Uniform κατανομή

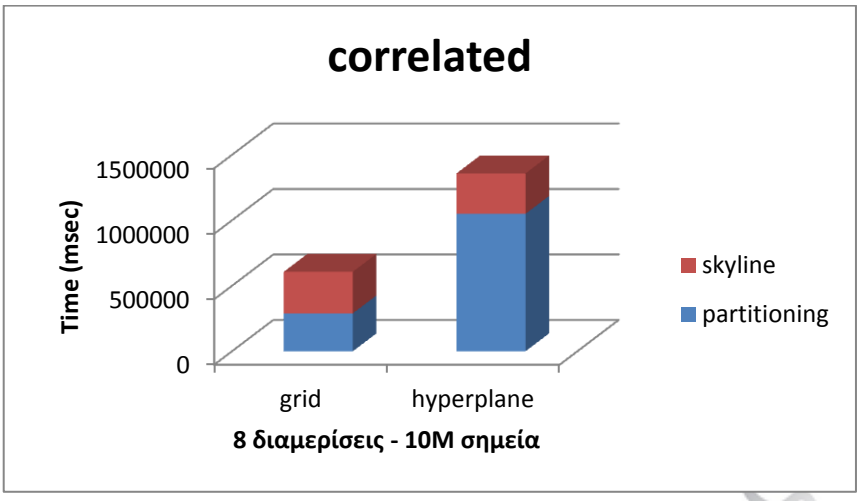
Στη uniform κατανομή, ο κώδικας εκτελέστηκε με αρχείο εισόδου το uniform10M.txt. Εδώ παρατηρούμε ότι ο χρόνος που διαρκεί ο διαμερισμός του χώρου στη hyperplane μέθοδο είναι μεγαλύτερος απ' ότι στην grid αλλά διαρκεί λιγότερο ο χρόνος που απαιτείται για τον υπολογισμό των skyline σημείων.



Εικόνα 26:Χρόνος εκτέλεσης σε uniform κατανομή ανά μέθοδο και job

6.2.2 Correlated κατανομή

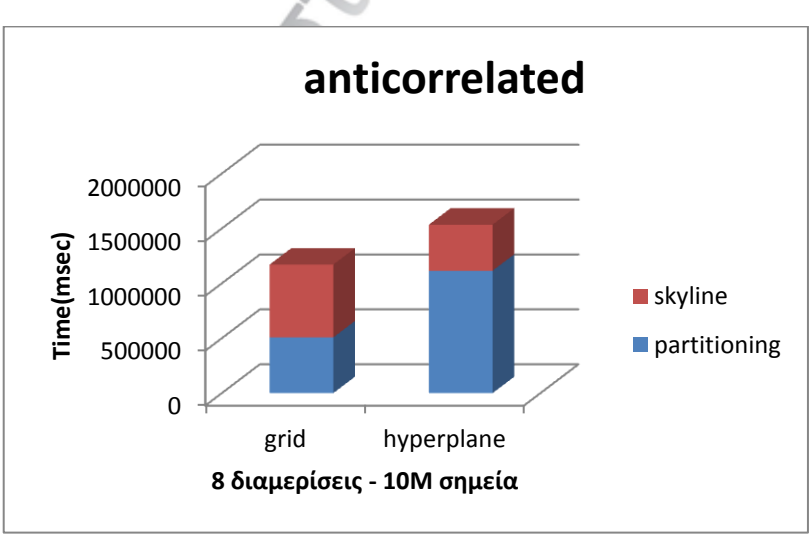
Στη correlated κατανομή, ο κώδικας εκτελέστηκε με αρχείο εισόδου το correlated10M.txt. Σε αυτή την περίπτωση, παρατηρείται μεγάλη διαφορά στο χρόνο εκτέλεσης του διαμοιρασμού του χώρου ενώ η διαφορά στο χρόνο υπολογισμού των skyline σημείων είναι αμελητέα.



Εικόνα 27:Χρόνος εκτέλεσης σε correlated κατανομή ανά μέθοδο και job

6.2.3 Anticorrelated κατανομή

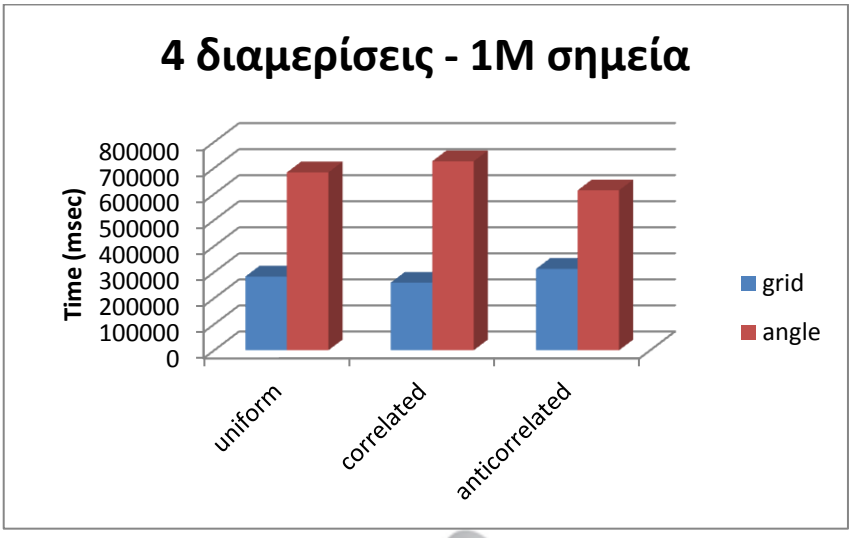
Στην anticorrelated κατανομή, ο κώδικας εκτελέστηκε με αρχείο εισόδου το anticorrelated10M.txt. Εδώ βλέπουμε ότι η φάση του διαμοιρασμού του χώρου διαρκεί περισσότερο στη hyperplane μέθοδο, ενώ διαρκεί πολύ λιγότερο ο υπολογισμός των skyline σημείων.



Εικόνα 28:Χρόνος εκτέλεσης σε anticorrelated κατανομή ανά μεθοδο και job

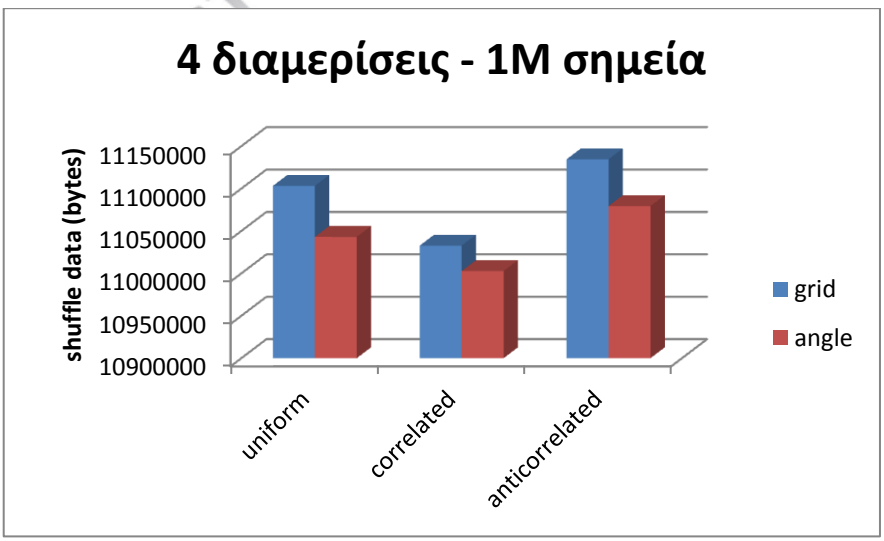
6.3 Σενάριο 3: Αρχείο εισόδου με 1.000.000 σημεία και 4 διαμερίσεις

Στο τρίτο σενάριο, ο κώδικας εκτελείται με αρχεία εισόδου με 1.000.000 σημεία τριών διαστάσεων για κάθε κατανομή, τα οποία πρέπει να μοιραστούν σε 4 διαμερίσεις. Παρατηρούμε ότι σε όλες τις κατανομές η Grid μέθοδος διαρκεί αισθητά λιγότερο χρόνο σε σχέση με την Angle μέθοδο.



Εικόνα 29:Χρόνος εκτέλεσης ανά κατανομή και μέθοδο διαμερισμού με 4 διαμερίσεις και 1M σημεία

Επίσης, σε αυτή την περίπτωση παρατηρούμε ότι στην Grid μέθοδο γίνονται περισσότερα δεδομένα shuffle σε σχέση με την Angle σε όλες τις κατανομές με την μικρότερη διαφορά μεταξύ των μεθόδων να καταγράφεται στην correlated κατανομή.

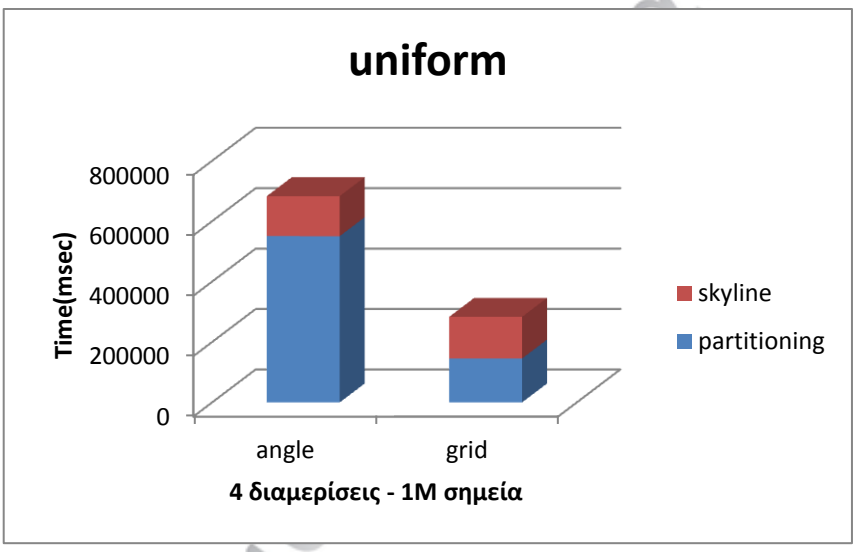


Εικόνα 30: Shuffle data ανά κατανομή και μέθοδο διαμερισμού με 4 διαμερίσεις και 1M σημεία

Ακολουθούν αναλυτικά διαγράμματα για κάθε κατανομή όπου απεικονίζεται ο χρόνος εκτέλεσης του κώδικα ξεχωριστά για κάθε μέθοδο.

6.3.1 Uniform κατανομή

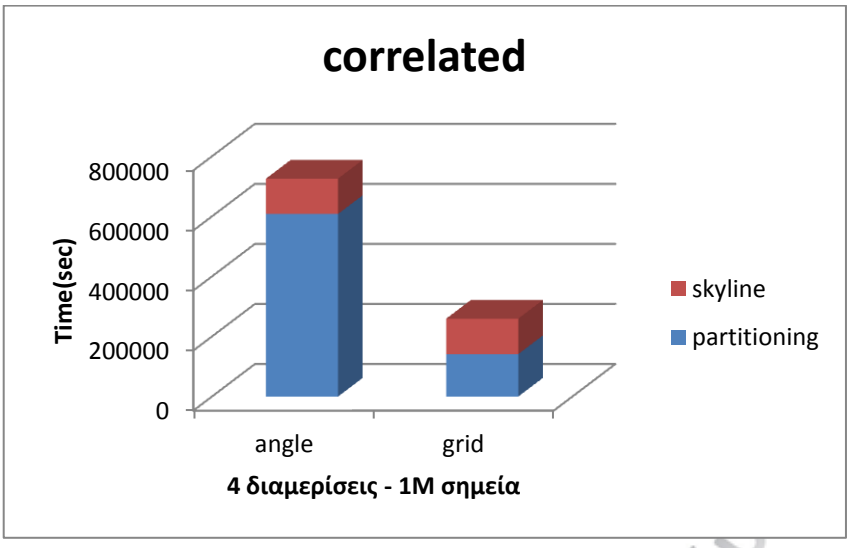
Στη uniform κατανομή, ως αρχείο εισόδου δόθηκε το uniform.txt . Όπως και στις προηγούμενες περιπτώσεις, έτσι και εδώ παρατηρούμε ότι η σημαντική διαφορά στο χρόνο εκτέλεσης του πειράματος καταγράφεται κατά τη φάση του διαμερισμού του χώρου ενώ ο χρόνος που απαιτείται για τον υπολογισμό των skyline σημείων είναι σχεδόν ίδιος και στις δυο μεθόδους.



Εικόνα 31:Χρόνος εκτέλεσης σε uniform κατανομή ανά μέθοδο διαμερισμού και job

6.3.2 Correlated κατανομή

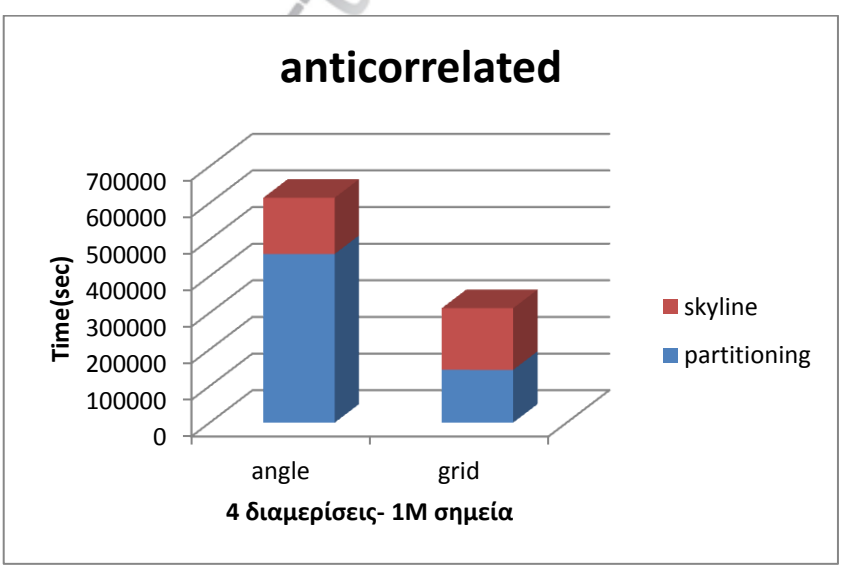
Στην correlated κατανομή, ως αρχείο εισόδου δόθηκε το correlated.txt. Σύμφωνα με τις μετρήσεις, παρατηρείται και εδώ μια διαφορά στο χρόνο που απαιτείται για την ολοκλήρωση της φάσης του διαμοιρασμού του χώρου. Στην angle μέθοδο, ο απαιτούμενος χρόνος για το διαμερισμό του χώρου είναι μεγαλύτερος.



Εικόνα 32:Χρόνος εκτέλεσης σε correlated κατανομή ανά μέθοδο διαμερισμού και job

6.3.3 Anticorrelated κατανομή

Στην anticorrelated κατανομή ως αρχείο εισόδου δόθηκε το anticorrelated.txt. Όπως και στις προηγούμενες κατανομές, έτσι και εδώ βλέπουμε στην Angle μέθοδο, η φάση του διαμερισμού του χώρου διαρκεί περισσότερο ενώ ο χρόνος για τον υπολογισμό των skyline είναι σχεδόν ο ίδιος και στις δυο μεθόδους.



Εικόνα 33:Χρόνος εκτέλεσης σε anticorrelated κατανομή ανά μέθοδο διαμερισμού και job

7. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα εργασία αρχικά παρουσιάστηκαν και περιγράφηκαν τα Skyline Queries και το Hadoop – MapReduce. Περιγράφηκε αναλυτικά η αρχιτεκτονική τους, ο τρόπος λειτουργίας τους καθώς και η χρησιμότητά τους.

Στη συνέχεια παρουσιάστηκαν προγενέστερες εργασίες που είχαν ασχοληθεί είτε με Skyline Επερωτήσεις είτε με το MapReduce είτε και με τις δυο τεχνολογίες. Σημαντικό βάρος δόθηκε στην περιγραφή των τριών τεχνικών για το διαμοιρασμό του χώρου αλλά και στην μέθοδο που χρησιμοποιείται για τον παράλληλο υπολογισμό των skyline σημείων.

Ακολούθως τέθηκαν οι στόχοι της εργασίας και ακολούθησε η περιγραφή της σχεδίασης της εργασίας σε επίπεδο αρχιτεκτονικής. Στη συνέχεια, περιγράφηκε η υλοποίηση του κώδικα σε επίπεδο κλάσεων, αναλύοντας τη χρησιμότητα της κάθε κλάσης.

Τέλος, παρουσιάστηκε η πειραματική μελέτη μέσω γραφημάτων στα οποία αναπαρίστανται οι μετρήσεις που έγιναν. Από τα γραφήματα προκύπτουν τα εξής συμπεράσματα:

Κατά την επεξεργασία αρχείου με 1.000.000 σημεία και τη δημιουργία 8 διαμερίσεων, η μέθοδος hyperplane διαρκεί περισσότερο από την grid σε όλες τις κατανομές με την μεγαλύτερη διαφορά χρόνου ανάμεσα στις δυο μεθόδους να καταγράφεται στην uniform κατανομή. Επίσης, παρατηρήθηκε ότι η φάση του διαμοιρασμού του χώρου στην hyperplane μέθοδο διαρκεί περισσότερο από την αντίστοιχη φάση στην grid μέθοδο σε όλες τις κατανομές. Ωστόσο, στην anticorrelated κατανομή η φάση του υπολογισμού των skyline σημείων στην grid μέθοδο διαρκεί περισσότερο απ’ ότι στην hyperplane.

Τα shuffle data είναι περισσότερα στην grid μέθοδο σε όλες τις κατανομές ενώ η μικρότερη διαφορά μεταξύ της grid και της hyperplane μεθόδου παρατηρείται στην correlated κατανομή.

Κατά την επεξεργασία αρχείου με 10.000.000 σημεία και τη δημιουργία 8 διαμερίσεων, η μέθοδος hyperplane διαρκεί περισσότερο σε όλες τις κατανομές και η μεγαλύτερη διαφορά χρόνου μεταξύ των μεθόδων grid και hyperplane καταγράφεται στην correlated κατανομή. Επίσης, σε όλες τις κατανομές η φάση του διαμοιρασμού του χώρου στην hyperplane μέθοδο διαρκεί περισσότερο απ' ότι η αντίστοιχη φάση στην grid μέθοδο ενώ στην correlated κατανομή, η φάση του υπολογισμού των skyline σημείων διαρκεί ελαφρώς περισσότερο στην grid μέθοδο.

Σε όλες τις κατανομές και οι δυο μέθοδοι έχουν σχεδόν τα ίδια shuffle data. Τα περισσότερα shuffle data παρατηρούνται στην correlated κατανομή, ενώ τα λιγότερα στη uniform.

Κατά την επεξεργασία αρχείου με 1.000.000 σημεία και τη δημιουργία 4 διαμερίσεων, η μέθοδος angle διαρκεί περισσότερο από την grid σε όλες τις κατανομές. Επίσης, σε όλες τις κατανομές, η φάση του διαμερισμού του χώρου στην angle μέθοδο διαρκεί περισσότερο απ' ότι η αντίστοιχη φάση στην grid μέθοδο, ενώ η φάση υπολογισμού των skyline σημείων διαρκεί σχεδόν το ίδιο και στις δυο μεθόδους.

Τα shuffle data είναι περισσότερα στην grid μέθοδο απ' ότι στην angle, σε όλες τις κατανομές και η μικρότερη διαφορά μεταξύ των δυο μεθόδων παρατηρείται στην correlated κατανομή.

Άρα, συγκεντρώνοντας τα στοιχεία της πειραματικής μελέτης βλέπουμε ότι αν κρατήσουμε σταθερό τον αριθμό των διαμερίσεων και αυξήσουμε τον αριθμό των σημείων προς επεξεργασία, αυξάνεται και ο χρόνος εκτέλεσης τόσο στην grid όσο και στην hyperplane μέθοδο σε όλες τις κατανομές. Το ίδιο αυξάνονται και τα shuffle data. Στην περίπτωση, που ο αριθμός των σημείων προς επεξεργασία μείνει σταθερός και μειώσουμε τον αριθμό των διαμερίσεων, ο χρόνος εκτέλεσης της grid μεθόδου μειώνεται κατά πολύ. Επίσης, παρατηρείται και μια πολύ μικρή μείωση στα shuffle data.

Κατά τη διάρκεια ανάπτυξης και εκτέλεσης του κώδικα υπήρξαν κάποιοι περιορισμοί. Ένας από αυτούς ήταν η εκτέλεση του κώδικα σε δύο jobs, πράγμα το οποίο αυξάνει το χρόνο εκτέλεσής του. Σημαντικοί περιορισμοί υπήρξαν στην μέθοδο διαμοιρασμού του χώρου μέσω hyperplane προβολών. Συγκεκριμένα, η κλάση δεν

μπορούσε να υλοποιηθεί με διαφορετικό αριθμό διαστάσεων μεγαλύτερο από 3, όπως και με διαφορετικό αριθμό διαμερίσεων.

Σαν μελλοντική έρευνα, θα μπορούσε να προταθεί η προσπάθεια εκτέλεσης του κώδικα με τη χρήση ενός job, για να διαπιστωθεί αν υπάρχει κάποια διαφορά στο χρόνο εκτέλεσης. Επίσης, ο διαμοιρασμός του χώρου και των σημείων θα μπορούσε να γίνει με δυναμικό τρόπο. Ακόμα, η τεχνική του διαμοιρασμού του χώρου μέσω hyperplane προβολών θα μπορούσε εκτός από το να εκτελείται δυναμικά, να εκτελείται και για περισσότερες από 3 διαστάσεις για διαφορετικό αριθμό διαμερίσεων.

Συνοπτικά, οι προτάσεις για τη βελτίωση του κώδικα και για μελλοντική έρευνα είναι οι εξής:

- Εκτέλεση του κώδικα με ένα job
- Δυναμικός διαμερισμός του χώρου για όλες τις τεχνικές
- Εκτέλεση του διαμερισμού του χώρου μέσω hyperplane προβολών για περισσότερες διαστάσεις και διαφορετικό αριθμό διαμερίσεων.

8. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Bo Dong, Qinghua Zheng, Jie Yang, Haifei Li, Mu Qiao: **An E-learning Ecosystem Based on Cloud Computing Infrastructure.** ICALT 2009: 125-127

[2] Yasuhiko Morimoto; Mohammad Shamsul Arefin; Mohammad Anisuzzaman Siddique: **Agent - Based Convex Skyline Set Query for Cloud Computing Environment**. (2012)

[3] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, Bongki Moon: **Parallel data processing with MapReduce: a survey.** SIGMOD Record 40(4): 11-20 (2011)

[4] Jörg Schad, Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz: **Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance.**PVLDB 3(1): 460-471 (2010)

[5] «Power of Cloud,» [Ηλεκτρονικό]. Available: http://transformcloud.blogspot.gr. [Πρόσβαση 2013].

[6] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis: **Angle-based space partitioning for efficient parallel skyline computation.** SIGMOD Conference 2008: 227-238

[7] Chuck Lam: **Hadoop in Action**. Manning Publications (2011)

[8] Jeffrey Dean, Sanjay Ghemawat: **MapReduce: a flexible data processing tool.** Commun. ACM 53(1): 72-77 (2010)

[9] Tom White: Hadoop The Definitive Guide.O'Reilly | Yahoo Press.

[10] Christos Doulkeridis, Kjetil Nørvåg: **A Survey of Large-Scale Analytical Query Processing in MapReduce**. *VLDB* Journal (to appear 2013)

[11] Jeffrey Dean, Sanjay Ghemawat: **MapReduce: Simplified Data Processing on Large Clusters.** OSDI 2004: 137-150

[12] Michael Stonebraker, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Erik Paulson, Andrew Pavlo, Alexander Rasin: **MapReduce and parallel DBMSs:**

friends or foes? Commun. ACM 53(1): 64-71 (2010)

[13] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, Avi Silberschatz: **HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads.** PVLDB 2(1): 922-933 (2009)

[14] Mohamed Y. Eltabakh, Yuanyuan Tian, Fatma Özcan, Rainer Gemulla, Aljoscha Krettek, John McPherson: **CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop.** PVLDB 4(9): 575-585 (2011)

[15] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, Raghotham Murthy: **Hive - A Warehousing Solution Over a Map-Reduce Framework.** PVLDB 2(2): 1626-1629 (2009)

[16] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, Jörg Schad: **Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing).** PVLDB 3(1): 518-529 (2010)

[17] Eaman Jahani, Michael J. Cafarella, Christopher Ré: **Automatic Optimization for MapReduce Programs.** PVLDB 4(6): 385-396 (2011)

[18] Katja Hose, Akrivi Vlachou: **A survey of skyline processing in highly distributed environments.** VLDB J. 21(3): 359-384 (2012)

[19] Henning Köhler, Jing Yang, Xiaofang Zhou: **Efficient parallel skyline processing using hyperplane projections.** SIGMOD Conference 2011: 85-96

[20] Yufei Tao, Greg Fu, Bernhard Seeger, Dimitris Papadias: **Skyline queries and its variations - An Optimal and Progressive Algorithm for Skyline Queries**. SIGMOD Conference 2003: 467-478

[21] Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger: **Progressive skyline computation in database systems.** ACM Trans. Database Syst. 30(1): 41-82 (2005)

[22] Jan Chomicki, Parke Godfrey, Jarek Gryz, Dongming Liang: **Skyline with Presorting.** ICDE 2003: 717-719

[23] «Wikipedia - The Free Encyclopedia,» [Ηλεκτρονικό]. Available: http://www.wikipedia.org.

[24] Kasper Mullesgaard, Jens Laurits Pedersen: **Efficient Skyline Computation for Large Volume Data in MapReduce Utilising Multiple Reducers**. (2013)

[25] Liang Chen, Kai Hwang, Jian Wu: **MapReduce Skyline Query Processing with a New Angular Partitioning Approach.** IPDPS Workshops 2012: 2262-2270

[26] Boliang Zhang, Shuigeng Zhou, Jihong Guan: **Adapting Skyline Computation to the MapReduce Framework: Algorithms and Experiments.** DASFAA Workshops 2011: 403-414

[27] Foto N. Afrati, Paraschos Koutris, Dan Suciu, Jeffrey D. Ullman: **Parallel skyline queries.** ICDT 2012: 274-284

[28] Wang Zuo: **Research on Fault-Tolerant Parallel Skyline Query Technology in Cloud Computing Environment**. (2011)

[29] «Eclipse,» The Eclipse Foundation, [Ηλεκτρονικό]. Available: http://www.eclipse.org. [Πρόσβαση 2011]

[30] «Cloudera,» Cloudera Inc., [Ηλεκτρονικό]. Available: http://www.cloudera.com. [Πρόσβαση 2011]

[31] Lawrence Manion, Keith Morrison, Louis Cohen: **Μεθοδολογία εκπαιδευτικής έρευνας**. Μεταίχμιο. (2008)

[32] **MapReduce Algorithms**. Cloudera. (2009)

[33] **Programming with Hadoop**. Cloudera. (2009)

[34] Wei -Yu Chen: **Programming Map-Reduce (Hadoop) with Eclipse.** NCHC. (2008)

[35] **Hadoop Map/Reduce Tutorial**. The Apache Software Foundation. (2008)

[36] **Hadoop**. The Apache Software Foundation, [Ηλεκτρονικό]. Available: hadoop.apache.org. [Πρόσβαση 9 2011].

[37] **Hadoop Wiki**. The Apache Software Fountation, [Ηλεκτρονικό]. Available: wiki.apache.org/hadoop. [Πρόσβαση 2009].

[38] **MapR Technologies.** MapR Technologies Inc., [Ηλεκτρονικό]. Available: http://www.mapr.com. [Πρόσβαση 2013].

[39] **Dissertation**. [Ηλεκτρονικό]. Available: http://www.dissertationtopic.net. [Πρόσβαση 2013].

[40] **CloudLounge**. [Ηλεκτρονικό]. Available: http://www.cloud-lounge.org.

[Πρόσβαση 2013].

[41] Jörg Schad, Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz: **Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance.**PVLDB 3(1): 460-471 (2010)

[42] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver,Ramana Yerneni: **PNUTS: Yahoo!'s hosted data serving platform.** PVLDB 1(2): 1277-1288 (2008)

[43] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu,Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Henry F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, Gerhard Weikum: **The Claremont report on database research.** SIGMOD Record 37(3): 9-19 (2008)

[44] Bo Peng, Bin Cui, Xiaoming Li: **Implementation Issues of A Cloud Computing Platform.** 59-66

[45] Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, Philip S. Yu: **Graph OLAP: a multi-dimensional framework for graph data analysis.** Knowl. Inf. Syst. 21(1): 41-63 (2009)

[46] Daniel Warneke, Odej Kao: **Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud.** IEEE Trans. Parallel Distrib. Syst. 22(6): 985-997 (2011)

[47] João B. Rocha-Junior, Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg: **AGiDS: A Grid-Based Strategy for Distributed Skyline Query Processing.** Globe 2009: 12-23

[48] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, Frank van Harmelen: **Scalable Distributed Reasoning Using MapReduce.** International Semantic Web Conference 2009: 634-649

[49] Jinbao Wang, Sai Wu, Hong Gao, Jianzhong Li, Beng Chin Ooi: **Indexing multi-dimensional data in a cloud system.** SIGMOD Conference 2010: 591-602

[50] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, Jingren Zhou: **SCOPE: easy and efficient parallel processing of massive data sets.** PVLDB 1(2): 1265-1276 (2008)

[51] Suvarna Bothe, Panagiotis Karras, Akrivi Vlachou: **eSkyline: Processing Skyline Queries over Encrypted Data.** PVLDB 6(12): 1338-1341 (2013)

[52] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis: **Dremel: Interactive Analysis of Web-Scale Datasets.** PVLDB 3(1): 330-339 (2010)

[53] Sai Wu, Dawei Jiang, Beng Chin Ooi, Kun-Lung Wu: **Efficient B-tree Based Indexing for Cloud Data Processing.** PVLDB 3(1): 1207-1218 (2010)

[54] **Autodesk Softimage**. [Ηλεκτρονικό]. Available: http://download.autodesk.com/. [Πρόσβαση 2013].

ΓΛΩΣΣΑΡΙ

Skyline επερώτηση = Skyline Query

Γεννήτριες Αριθμών = Data Generators

Διαμερίσεις = Partitions

Διαμοιρασμός Χώρου = Space Partitioning

Διαμοιρασμός Χώρου με βάση Grid τεχνική = Grid-based Partitioning

Διαμοιρασμός Χώρου με βάση Hyperplane προβολή = Partitioning via Hyperplane Projection

Διαμοιρασμός Χώρου με βάση Γωνιακές Συντεταγμένες = Angle-based Partitioning

Επικράτηση Σημείων = Point Domination

Κατανομή = Distribution

Κόμβοι = Nodes

Κομμάτια = Tokens

Μονοπάτι Εισόδου = Input Path

Μονοπάτι Εξόδου = Output Path

Παράλληλη Επεξεργασία = Parallel Execution

Πολυδιάστατα Δεδομένα = Multidimensional Data

Ρύθμιση Παραμέτρων = Configuration

Σύστημα αρχείων του Hadoop = HDFS

Συστοιχία = Cluster

Τοπικά Skyline Σημεία = Local Skyline

Υπολογισμός Skyline Σημείων = Skyline Computation

Υπολογιστικό Νέφος = Cloud Computing

ΠΑΡΑΡΤΗΜΑ

Παρατίθενται ενδεικτικά τα κομμάτια κώδικα που αφορούν στο διαμοιρασμό του χώρου για όλες τις τεχνικές και τον υπολογισμό των skyline σημείων.

Grid Partitioning

```
public int getPartition(Text key,  Text value, int m_nNrPartitions){
   StringTokenizer itr = new StringTokenizer(key.toString());
   String pointId = itr.nextToken();
       for (int i=0; i< m_nDim; i++){
arPoint[i]=Double.parseDouble(itr.nextToken());
                   }
       int nPartitionId = 0;
       nPartitionId = (int)Math.floor(arPoint[0]/(L/(double)K[0]));
       int off = 1;
       for (int i=1 ; i < m_nDim ; i++) {
             int a = (int)Math.floor(arPoint[i]/(L/(double)K[i]));
                   off = K[i-1]*off;
                   nPartitionId += a*off;
                   }
                   if (nPartitionId > m_nNrPartitions){
                   nPartitionId = m_nNrPartitions;
                   }
             System.out.println();
             return nPartitionId;
             }
         }
```

Angle-based Partitioning

```
public int getPartition(Text key,  Text value, int m_nNrPartitions){
	m_dPartVolume = m_arVolumeAtD[m_nDim]/(double)m_nNrPartitions;
	computeBounds();
	StringTokenizer itr = new StringTokenizer(key.toString());
	String pointId = itr.nextToken();
	 for (int i=0; i< m_nDim; i++){
		arPoint[i]=Double.parseDouble(itr.nextToken());
		}
	int nPartitionId = 0;
	double[] arPointNew = convertCartesianToPolar(arPoint);
	nPartitionId= mapPoint2Partition(arPointNew);
	if ( nPartitionId > m_nNrPartitions){
		nPartitionId=m_nNrPartitions-1;
			}
	return nPartitionId;
}
```

Hyperplane projection – based partitioning

```
public int getPartition(Text key,  Text value, int m_nNrPartitions){
	StringTokenizer itr = new StringTokenizer(key.toString());
	String pointId = itr.nextToken();
	for (int i=0; i< m_nDim; i++){
		arPoint[i]=Double.parseDouble(itr.nextToken());
		}
	int nPartitionId = 0;
	double[] arPointNew = convertCoord(arPoint);
	nPartitionId= mapPoint2Partition(arPointNew);
	if ( nPartitionId > m_nNrPartitions){
		nPartitionId=m_nNrPartitions-1;
		}
	return nPartitionId;
}
```

Skyline Computation

```
public void map(LongWritable key,Text value,
OutputCollector<DoubleWritable,Text> output, Reporter reporter)
throws IOException {
      DoubleWritable dw = new DoubleWritable();
      Text word = new Text();
      String line= value.toString();
      StringTokenizer st = new StringTokenizer(line);
            int y =st.countTokens();
            int f=0;
            String nLine="";
            while(st.hasMoreTokens()){
                  String s =  st.nextToken();
                  while(f<(y-1)){
                        nLine = nLine+" "+ st.nextToken();
                        f++;
                       }
                  double d[] = parseLine(nLine);
                  double dScore = computeScoreEntropy(d);
                  if(memoryPoints.isEmpty()){
                  memoryPoints.add(d);
                  dw.set(dScore);
                  word.set(s+" "+nLine);
                  break;
            }else {
                  int sz = memoryPoints.size();
                  for(int i=0;i<=sz-1;i++){
                        double x[] = (double[])memoryPoints.get(i);
                        int k = dominate(d,x);
                              if(k==-1){
                              break;
                              }else if (k==0){
                              if(i== (sz-1)){
                              memoryPoints.add(d);
                              sz++;
                              dw.set(dScore);
                              word.set(s + " "+nLine);
                              break;
                              }
                        }else if(k==1){
```

```
                        memoryPoints.remove(i);
                        sz--;
                        i--;
                        if(i== (sz-1)){
                        memoryPoints.add(d);
                        sz++;
                        dw.set(dScore);
                        word.set(s + " "+nLine);
                        break;
                              }
                        }
                  }
            }
            output.collect(dw,word);
      }
}
```