Πανεπιστήμιο Θεσσαλίας

Διπλωματική Εργασία

Αποτίμηση του αντικτύπου του αποθηκευτικού μέσου στους Hadoop density-based αλγορίθμους clustering για big data

Impact of storage medium on Hadoop density-based big data clustering algorithms

Χατζηιωάννου Πέτρος

ΑΕΜ: 1812

Επιβλέπων Καθηγητής:
καθ. Κατσαρός Δημήτριος

Βόλος 2018

Ευχαριστίες

Πρώτα απ’ όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής εργασίας μου, Καθηγητή κ. Κατσαρό Δημήτριο, για την καταλυτική βοήθεια και πολύτιμη καθοδήγησή του κατά τη διάρκεια της δουλειάς μου. Ευχαριστώ επίσης, τον συνάδελφο μου Τσολακίδη Γρηγόριο για την ηθική υποστήριξή του, καθώς και για τη βοήθεια που μου προσέφερε σε θέματα που αφορούσαν προγραμματισμό σε JAVA καθώς και την κατανόηση του προβλήματος. Θα ήθελα να απονείμω ανώνυμες ευχαριστίες και σε όλους όσους συντέλεσαν στη δημιουργία αυτής της εργασίας καθώς, χωρίς τη βοήθειά τους το τελικό αποτέλεσμα δεν θα ήταν το ίδιο. Πάνω απ’ όλα, είμαι ευγνώμων στους γονείς μου, Ιωάννη και Σοφία Χατζηιωάννου για την ολόψυχη αγάπη και υποστήριξή τους όλα αυτά τα χρόνια.

Πέτρος Χατζηιωάννου

Πίνακας Περιεχομένων

Contents

Πίνακας Εικόνων

Περίληψη

Το MapReduce είναι μία από τις πιο δημοφιλείς κατανεμημένες πλατφόρμες υπολογιστών για εφαρμογές που επεξεργάζονται δεδομένα μεγάλης κλίμακας. Έχει εφαρμοστεί σε πολλές περιοχές διαίρεσης και επίλυσης (divide and conquer) προβλημάτων, όπως μηχανές αναζήτησης, εξόρυξη δεδομένων και καταγραφή δεδομένων σε ευρετήρια. Το Hadoop (το οποίο αναπτύχθηκε από την Yahoo) είναι μία open source υλοποίηση του μοντέλου MapReduce σε Java. Στην παρούσα διπλωματική εργασία, εστιάζουμε στις επιδόσεις που παρουσιάζει το Hadoop όταν καλείται να κάνει εγγραφή δεδομένων σε σκληρούς δίσκους Hard Disk Drive αλλά και Solid State Disk.

Πιο συγκεκριμένα, η διπλωματική εργασία έχει σαν αντικείμενο μελέτης την επίδοση αλγορίθμων ομαδοποίησης, υλοποιημένοι στο framework Apache Hadoop. Η εργασία αποτελείται από δύο κύρια μέρη. Το πρώτο μέρος αφορά την υλοποίηση density-based clustering αλγορίθμων ενώ το δεύτερο μέρος εστιάζει στις διαφορές που παρουσιάζουν αυτοί οι αλγόριθμοι όταν πραγματοποιούν εγγραφές σε Hard Disk Drive (HDD) και Solid State Disks (SSD). Η υλοποίηση των αλγορίθμων έγινε σε γλώσσα προγραμματιστικού περιβάλλοντος JAVA και τα πειράματα πραγματοποιήθηκαν με τη χρήση εικονικού μηχανήματος (Virtual Machine – VM) που κατασκευάστηκε για το σκοπό αυτό. Απώτερος στόχος της διπλωματικής αυτής εργασίας είναι η εξαγωγή συμπερασμάτων σχετικά με το χρόνο εκτέλεσης παράλληλων, κατανεμημένων αλγορίθμων ομαδοποίησης χρησιμοποιώντας το προγραμματιστικό μοντέλο MapReduce σε ένα computer cluster.

Abstract

MapReduce is one of the most popular distributed computing platforms for applications that process largescale data. It has been applied to many areas of divide and conquer problems such as search engines, data mining and indexing of data. Hadoop, initially developed by Yahoo, is an open source software with the ability of implementing the MapReduce model in Java environment. This Dissertation is focused on the examination of Hadoop's performance when operating in Hard Disks (HDDs) and Solid-State Disks (SSDs) disks. More specifically, the dissertation aims at studying the performance of Density-based algorithms implemented within Apache Hadoop. The present study can be divided in two parts. In the first part, the implementation of density based algorithmic methods is explored while in the second part, the differences that arise HDD and SSD are quantified through numerical tests. Implementation of the algorithms are utilized in JAVA programming language and the tests are performed using a Virtual Machine (VM) built for this purpose. The ultimate goal of this diploma thesis is to draw conclusions about the duration of executing parallel, distributed algorithm groups using the MapReduce programming model in a cluster of computers.

1. Εισαγωγή

Είναι γενικά αποδεκτό πως ο όγκος των δεδομένων αυξάνεται με πολύ γρήγορους ρυθμούς. Η ταχεία αύξηση των δεδομένων, δημιούργησε σύνολα με αρκετά μεγάλο όγκο και πολυπλοκότητα. Αυτό έχει σαν αποτέλεσμα το μεγαλύτερο μέρος των εφαρμογών επεξεργασίας να καταστεί ανεπαρκές για την διαχείρισή τους. Τα σύνολα αυτά αναφέρονται ως Big Data και αποτέλεσαν μια πολλά υποσχόμενη τάση όταν πρωτοεμφανίστηκαν στον τομέα των τεχνολογιών της πληροφορικής (Information Technology - IT). Τώρα πια τα Big Data είναι πολύ δημοφιλή και οι ειδικοί αναφέρονται στην εποχή μας, σαν μια εποχή που χαρακτηρίζεται από αυτά. Τεράστια ποσά δεδομένων παράγονται από διάφορες πηγές όπως αισθητήρες, κινητά τηλέφωνα και άλλες ηλεκτρονικές συσκευές, δεδομένα διαδικτύου από αναζητήσεις στο διαδίκτυο και υπηρεσίες κοινωνικών μέσων όπως το Facebook, Twitter, Instagram κ.λ.π.. Το 2017, η Διεθνή Εταιρία Δεδομένων (International Data Corporation-IDC) ανέφερε ότι ο όγκος των δεδομένων που θα παράγεται ετησίως μετά το 2025 θα είναι 160 zettabytes (ZB).

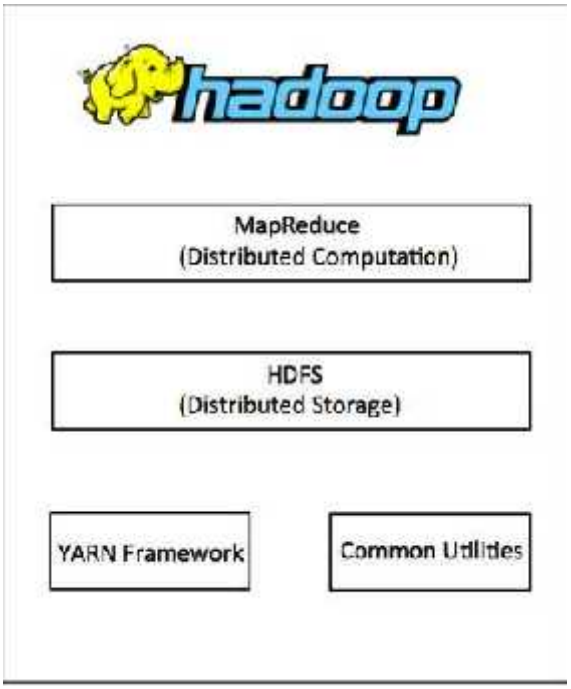
Ως εκ τούτου, μεγάλες βάσεις δεδομένων (Big Data) χρειάζονται ένα νέο μοντέλο επεξεργασίας που έχει καλύτερες δυνατότητες αποθήκευσης, λήψης αποφάσεων και ανάλυσης. Αυτός είναι ο λόγος για τον οποίο γεννήθηκε η τεχνολογία των Big Data. Η τεχνολογία αυτή παρέχει έναν νέο τρόπο εξαγωγής, αλληλεπίδρασης, ενσωμάτωσης και ανάλυσης με απώτερο σκοπό την εξόρυξη σημαντικών πολύτιμων πληροφοριών που προκύπτουν από αυτή την εξειδικευμένη επεξεργασία. Για τον λόγο αυτό δημιουργήθηκαν αρκετά framework με σκοπό την επεξεργασία των Big Data. Ενδεικτικά παραδείγματα τέτοιων framework αποτελούν το Spark, Hadoop, Storm.

Το αρχικό κίνητρο της διπλωματικής εργασίας ήταν η μελέτη της ταχύτητας εγγραφής παράλληλων, κατανεμημένων αλγορίθμων ομαδοποίησης χρησιμοποιώντας το προγραμματιστικό μοντέλο MapReduce σε ένα computer cluster. Δεδομένου ότι ο χρόνος που απαιτεί ένας σειριακός κώδικας να επεξεργαστεί μεγάλου όγκου δεδομένα συνήθως αυξάνεται εκθετικά σε συνάρτηση με τον όγκο, τέθηκε ως πρώτος στόχος η υλοποίηση παράλληλων, κατανεμημένων αλγορίθμων και ως δεύτερος η αξιολόγηση της απόδοσης τους. Με τη χρήση κωδίκων όπου ο κάθε ένας επεξεργάζεται με διαφορετικό τρόπο τα δεδομένα αλλά έχουν όλοι ως κοινό σημείο την ιδιότητα ομαδοποίησης με βάση την πυκνότητα, επιτεύχθηκε ο τελικός στόχος της εργασίας, ο οποίος ήταν να ερευνηθούν οι διαφορές στις επιδόσεις αλγορίθμων όταν κάνουν εγγραφή σε δίσκους Hard Disk και Solid State.

2. Hadoop

Το Hadoop είναι η πιο δημοφιλής ανοιχτού κώδικα εφαρμογή του μοντέλου προγραμματισμού MapReduce. Το Apache Hadoop είναι ένα πλαίσιο λογισμικού για αξιόπιστο, κλιμακωτό, παράλληλο και κατανεμημένο υπολογιστικό σύστημα. Αντί να βασίζεται σε δαπανηρό υλικό και δαπανηρά συστήματα επεξεργασίας και αποθήκευσης δεδομένων, το Apache Hadoop διευκολύνει την παράλληλη επεξεργασία μεγάλων δεδομένων σε συστήματα χαμηλών προδιαγραφών. Το μοντέλο προγραμματισμού HDFS και MapReduce επιχορηγείται σε αυτές τις εργασίες αναλυτικής ανάλυσης μεγάλου όγκου δεδομένων, λόγω της αρχιτεκτονικής κλιμάκωσης και της ικανότητάς του να επεξεργάζεται παράλληλα δεδομένα σε ομάδες πολλαπλών κόμβων. Το MapReduce και το υπάρχον open source project που ονομάζεται Hadoop, επιτρέπουν την παράλληλη επεξεργασία τεράστιου όγκου δεδομένων και αυτόματη κατανομή δεδομένων, διανομή δεδομένων, ανοχή σφάλματος και διαχείριση εξισορρόπησης φορτίου, η οποία τελικά αποδίδει αξιόπιστα υπολογιστικά αποτελέσματα [1]. Το Hadoop αποτελείται από τέσσερα μέρη όπου το καθένα εκτελεί ένα συγκεκριμένο έργο απαραίτητο για ένα σύστημα υπολογιστών σχεδιασμένο για αναλύσεις Big Data. Τα επιμέρους μέρη είναι:

1. Σύστημα αποθήκευσης αρχείων του Hadoop (Hadoop Distributed File System-HDFS).
2. “MapReduce” το οποίο ονομάστηκε έτσι από τις δύο βασικές λειτουργίες που εκτελεί. Την “map”, η οποία είναι υπεύθυνη για την ανάγνωση δεδομένων και τα μετατρέπει σε μορφή κατάλληλη για επεξεργασία και την “reduce”, η οποία εκτελεί μαθηματικούς υπολογισμούς.
3. Το “Hadoop Common”, το οποίο παρέχει τα εργαλεία της Java που χρειάζεται ο χρήστης για να διαβάσει τα δεδομένα που είναι αποθηκευμένα στο σύστημα αρχείων του Hadoop.
4. Το τελευταίο μέρος είναι το “Yarn”, το οποίο διαχειρίζεται τους πόρους των συστημάτων που αποθηκεύουν τα δεδομένα και εκτελούν αναλύσεις.



Εικόνα 1: Τα συστατικά μέρη του Hadoop.(Source: [2])

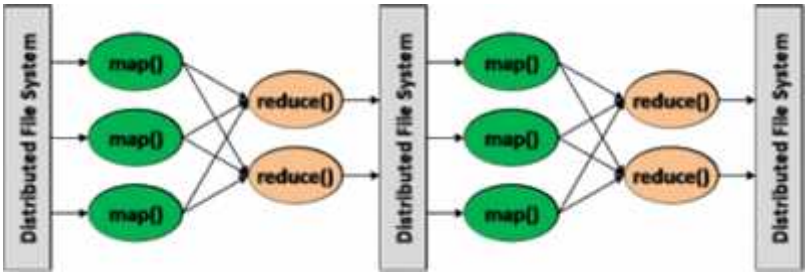
2.1 Προγραμματιστικό Μοντέλο MapReduce

Το Map Reduce είναι ένα μοντέλο προγραμματισμού για την επεξεργασία και παραγωγή μεγάλων συνόλων δεδομένων πάνω σε ένα δίκτυο υπολογιστών (Cluster) που αρχικά προτάθηκε από τη Google το 2004. Τα προγράμματα γραμμένα σε αυτό το λειτουργικό στυλ παραλληλίζονται αυτόματα και εκτελούνται σε ένα μεγάλο σύνολο υπολογιστών. Το σύστημα εκτέλεσης φροντίζει για τις λεπτομέρειες του διαχωρισμού των δεδομένων εισόδου, τον προγραμματισμό της εκτέλεσης του προγράμματος σε ένα σύνολο υπολογιστών, τον χειρισμό των βλαβών των υπολογιστών και τη διαχείριση της απαιτούμενης επικοινωνίας μεταξύ τους. Αυτό επιτρέπει στους προγραμματιστές χωρίς εμπειρία σε παράλληλα και κατανεμημένα συστήματα να χρησιμοποιούν εύκολα τους πόρους ενός μεγάλου κατανεμημένου συστήματος.

Το μοντέλο είναι αρκετά απλό στη χρήση, πολύ διαδεδομένο καθώς έχουν υλοποιηθεί εκατοντάδες προγράμματα MapReduce και πάνω από χίλιες εργασίες MapReduce εκτελούνται καθημερινά στα clusters της Google. Βασική ιδέα του MapReduce αποτελεί η υλοποίηση του κώδικα σε δύο συναρτήσεις, την map και την reduce. Και οι 2 αυτές συναρτήσεις δέχονται ως είσοδο τα ζευγάρια <keyIn,value> και έχουν ως έξοδο τα ζευγάρια <keyOut,value>.

Η map δέχεται μια είσοδο από ζεύγη <key1,value1> και παράγει μία έξοδο από ζεύγη <key2,value2> όπου ταξινομούνται με βάση το κλειδί, και το αποτέλεσμα τους αποτελεί την

είσοδο της reduce. Η reduce λαμβάνει τα ζευγάρια από την map, τα ομαδοποιεί με βάση το κλειδί και παράγει έξοδο <key3,value3> όπου είναι ταξινομημένη με βάση το key. Σε πολλές εργασίες η χρήση της reduce είναι περιττή.



Εικόνα 2: Αναπαράσταση αλγορίθμων MapReduce (Source:[3])

Το MapReduce λειτουργεί ακολουθώντας το μοντέλο του master-slave. Ο κεντρικός κόμβος (NameNode) είναι αυτός που διαχωρίζει τις εργασίες που πρέπει να υλοποιηθούν στους slaves. Για να συμβεί αυτό, ο NameNode είναι αυτός που παρακολουθεί τις δουλειές (job tracker) ενώ οι slaves (DataNodes) έχουν την ευθύνη για την διεκπεραίωση των εργασιών (task trackers) . Εν κατακλείδι, οι task trackers εκτελούν τις εργασίες που τους αναθέτει ο job tracker [4].

2.2 Το σύστημα αρχείων του Hadoop

Το σύστημα κατανομής αρχείων Hadoop (HDFS) έχει σχεδιαστεί για την αξιόπιστη αποθήκευση πολύ μεγάλων συνόλων δεδομένων και τη ροή αυτών των συνόλων δεδομένων σε υψηλό εύρος ζώνης σε εφαρμογές χρηστών. Το HDFS είναι ένα σύστημα αυτοματοποίησης και κατανομής αρχείων που παρέχει αξιόπιστη, κλιμακούμενη και ανθεκτική σε σφάλματα αποθήκευση δεδομένων σε χαμηλού προϋπολογισμού υπολογιστικά συστήματα. Συνεργάζεται στενά με το MapReduce, διανέμοντας την αποθήκευση και τον υπολογισμό σε μεγάλα cluster, συνδυάζοντας πόρους αποθήκευσης που μπορούν να κλιμακώνονται ανάλογα με τις αιτήσεις και τα ερωτήματα. Το κύριο πλεονέκτημα του HDFS είναι η ανοχή σφάλματος. Παρέχοντας γρήγορη μεταφορά δεδομένων μεταξύ των κόμβων και επιτρέποντας στον Hadoop να συνεχίσει να παρέχει υπηρεσίες ακόμη και σε περίπτωση αποτυχίας κόμβου μειώνει τον κίνδυνο καταστροφικής αποτυχίας[1, 5]. Το Hadoop Distributed File System (HDFS) ακολουθεί το μοντέλο “Write once-Ready many”. Για τον λόγο αυτό, δεν επιτρέπεται η επεξεργασία του στοιχείου που έχει εγγραφθεί, παρά μόνο η προσάρτηση επιπλέον πληροφοριών ανοίγοντάς το ξανά. Στις λειτουργίες Εγγραφής-Ανάγνωσης ο χρήστης πρέπει πρώτα να επικοινωνήσει με τον master (NameNode) για να λάβει τα δικαιώματα ώστε να μπορεί να κάνει εύκολα ανάγνωση και εγγραφή data blocks από και προς τους υποτελούς (DataNodes) κόμβους

2.3 Hadoop Common

Επόμενη ενότητα αποτελεί το Hadoop Common, το οποίο παρέχει τα εργαλεία (σε Java) που απαιτούνται για τα συστήματα ηλεκτρονικών υπολογιστών του χρήστη (Windows, Unix ή οτιδήποτε άλλο) για την ανάγνωση των δεδομένων που είναι αποθηκευμένα στο σύστημα αρχείων Hadoop.

2.4 Hadoop Yarn

Τελευταίο κομμάτι του Hadoop αποτελεί το Hadoop Yarn. Κύρια του εργασία αποτελεί η κατανομή των πόρων με τέτοιο τρόπο ώστε να μπορούν αυθαίρετες εφαρμογές να εκτελεστούν σε ένα Hadoop cluster. Για να συμβεί αυτό, το Yarn αποτελείται από δύο συστατικά μέρη. Το πρώτο είναι το Resource Manager το οποίο παρακολουθεί τη χρήση των πόρων και τη ζωντάνια των κόμβων, επιβάλει αναλογικές μεταβολές και διαιτητεύει τους ισχυρισμούς μεταξύ των ενοικιαστών. Δεύτερο συστατικό μέρος του Yarn αποτελεί ο Application Master, σκοπός του οποίου είναι ο συντονισμός του λογικού πλάνου μιας δουλείας (Job). Για να το επιτύχει αυτό ζητάει τους απαραίτητους πόρους από τον Resource Manager και δημιουργεί ένα physical plan. Από τους πόρους που έχει λάβει συντονίζει το σχέδιο γύρω από τα σφάλματα που συμβαίνουν[6].

3. Βιβλιογραφική Ανασκόπηση

Ο όγκος δεδομένων που παράγεται καθημερινά γίνεται όλο και μεγαλύτερος και υπάρχει μεγάλη ζήτηση για μεθόδους και εργαλεία που μπορούν να διαχειρίζονται, να διοχετεύουν και να επεξεργάζονται αποτελεσματικά αυτούς τους μεγάλους όγκους δεδομένων, γνωστό ως Big Data. Τα κλασικά εργαλεία και τα συστήματα διαχείρισης δεν είναι κατάλληλα για μεγάλα δεδομένα. Ως εκ τούτου, έχουν προταθεί αρκετά νέα μοντέλα προγραμματισμού και πλαίσια για την επεξεργασία των Big Data. Για τον λόγο αυτό υπάρχει μια αυξανόμενη τάση των εφαρμογών που αναμένεται να ασχοληθούν με Big Data. Για τέτοιου είδους εφαρμογές, που ασχολούνται δηλαδή με δεδομένα τέτοιου μεγέθους και πολυπλοκότητας το MapReduce framework απέκτησε μεγάλη δημοφιλία. Το MapReduce είναι ένα προγραμματιστικό μοντέλο που επιτρέπει την εύκολη ανάπτυξη κλιμακούμενων παράλληλων εφαρμογών για να επεξεργάζεται Big Data. Google's MapReduce ή αλλιώς το open source Hadoop είναι ένα χρήσιμο εργαλείο για την ανάπτυξη τέτοιων εφαρμογών. Αρκετοί αλγόριθμοι εξόρυξης δεδομένων έχουν υλοποιηθεί σε παράλληλη μορφή πρόσφατα με το MapReduce.

Η ομαδοποίηση (clustering) είναι μια μέθοδος μάθησης χωρίς επίβλεψη που αποτελεί τον ακρογωνιαίο λίθο μιας ευφυούς διαδικασίας ανάλυσης δεδομένων. Χρησιμοποιείται για την διερεύνηση των αλληλεπιδράσεων μεταξύ μιας συλλογής μοτίβων, οργανώνοντάς τα σε ομοιογενείς συστάδες. Η ομαδοποίηση εφαρμόστηκε δυναμικά σε μια ποικιλία εργασιών στον τομέα της Ανάκτησης Πληροφοριών (IR). Έχει γίνει ένας από τους πιο δραστήριους τομείς της έρευνας και της ανάπτυξης (research and development). Σκοπός του clustering είναι να ανακαλύψει ένα σύνολο ομάδων όπου τα στοιχεία εντός της κάθε ομάδας συνδέονται πιο στενά μεταξύ τους από ό,τι με τα άλλα που ανήκουν σε διαφορετικές ομάδες. Η ομαδοποίηση χρησιμοποιείται σε πολλούς τομείς έρευνας όπως η εξόρυξη δεδομένων, η ανάλυση στατιστικών δεδομένων, η εκμάθηση μηχανών, η αναγνώριση προτύπων, η ανάλυση εικόνων και η ανάκτηση πληροφοριών. Το πρόβλημα της ομαδοποίησης δεν μπορεί να λυθεί με έναν συγκεκριμένο αλγόριθμο, αλλά απαιτεί διάφορους αλγόριθμους που διαφέρουν σημαντικά στην αντίληψή τους για το τι κάνει ένα σύμπλεγμα και πώς μπορεί να τα βρει αποτελεσματικά. Γενικά, οι ομάδες περιλαμβάνουν ομάδες με μικρές αποστάσεις μεταξύ των μελών του συμπλέγματος, πυκνές περιοχές του χώρου δεδομένων, διαστήματα ή συγκεκριμένες στατιστικές κατανομές. Επομένως, η ομαδοποίηση μπορεί να διατυπωθεί ως πρόβλημα πολλαπλών στόχων βελτιστοποίησης. Ο κατάλληλος αλγόριθμος ομαδοποίησης και οι ρυθμίσεις παραμέτρων εξαρτώνται από το μεμονωμένο σύνολο δεδομένων και την προβλεπόμενη χρήση των αποτελεσμάτων [7, 8].

Η διαδικασία της ομαδοποίησης αποτελείται από επτά βήματα [9]:

1. Data Collection : Περιλαμβάνει την προσεκτική εξαγωγή σχετικών αντικειμένων δεδομένων από τις υποκείμενες πηγές δεδομένων. Στο πλαίσιο μας, τα αντικείμενα δεδομένων διακρίνονται από τις ατομικές τους τιμές για ένα σύνολο χαρακτηριστικών (ή μέτρων).
2. Data Cleaning: Αναφέρεται στον καθαρισμό των δεδομένων μετά την εξαγωγή τους από τις πηγές.
3. Representation: Περιλαμβάνει την κατάλληλη προετοιμασία των δεδομένων προκειμένου να γίνει κατάλληλη για τον αλγόριθμο ομαδοποίησης. Εδώ επιλέγεται το μέτρο ομοιότητας, εξετάζονται τα χαρακτηριστικά και η διαστασιολόγηση των δεδομένων.
4. Clustering Tendency: Ελέγχει αν τα δεδομένα έχουν φυσική τάση συσπείρωσης ή όχι. Πρόκειται για ένα στάδιο το οποίο αγνοείται συχνά, ειδικά όταν γίνεται επεξεργασία μεγάλων συνόλων δεδομένων.
5. Clustering Strategy: Περιλαμβάνει την προσεκτική επιλογή του αλγόριθμου ομαδοποίησης και των αρχικών παραμέτρων.
6. Validation: Πρόκειται για ένα από τα τελευταία στάδια της διαδικασίας. Η αξιολόγηση των αποτελεσμάτων μπορεί να φανεί πολύ χρήσιμη όταν το σύνολο δεδομένων είναι μικρό. Όσο ο όγκος των δεδομένων αυξάνεται αυτό το στάδιο τείνει να μην χρησιμοποιείται.
7. Interpretation: Αυτό το στάδιο περιλαμβάνει τον συνδυασμό αποτελεσμάτων ομαδοποίησης.

Οι αλγόριθμοι ομαδοποίησης ταξινομούνται σε : “partitional”, “hierarchical” , “Grid Based” και σε αυτούς που βασίζονται στην πυκνότητα (Density-Based). Οι partitional αλγόριθμοι όπως ο K-means , K-medoids , K-modes , PAM , CLARANS , CLARA και FCM βρίσκουν ομάδες δεδομένων που έχουν σφαιρικό σχήμα μόνο [10]. Η hierarchical ομαδοποίηση ανακαλύπτει ομάδες δεδομένων συγχωνεύοντας επαναληπτικά τα πιο κοντινά ζευγάρια από ομάδες δεδομένων μέχρι να φτάσουν τον επιθυμητό αριθμό από clusters. Ωστόσο, η εύρεση του πλησιέστερου ζεύγους ένα προς ένα κατ’ επανάληψη είναι πολύ δύσκολο να υλοποιηθεί σε παράλληλη μορφή επειδή το πλησιέστερο ζεύγος σε κάθε βήμα εξαρτάται από τις δημιουργούμενες ομάδες δεδομένων στο προηγούμενο στάδιο. Χαρακτηριστικά παραδείγματα αλγορίθμων τέτοιου είδους αποτελούν οι Birch , Cure , Rock [8]. Επόμενη κατηγορία στους αλγορίθμους ομαδοποίησης αποτελεί η “Grid based”. Οι αλγόριθμοι αυτού του τύπου έχουν το χαρακτηριστικό ότι λειτουργούν καλύτερα για spatial data. Ο χώρος που περιέχει τα σημεία του συνόλου δεδομένων χωρίζεται σε κελιά και οι απαραίτητες ενέργειες πραγματοποιούνται μέσα σε αυτά. Παραδείγματα τέτοιων αλγορίθμων αποτελούν οι STING, CLIQUE [8]. Τέλος η τέταρτη και τελευταία κατηγορία αλγορίθμων σχετικά με την ομαδοποίηση δεδομένων είναι αυτή που ασχολείται με την πυκνότητα των στοιχείων (Density-Based). Χαρακτηριστικά παραδείγματα τέτοιου είδους αλγορίθμων αποτελούν ο

DBSCAN, OPTICS, DENCLUE, DBCURE και CFSFDP οι οποίοι έχουν ως χαρακτηριστικό την ανακάλυψη ομαδοποιημένων δεδομένων (clusters) τα οποία είναι πυκνές περιοχές σημείων δεδομένων και διαχωρίζονται από αραιές περιοχές σε σχέση με τις δεδομένες παραμέτρους πυκνότητας (Density Parameters). Η ομαδοποίηση με βάση την πυκνότητα μπορεί να ανακαλύψει ομάδες σημειακών δεδομένων με αυθαίρετα σχήματα [8].

4. Διαφορές των HDD και SSD

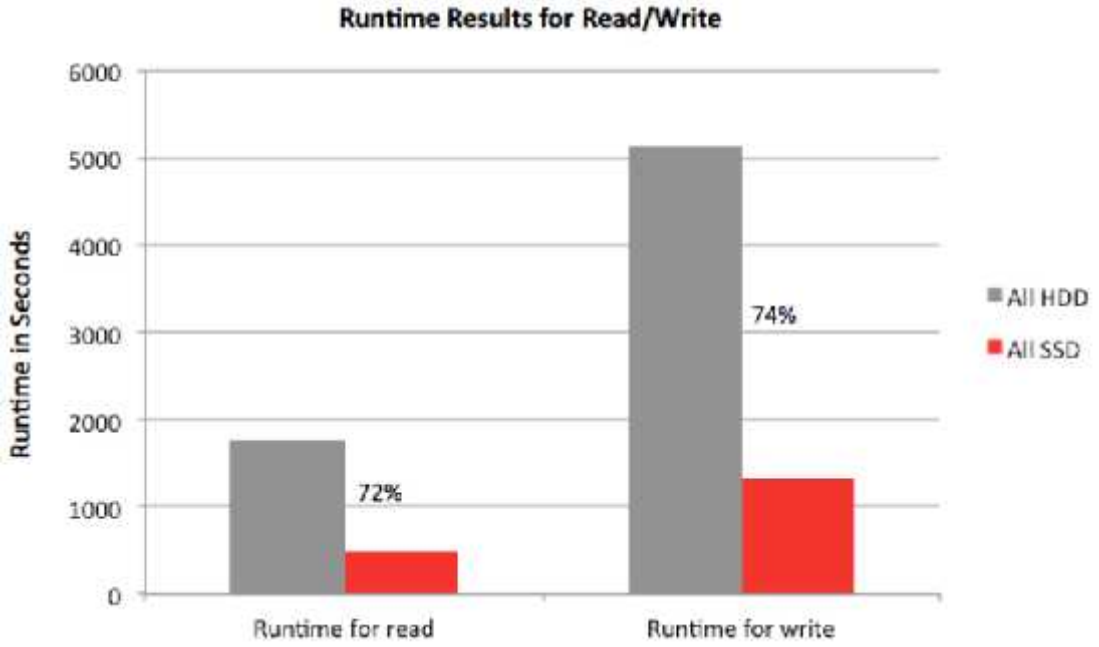
Το παραδοσιακό στήριγμα στην τεχνολογία αποθήκευσης είναι ο σκληρός δίσκος (HDD). Ωστόσο, ενώ η χωρητικότητα των σκληρών δίσκων αυξήθηκε κατά 40% ετησίως, η απόδοση τυχαίων εισόδων / εξόδων (Input / Output) αυξήθηκε μόνο κατά 2% ετησίως. Αυτό σημαίνει ότι για μερικές από τις σημερινές επιχειρήσεις, το Web, το cloud και τις εφαρμογές virtualization που απαιτούν υψηλή χωρητικότητα και απόδοση, οι σκληροί δίσκοι ενδέχεται να μην παρέχουν μια οικονομικά αποδοτική λύση αποθήκευσης, ακόμη και με τη σημαντική μείωση του κόστους ανά GB. Το 2011, μια νέα τεχνολογία εναλλακτικών συσκευών αποθήκευσης, η SSD, άρχισε να κερδίζει προβολή. Οι SSD προσφέρουν εξαιρετική απόδοση, αλλά έχουν πολύ λιγότερη χωρητικότητα ανά μονάδα δίσκου. Είναι επίσης σχετικά ακριβοί σε σύγκριση με τους σκληρούς δίσκους και έχουν ένα όριο αντοχής εγγραφής. Οι σκληροί δίσκοι είναι περιστρεφόμενοι δίσκοι και οι μηχανικές τους ενέργειες μπορούν να προκαλέσουν καθυστέρηση στην ανάκτηση δεδομένων. Οι SSD, που είναι κατασκευασμένοι από μάρκες μνήμης πυριτίου, δεν έχουν κινούμενα μέρη, καθυστερημένη περιστροφή και σχεδόν μηδενική ώρα αναζήτησης, γεγονός που μειώνει δραματικά τους χρόνους απόκρισης. Οι μονάδες SSD μπορούν να είναι εξαιρετικά πολύτιμες για εφαρμογές που χρειάζονται υψηλή απόδοση που μετράται με access latency και λειτουργίες input/output ανά δευτερόλεπτο (IOPS) από περιορισμένη χωρητικότητα [11].

4.1 Διαφορές επιδόσεων σε read και write εργασίες

Οι σύγχρονοι SSD παρέχουν ταχύτητες ανάγνωσης / εγγραφής μέχρι 500Mb / sec. Συγκρίνοντας το με περίπου 100Mb / sec που είναι το άνω κατώφλι στην ταχύτητα των HDD, βγαίνει εύκολα το συμπέρασμα ότι μία αίτησή σας θα παράγει την έξοδο 5 φορές πιο γρήγορα από ό, τι προηγουμένως. Δύο δίσκοι ίδιας χωρητικότητας χρονομετρήθηκαν σε λειτουργίες όπως η ανάγνωση και η εγγραφή δεδομένων από και σε ένα αρχείο αντίστοιχα. Τα αποτελέσματα έδειξαν ότι για απλή εγγραφή δεδομένων χωρίς κάποια επεξεργασία αυτών ο SSD ήταν κοντά τρεις φορές πιο γρήγορος , ενώ όταν τα δεδομένα χρειάζονταν επεξεργασία τότε ο SSD έδειξε να χρειάζεται τον μισό χρόνο από ότι ο HDD. Εκτός από τη βελτίωση της ταχύτητας ανάγνωσης / εγγραφής μιας ροής δεδομένων, ο SSD έχει πολύ πιο σημαντικό πλεονέκτημα έναντι του HDD αφού ο χρόνος αναζήτησης είναι μια τάξη μεγέθους μικρότερη. Σε πειράματα που συνέβησαν σχετικά με την αναζήτηση μέσα σε ένα αρχείο, όταν το αρχείο ήταν μεγάλου μεγέθους ο SSD φάνηκε να επιτυγχάνει την αναζήτηση περίπου είκοσι φορές πιο γρήγορα σε σχέση με τον HDD [12].

4.2 Διαφορές επιδόσεων σε read και write εργασίες στο περιβάλλον του Hadoop

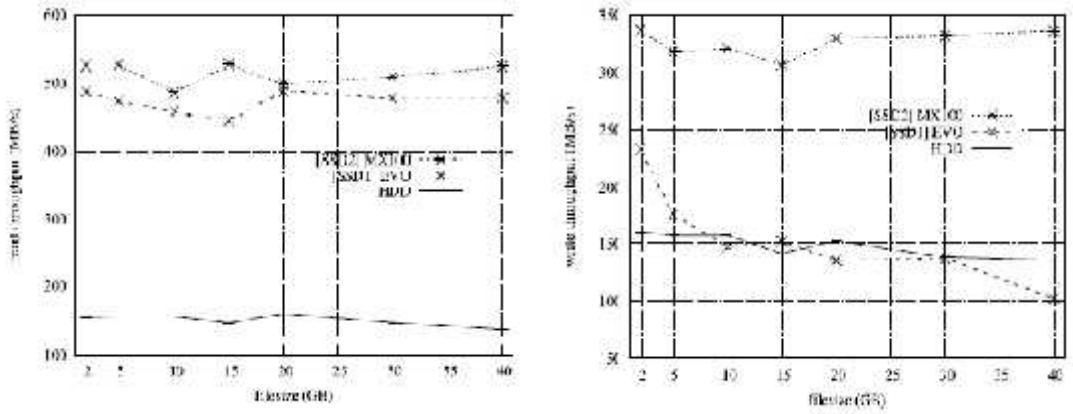
Όπως και πριν έτσι και τώρα δύο δίσκοι ένας HDD και ένας SSD ίδιας χωρητικότητας μελετήθηκαν στις ταχύτητες ανάγνωσης και εγγραφής από και προς ένα αρχείο στο Hadoop Distributed File System. Τα αποτελέσματα όπως και πριν έδειξαν ότι οι δίσκοι SSD είναι αρκετά πιο γρήγοροι από τους HDD. Πιο συγκεκριμένα οι SSD παρουσιάζουν μια βελτίωση του χρόνου κατά 70%-75% σε σχέση με τους HDD στις εντολές ανάγνωσης και εγγραφής στο σύστημα του Hadoop [13].



Εικόνα 3: Διαφορές επιδόσεων ανάγνωσης και εγγραφής(Source : [13])

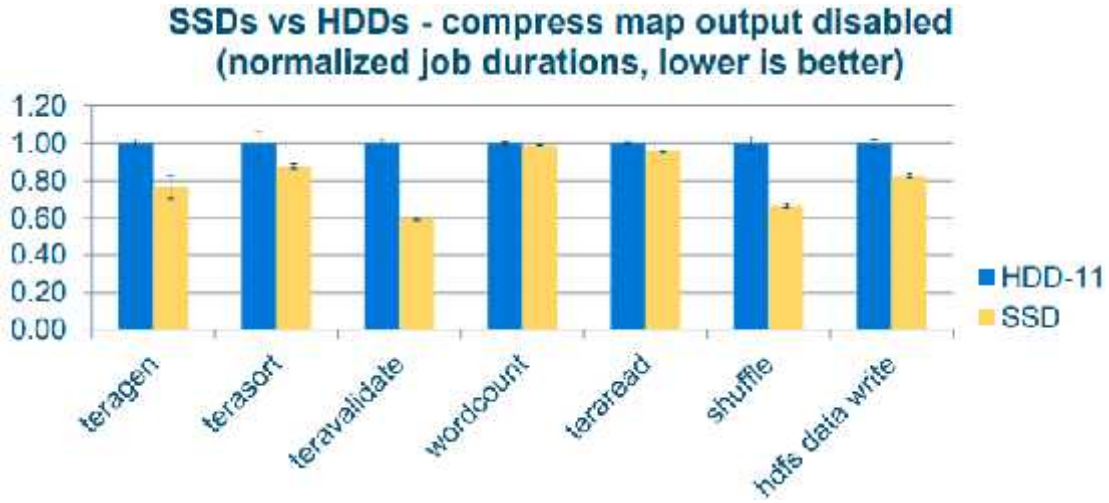
Επιπλέον μελέτες έχουν διεξαχθεί με απώτερο σκοπό την εξαγωγή συμπερασμάτων σχετικά με τις διαφορές που παρουσιάζουν οι επιδόσεις των HDD και SSD. Αυτή τη φορά η έρευνα επικεντρώθηκε στην διαφορά διακίνησης megabyte ανά δευτερόλεπτο (Mb/s throughput) ανάλογα με το μέγεθος του αρχείου εισόδου (file size) ανάμεσα σε τρεις διαφορετικούς δίσκους. Οι δύο από αυτούς ήταν SSD και ο ένας HDD. Ο πρώτος SSD (SSD1) είναι μεγέθους 120 GB , ο δεύτερος (SSD2) είναι 500 GB ενώ ο HDD είναι 1 TB. Στις λειτουργίες ανάγνωσης (read) οι δύο δίσκοι SSD έδειξαν εντυπωσιακές διαφορές σε σχέση με τον HDD. Το ποσοστό των MB/s που διακινούσαν ήταν περίπου τρεις φορές μεγαλύτερο από το ποσοστό του HDD. Αυτό όμως δεν συνέβαινε και στην λειτουργία της εγγραφής (write). Αντίθετα με τα αποτελέσματα που πήραμε από τις προηγούμενες μελέτες, αυτή τη φορά ο ένας SSD δίσκος (SSD1) ενώ είχε αρκετά καλές επιδόσεις στην αρχή, όσο το μέγεθος του αρχείου εισόδου αυξανόταν οι επιδόσεις του μειώνονταν. Αυτό βέβαια δεν ίσχυε και για τον δεύτερο SSD (SSD2) ο οποίος έδειξε να έχει ταχύτητα εγγραφής σταθερή με μερικές μικρές

διακυμάνσεις. Οι επιδόσεις του ήταν δυόμιση φορές καλύτερες από ότι του HDD του οποίου η ταχύτητα εγγραφής μειωνόταν με μικρό ποσοστό ανάλογα με τον input filesize. Τα αποτελέσματα παρουσιάζονται παρακάτω [14].



Εικόνα 4: Διαφορές στην διακίνηση δεδομένων(Source:[14])

Πέρα από τις μελέτες για τις ταχύτητες εγγραφής και ανάγνωσης , έχουν επίσης σημειωθεί έρευνες και σε άλλες ενέργειες. Σε περιβάλλον όπου αποτελείται από δύο δίσκους έναν HDD και έναν SSD έχουν γίνει πειράματα για τις διαφορές που εμφανίζουν οι δύο δίσκοι όταν καλούνται να παράγουν δεδομένα (Teragen) , να ταξινομήσουν (Terasort) , να καταμετρήσουν την εμφάνιση διαφορετικών λέξεων σε ένα κείμενο (WordCount), να εκτελέσουν την λειτουργία της ανάμειξης (Shuffle) η οποία εκτελείται στο ενδιάμεσο στάδιο του map και reduce στο Hadoop και τέλος για ενέργειες που επικυρώνουν ότι η έξοδος είναι ταξινομημένη(Teravalidate). Όλες τις παραπάνω ενέργειες ο SSD δίσκους παρουσίασε καλύτερες επιδόσεις σε σχέση με το HDD όπως φαίνεται στην παρακάτω εικόνα [15].

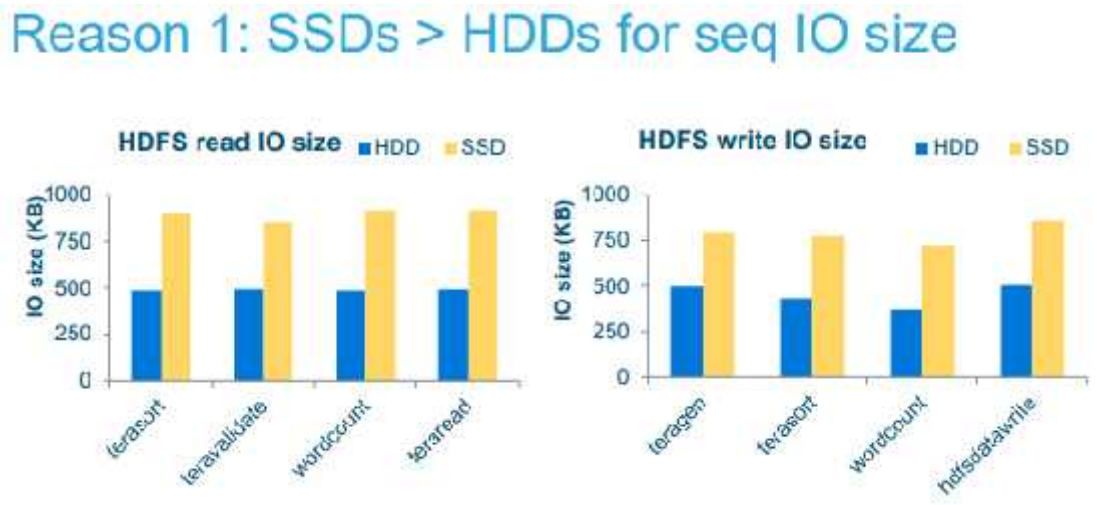


Εικόνα 5: Διαφορές HDD και SSD σε MapReduce tasks με normalize αποτελέσματα (Source:[15])

Τα αποτελέσματα είναι normalize. Βλέπουμε λοιπόν ότι σε όλες τις παραπάνω ενέργειες οι δίσκου τύπου SSD έχουν το λιγότερο μια βελτίωση περίπου κατά 10% (Teraread) και το περισσότερο 40% (Teravalidate). Η μόνη περίπτωση όπου οι δίσκοι δείχνουν να ανταποκρίνονται το ίδιο είναι αυτή του WordCount.

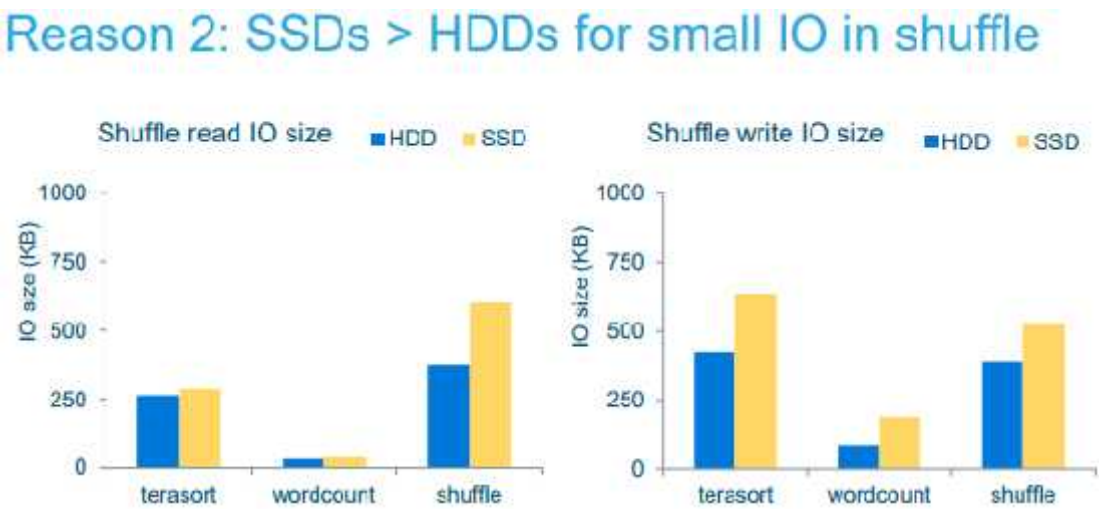
Τα παραπάνω αποτελέσματα πηγάζουν από δύο λόγους οι οποίοι παρουσιάζονται παρακάτω. Ο πρώτος είναι ο ρυθμός ανάγνωσης και εγγραφής που έχει κάθε μία από τις παραπάνω εργασίες. Όπως φαίνεται παρακάτω ο SSD δίσκος έχει σημαντικές διαφορές στον αριθμό των KB που διαβάζει και γράφει κάθε λειτουργία σε σχέση με τον

HDD.



Εικόνα 6: Επιδόσεις ανάγνωσης και εγγραφής διαφόρων λειτουργιών. (Source : [15])

Δεύτερος λόγος για τις διαφορές που παρουσιάζουν οι δύο δίσκοι είναι ο ρυθμός που γίνεται η ανάγνωση και η εγγραφή κατά την διαδικασία του Shuffle που συμβαίνει ενδιάμεσα από την map και την reduce. Όπως φαίνεται στην εικόνα παρακάτω εκτός από την περίπτωση της ανάγνωσης κατά την λειτουργία της WordCount, όλες οι άλλες περιπτώσεις δείχνουν τον SSD να προσπερνάει σε ρυθμό τον HDD.



Εικόνα 7: Επιδόσεις ανάγνωσης και εγγραφής κατά την διαδικασία της ανάμειξης (Shuffle). (Source: [15])

5. Density-Based Algorithms

Επιλέχθηκε να γίνει αναφορά σε αυτού του τύπου αλγορίθμους διότι είναι αυτοί που παρουσιάζουν ιδιαίτερο ενδιαφέρον στον τρόπο λειτουργίας τους και μελετήθηκαν σε βάθος σε όλη τη διάρκεια της διπλωματικής εργασίας. Όπως ήδη αναφέρθηκε πρόκειται για αλγορίθμους που σχηματίζουν ομάδες σημείων δεδομένων ανάλογα με τις παραμέτρους πυκνότητας τις οποίες λαμβάνουν.

Ξεκινώντας, ένας από τους πιο διαδεδομένους αλγορίθμους ομαδοποίησης με βάσει την πυκνότητα για αριθμητικά δεδομένα είναι ο DBSCAN [16]. Η διαδικασία που ακολουθεί ο αλγόριθμος στη σειριακή εκτέλεσή του είναι η εξής:

1. Για κάθε ένα στοιχείο στο σύνολο δεδομένων D υπολογίζει τις αποστάσεις του με όλα τα υπόλοιπα σημεία. Έπειτα βρίσκει πόσοι από αυτούς τους γείτονες βρίσκονται σε ακτίνα μικρότερη από την eps (παράμετρος που δίνει ο χρήστης).
2. Αν ο αριθμός των γειτόνων ενός στοιχείου είναι μεγαλύτερος ή ίσως από minPts (επίσης παράμετρος που ορίζει ο χρήστης) τότε ονομάζει το στοιχείο ως πυρήνας (core).
3. Για κάθε σημείο πυρήνα εάν δεν έχει ήδη ανατεθεί σε άλλο σύμπλεγμα σημείων (cluster) δημιουργεί ένα νέο cluster με πυρήνα το ίδιο.
4. Έπειτα βρίσκει όλα τα density-connected σημεία και τα εισάγει στο σύμπλεγμα σημείων του εξεταζόμενου πυρήνα.
5. Η διαδικασία είναι επαναληπτική και συμβαίνει μέχρι να επισκεφθούν όλα τα σημεία του Dataset. Τέλος αν κάποιο σημείο δεν ανήκει σε κανένα σύμπλεγμα θεωρείται θόρυβος (noise).

Επόμενος αλγόριθμος που επίσης χρησιμοποιείται για την ομαδοποίηση αριθμητικών δεδομένων είναι ο Clustering by fast search and find of density peaks (dc, minPts) [17]. Η διαδικασία που ακολουθείται αποτελείται από πέντε βήματα και είναι η παρακάτω:

1. Υπολογίζει τον πίνακα αποστάσεων (distance matrix) μεταξύ των σημείων του συνόλου του Dataset.
2. Για κάθε ένα στοιχείο βρίσκει των αριθμών των σημείων που βρίσκονται εντός μιας απόστασης αποκοπής (cut-off distance – dc) που δίνεται σαν παράμετρος από τον χρήστη. Ο αριθμός αυτός ονομάζεται πυκνότητα.
3. Εφόσον έχει βρει την πυκνότητα όλων των στοιχείων, υπολογίζει για κάθε σημείο την ελάχιστη απόσταση από σημείο με μεγαλύτερη πυκνότητα (δ). Το στοιχείο με

την μεγαλύτερη πυκνότητα στο Dataset θέτει ως δ την μέγιστη απόσταση του από άλλο σημείο.

4. Τα στοιχεία με υψηλό δ και σχετικά ψηλό ρ ταξινομούνται ως πυρήνες (cluster centers) και όσα στοιχεία ανήκουν στις γειτονιές των cluster centers γίνονται γείτονές του.
5. Cluster centers σε συνδυασμό με τους γείτονες αποτελούν μια ομάδα σημείων (cluster).

Τέλος άλλος ένας αλγόριθμος που ασχολείται με την ομαδοποίηση των δεδομένων είναι ο DBCURE (τ,δ) [18]. Τα βήματα αυτού του αλγορίθμου παρουσιάζονται παρακάτω:

1. Για κάθε ένα στοιχείο στο σύνολο δεδομένων D υπολογίζει την έλλειψη που σχηματίζεται γύρω από το στοιχείο.
2. Έπειτα επιλέγει ένα τυχαίο σημείο (το οποίο δεν έχει εξερευνηθεί ακόμα) και βρίσκει τα στοιχεία που περιλαμβάνονται μέσα στην έλλειψη που έχει σχηματιστεί με βάση τις συντεταγμένες του και τα βάζει σε ένα νέο σύνολο S.
3. Έπειτα για κάθε στοιχείο που βρίσκεται στο S, βρίσκει των αριθμών των γειτόνων που υπάρχουν μέσα στην έλλειψη που σχηματίζεται. Αν ο αριθμός είναι μεγαλύτερος από *δ*, τότε στο σύνολο S προστίθενται τα νέα σημεία. Αν δεν είναι προχωράει στο επόμενο σημείο.
4. Η διαδικασία 3 επαναλαμβάνεται μέχρι το S να μην μπορεί να επεκταθεί άλλο. Όταν συμβεί αυτό ο αλγόριθμος επιστρέφει στο βήμα 2.

Ο αλγόριθμος που μελετήθηκε και αφορά την ομαδοποίηση σε κατηγορικά δεδομένα είναι ο **Transitive heuristic [19]** και εκτελείται σε τέσσερα βασικά βήματα τα οποία είναι η αρχικοποίηση, κατασκευή, ομαδοποίηση και αξιολόγηση.

1. Επιλέγει ένα τυχαίο σημείο από το Dataset για εκπρόσωπο.
2. Συγκεντρώνει όλα τα στοιχεία από το σύνολο δεδομένων που είναι παρόμοιες με τον εκπρόσωπο που επιλέχθηκε στο προηγούμενο βήμα χρησιμοποιώντας το κριτήριο του Condorcet (για κάθε ένα attribute που είναι όμοιο με αυτό του cluster ο μετρητής αυξάνεται κατά ένα ενώ όταν είναι ανόμοιο μειώνεται κατά ένα.
3. Αν το συνολικό άθροισμα είναι μεγαλύτερο του μηδέν τότε καταχωρείται στο αντίστοιχο cluster ενώ αν είναι μικρότερο δημιουργείται νέο cluster και το στοιχείο είναι το cluster center).
4. Υπολογίζει την τομή των υπάρχων ομάδων δεδομένων. Αν είναι διαφορετική από το μηδέν τότε υπολογίζει τη συνεισφορά των στοιχείων στις ομάδες που βρίσκονται, η ομάδα στην οποία προσφέρει περισσότερο είναι αυτή που το κρατάει ενώ από τις

άλλες διαγράφεται και τέλος γίνεται ενημέρωση για το ποια στοιχεία είναι ήδη ομαδοποιημένα.

5. Τέλος γίνεται έλεγχος για το αν όλα τα στοιχεία είναι ομαδοποιημένα, αν είναι τότε ο αλγόριθμος τελειώνει αλλιώς επιστρέφει στο βήμα 1.

6. Υλοποίηση κατανεμημένης εκδοχής αλγορίθμων και πειραματική αξιολόγηση

Σε αυτό το κεφάλαιο της εργασίας παρουσιάζεται η μέθοδος υλοποίησης των παραπάνω αλγορίθμων σε μορφή MapReduce και τα αποτελέσματα των αριθμητικών πειραμάτων.

6.1 DB-CURE MapReduce

Η μέθοδος DB-CURE MR υλοποιείται σε 4 βήματα:

1. Εκτίμηση των πινάκων συνδιακύμανσης (covariance matrix) της γειτονιάς. Αρχικά υπολογίζουμε τον πίνακα συνδιακύμανσης της γειτονιάς για κάθε κελί παράλληλα.
2. Υπολογισμός ελλειψοειδών γειτονιών: Στη συνέχεια ανακαλύπτει όλα τα ζεύγη σημείων, καθένα από τα οποία βρίσκεται μέσα στις δύο γειτονιές τους, πραγματοποιώντας ομοιότητες.
3. Ανακάλυψη των πυρήνων συμπλέγματος (core points): Για ένα core point p, το σύνολο του p και των σημείων που είναι density reachable ονομάζεται πυρήνας συμπλέγματος. Παίρνουμε τον κατάλογο των ελλειψοειδών σημείων της γειτονιάς για κάθε σημείο του D χρησιμοποιώντας κάθε ξεχωριστό σημείο ως κλειδί ομαδοποίησης. Για κάθε κλειδί ομαδοποίησης, αν ο αριθμός των ελλειψοειδών σημείων της γειτονιάς τ είναι τουλάχιστον δ, το σημείο είναι ένα σημείο πυρήνα και εξάγουμε το σημείο με τα ελλειψοειδή σημεία του.
4. Συγχώνευση πυρήνων συμπλέγματος (core points): Συνδυάζουμε επανειλημμένα τα core points τα οποία μοιράζονται ένα κοινό σημείο για να δημιουργηθούν οι τελικές ομάδες σημείων δεδομένων.

Η διαφορά με το paper που μελετήθηκε για να υλοποιηθεί ο αλγόριθμος αφορά το κομμάτι κατασκευής ε-Tree. Για να αποφύγουμε την κατασκευή αυτή στην τελευταία εργασία του αλγορίθμου συγκεντρώνουμε όλα τα τοπικά clusters και ανακαλύπτουμε τα global. Πιο αναλυτικά ο αλγόριθμος υλοποιείται με τη χρήση πέντε διαφορετικών εργασιών στο MapReduce. Στην προ επεξεργασία ο αλγόριθμος διαβάζει το σύνολο δεδομένων (Dataset) D και διαιρεί το χώρο που περιέχει τα σημεία του D σε ορθογώνια πλέγματα (cells) διαιρώντας κάθε διάσταση σε διαστήματα M με το ίδιο μήκος. Έπειτα τοποθετεί κάθε στοιχείο του συνόλου σε ορθογώνιο ανάλογα με τις συντεταγμένες του και δημιουργεί τα ζεύγη αριθμός ορθογωνίου(cell-ID)–συντεταγμένες σημείου. Θεωρούμε Si το άθροισμα των i-συντεταγμένων όλων των στοιχείων που ανήκουν σε ένα κελί G_{k1_kd} και $S_{i,l}$ το άθροισμα των γινομένων μεταξύ της διάστασης *i* και *j* όλων των σημείων σε ένα κελί G_{k1_kd}. Το σύνολο των απαραίτητων εργασιών παρουσιάζεται παρακάτω:

Εργασία 1.

Για κάθε στοιχείο pi το οποίο ανήκει σε ένα κελί με ID g καλείται η **map** συνάρτηση της και παράγει ως έξοδο το ζευγάρι <g,pi>. Στη συνέχεια η συνάρτηση **reduce** παράγει τα ζευγάρια <g,ListOfItems<g>>.

Εργασία 2.

Στη συνέχεια η **map** δέχεται ως είσοδο τα ζεύγη <g,ListOfItems<g>> από την έξοδο της reduce της εργασίας 1 και υπολογίζει τα αθροίσματα $S_{xx}S_{yy}S_xS_y$ αλλά και τον αριθμό των στοιχείων που υπάρχουν σε κάθε κελί(n) και παράγει ως έξοδο τα ζεύγη <g', n, <$S_{xx}S_{yy}S_xS_y$'> > και <g,ListOfItems> , για όλα τα γειτονικά κελιά g' του g. Η **reduce** ομαδοποιεί τα δεδομένα με βάση το ID του κάθε κελιού (g) και δημιουργεί για κάθε ένα κελί το ζεύγος <g, $\Sigma n, list < S_{xx}S_{yy}S_xS_y >$, ListOfItems<g>>.

Εργασία 3.

Έπειτα σε επόμενη εργασία η συνάρτηση **map** για κάθε ένα ζεύγος <g,Σn $list < S_{xx}S_{yy}S_xS_y >$,ListOfItems<g>> που παίρνει για είσοδο από την έξοδο της reduce από την εργασία 2 παράγει τον πίνακα συνδιακύμανσης και έπειτα υπολογίζονται οι άξονες τις έλλειψης κάθε κελιού g.

Για τον πίνακα συνδιακύμανσης χρησιμοποιείτε ο τύπος:

$$\Sigma(u,v) = \frac{1}{n-1}\sum S_{uv}[k_1,\dots,k_d] - \frac{n}{n-1}\mu(u)\mu(v)$$

Όπου n είναι ο αριθμός των στοιχείων στα γειτονικά κελιά όπου έχει υπολογισθεί στην Εργασία 2 και $\mu_i(u) = \frac{1}{n}\sum[S_u[k_1],\dots,k_d]$ δηλώνει την μέση τιμή των σύνολο των σημείων των γειτονικών κελιών των u συντεταγμένων. Τέλος η map γράφει για κάθε στοιχείο για κλειδί(key) το <cell-ID, πίνακα συνδιακύμανσης για τους δύο άξονες> και σαν τιμή (value) τις συντεταγμένες του σημείου. Η **reduce** παράγει την λίστα από τα σημεία που ανήκουν στα ίδια κελιά μαζί με του άξονες της έλλειψης οι οποίοι θα χρησιμοποιηθούν αργότερα για την εύρεση ομάδων σημείων (clusters).

Εργασία 4.

Η **map** συνάρτηση έχει για είσοδο το αποτέλεσμα της reduce από την εργασία 3 και για κάθε ένα κελί πραγματοποιεί τις ακόλουθες ενέργειες. Αρχικά σχηματίζει τις ελλείψεις για κάθε ένα σημείο και μετράει των αριθμό γειτόνων που περιλαμβάνονται. Αν ο αριθμός αυτός είναι αρκετά μεγάλος το σημείο θεωρείται πυρήνας σημείων και όσα σημεία περιλαμβάνονται στην έλλειψη που σχηματίζει είναι οι γείτονες του. Ο πυρήνας μαζί με τους γείτονες του αποτελούν μία ομάδα σημείων (cluster). Αν ο αριθμός των σημείων δεν είναι επαρκής τότε η συνάρτηση μελετάει το επόμενο σημείο. Κάθε στοιχείο έχει το δικαίωμα να ανήκει σε ένα μόνο cluster. Αφού τελειώσει με την εύρεση των cluster η συνάρτηση ψάχνει επαναληπτικά

να βρει ποια cluster πρέπει να ενωθούν. Για να συμβεί αυτό αρκεί να υπάρχουν έστω ένας γείτονας από ένα cluster που να περιλαμβάνει έστω ένα σημείο από το άλλο cluster στην έλλειψη που σχηματίζει. Εφόσον τελειώσει αυτή η διαδικασία η τοπική εύρεση cluster έχει τελειώσει και απομένει να υλοποιηθεί η συνένωση μεταξύ cluster από διαφορετικά cell. Η map έχει ως έξοδο ένα κλειδί που είναι κοινό για όλα τα κελιά και σαν τιμή για κάθε τοπικό cluster ορίζει μία λίστα από τα σημεία του αλλά και τις μεταβλητές για την εύρεση ελλείψεων στο επόμενο βήμα. Για την εύρεση των άκρων της έλλειψης των στοιχείων για κάθε κελί ακολουθείται ο παρακάτω τύπος.

$$[p]_i(l) - \sqrt{c * \Sigma(l,l)}, p_i(l) + \sqrt{c * \Sigma(l,l)}$$

όπου $c = -2\log(\tau(2\pi)^{\tau}(d/2)\ |\Sigma|^{1/2})$. Η πρώτη εξίσωση μας δίνει τα άκρα μιας έλλειψης για ένα στοιχείο p ενώ η δεύτερη προκύπτει από το την ιδιότητα μιας Gaussian συνάρτησης (στην περίπτωση μας είναι η ελλειψοειδή γειτονιά κάθε σημείου) να έχει ένα κατώτατο όριο τα δηλαδή $f(p_i) \geq \iota$. Η reduce συνενώνει όλα τα σημεία βάση του κοινού τους κλειδιού και δημιουργεί την έξοδο <key, list(clusters)>.

Εργασία 5.

Η **map** δέχεται ως είσοδο την έξοδο της reduce από την προηγούμενη εργασία και με κατάλληλο διαχωρισμό χωρίζει τα clusters , μεταβλητές έλλειψης και προσπαθεί να βρει ποια από τα τοπικά clusters μπορούν να συνενωθούν. Για να το πετύχει αυτό λειτουργεί όπως πριν, ψάχνει δηλαδή να βρει εάν τα σημεία από ένα τοπικό cluster συμπεριλαμβάνουν στις ελλείψεις τους σημεία από άλλα τοπικά clusters διαφορετικού κελιού. Εάν αυτό συμβαίνει τότε πραγματοποιεί την συνένωση. Τέλος η map γράφει σαν έξοδο τον αριθμό του cluster (cluster-ID) και σαν τιμή την λίστα των σημείων που συμπεριλαμβάνονται σε κάθε cluster.

Δεν υπάρχει **reduce**.

6.2 DBSCAN-MR

Η διαφοροποίηση που υπάρχει σε σχέση με το paper στο οποίο βασίστηκε η κατασκευή του αλγορίθμου είναι η παρακάτω. Επειδή δεν βρέθηκε τρόπος για την δημιουργία και χρήση του R-TREE ώστε να μπορέσουμε να κάνουμε την συγχώνευση των cluster σε διαφορετικά τμήματα (εργασία 3 αλγορίθμου), προχωρήσαμε στην παρακάτω διαφοροποιημένη υλοποίηση του αλγορίθμου. Στην προ επεξεργασία ο αλγόριθμος διαβάζει το σύνολο δεδομένων (Dataset) D και διαιρεί το χώρο που περιέχει τα σημεία του D με σκοπό να χωρίσει τα δεδομένα με τέτοιο τρόπο ώστε να μπορούν να είναι διακριτές οι ομάδες δεδομένων που παράγονται μεταξύ τους ώστε να μπορέσουμε να εφαρμόσουμε τοπική

ομαδοποίηση δεδομένων. Κάθε ένα κελί (cell) που παράγεται αποκτά και έναν αριθμό που το αντιπροσωπεύει (ID). Κάθε στοιχείο ανάλογα με τις συντεταγμένες του τοποθετείται σε κάποιο κελί και αποκτά ένα cell ID. Ο αλγόριθμος εκτελείται σε 3 εργασίες.

Εργασία 1.

Η συνάρτηση **map** διαβάζει από το αρχείο τα στοιχεία ένα προς ένα μαζί με τον αριθμό κελιού (ID) που βρίσκονται. Στη συνέχεια γράφει ως κλειδί (key) τον αριθμό κελιού (ID) και ως value τις συντεταγμένες των σημείων. Η **reduce** δημουργεί μια λίστα από τα σημεία που βρίσκονται στο κάθε ένα από τα κελία και παράγει ως έξοδο για τη επομένη συναρτηση των συνδυασμό <ID , list(values(key))>.

Εργασία 2.

Η συνάρτηση **map** δέχεται ως είσοδο την έξοδο της reduce από την εργασία 1 και παίρνει επίσης ως παραμέτρους μια απόσταση αποκοπής (dc) και έναν αριθμό ελαχίστων γειτόνων(minPts) ώστε να μπορέσει να διακρίνει τις ομάδες σημείων. Στη συνέχεια για κάθε ένα σημείο βρίσκει τις αποστάσεις με τους γείτονες του και μετράει πόσοι από αυτούς έχουν απόσταση μικρότερη από την dc. Αν ο αριθμός αυτός είναι μεγαλύτερος από το minPts τότε το σημείο ονομάζεται πυρήνας (core point ή cluster center) και όλα τα στοιχεία με απόσταση μικρότερη από την dc γείτονές του. Ο πυρήνας μαζί με τους γείτονες του αποτελούν ένα σύμπλεγμα γειτόνων (cluster). Ένα στοιχείο μπορεί να ανήκει σε ένα μόνο cluster. Μόλις ολοκληρωθεί η εύρεση των τοπικών cluster ο αλγόριθμος βρίσκει επαναληπτικά όλους τους density-reachable γείτονες του κάθε cluster. Για να συμβεί αυτό , για κάθε ένα από τα cluster βρίσκει τις γειτονιές των γειτόνων του και όσα σημεία ανήκουν σε αυτές , τις προσθέτει στη λίστα με τους γείτονες του συγκεκριμένου cluster. Εάν κάποιος από τους νέους υποψήφιους γείτονες ανήκει σε άλλο cluster τότε ο αλγόριθμος συνενώνει τα δύο αυτά συμπλέγματα σημείων σε ένα. Τέλος γράφει για όλες τις ομάδες σημείων (clusters) ένα κοινό key και για value μια λίστα με πρώτο σημείο τον πυρήνα των σημείων και έπειτα όσα στοιχεία ανήκουν σε αυτό καθώς η **reduce** δημιουργεί μια λίστα με όλα τα clusters που υπάρχουν στο επίπεδο και ένα κλειδί ώστε να μπορέσει η map της επόμενης εργασίας να συνενώσει clusters από διαφορετικά κελιά.

Εργασία 3.

Η **map** συνάρτηση λαμβάνει σαν είσοδο το ζεύγος <key,list(clusters)>. Στη συνέχεια με τη χρήση της dc αποστάσεως βρίσκει ποια core points υπολογίζοντας τις νέες density-reachable γειτονιές έχουν κοινά σημεία και συνενώνει τα clusters αυτά. Τέλος κάθε ξεχωριστό σύμπλεγμα σημείων αποκτά έναν αριθμό cluster (cluster ID) το οποίο χρησιμοποιείται σαν key στην έξοδο της map. Η map για κάθε ένα cluster δημιουργεί τα στοιχεία που ανήκουν σε αυτό και γράφει τον συνδυασμό <clusterID,ListOfItems>. Δεν υπάρχει **reduce.**

6.3 CFSFDP-MR (Clustering by fast search and find of density Peaks)

Ο αλγόριθμος αυτός αποτελείται από 3 εργασίες.

Εργασία 1.

Στην συνάρτηση **map** γίνεται υπολογισμός του πίνακα αποστάσεων μεταξύ των σημείων. Έπειτα για κάθε σημείο υπολογίζεται ποια σημεία βρίσκονται εντός μια απόστασης αποκοπής (dc) -η οποία ορίζεται μέσα στην κλάση της εργασίας 1- και προστίθενται σε μια λίστα. Η συνάρτηση map έχει ως έξοδο για κλειδί (key) των αριθμό των γειτόνων (πυκνότητα) κάθε στοιχείου και σαν τιμή (value) την λίστα με τους γείτονες, πυκνότητα γειτόνων, μεταξύ τους απόσταση. Στην συνάρτηση **reduce** γίνεται χρήση identity reducer για την ταξινόμηση των στοιχείων ανάλογα με την πυκνότητα του καθενός.

Εργασία 2.

Η συνάρτηση **map** κατασκευάζει για κάθε ένα στοιχείο μια λίστα με τους γείτονες που έχουν μεγαλύτερη πυκνότητα από το ίδιο και αποθηκεύει την μικρότερη απόσταση. Σαν κλειδί (key) εξάγει έναν κοινό αριθμό για όλα τα στοιχεία (ώστε να μπορέσουν να γίνουν συγκρίσεις στο επόμενο βήμα) και σαν τιμή (value) γράφει το συνδυασμό <στοιχείο, πυκνότητα, απόσταση από μεγαλύτερη πυκνότητα>. Η **reduce** ομαδοποιεί τα δεδομένα και εξάγει ένα κλειδί (key) και μια λίστα από τους συνδυασμούς της map της εργασίας 2.

Εργασία 3.

Η συνάρτηση **map** δέχεται τις εξόδους από την εργασία 2 και για κάθε στοιχείο ελέγχει τις τιμές της πυκνότητας και της απόστασης από σημείο με μεγαλύτερη πυκνότητα και ορίζει 2 κατηγορίες. Αν το στοιχείο έχει υψηλό δ και σχετικά ψηλό ρ τότε το στοιχείο ονομάζεται cluster center αλλιώς αν το στοιχείο έχει υψηλό δ και χαμηλό ρ τότε ονομάζεται θόρυβος (nose).Για κάθε cluster center η απόσταση από σημείο με μεγαλύτερη πυκνότητα ορίζεται να δείχνει τον εαυτό του. Αφού έχουν οριστεί και αποθηκευτεί τα cluster center για κάθε ένα στοιχείο που δεν ανήκει στις παραπάνω δύο κατηγορίες το κατατάσσει σε ένα cluster με τον παρακάτω τρόπο. Ορίζει ως cluster το οποίο θα το περιλαμβάνει βλέποντας το στοιχείο με την κοντινότερη απόσταση με μεγαλύτερη πυκνότητα που ανήκει και εντάσσεται και αυτό στο ίδιο cluster. Η συνάρτηση **reduce** βάζει σε λίστα όλα τα στοιχεία με το ίδιο key (Cluster Number) και γράφει στο αρχείο <key,list(values)>.

6.4 Transitive heuristic-MR

Ο αλγόριθμος εκτελείται σε μια εργασία. Η συνάρτηση **map** διαβάζει ως είσοδο ένα στοιχείο με n attributes κάθε φορά. Με τη χρήση του κριτηρίου Condorcet ψάχνει να δει εάν υπάρχει κάποιο cluster center που μπορεί να το εντάξει στην ομάδα του. Αν δεν υπάρχει δημιουργείται ένα νέο cluster center με attributes ίσα με αυτά του στοιχείου. Η map έχει ως έξοδο το cluster center attributes ως κλειδί, και σαν value τα attributes του κάθε νέου στοιχείου.

Κριτήριο Condorcet: Για κάθε νέο στοιχείο που διαβάζει η **map** δημιουργεί έναν μετρητή ίσο με το μηδέν. Στη συνέχεια ψάχνει στη λίστα με τα cluster centers και υπολογίζει κάθε φορά το άθροισμα του Condorcet. Για τον υπολογισμό του αθροίσματος, συγκρίνει τα attributes του νέου στοιχείου με αυτά των cluster centers. Αν τα attributes είναι ίδια τότε το άθροισμα αυξάνεται κατά ένα, αν όχι τότε μειώνεται κατά ένα. Τέλος, αν το άθροισμα είναι μεγαλύτερο του μηδέν τότε το νέο στοιχείο μπορεί να εισαχθεί στη λίστα του συγκεκριμένου cluster centers ειδάλλως όχι. Αφού έχει υπολογίσει όλα τα αθροίσματα ψάχνει αυτό με την μεγαλύτερη θετική τιμή και εισάγει το στοιχείο στο συγκεκριμένο cluster. Αν όλα τα αθροίσματα είναι μικρότερα του μηδέν τότε δημιουργεί νέο cluster και τα attributes του cluster center είναι του στοιχείου που μόλις διαβάστηκε. Η **reduce** ομαδοποιεί τα δεδομένα με ίδια cluster attributes και εξάγει για κλειδί το cluster attributes και σαν τιμή (value) την λίστα με τα στοιχεία που ανήκουν σε αυτό το cluster, list(elements).

7. Αποτίμηση επίδοσης των αλγορίθμων

Κάθε ένας από τους παραπάνω αλγορίθμους εξετάστηκε σε διαφορετικά σύνολα δεδομένων με απώτερο σκοπό την εξαγωγή συμπερασμάτων σχετικά με τους χρόνους εκτέλεσης ανάμεσα σε HDD και SSD. Οι αλγόριθμοι που ασχολούνται με αριθμητικά δεδομένα εξετάστηκαν σε πέντε σύνολα δεδομένων ενώ ο αλγόριθμος που ασχολείται με σχεσιακά δεδομένα εξετάστηκε σε τρία.

7.1 Περιγραφή των Dataset και ανάλυση συστήματος λειτουργίας

Πίνακας 1: Προδιαγραφές υπολογιστή.

4 CPU	Intel® Xeon® W3550 @3.07 GHz
RAM	4 GB
DISK 1(HDD)	100GB Samsung 850 Evo
DISK 2(SSD)	100GB Seagate ST31000524AS

Πίνακας 2: Εγκατεστημένο Λογισμικό

OS	Centos 7
Java SDK	Oracle Java 1.8.0
Hadoop version	Hadoop-3.1

Πίνακας 3 Περιγραφή των αριθμητικών Dataset

Dataset 1	Size : 500 items	Number of Attributes : 2
Dataset 2	Size : 1500 items	Number of Attributes: 2
Dataset 3	Size: 4000 items	Number of Attributes: 2

Τα τρία παραπάνω Dataset έχει παραχθεί με την εντολή make_blobs της python. Κύριο χαρακτηριστικό αυτής της εντολής αποτελεί η κανονική κατανομή που έχουν τα παραγόμενα δεδομένα και η παράμετρος για τον αριθμό των κέντρων που θέλει ο χρήστης. Με τη χρήση αριθμού κέντρων ο χρήστης έχει την δυνατότητα να συγκεντρώσει τα

δεδομένα του γύρω από n σημεία το οποίο ενδείκνυται για λειτουργίες clustering.
Για size = 500 items χρησιμοποιήθηκαν n=4 αριθμός κέντρων.
Για size = 1500 items ορίσαμε των αριθμών των κέντρων n=6. Ώστε τα δεδομένα να έχουν μια ικανοποιητική διασπορά στο επίπεδο. Τέλος για size = 4000 θέσαμε n=10 ώστε να μην είναι τα δεδομένα μας σε πολύ μικρές αποστάσεις

Dataset 4	Size: 22000 items	Number of Attributes: 2
Dataset 5	Size: 30000 items	Number of Attributes: 2

Τα δύο αυτά Dataset αποτελούν μέρος του Birch Dataset που υπάρχει διαθέσιμο online. (Source : [20]). Αποτελείται από δύο συντεταγμένες και είναι ένα ενδεικτικό Dataset για clustering με διάφορα μεγέθη.

Πίνακας 4 Περιγραφή των σχεσιακών δεδομένων

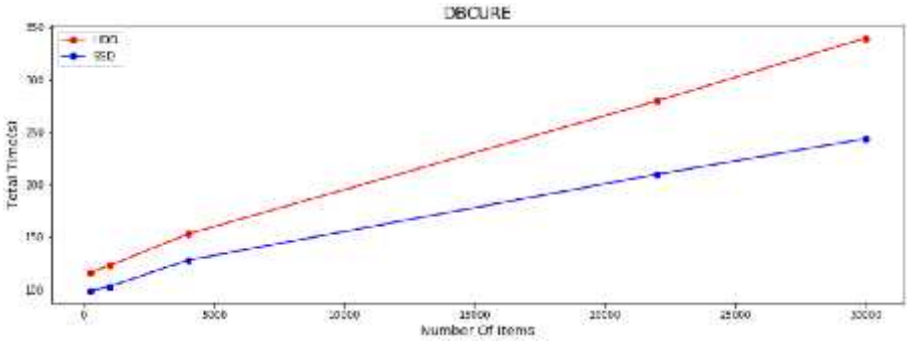
Dataset 1	Size: 8124 items	Number of Attributes: 22
Dataset 2	Size: 15000items	Number of Attributes: 22
Dataset 3	Size: 22000items	Number of Attributes: 22

Τα σχεσιακά δεδομένα που χρησιμοποιήθηκαν για την αξιολόγηση του Transitive Heuristic-MR ακολουθούσαν το μοτίβο του Dataset Mushroom που υπάρχει σε πηγή δεδομένων στο [21]. Το συγκεκριμένο Dataset έχει Number of Attributes = 22. Για να παράγουμε μεγαλύτερου μεγέθους σύνολα δεδομένων κάναμε χρήση της python και ως παραμέτρους βάλαμε τα σύνολα τιμών που παίρνει το εκάστοτε Attribute.

7.2 Επιδόσεις αλγορίθμων

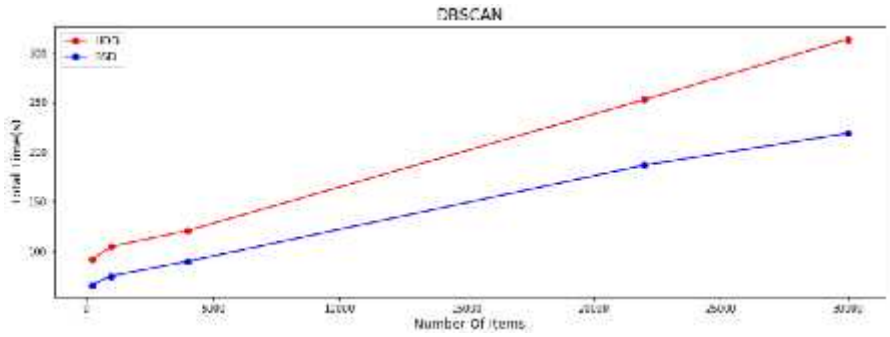
Ως μετρική απόδοσης αλγορίθμων χρησιμοποιήθηκε ο συνολικός χρόνος που χρειάστηκε ο κάθε αλγόριθμος για να ομαδοποιήσει τα εκάστοτε δεδομένα.

Ξεκινώντας από των DBCURE μπορούμε εύκολα να διακρίνουμε από το παρακάτω διάγραμμα ότι ο χρόνος εκτέλεσης ανάμεσα στους δύο δίσκους έχει σημαντικές διαφορές. Από την Εικόνα 5 είναι εμφανές ότι όσο ο αριθμός των στοιχείων μεγαλώνει , η διαφορά ανάμεσα στους χρόνους εκτέλεσης μεγαλώνει επίσης. Από τις μετρήσεις που παράχθηκαν, βγαίνει το συμπέρασμα ό,τι η εκτέλεση του DBCURE σε SSD είναι από 15% έως 30% πιο γρήγορη από ότι σε HDD.



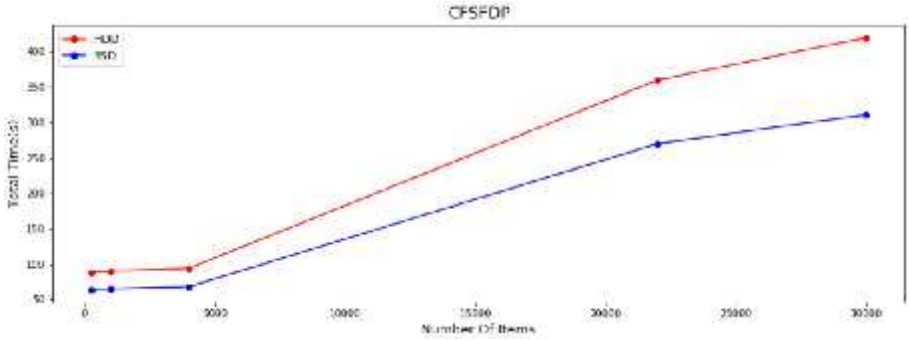
Εικόνα 8: Επιδόσεις του αλγορίθμου DBCURE σε HDD και SSD.

Επόμενος αλγόριθμος ο οποίος εξετάστηκε είναι ο DBSCAN. Παρόμοια με τον DBCURE, ο DBSCAN επίσης παρουσίασε μεγάλες διαφορές στην εκτέλεση του όταν αυτή γινόταν σε HDD και SSD. Αυτή τη φορά βλέπουμε ό,τι οι διαφορές είναι πιο έντονες σε σχέση με πριν. Το εύρος τους κυμαίνεται από 25% έως 30%. Είναι άξιο να σημειωθεί ό,τι η διαφορά στους χρόνους εκτέλεσης μεγάλωνε όσο μεγάλωνε ο αριθμός των σημείων δεδομένων.



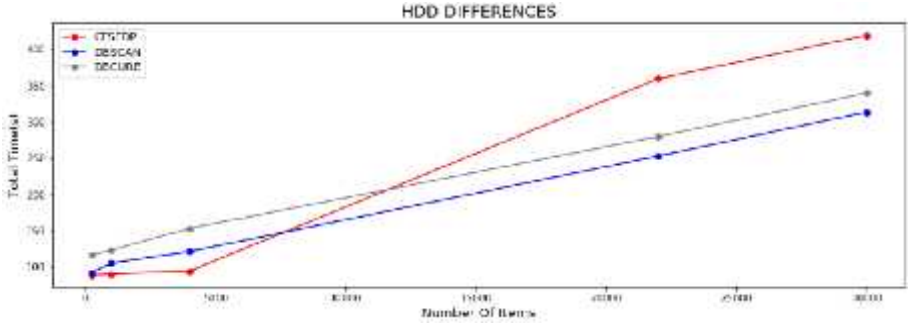
Εικόνα 9: Επιδόσεις του αλγορίθμου DBSCAN σε HDD και SSD.

Τέλος, τελευταίος αλγόριθμος που εξετάστηκε για αριθμητικά δεδομένα είναι ο Clustering by fast search and find of density Peaks. Τα αποτελέσματα του παρουσιάζονται από κάτω. Ο συγκεκριμένος αλγόριθμος ενώ για μικρά σύνολα σημείων είχε καλές επιδόσεις, οσο αυτά έτειναν να μεγαλώνουν τόσο μεγάλωνε και ο χρόνος που χρειαζόνταν να τα ομαδοποιήσει. Για αυτό οφείλεται κυρίως η κατασκευή του πίνακα αποστάσεων στην πρώτη εργασία του κώδικα. Τέλος, όπως οι άλλοι δύο αλγόριθμοι έτσι και αυτός παρουσιάζει μια μείωση από 26% με 28% στον χρόνο εκτέλεσης του ανάμεσα σε HDD και SSD.

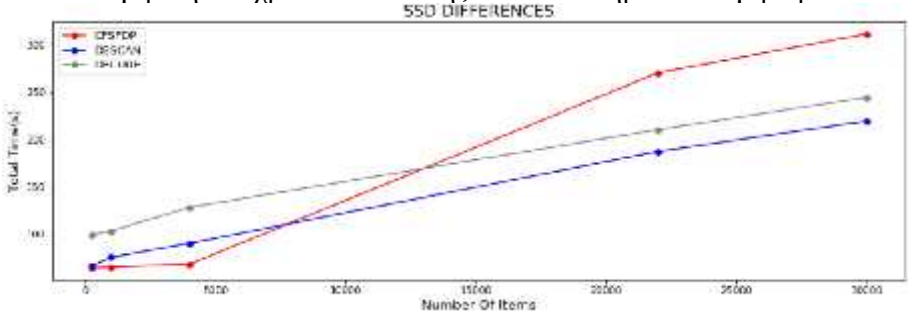


Εικόνα 10: Επιδόσεις του αλγορίθμου CFSFDP σε HDD και SSD.

Έπειτα προχωρήσαμε στη σύγκριση των τριών αυτών αλγορίθμων. Όπως φαίνεται και από τα δύο επόμενα σχεδιαγράμματα πιο γρήγορος αλγόριθμος για δεδομένα μικρής κλίμακας και στις δύο περιπτώσεις αποτελεί ο CFSFDP, ενώ πιο αργός ο DBCURE. Όσο ο αριθμός των δεδομένων μεγαλώνει, μπορούμε εύκολα να διακρίνουμε ό,τι βέλτιστη επιλογή αποτελεί ο DBSCAN ενώ δεύτερος ακολουθεί ο DBCURE. Η παραπάνω παρατήρηση αποτελεί λογική μιας και ο CFSFDP είναι ο μόνος αλγόριθμος ο οποίος χρειάζεται να κατασκευάσει ολοκλήρο τον πίνακα αποστάσεων. Επίσης, για τους κακούς σχετικά χρόνους του DBCURE σε σχέση με τον DBSCAN (ακολουθούν την ίδια φιλοσοφία) αποτελεί ο μεγάλος αριθμός των εργασιών. Το σύστημα κάνει ανάγνωση και εγγραφή δεδομένων δύο φορές παραπάνω από ότι χρειάζεται ο DBSCAN πράγμα χρονοβόρο όταν τα δεδομένα είναι μεγάλου όγκου. Τέλος είναι σημαντικό να τονιστεί πως ο CFSFDP είναι ο μόνος αλγόρ ιθμος ο οποίος δεν υποβάλει τα δεδομένα σε καποιού είδους προεπεξεργασία αντίθετα με τους άλλους δύο.

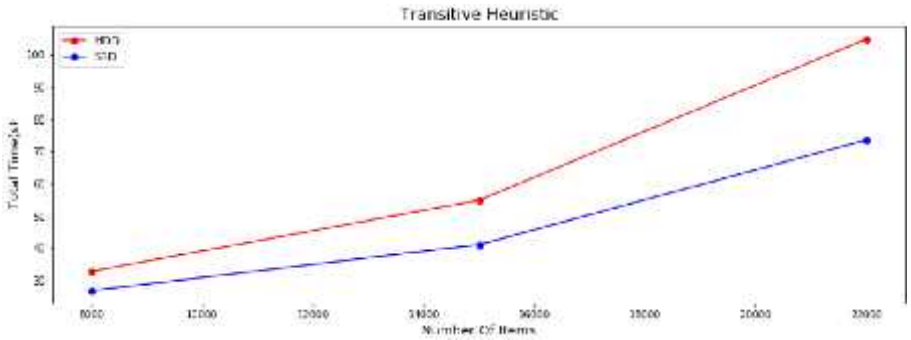


Εικόνα 11: Σύγκριση του χρόνου εκτέλεσης των υλοποιημένων αλγορίθμων σε HDD.



Εικόνα 12: Σύγκριση του χρόνου εκτέλεσης των υλοποιημένων αλγορίθμων σε SSD.

Τέλος ο μοναδικός αλγόριθμος που κατασκευάστηκε για την ομαδοποίηση σχεσιακών δεδομένων είναι ο Transitive Heuristic. Τα αποτελέσματα του και οι επιδόσεις του σε HDD και SSD παρουσιάζονται παρακάτω. Όπως φαίνεται από την γραφική παράσταση, πάλι η εκτέλεση ενός αλγορίθμου σε SSD είναι γρηγορότερη από ότι σε HDD. Στην περίπτωση ειδικά του συγκεκριμένου αλγορίθμου η διαφόρα στις επιδόσεις αυξήθηκε όταν το μέγεθος των δεδομένων μεγάλωνε.



Εικόνα 13: Επιδόσεις του αλγορίθμου Transitive Heuristic σε HDD και SSD.

8. Συμπεράσματα και προτάσεις για μελλοντική έρευνα

Ο στόχος της διπλωματικής εργασίας ήταν να διευκρινιστούν οι διαφορές στις επιδόσεις αλγορίθμων ομαδοποίησης που βασίζονται στην πυκνότητα κατά την εκτέλεση τους σε HDD και SSD. Για το σκοπό αυτό, αρχικά μελετήθηκαν οι σειριακής μορφής αλγόριθμοι με τα παραπάνω χαρακτηριστικά. Έπειτα εφόσον είχε γίνει κατανοητή η λειτουργία τους για μικρού όγκου δεδομένα, έγινε η αντίστοιχη υλοποίηση σε μορφή MapReduce (παράλληλη , κατανεμημένη υλοποίηση). Για να επιτευχθεί αυτό, χρησιμοποιήθηκαν τα εργαλεία του Hadoop και της Java. Όλες οι παραπάνω ενέργειες έκαναν δυνατή την διεξαγωγή των πειραμάτων που είχαν τεθεί σαν κύριος στόχος. Χρησιμοποιώντας τον συνολικό χρόνο ως μέτρο σύγκρισης της απόδοσης των αλγορίθμων και εκτελώντας τους αλγορίθμους που υλοποιήθηκαν για διαφόρου όγκου δεδομένα αναδείχθηκαν τα εξής συμπεράσματα. Πρωτίστως, όσοι αλγόριθμοι κατασκευάστηκαν, έδειξαν κατά την εκτέλεση τους σε SSD μια βελτίωση στο χρόνο από 15% έως 30% είτε η ομαδοποίηση αφορούσε αριθμητικά δεδομένα είτε σχεσιακά. Έπειτα, όσοι αλγόριθμοι εξειδικεύονται σε αριθμητικά δεδομένα και απαιτούν την προεπεξεργασία τους, τείνουν να έχουν καλύτερες επιδόσεις σε μεγάλου όγκου δεδομένα σε σχέση με αυτούς που τα χρησιμοποιούν ατόφια. Παρόλα αυτά όταν η κλίμακα των δεδομένων είναι μικρή η χρήση αλγορίθμων που δεν απαιτούν προεπεξεργασία τείνει να αποδίδει καλύτερα αποτελέσματα στο κομμάτι του χρόνου εκτέλεσης. Επιπλέον, παρατηρήθηκε ότι όσο ο αριθμός των στοιχείων προς επεξεργασία αυξανόταν, τόσο η διαφορά στις επιδόσεις ανάμεσα στους δύο δίσκους έτεινε να αυξάνει ενώ όσο ο όγκος των δεδομένων παρέμενε μικρός οι δύο δίσκοι φάνηκαν να παρουσιάζουν τις μικρότερες χρονικά διαφορές. Επιπλέον παρατηρήθηκε ότι οι αλγόριθμοι που αποτελούνταν από μεγάλο αριθμό εργασιών όπως ο DBCURE-MR υστερούσαν σε ταχύτητα σε σχέση με άλλους αλγόριθμους ίδια φιλοσοφίας όπως ο DBSCAN-MR. Αυτό οφείλεται κυρίως στην συνεχή εγγραφή κ ανάγνωση δεδομένων που χρειάζεται να κάνει ο υπολογιστής. Αξίζει βέβαια να σημειωθεί πως λόγο των πολλών εργασιών ο συγκεκριμένος αλγόριθμος είχε μεγαλύτερες χρονικές διαφορές από όλους τους ανάμεσα σε SSD και HDD. Αυτό συνέβαινε επειδή σε κάθε εργασία του αλγορίθμου τα map και reduce task ολοκληρωνόντουσαν πιο γρήγορα στον SSD από ότι στον HDD με αποτέλεσμα να αυξάνουν την χρονική διαφορά.

Οι προτάσεις για μελλοντική ερεύνα συνοψίζονται στα εξής:

1. Δημιουργία περισσότερων αλγορίθμων που ομαδοποιούν σχεσιακά δεδομένα για συγκρίσεις αποδόσεων.
2. Συγκριτική μελέτη όλων των δυνατών υλοποιήσεων για την επιλογή της βέλτιστης.

3. Δημιουργία Hadoop cluster με προδιαγραφές για την εφαρμογή των αλγορίθμων σε δεδομένα μεγάλου όγκου.

Βιβλιογραφία

1. Ghazi, M.R. and D. Gangodkar, *Hadoop, MapReduce and HDFS: A Developer's Perspective*, in *International Conference on Intelligent Computing, Communication & Convergence*. 2015, Elsevier: Bhubaneswar, Odisha, India. p. 45-50.
2. https://www.tutorialspoint.com/.
3. www.storegaga.com.
4. Dean, J. and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters.* ACM, 2008. **51**(1): p. 107-113
5. Shvachko, K., et al., *The Hadoop Distributed File System* in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 2010, IEEE: Incline Village, NV, USA. p. 1-10
6. Vavilapalli, V.K., et al., *Apache Hadoop YARN: Yet Another Resource Negotiator*, in *4th annual Symposium on Cloud Computing*. 2013, ACM: Santa Clara, California.
7. Ghuman, S.S., *Clustering Techniques- A Review* International Journal of Computer Science and Mobile Computing, 2016. **Vol. 5**(5): p. 524-530.
8. Garima, H. Gulati, and P.K.Singh, *Clustering Techniques in Data Mining: A Comparison* in *2nd International Conference on Computing for Sustainable Global Development*. 2015, IEEE: New Delhi, India.
9. Jain, A.K. and R.C. Dubes, *Algorithms for clustering data*. 1988: Prentice Hall PTR.
10. Gandhi, G. and R. Srivastava, *Review Paper: A Comparative Study on Partitioning Techniques of Clustering Algorithms.* IJCA, 2014. **87**: p. 10-13.
11. Kasavajhala, V., *Solid State Drive vs. Hard Disk Drive Price and Performance Study*. 2011: Dell PowerVault Storage Systems.
12. www.java-performance.info.
13. https://blog.westerndigital.com.
14. M.Bakratsas, et al., *Hadoop MapReduce Performance on SSDs for Analyzing Social Networks.* Big Data Research, 2017. **11**: p. 1-10.
15. Kambatla, K. and Y. Chen, *The Truth About MapReduce Performance on SSDs*, in *28th Large Installation System Administration Conference*. 2014: Seattle, WA.
16. He, Y., et al., *MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data.* Frontiers of Computer Science, 2014. **8**(1): p. 83-99.
17. Pang, L. and F.A. Liu, *Density-Based Algorithm in MapReduce*, in *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*. 2016. p. 394-397.
18. Kim, Y., et al., *DBCURE-MR:An efficient density-based clustering algorithm for large data using MapReduce.* Information Systems 2013. **42**: p. 15-35.
19. Lamari, Y. and S.C. Slaoui, *Clustering categorical data based on the relational analysis approach and MapReduce.* Journal of Big Data, 2017. **4**(1): p. 28.
20. https://cs.joensuu.fi/sipu/datasets/.
21. https://archive.ics.uci.edu/ml/datasets.html.