# Μεταπτυχιακή Διπλωματική Εργασία

# Υπολογισμός ανάστροφων ερωτημάτων κατάταξης σε κατανεμημένο περιβάλλον

Γεώργιος Σφυρής - ΜΕ14032

Επιβλέπων καθηγητής: Χρήστος Δουλκερίδης

**Πειραιάς 2016**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

ΤΜΗΜΑ: ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ: ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ & ΥΠΗΡΕΣΙΕΣ

ΚΑΤΕΥΘΥΝΣΗ: ΔΙΚΤΥΟΚΕΝΤΡΙΚΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

## Περίληψη

Τα τελευταία χρόνια προτάθηκε ένας νέος τύπος ερωτήματος, το Reverse Top-k ερώτημα. Το ερώτημα αυτό προσέλκυσε το ενδιαφέρον τόσο της ερευνητικής όσο και της επιχειρηματικής κοινότητας. Το ερώτημα αυτό μπορεί να εντοπιστεί για παράδειγμα σε εφαρμογές επιχειρηματικής ευφυΐας καθώς θα μπορούσε να χρησιμοποιηθεί για να προβλέψει την απήχηση ενός προϊόντος στο καταναλωτικό κοινό. Από την άλλη πλευρά τα μεγάλα δεδομένα αποτελούν έναν τομέα με πολύ μεγάλη δραστηριότητα, τόσο από τον επιχειρηματικό όσο και από τον ερευνητικό χώρο καθώς η παραγωγή δεδομένων τόσο από επιχειρήσεις όσο και ατομικά είναι πρωτόγνωρη. Έτσι σε αυτή την εργασία αντιμετωπίζεται το πρόβλημα της επεξεργασίας των Reverse Top-k ερωτημάτων με παράλληλο και κατανεμημένο τρόπο. Αξίζει να σημειωθεί ότι το πρόβλημα αυτό δεν έχει αντιμετωπιστεί ακόμα. Ακόμα για την παράλληλη και κατανεμημένη επεξεργασία δεδομένων χρησιμοποιήθηκε το MapReduce framework καθώς αποτελεί έναν από τους πιο δημοφιλείς τρόπους για τέτοιου είδους επεξεργασία. Στην εργασία αυτή, αρχικά προτείνονται δύο αλγόριθμοι για την αντιμετώπιση του προβλήματος εκ των οποίων ο ένας αποδεικνύεται καλύτερος καθώς πετυχαίνει χαμηλότερους χρόνους απόκρισης και μπορεί να διαχειριστεί μεγάλους όγκους δεδομένων. Κάτι που προκύπτει από το μεγάλο πλήθος πειραμάτων που εκτελέστηκαν κατά την διάρκεια της εργασίας αυτής. Ο αλγόριθμος ο οποίος προτείνεται βασίζεται σε ιδιότητες οι οποίες παρουσιάζονται κατά την διάρκεια της εργασίας και αποτελεί μία πρώτη προσπάθεια για την αντιμετώπιση του προβλήματος της κατανεμημένης και παράλληλης επεξεργασίας των Reverse Top-k ερωτημάτων.



## Abstract

The last years has been proposed a new type of query, the Reverse Top-k query. This query has attracted the interest of both the research and the business community. This query can be found for example in business intelligence applications as could be used in order to predict the impact of a product to consumers. On the other hand, big data is an area with great activity, both from the business and the research area, as the data production from both business and individual is unprecedented. Thus in this thesis addressed the problem of processing Reverse Top-k query with parallel and distributed way. It is worth mentioning that this problem has not been tackled yet. Furthermore about the parallel and distributed process it used the MapReduce framework as is one of the most popular ways for such processing. In this thesis, initially two algorithms proposed to address this problem of which one proved better as achieves lower response times and can handle large volumes of data. Something resulting from the large number of experiments which performed during this work. The proposed algorithm is based on properties which are presented during this thesis and is a first attempt to address the problem of processing the Reverse Top-k query with parallel and distributed way.

# Πίνακας περιεχομένων

Περίληψη ..... 1
Abstract ..... 1
1. Εισαγωγή ..... 5
1.1 Περιγραφή ..... 5
1.2 Εισαγωγή ..... 5
1.3 Top-k ερωτήματα ..... 7
1.4 Reverse Top-k ερωτήματα ..... 10
1.5 Skyline ερωτήματα ..... 11
1.6 Map Reduce (Hadoop) ..... 12
1.6.1 HDFS (Hadoop Distributed File System) ..... 12
1.6.2 Μοντέλο Map Reduce ..... 14
1.6.3 Λεπτομέρειες MapReduce στο Hadoop ..... 15
2. Αποτύπωση του προβλήματος ..... 18
2.1 Προβληματική κατάσταση ..... 18
2.2 Βασικές ιδιότητες ..... 21
2.2.1 Περικοπή βάση κυριαρχίας ..... 21
2.2.2 Περικοπή Weighting Vector ..... 22
2.3 Απλοϊκός αλγόριθμος και ορθότητα ..... 22
3. Προχωρημένες Ιδιότητες ..... 24
3.1 Ομαδοποίηση και περικοπή δεδομένων ..... 24
3.1.1 Ομαδοποίηση weighting vectors ..... 24
3.1.2 Περικοπή στοιχείων ..... 26
3.2 Απόρριψη Weighting Vectors ..... 28
4. Reverse Top-k αλγόριθμοι ..... 31
4.1 RTA αλγόριθμος ..... 31
5. Απλοϊκός αλγόριθμος ..... 33
5.1 Map φάση ..... 33
5.2 Reduce φάση ..... 35
5.3 Ανάλυση Πολυπλοκότητας ..... 36
5.4 Περιγραφή υλοποίησης ..... 36
5.4.1 Βασικά χαρακτηριστικά ..... 36
5.4.2 Περιγραφή υλοποίησης των στοιχείων του Hadoop ..... 37
5.4.3 Στοιχεία μέτρησης ..... 40

6. Σύνθετος Αλγόριθμος ..... 42
6.1 Ομαδοποίηση weighting vectors ..... 42
6.1.1 Κατασκευή ..... 42
6.1.2 Πλέγμα μη υπαρκτών ορίων (Not Real Bounds Grid) ..... 43
6.2 Αλγόριθμος περικοπής στοιχείων S ..... 44
6.2.1 ExtremeScore ..... 44
6.2.2 R-Lists ..... 46
6.2.3 ExtremeScore & R-Lists ..... 47
6.3 Πλέγμα S ..... 49
6.3.1 Κατασκευή ..... 49
6.3.2 Simple ..... 50
6.3.3 Summarized ..... 52
6.3.4 Tree ..... 54
6.3.5 Summarized Tree ..... 58
6.4 Αλγόριθμος περικοπής W ..... 62
6.5 Map φάση ..... 63
6.6 Reduce φάση ..... 64
6.7 Περιγραφή υλοποίησης ..... 65
6.7.1 Βασικά χαρακτηριστικά ..... 65
6.7.2 Περιγραφή υλοποίησης των στοιχείων του Hadoop ..... 65
6.7.3 Στοιχεία μέτρησης ..... 68
7. Πειραματική Αξιολόγηση ..... 70
7.1 Περιβάλλον ..... 70
7.1.1 Υπολογιστικό περιβάλλον ..... 70
7.1.2 Σύνολα δεδομένων ..... 71
7.2 Σύγκριση πλεγμάτων Gs ..... 72
7.3 Σύγκριση αλγορίθμων Περικοπής στοιχείων S ..... 74
7.4 Σύγκριση Απλοϊκού και Σύνθετου αλγορίθμου ..... 77
7.5 Ανάλυση ευαισθησίας του Σύνθετου αλγορίθμου ..... 82
7.5.1 Πειράματα μεταβολής μεγέθους S ..... 83
7.5.2 Πειράματα μεταβολής μεγέθους W ..... 88
7.5.3 Πειράματα μεταβολής μεγέθους k ..... 93
7.5.4 Πειράματα μεταβολής κατανομής S ..... 98
7.5.5 Πειράματα μεταβολής των διαστάσεων ..... 103

8. Σχετικές έρευνες ..... 108
9. Συμπεράσματα ..... 110
Παράρτημα ..... 111
Περιβάλλον εκτέλεσης αλγορίθμων ..... 111
Εγχειρίδιο χρήσης Απλού αλγορίθμου ..... 111
Εγχειρίδιο χρήσης Σύνθετου αλγορίθμου ..... 112
Βιβλιογραφία ..... 114

# 1. Εισαγωγή

## 1.1 Περιγραφή

Στην εργασία αυτή αντιμετωπίζεται το πρόβλημα υπολογισμού των Reverse Top-k ερωτημάτων σε κατανεμημένο περιβάλλον. Δοσμένης μιας βάσης δεδομένων με αντικείμενα, προτιμήσεις χρηστών και δοσμένου ενός ερωτήματος q, το Reverse Top-k ερώτημα θα επιστρέψει ένα υποσύνολο των προτιμήσεων των χρηστών για τα οποία το ερώτημα q ανήκει στα Top-k αποτελέσματά τους. Παρόλο που το Reverse TopK ερώτημα έχει προσελκύσει το ενδιαφέρον και έχει πολλές εφαρμογές στην πραγματική ζωή, ο υπολογισμός αυτού σε κατανεμημένο περιβάλλον, για μεγάλα σύνολα δεδομένων δεν έχει αντιμετωπιστεί ακόμα.

Σε αυτή την εργασία θα προταθούν δύο αλγόριθμοι για την επίλυση του υπολογισμού Reverse Top-k ερωτημάτων σε κατανεμημένο περιβάλλον. Ο πρώτος αλγόριθμος αφορά μια απλή λύση που στηρίζεται σε πολύ βασικές ιδιότητες. Αντίθετα ο δεύτερος αλγόριθμος που θα προταθεί εκμεταλλεύεται πιο προχωρημένες ιδιότητες. Έπειτα αξιολογώντας τους αλγορίθμους αυτούς και συγκρίνοντάς τους βάση των πειραμάτων που εκτελέστηκαν, προκύπτει ότι ο δεύτερος αλγόριθμος είναι καλύτερος από τον πρώτο καθώς αντιμετωπίζει καταλληλότερα το πρόβλημα.

Στην συνέχεια θα ακολουθήσει η περιγραφή των βασικών στοιχείων που χρειάζονται για την κατανόηση του προβλήματος. Έπειτα θα αποτυπωθεί το πρόβλημα με έναν πιο επίσημο τρόπο. Μετά θα παρουσιαστούν τόσο οι βασικές όσο και οι προχωρημένες ιδιότητες. Αντίστοιχα θα παρουσιαστούν ο απλοϊκός και ο σύνθετος αλγόριθμος με λεπτομερή περιγραφή. Ύστερα θα ακολουθήσει η πειραματική αξιολόγηση, ενώ στο τέλος θα γίνει μια αναφορά σε σχετικές έρευνες καθώς και σε μελλοντική έρευνα με προτάσεις διαφορετικών τρόπων αντιμετώπισης του προβλήματος.

## 1.2 Εισαγωγή

Στην σημερινή εποχή η εξατομίκευση των εφαρμογών έχει προσελκύσει το ενδιαφέρον τόσο της ερευνητικής όσο και της επιχειρηματικής κοινότητας. Για παράδειγμα είναι το στοιχείο κλειδί για εφαρμογές όπως κοινωνικά δίκτυα (Social Networks) και συστήματα συστάσεων (Recommendation Systems). Αξίζει να σημειωθεί ότι πολλές επιχειρήσεις έχουν δημιουργηθεί για τον σκοπό αυτό, προκειμένου να προτείνουν στον χρήστη προϊόντα τα οποία τον ενδιαφέρουν. Για να πετύχουν αυτό πρέπει να γνωρίζουν τις προτιμήσεις των χρηστών. Μια συνηθισμένη τακτική εκμαίευσης των προτιμήσεων των χρηστών είναι μέσα από τις αναζητήσεις που αυτοί πραγματοποιούν. Για παράδειγμα ένα σύστημα συστάσεων ξενοδοχείων θα μπορούσε ανάλογα με το ιστορικό αναζήτησης ενός χρήστη να δημιουργήσει ένα προφίλ σχετικά με τις προτιμήσεις αυτού. Το οποίο θα περιέγραφε ότι τον ενδιαφέρει κατά 20% η τιμή ενός δωματίου, κατά 50% η αξιολόγησή του από άλλους χρήστες και κατά 30% τα αστέρια του ξενοδοχείου στο οποίο ανήκει. Γνωρίζοντας τις προτιμήσεις των χρηστών μια εφαρμογή καλείται να προσαρμόσει την συμπεριφορά της στον εκάστοτε χρήστη, ο πιο δημοφιλής τρόπος για την απάντηση ερωτημάτων με επίγνωση των προτιμήσεων ενός χρήστη ή ερωτημάτων σχετικά με την κατάταξη είναι το Top-k ερώτημα, το οποίο βρίσκεται στα περισσότερα σύγχρονα συστήματα.

Ένα Top-k ερώτημα βασίζεται σε μία συνάρτηση βαθμολόγησης η οποία παίρνει ως παραμέτρους τις προτιμήσεις του χρήστη σε μορφή ερωτήματος και βαθμολογεί κάθε στοιχείο. Έπειτα θα επιστρέψει στον χρήστη τα k στοιχεία με την καλύτερη βαθμολογία. Η επιλογή κάποιου στοιχείου από τον χρήστη σε σχεδόν όλες τις περιπτώσεις εξαρτάται από τα Top-k στοιχεία καθώς κατά πάσα πιθανότητα θα είναι ένα από αυτά. Για παράδειγμα ένα ξενοδοχείο που εμφανίζεται στο Top-k για την αναζήτηση ενός χρήστη έχει υψηλές πιθανότητες να επιλεγεί από αυτόν. Οι επιχειρήσεις από την άλλη πλευρά όταν πρόκειται να δημιουργήσουν ένα προϊόν πραγματοποιούν έρευνες σχετικά με την απήχηση του προϊόντος στο καταναλωτικό κοινό. Για παράδειγμα για την κατασκευή μιας ξενοδοχειακής μονάδας θα μελετηθεί η ζήτηση των τουριστών για κάποιες περιοχές έτσι ώστε να επιλεγεί αυτή που θα αποδώσει περισσότερο κέρδος. Οπότε η μελέτη αυτή θα μπορούσε να γίνει αναζητώντας σε πόσα Top-k ερωτήματα χρηστών θα άνηκε η ξενοδοχειακή μονάδα αν υπήρχε σε κάθε μια περιοχή. Όποια περιοχή είχε το μεγαλύτερο πλήθος Top-k ερωτημάτων θα είχε αντίστοιχα μεγαλύτερο ενδιαφερόμενο κοινό. Από τα παραπάνω προκύπτει το Reverse Top-k ερώτημα.

Ένα Reverse Top-k ερώτημα απαντά σε ποιά Top-k ερωτήματα θα άνηκε ένα στοιχείο. Αυτός ο τύπος ερωτήματος είναι στραμμένος περισσότερο προς την επιχειρηματική αναλυτική (Business Analytics) παρά για να εξυπηρετήσει ανάγκες των χρηστών. Όπως προαναφέρθηκε μια επιχείρηση θα μπορούσε να μελετήσει την απήχηση ενός προϊόντος. Σήμερα τα δεδομένα των επιχειρήσεων είναι πολύ μεγάλα για να επεξεργαστούν από έναν υπολογιστή και ο ρυθμός αύξησής τους μεγαλώνει διαρκώς. Οπότε κρίνεται αναγκαία η ύπαρξη ενός αλγορίθμου ο οποίος να μπορεί να επεξεργάζεται Reverse Top-k ερωτήματα σε κατανεμημένο περιβάλλον. Αυτός είναι και ο στόχος της παρούσας εργασίας.

| Top-k | | | |
|---|---|---|---|
| **Top-5** | **W1** | **W2** | **W3** |
| **1ο** | S3 | S7 | S3 |
| **2ο** | S1 | S3 | S10 |
| **3ο** | S5 | S9 | S7 |
| **4ο** | S8 | S1 | S6 |
| **5ο** | S10 | S4 | S9 |

*Εικόνα 1: Στην εικόνα αυτή παρουσιάζονται τα στοιχεία τα οποία ανήκουν στα Top-k ερωτήματα των W1, W2, W3. Και πιο συγκεκριμένα εμφανίζεται το Top-5 αυτών. Με W συμβολίζονται οι προτιμήσεις των χρηστών ενώ με S τα στοιχεία.*

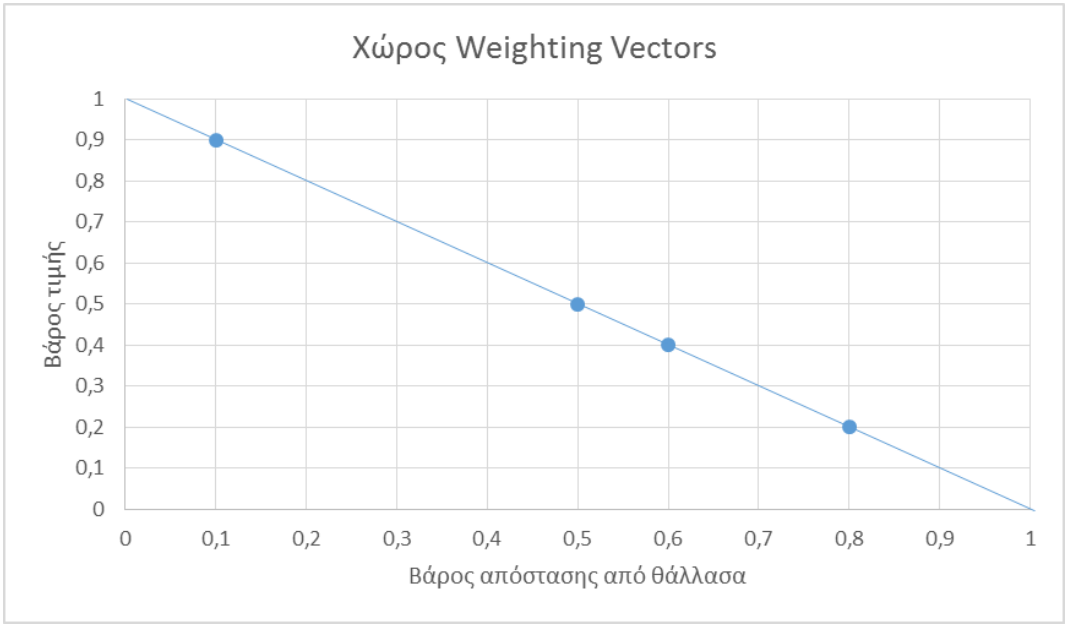
| Reverse Top-k | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **S1** | **S2** | **S3** | **S4** | **S5** | **S6** | **S7** | **S8** | **S9** | **S10** |
| W1 | | W1 | W2 | W1 | W3 | W2 | W1 | W2 | W1 |
| W2 | | W2 | | | | W3 | | W3 | W3 |
| | | W3 | | | | | | | |

*Εικόνα 2: Στην εικόνα αυτή παρουσιάζονται τα αποτελέσματα των Reverse Top-k ερωτημάτων για όλα τα στοιχεία της Εικόνα 1. Όπως παρατηρείται το στοιχείο S3 ανήκει στα περισσότερα Top-k ενώ το στοιχείο S2 ανήκει στα λιγότερα Top-k.*

## 1.3 Top-k ερωτήματα

Ένα Top-k ερώτημα καθορίζεται από μία συνάρτηση βαθμολόγησης $f$ η οποία βαθμολογεί ένα στοιχείο λαμβάνοντας υπόψη τις τιμές αυτού σε κάθε διάσταση. Έτσι με μία γενική βαθμολογία ταξινομούνται τα στοιχεία, όπου τα πρώτα k από αυτά αποτελούν την απάντηση του Top-k ερωτήματος. Αξίζει στο συγκεκριμένο σημείο να αναφερθεί ότι στα πλαίσια της εργασίας αυτής θεωρείται ως καλύτερη βαθμολογία αυτή που έχει την χαμηλότερη τιμή. Η πιο σημαντική και η πιο δημοφιλής συνάρτηση βαθμολόγησης είναι η weighting sum συνάρτηση, όπου είναι γραμμική. Κάθε διάσταση $d_i$ συσχετίζεται με ένα weighting vector του ερωτήματος $w[i]$ όπου δηλώνει την σημαντικότητα του $d_i$ για το ερώτημα. Η συγκεντρωτική βαθμολογία $f_w(p)$ για ένα στοιχείο $p$ καθορίζεται ως το weighting sum των ανεξάρτητων βαθμολογιών κάθε διάστασης. Πιο συγκεκριμένα η συνάρτηση βαθμολόγησης είναι η εξής: $f_w(p) = \sum_{i=0}^{d} w[i] * p[i]$, όπου $w[i] \geq 0,\ (1 \leq i \leq d)$ και $\sum_{i=0}^{d} w[i] = 1$. Εφόσον τα βάρη αντιπροσωπεύουν την σημαντικότητα μεταξύ διαφορετικών διαστάσεων η υπόθεση $\sum_{i=0}^{d} w[i] = 1$ δεν επηρεάζει τον ορισμό των Top-k ερωτημάτων. Μία πιο ευαίσθητη κατάσταση υπάρχει όταν δύο ή περισσότερα στοιχεία έχουν την ίδια βαθμολογία και συναγωνίζονται για την k-οστή θέση. Σε αυτή την περίπτωση για λόγους απλότητας, θεωρούμε ότι μπορεί να αναφερθεί οποιοδήποτε στοιχείο από αυτά ως k-οστό.

*<u>Ορισμός 1 (Top-k ερώτημα):</u> Δοσμένων ενός θετικού ακεραίου αριθμού k και ενός weighting vector καθορισμένο από τον χρήστη, το αποτέλεσμα $T(w,k)$ του Top-k ερωτήματος είναι ένα σύνολο δεδομένων τέτοιο ώστε $T(w,k) \subseteq S,\quad |T(w,k)| = k$ και $\forall p_i, p_j : p_i \in T(w,k),\ p_j \in S - T(w,k)$ ισχύει ότι $f_{w_i}(p_i) \leq f_{w_i}(p_j)$.*




*Εικόνα 3: Στην εικόνα αυτή παρουσιάζεται ο χώρος των weighting vectors στις 2 διαστάσεις. Όλα τα weighting vectors βρίσκονται απάνω στην μπλε γραμμή καθώς ισχύει $\sum_{i=0}^{d} w[i] = 1$.*

Για παράδειγμα, έστω ότι χρειάζεται να υλοποιηθεί ένα σύστημα το οποίο θα προτείνει σε ταξιδιώτες ξενοδοχεία, δηλαδή ένα σύστημα συστάσεων ξενοδοχείων. Έστω επίσης ότι οι πληροφορίες για τα ξενοδοχεία που ενδιαφέρουν τους ταξιδιώτες είναι η τιμή και η απόσταση από την θάλασσα. Όσο μικρότερες είναι αυτές οι τιμές τόσο πιο πολύ προτιμάται το ξενοδοχείο. Οι χρήστες προκειμένου να δηλώσουν τι τους ενδιαφέρει πιο πολύ και κατά πόσο, θεωρούμε ότι έχουν μία μπάρα και την μεταβάλλουν είτε προς την τιμή αν τους ενδιαφέρει να βρούνε φθηνό ξενοδοχείο ή προς την απόσταση από την θάλασσα αν τους ενδιαφέρει περισσότερο να είναι κοντά στην θάλασσα. Έτσι η μεταβολή της μπάρας αυτή μεταφράζεται σε ένα ποσοστό επί της εκατό, που δηλώνει το βάρος που δίνει ο χρήστης στην τιμή και στην απόσταση από την θάλασσα.

Όπως φαίνεται στην Εικόνα 4 υπάρχουν 5 ξενοδοχεία που διαφέρουν τόσο στις τιμές όσο και στην απόσταση από την θάλασσα. Για παράδειγμα το ξενοδοχείο Βυθός απέχει μόλις 100 μέτρα από την θάλασσα αλλά είναι το ακριβότερο καθώς κοστίζει 300 ευρώ την βραδιά. Αντίθετα το ξενοδοχείο Αστερίας είναι 800 μέτρα από την θάλασσα αλλά η τιμή του είναι μόλις 50 ευρώ η βραδιά. Επίσης στην Εικόνα 5 υπάρχουν 4 ταξιδιώτες με διαφορετικές προτιμήσεις. Για παράδειγμα ο Μάρκος είναι φοιτητής και δεν ενδιαφέρεται τόσο πολύ να είναι το ξενοδοχείο που θα μένει κοντά στην θάλασσα (βάρος 0,1) αλλά τον ενδιαφέρει περισσότερο να είναι οικονομικό (βάρος 0,9). Αντίθετα ο Θανάσης είναι συνταξιούχος και τον ενδιαφέρει το ξενοδοχείο να είναι κοντά στην θάλασσα (βάρος 0,8) γιατί δεν μπορεί να περπατά μεγάλες αποστάσεις περισσότερο από την τιμή (βάρος 0,2).

| Ξενοδοχείο | Τιμή | Απόσταση από θάλασσα |
|---|---|---|
| Αστερίας | 50 € | 800 m. |
| Βυθός | 300 € | 100 m. |
| Πανόραμα | 70 € | 700 m. |
| Ανεμόμυλος | 40 € | 250 m. |
| Ακρογιαλιά | 50 € | 500 m. |

*Εικόνα 4: Στην εικόνα αυτή παρουσιάζονται 5 ξενοδοχεία. Οι διαστάσεις είναι 2, η τιμή και η απόσταση από την θάλασσα.*

| Ταξιδιώτης | Βάρος τιμής | Βάρος απόστασης από την θάλασσα |
|---|---|---|
| Θανάσης | 0,2 | 0,8 |
| Παναγιώτης | 0,4 | 0,6 |
| Μάρκος | 0,9 | 0,1 |
| Έλενα | 0,5 | 0,5 |

*Εικόνα 5: Παρουσιάζονται τα weighting vectors 4 χρηστών. Η πρώτη διάσταση αφορά το βάρος της τιμής ενώ η δεύτερη διάσταση το βάρος της απόστασης από την θάλασσα.*

Δοσμένης της συνάρτησης βαθμολόγησης που δόθηκε προηγουμένως τα Top-k ξενοδοχεία που θα προταθούν τόσο στον Μάρκο όσο και στον Θανάση υπολογίζονται ως εξής:

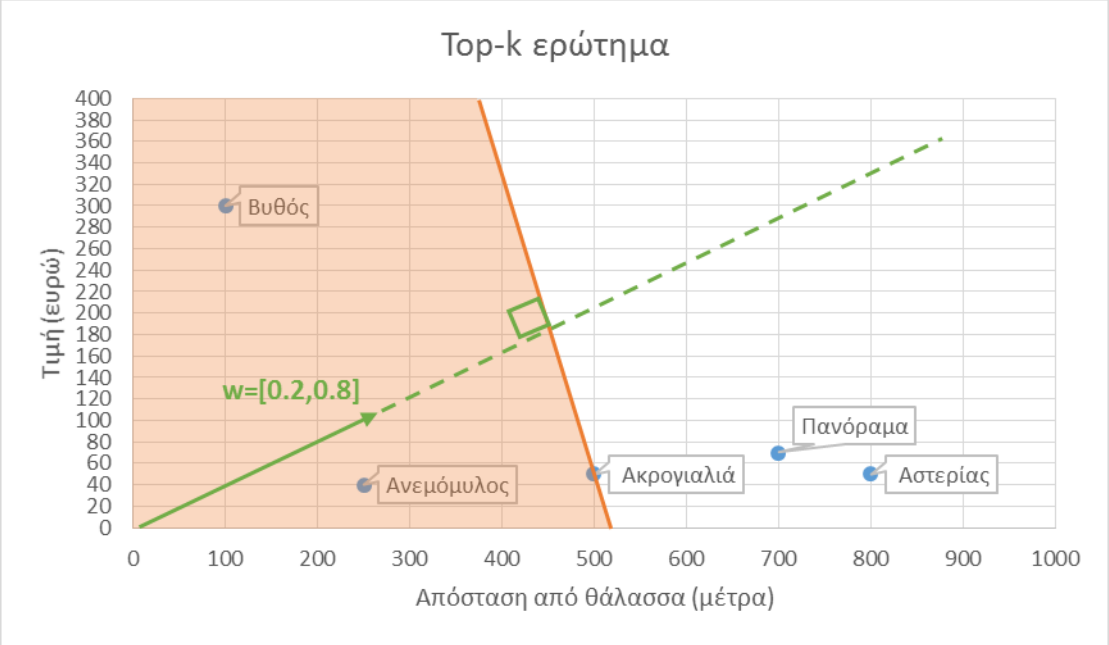
- $f_{Μάρκος}(Αστερίας) = 50 * 0{,}9 + 800 * 0{,}1 = 125$
- $f_{Μάρκος}(Βυθός) = 300 * 0{,}9 + 100 * 0{,}1 = 280$
- $f_{Μάρκος}(Πανόραμα) = 70 * 0{,}9 + 700 * 0{,}1 = 133$
- $f_{Μάρκος}(Ανεμόμυλος) = 40 * 0{,}9 + 250 * 0{,}1 = 61$
- $f_{Μάρκος}(Ακρογιαλά) = 50 * 0{,}9 + 500 * 0{,}1 = 95$
- $f_{θανάσης}(Αστερίας) = 50 * 0{,}2 + 800 * 0{,}8 = 650$
- $f_{θανάσης}(Βυθός) = 300 * 0{,}2 + 100 * 0{,}8 = 140$
- $f_{θανάσης}(Πανόραμα) = 70 * 0{,}2 + 700 * 0{,}8 = 574$
- $f_{θανάσης}(Ανεμόμυλος) = 40 * 0{,}2 + 250 * 0{,}8 = 208$
- $f_{θανάσης}(Ακρογιαλά) = 50 * 0{,}2 + 500 * 0{,}8 = 410$

Αντίστοιχα υπολογίζεται η βαθμολογία του κάθε ξενοδοχείου και για τους υπόλοιπους ταξιδιώτες. Έτσι ταξινομώντας τα ξενοδοχεία για κάθε χρήστη βάση της βαθμολογίας τους κατά αύξουσα σειρά και επιλέγοντας τα k πρώτα ξενοδοχεία προκύπτει το Top-k. Η ταξινόμηση γίνεται κατά αύξουσα σειρά καθώς όσο μικρότερη είναι η βαθμολογία τόσο περισσότερο ταιριάζει στις προτιμήσεις του χρήστη. Παρακάτω παρουσιάζονται τα Top-k αποτελέσματα των χρηστών:

| Ταξιδιώτης | Top-1 | Top-2 | Top-3 | Top-4 | Top-5 |
|---|---|---|---|---|---|
| Θανάσης | Βυθός | Ανεμόμυλος | Ακρογιαλιά | Πανόραμα | Αστερίας |
| Παναγιώτης | Ανεμόμυλος | Βυθός | Ακρογιαλιά | Πανόραμα | Αστερίας |
| Μάρκος | Ανεμόμυλος | Ακρογιαλιά | Αστερίας | Πανόραμα | Βυθός |
| Έλενα | Ανεμόμυλος | Βυθός | Ακρογιαλιά | Πανόραμα | Αστερίας |

*Εικόνα 6: Στην εικόνα αυτή παρουσιάζονται τα Top-k αποτελέσματα των χρηστών.*

Στην Εικόνα 7 παρουσιάζεται το weighting vector που αντιστοιχεί στις προτιμήσεις του Θανάση. Η επιλογή των Top-k ερωτημάτων γίνεται διατρέχοντας τον χώρο των στοιχείων S με μία κάθετη γραμμή (γραμμή βαθμολογίας) στο weighting vector ξεκινώντας από την αρχή των αξόνων, μέχρι η γραμμή να αγγίξει k-στοιχεία. Η γραμμή αυτή λέγεται γραμμή βαθμολογίας καθώς όλα τα στοιχεία που βρίσκονται απάνω στην γραμμή αυτή έχουν την ίδια βαθμολογία. Ανάλογα τις προτιμήσεις των χρηστών το weighting vector μεταβάλει την γωνία του ώστε τα στοιχεία που επιλέγονται να προσαρμόζονται στις προτιμήσεις του εκάστοτε χρήστη. Στο συγκεκριμένο παράδειγμα μπορεί να παρατηρήσει κανείς ότι δίνεται προτεραιότητα στην απόσταση από την θάλασσα σε σχέση με την τιμή, καθώς αυτή είναι η προτίμηση του Θανάση. Επίσης αξίζει να σημειωθεί ότι στην περίπτωση που δύο ή παραπάνω στοιχεία βρίσκονταν στην γραμμή και διεκδικούσαν την k-οστή θέση, τότε όπως αναφέρθηκε και προηγουμένως θα μπορούσε να επιλεγεί ένα από αυτά με τυχαίο τρόπο.




*Εικόνα 7: Στην εικόνα αυτή παρουσιάζεται το weighting vector που αντιστοιχεί στις προτιμήσεις του χρήστη Θανάση. Επίσης παρουσιάζονται όλα τα ξενοδοχεία καθώς και ο τρόπος με τον οποίο επιλέγονται τα Top-k ξενοδοχεία για τις προτιμήσεις του χρήστη αυτού και πιο συγκεκριμένα το Top-3.*

## 1.4 Reverse Top-k ερωτήματα

Δοσμένου ενός ερωτήματος q, το Reverse Top-k ερώτημα ανακαλύπτει όλα εκείνα τα weighting vectors στα οποία το q ανήκει στα αποτελέσματα των Top-k ερωτημάτων τους. Ο ορισμός του Reverse Top-k ερωτήματος ακολουθεί στο πεδίο Ορισμός 2. Παρατηρείται ότι αυτός ο ορισμός αντιστοιχεί στην bichromatic εκδοχή του Reverse Top-k ερωτήματος (1), όπου θεωρείται ότι υπάρχει ένα σύνολο W με τις προτιμήσεις των χρηστών.

> *Ορισμός 2 (Reverse Top-k ερώτημα): Δοσμένων ενός στοιχείου q, ενός θετικού ακεραίου αριθμού k, επίσης δύο σύνολα $S$ και $W$ από στοιχεία και weighting vectors αντίστοιχα, ένα weighting vector $w_i \in W$ ανήκει στα Reverse Top-k αποτελέσματα ( $R(S, W, k, q)$ ) του q αν και μόνο αν $\exists\, p \in T(w_i, q)$ τέτοιο ώστε $f_{w_i}(q) \leq f_{w_i}(p)$.*

Στην συνέχεια του παραδείγματος για το Top-k ερώτημα θα εξηγηθεί το Reverse Top-k ερώτημα. Έστω ότι στο σύστημα συστάσεων ξενοδοχείων προτείνονται τα 2 καλύτερα ξενοδοχεία για τις προτιμήσεις του κάθε χρήστη, δηλαδή το Top-2. Έστω επίσης ένας επενδυτής ο οποίος σκέφτεται να ανοίξει ένα νέο ξενοδοχείο και θέλει να υπολογίσει την απήχησή του στο καταναλωτικό κοινό μέσω ενός Reverse Top-k ερωτήματος. Μετά από αναζήτηση ακινήτων και οικονομικές μελέτες το ξενοδοχείο που είναι υπό συζήτηση θα απέχει 150 μέτρα από την θάλασσα ενώ η τιμή του θα ανέρχεται στα 100 ευρώ την βραδιά, το ξενοδοχείο αυτό θα ονομάζεται Πέτρινο.

Στην Εικόνα 8 παρουσιάζεται το Top-2 των ξενοδοχείων για κάθε χρήστη μαζί με την βαθμολογία του κάθε ξενοδοχείου. Για τον υπολογισμό του Reverse Top-k ερωτήματος υπολογίζεται η βαθμολογία για το ξενοδοχείο Πέτρινο καθώς και οι βαθμολογίες για τα υπόλοιπα ξενοδοχεία, όπως στο παράδειγμα που δόθηκε προηγουμένως. Οι βαθμολογίες για το ξενοδοχείο Πέτρινο υπολογίζονται ως εξής:

- $f_{\text{Θανάσης}}(\text{Πέτρινο}) = 100 * 0{,}2 + 150 * 0{,}8 = 140$
- $f_{\text{Παναγιώτης}}(\text{Πέτρινο}) = 100 * 0{,}4 + 150 * 0{,}6 = 130$
- $f_{\text{Μάρκος}}(\text{Πέτρινο}) = 100 * 0{,}9 + 150 * 0{,}1 = 105$
- $f_{\text{Έλενα}}(\text{Πέτρινο}) = 100 * 0{,}5 + 150 * 0{,}5 = 125$

Το Reverse Top-k ερώτημα υπολογίζει σε ποια Top-k ερωτήματα ανήκει το q, δηλαδή το ξενοδοχείο Πέτρινο. Οπότε όπως φαίνεται στην Εικόνα 8, το Reverse Top-2 ερώτημα θα επιστρέψει τις προτιμήσεις των χρηστών Θανάση, Παναγιώτη και Έλενας.

| Ταξιδιώτης | Top-1 | Top-2 | | Top-1 | Top-2 |
|---|---|---|---|---|---|
| Θανάσης | Βυθός (140) | Ανεμόμυλος (208) | | Πέτρινο (140) | Βυθός (140) |
| Παναγιώτης | Ανεμόμυλος (166) | Βυθός (180) | | Πέτρινο (130) | Ανεμόμυλος (166) |
| Μάρκος | Ανεμόμυλος (61) | Ακρογιαλιά (95) | | Ανεμόμυλος (61) | Ακρογιαλιά (95) |
| Έλενα | Ανεμόμυλος (145) | Βυθός (200) | | Πέτρινο (125) | Ανεμόμυλος (145) |

*Εικόνα 8: Στην εικόνα αυτή παρουσιάζονται αριστερά το Top-2 των χρηστών για τα υπάρχοντα ξενοδοχεία, ενώ στην δεξιά πλευρά παρουσιάζονται τα Top-2 των χρηστών με την παραδοχή ύπαρξης του ξενοδοχείου Πέτρινο. Μέσα στις παρενθέσεις παρουσιάζεται η βαθμολογία των ξενοδοχείων για τον κάθε χρήστη.*

## 1.5 Skyline ερωτήματα

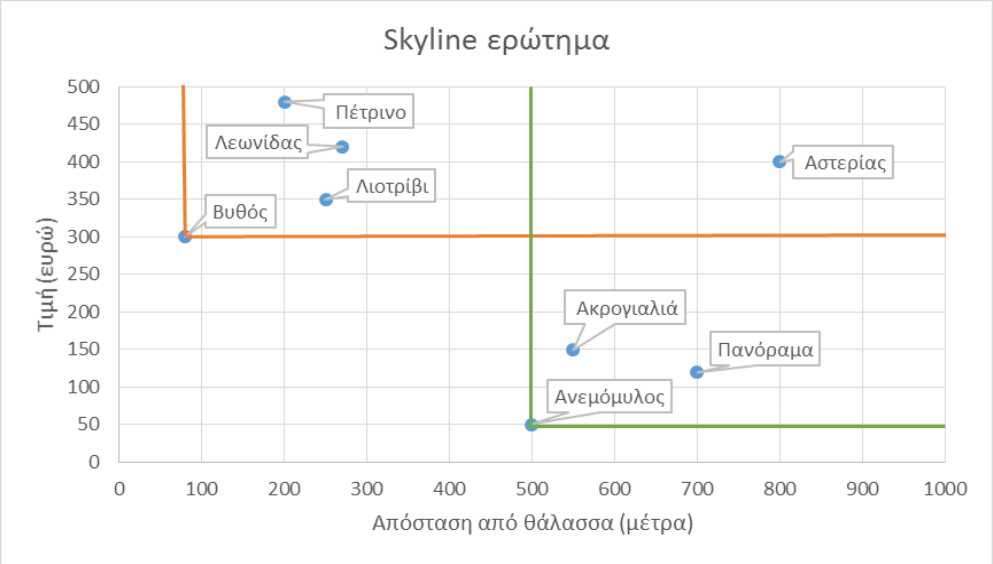
Τα Skyline ερωτήματα είναι σχετικά με τα Reverse Top-k ερωτήματα. Δοσμένου ενός συνόλου $S$, το Skyline αποτέλεσμα $S(S)$ του συνόλου $S$ ορίζεται όπως παρατηρείται στο πεδίο Ορισμός 3. Σε αυτή την εργασία θεωρούμε ότι τα skyline ερωτήματα υπολογίζονται σε στοιχεία τα οποία έχουν μη αρνητικές τιμές.

*Ορισμός 3 (Skyline ερώτημα):* *Ένα στοιχείο $p \in S$, λέγεται ότι κυριαρχεί ενός άλλου στοιχείου $p' \in S$ (συμβολίζεται ως $p \prec p'$), αν ισχύουν τα εξής:*

1. *Αν για κάθε διάσταση $d_j \in D$ ισχύει $p[j] \leq p'[j]$.*
2. *Αν για μία τουλάχιστον διάσταση ισχύει $p[j] < p'[j]$.*

*Το Skyline αποτέλεσμα είναι ένα σύνολο στοιχείων $S(S) \subseteq S$ το οποίο δεν κυριαρχείται από κανένα άλλο στοιχείο. Τα στοιχεία που ανήκουν στο $S(S)$ ονομάζονται Skyline στοιχεία.*

Σε συνέχεια των προηγούμενων παραδειγμάτων έστω ότι υπάρχουν πλέον 8 ξενοδοχεία τα οποία παρουσιάζονται στην Εικόνα 9. Όπως παρατηρείται στην εικόνα αυτή το ξενοδοχείο Βυθός είναι καλύτερο σε όλες τις διαστάσεις του (κυριαρχεί), δηλαδή τόσο στην τιμή όσο και στην απόσταση από την θάλασσα από τα ξενοδοχεία Λεωνίδας, Πέτρινο, Λιοτρίβι και Αστερίας. Αντίστοιχα το ξενοδοχείο Ανεμόμυλος είναι καλύτερο σε όλες τις διαστάσεις του (κυριαρχεί) από τα ξενοδοχεία Ακρογιαλιά, Πανόραμα και Αστερίας. Αυτό σημαίνει ότι τα ξενοδοχεία τα οποία κυριαρχούν έναντι κάποιων άλλων θα έχουν πάντα καλύτερη βαθμολογία από τα εκάστοτε κυριαρχημένα. Για παράδειγμα κανένας χρήστης δεν θα επέλεγε το ξενοδοχείο Πανόραμα έναντι του Βυθός καθώς το Βυθός είναι φθηνότερο και είναι πιο κοντά στην θάλασσα από το Πανόραμα. Αυτό σημαίνει ότι οι κυρίαρχοι έχουν πάντα υψηλότερη θέση σε ένα Top-k ερώτημα από τα κυριαρχημένα στοιχεία.




*Εικόνα 9: Στην εικόνα αυτή παρουσιάζονται 8 ξενοδοχεία, ενώ τα ξενοδοχεία Ανεμόμυλος και Βυθός δεν κυριαρχούνται από κανένα άλλο ξενοδοχείο.*

## 1.6 Map Reduce (Hadoop)

Το Hadoop παρέχεται από την Apache και είναι ένα framework για παράλληλη και κατανεμημένη αποθήκευση και επεξεργασία δεδομένων. Η αποθήκευση των δεδομένων στο Hadoop υποστηρίζεται μέσω του Hadoop Distributed File System (HDFS) ενώ η επεξεργασία αυτών βασίζεται στο μοντέλο MapReduce. Το μοντέλο MapReduce προτάθηκε από την Google και έχει ως δύο κύριες λειτουργίες, την λειτουργία Map και την λειτουργία Reduce. Στην συνέχεια θα ακολουθήσει αναλυτικότερη περιγραφή του framework αυτού.

### 1.6.1 HDFS (Hadoop Distributed File System)

Το HDFS (Hadoop Distributed File System) παρέχει την δυνατότητα να αποθηκεύονται δεδομένα σε κατανεμημένο περιβάλλον. Αντίθετα με άλλα κατανεμημένα συστήματα το HDFS είναι πολύ ανεκτικό σε σφάλματα και έχει σχεδιαστεί για χαμηλού κόστους υπολογιστικά μηχανήματα. Το HDFS μπορεί να διαχειριστεί τεράστιου όγκου δεδομένα ενώ παράλληλα προσφέρει εύκολη πρόσβαση. Προκειμένου να είναι δυνατή η αποθήκευση τεράστιου όγκου δεδομένων, τα αρχεία αποθηκεύονται σε πολλούς κόμβους. Επίσης τα αρχεία αυτά αποθηκεύονται πλεονασματικά (replication factor), δηλαδή υπάρχουν διπλότυπα, τριπλότυπα κλπ. προκειμένου να αποφευχθεί η απώλεια δεδομένων σε περίπτωση κάποιας βλάβης. Ακόμα το HDFS δίνει την δυνατότητα σε εφαρμογές για παράλληλη επεξεργασία. Μερικά από τα χαρακτηριστικά του συστήματος είναι τα εξής:

- Είναι κατάλληλο για την κατανεμημένη αποθήκευση και επεξεργασία.
- Το Hadoop παρέχει μία διεπαφή εντολών ώστε να αλληλοεπιδρά με το HDFS.
- Υπάρχει η δυνατότητα πρόσβασης στα δεδομένα με streaming τρόπο.
- Παρέχει δικαιώματα στα αρχεία καθώς και αυθεντικοποίηση.

Το HDFS ακολουθεί την αρχιτεκτονική master-slave και έχει τα εξής στοιχεία:

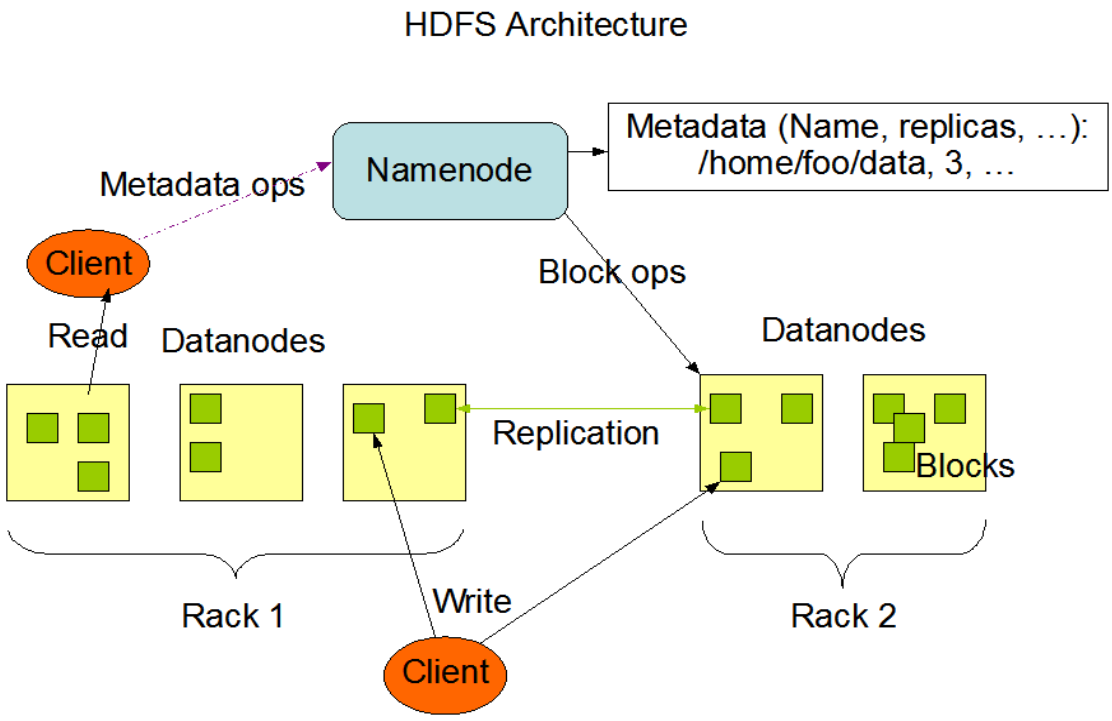
*Namenode:* Ο Namenode είναι ένας κόμβος ο οποίος περιέχει ένα λειτουργικό σύστημα και το λογισμικό του Namenode. Ο κόμβος αυτός λειτουργεί ως master κόμβος και έχει τις ακόλουθες αρμοδιότητες:

- Διαχειρίζεται τον χώρο ονομάτων (namespace) των αρχείων.
- Ρυθμίζει την πρόσβαση των clients στα αρχεία.
- Εκτελεί επίσης λειτουργίες ενός συστήματος αρχείων, όπως μετονομασία, το κλείσιμο και το άνοιγμα των αρχείων και φακέλων.

*Datanode:* Ο Datanode είναι ένας κόμβος ο οποίος περιέχει ένα λειτουργικό σύστημα και το λογισμικό του Datanode. Για κάθε κόμβο μέσα στο cluster, θα υπάρχει ένας Datanode. Αυτοί οι κόμβοι διαχειρίζονται την αποθήκευση των αρχείων στο σύστημά τους. Πιο συγκεκριμένα πράττουν τις εξής λειτουργίες:

- Οι Datanodes εκτελούν λειτουργίες ανάγνωσης και εγγραφής στο σύστημα αρχείων, σύμφωνα με τα αιτήματα του client.
- Επίσης εκτελούν λειτουργίες όπως δημιουργία blocks, διαγραφή και αντιγραφή βάση των οδηγιών του Namenode.

*Block:* Γενικά τα δεδομένα του χρήστη αποθηκεύονται σε αρχεία στο HDFS. Το αρχείο θα διασπαστεί σε πολλά τμήματα τα οποία αποθηκεύονται σε ανεξάρτητους Datanodes (ένα τμήμα μπορεί να αποθηκευτεί σε περισσότερους από 1 Datanodes). Αυτά τα τμήματα δεδομένων ονομάζονται Blocks. Δηλαδή το ελάχιστο μέγεθος δεδομένων που μπορεί να διαβάσει και να γράψει το HDFS είναι το Block. Το προκαθορισμένο μέγεθος ενός Block είναι 64 MB, αλλά αυτό μπορεί να αυξηθεί ή να μειωθεί από τον χρήστη. Επίσης το πλήθος των Datanodes που αποθηκεύεται ένα Block μπορεί επίσης να αυξηθεί ή να μειωθεί ανάλογα με τις προτιμήσεις του χρήστη.




*Εικόνα 10: Στην εικόνα αυτή παρουσιάζεται η αρχιτεκτονική του HDFS (2).*

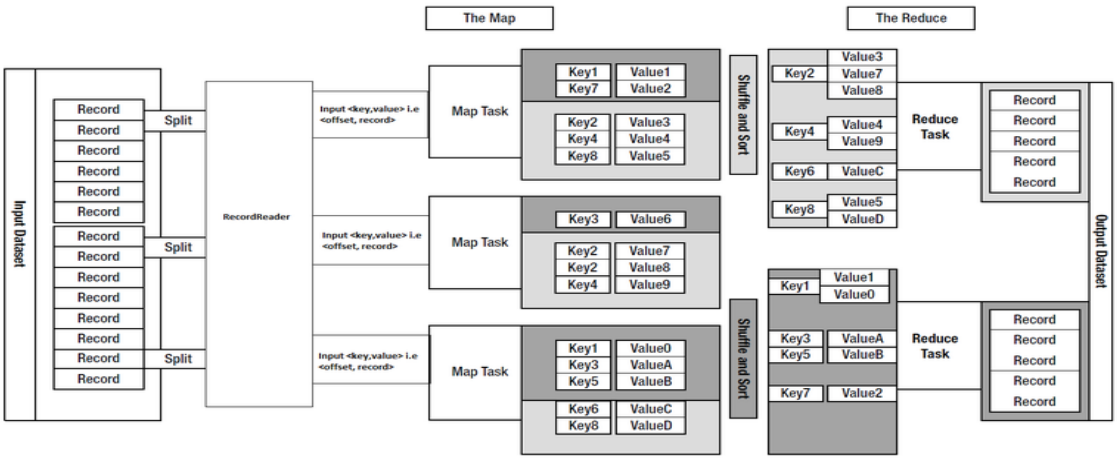
### 1.6.2 Μοντέλο Map Reduce

Το MapReduce είναι ένα προγραμματιστικό μοντέλο που δίνει την δυνατότητα να δημιουργηθούν εφαρμογές οι οποίες να επεξεργάζονται τεράστια σύνολα δεδομένων, παράλληλα, σε μεγάλα clusters με αξιόπιστο τρόπο. Πιο αναλυτικά ένας MapReduce αλγόριθμος αποτελείται από δύο σημαντικές λειτουργίες (φάσεις) την Map φάση και την Reduce φάση. Η Map φάση δέχεται ως είσοδο ένα σύνολο δεδομένων και εξάγει ένα δεύτερο σύνολο δεδομένων, όπου τα στοιχεία διασπώνται σε πλειάδες κλειδιού-τιμής (key-value pairs). Έπειτα η Reduce φάση δέχεται ως είσοδο την έξοδο της Map φάσης και συνδυάζει τις πλειάδες αυτές σε μικρότερα σύνολα πλειάδων. Όπως είναι αντιληπτό και από το όνομα του MapReduce μοντέλου, η Map φάση εκτελείται πάντα πριν από την Reduce φάση.

Το βασικό πλεονέκτημα του MapReduce είναι ότι εύκολα μπορεί να κλιμακωθεί η επεξεργασία των δεδομένων πάνω από πολλούς υπολογιστικούς κόμβους. Τα στοιχεία τα οποία επεξεργάζονται τα δεδομένα στο MapReduce καλούνται Mappers και Reducers. Οπότε ο χρήστης καλείται να υλοποιήσει τις δύο αυτές λειτουργίες, δηλαδή τον αλγόριθμο που θα εκτελούν οι Mappers και τον αλγόριθμο που θα εκτελούν οι Reducers. Έπειτα αυτός ο MapReduce αλγόριθμος μπορεί να εκτελεστεί σε εκατοντάδες, χιλιάδες ή ακόμα και δεκάδες χιλιάδες κόμβους στο cluster απλά με μία αλλαγή στις ρυθμίσεις του συστήματος. Αυτή η εύκολη επεκτασιμότητα έχει κάνει δημοφιλές το MapReduce μοντέλο.

Γενικά η φιλοσοφία του MapReduce είναι να πραγματοποιείται ο υπολογισμός στην τοποθεσία που βρίσκονται τα δεδομένα. Πιο αναλυτικά ο MapReduce αλγόριθμος χωρίζεται σε 3 φάσεις (στάδια), την Map φάση, την Shuffle φάση και την Reduce φάση όπως αναφέρθηκε προηγουμένως. Ακολουθεί επεξήγηση της λειτουργίας των φάσεων αυτών:

- *Map φάση:* Κατά την Map φάση εκτελούνται οι Mappers, οι οποίοι επεξεργάζονται τα δεδομένα εισόδου. Γενικά τα δεδομένα εισόδου είναι αρχεία τα οποία αποθηκεύονται στο HDFS. Κάθε αρχείο εισόδου διαβάζεται από τον Mapper γραμμή-γραμμή. Ο Mapper επεξεργάζεται το σύνολο δεδομένων που του δίνεται ως είσοδος και δημιουργεί πολλά μικρά τμήματα δεδομένων (key-value pairs).
- *Reduce φάση:* Η φάση αυτή είναι ένας συνδυασμός της Shuffle φάσης και της Reduce φάσης. Η δουλειά ενός Reducer είναι να επεξεργαστεί τα δεδομένα που λαμβάνει από τους Mappers. Αφού τα επεξεργαστεί εξάγει ένα νέο σύνολο δεδομένων που αποθηκεύεται στο HDFS και αποτελεί την έξοδο του Reducer.




*Εικόνα 11: Παρουσιάζεται το μοντέλο MapReduce στο Hadoop* (3).

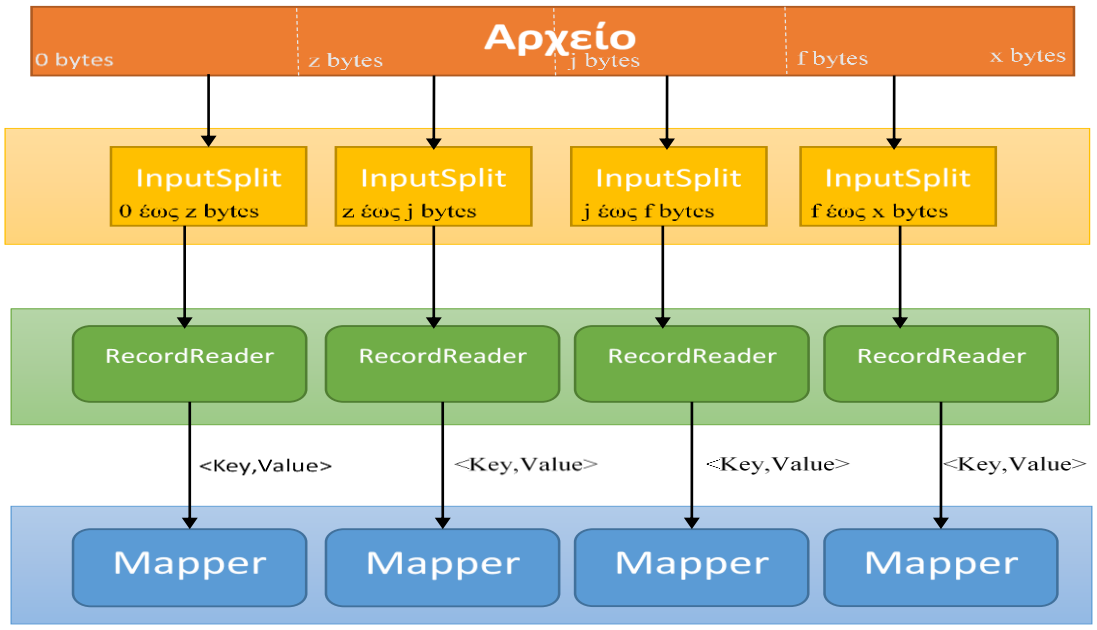
### 1.6.3 Λεπτομέρειες MapReduce στο Hadoop

Προηγουμένως αναφέρθηκαν οι βασικές αρχές καθώς και η λογική του MapReduce μοντέλου. Σε αυτό το σημείο θα αναλυθεί ο τρόπος ο οποίος υλοποιείται το μοντέλο αυτό στο Hadoop με περισσότερες λεπτομέρειες. Το επίπεδο της ανάλυσης θα είναι τέτοιο ώστε να παρουσιαστούν τα στοιχεία εκείνα τα οποία χρειάζονται για την κατανόηση της εργασίας αυτής και μόνο.

Όπως αναφέρθηκε προηγουμένως οι Mappers δέχονται ως είσοδο ένα σύνολο δεδομένων. Πιο συγκεκριμένα ένα ή περισσότερα αρχεία που είναι αποθηκευμένα στο HDFS. Κάθε Mapper επεξεργάζεται ένα τμήμα του συνόλου αυτού, το τμήμα που δέχεται ως είσοδο ο κάθε Mapper ονομάζεται InputSplit. Το μέγεθος του InputSplit καθορίζεται από τον χρήστη, για λόγους απόδοσης προτείνεται το μέγεθος InputSplit να είναι ίσο με το μέγεθος του Block (4) έτσι ώστε να μην μεταφερθεί μεγάλος όγκος δεδομένων πάνω από το δίκτυο γιατί το Hadoop θα προσπαθήσει να εκτελέσει τον Mapper στον ίδιο κόμβο που βρίσκεται το InputSplit του.

Τα InputSplits δημιουργούνται και μοιράζονται στους Mappers μόλις ξεκινήσει το Job. Υπεύθυνο για την δημιουργία των InputSplits είναι το InputFormat. Το InputFormat έχει πρόσβαση στα μεταδεδομένα του αρχείου που δόθηκε ως είσοδο, μαζί με αυτά και το συνολικό μήκος του αρχείου. Έτσι διαιρεί το συνολικό μήκος του αρχείου με το μέγεθος του InputSplit που όρισε ο χρήστης και έτσι χωρίζεται το αρχείο σε InputSplits. Αξίζει να σημειωθεί σε αυτό το σημείο ότι ένα InputSplit δεν είναι τίποτε άλλο παρά δείκτες προς το τμήμα των δεδομένων που καλείται να επεξεργαστεί ένας Mapper (5).

Έπειτα μαζί με κάθε Mapper εκτελείται και ένας RecordReader ο οποίος είναι υπεύθυνος για την ανάγνωση του InputSplit προκειμένου να δώσει στον Mapper τα δεδομένα του InputSplit σε τμήματα κλειδιού τιμής (key-value pairs). Για παράδειγμα ένας Record Reader μπορεί να διαβάζει το InputSplit και να δίνει στον Mapper τα στοιχεία του αρχείου γραμμή-γραμμή με κλειδί τον αριθμό της γραμμής και τιμή το περιεχόμενο της γραμμής.




*Εικόνα 12: Στην εικόνα αυτή παρουσιάζεται ο τρόπος με τον οποίο χωρίζεται ένα αρχείο και δίνεται προς επεξεργασία στους Mappers.*

Σε αυτό το σημείο αξίζει να σημειωθεί ότι κάθε κλειδί (key) είναι μία υλοποίηση της διεπαφής WritableComparable έτσι ώστε να δίνεται η δυνατότητα σύγκρισης μεταξύ των κλειδιών (5). Από την άλλη κάθε τιμή (value) είναι μια υλοποίηση της διεπαφής Writable. Το Hadoop API παρέχει τους βασικούς τύπους δεδομένων όπως Long, Text κλπ. αλλά είναι στην ευχέρεια του χρήστη να υλοποιήσει τους δικούς του τύπους πχ. Person.

Έπειτα κάθε Mapper δέχεται τα δεδομένα που του ανατέθηκαν σε μορφή κλειδιού τιμής (key-value pair). Σε αυτό το σημείο αξίζει να σημειωθεί ότι αν έχει δοθεί από τον χρήστη μηδενικό πλήθος Reducers τότε θα εκτελεστεί μόνο η Map φάση και το αποτέλεσμα αυτής θα είναι το αποτέλεσμα του MapReduce Job. Κάθε Mapper έχει τις εξής μεθόδους, την setup η οποία εκτελείται πριν την εκτέλεση του Mapper και χρησιμοποιείται συνήθως για την ανάθεση τιμών που δόθηκαν από τον χρήστη ή οτιδήποτε άλλο χρειάζεται πριν την εκτέλεση ενός Mapper. Έπειτα υπάρχει η μέθοδος map η οποία εκτελείται για κάθε πλειάδα κλειδιού τιμής. Αφού τελειώσει η ανάγνωση του InputSplit καλείται η cleanup μέθοδος όπου συνήθως χρησιμοποιείται για την αποδέσμευση κάποιων πόρων ή οτιδήποτε χρειάζεται να εκτελεστεί στο τέλος του Mapper. Η μέθοδος η οποία καλεί τις παραπάνω μεθόδους είναι η run όπου πρώτα εκτελεί την setup, έπειτα για κάθε key-value που της δίνει ο RecordReader εκτελεί την map και αφού δεν έχει άλλα στοιχεία καλεί την cleanup πριν το τέλος της. Η run μέθοδος μπορεί να παραμετροποιηθεί προκειμένου να εκτελεστεί σε πολλά Threads ο Mapper επίσης μπορεί να χρησιμοποιηθεί για οτιδήποτε άλλο φανεί χρήσιμο στον χρήστη. Από οποιαδήποτε μέθοδο από τις παραπάνω μπορούν να εξαχθούν τα επιθυμητά δεδομένα σε μορφή κλειδιού τιμής προς τους Reducers.

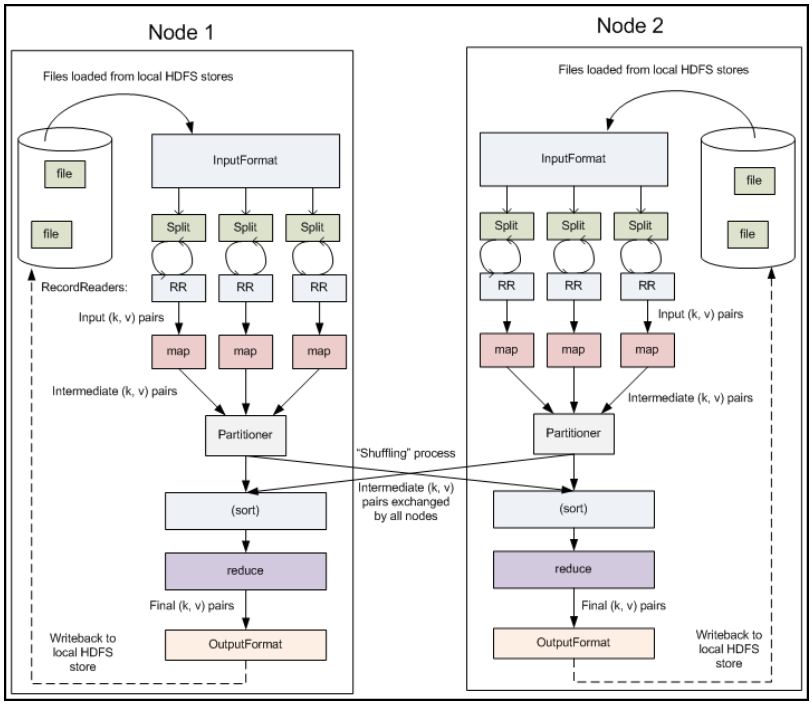
Παραμένοντας στον κόμβο του Mapper, κάθε στοιχείο που εξάγει ο Mapper επεξεργάζεται από έναν Partitioner (5). Ο Partitioner αυτός είναι συνήθως η υλοποίηση μιας συνάρτησης κατακερματισμού αλλά μπορεί αν θελήσει ο χρήστης να την παραμετροποιήσει. Η συνάρτηση αυτή διαβάζει τα κλειδιά από τις πλειάδες και είναι υπεύθυνη να στείλει κάθε πλειάδα στον Reducer που της αντιστοιχεί. Σε αυτό το σημείο υπενθυμίζεται ότι στοιχεία με τα ίδια κλειδιά θα σταλούν στον ίδιο Reducer. Προκειμένου να υπολογίσει και να στείλει τα στοιχεία σε x Reducers που όρισε ο χρήστης η συνάρτηση αυτή δεν μπορεί να αγνοεί τον μέγιστο αριθμό των Reducers που όρισε ο χρήστης.

Στην συνέχεια κατά την αποστολή των στοιχείων στους κόμβους που βρίσκονται οι Reducers, εκεί λαμβάνει χώρα πρώτα η Shuffle φάση κατά την οποία ο Reducer αντιγράφει τα ταξινομημένα στοιχεία που δόθηκαν ως έξοδο από τους Mappers μέσω του HTTP πρωτοκόλλου πάνω από το δίκτυο. Έπειτα ταξινομούνται πάλι τα δεδομένα βάση των κλειδιών τους, καθώς κάθε Mapper μπορεί να έχει διαφορετικά κλειδιά. Η ταξινόμηση αυτή γίνεται συνήθως βάση της μεθόδου σύγκρισης που ορίζεται στο κλειδί αλλά μπορεί να χρησιμοποιηθεί και ένας SortComparator ο οποίος να ελέγξει την ταξινόμηση των στοιχείων. Έπειτα ο GroupComparator συγκρίνει την ισότητα των κλειδιών προκειμένου να βάλει τα στοιχεία με τα ίδια κλειδιά στην ίδια λίστα, δίνεται επίσης η δυνατότητα στον χρήστη να φτιάξει την δικιά του υλοποίηση. Έτσι ο Reducer δέχεται ως είσοδο ένα κλειδί και μία λίστα από τιμές οι οποίες αντιστοιχούν στο κλειδί αυτό.

Σε αυτό το σημείο αξίζει να σημειωθεί ότι αν για κάποιο λόγο μπορεί να χρειάζεται να έρχονται και οι τιμές ταξινομημένες στον Reducer, αυτό ονομάζεται SecondarySort. Προκειμένου να είναι δυνατό αυτό συνήθως θέτουμε ένα σύνθετο κλειδί το οποίο αποτελείται από το κλειδί μαζί με την τιμή ή αν όχι όλη την τιμή, το στοιχείο βάση του οποίου θέλουμε να γίνει η ταξινόμηση. Αυτό όπως είναι φυσικό αποτελεί ένα βάρος καθώς στέλνεται η ίδια πληροφορία δύο φορές πάνω από το δίκτυο. Έπειτα προκειμένου να φτάσουν τα στοιχεία με το ίδιο πραγματικό κλειδί (όχι το σύνθετο) στον ίδιο Reducer πρέπει να υλοποιηθούν ο Partitioner και ο GroupComparator ώστε να μην ελέγχουν όλο το σύνθετο κλειδί αλλά μόνο το πραγματικό κλειδί που εμπεριέχεται στο σύνθετο. Έτσι είναι δυνατόν να φτάσουν τα στοιχεία ταξινομημένα και ανά τιμή στους Reducers.

Έπειτα κάθε Reducer δέχεται τα δεδομένα που του ανατέθηκαν σε μορφή κλειδιού και λίστας τιμών (key-List<value> pair). Κάθε Reducer έχει τις εξής μεθόδους, την setup η οποία εκτελείται πριν την εκτέλεση του Reducer και χρησιμοποιείται συνήθως για την ανάθεση τιμών που δόθηκαν από τον χρήστη ή οτιδήποτε άλλο χρειάζεται πριν την εκτέλεση ενός Reducer. Έπειτα υπάρχει η μέθοδος reduce η οποία εκτελείται για κάθε πλειάδα κλειδιού τιμών. Αφού τελειώσει η ανάγνωση των ομάδων καλείται η cleanup μέθοδος όπου συνήθως χρησιμοποιείται για την αποδέσμευση κάποιων πόρων ή οτιδήποτε χρειάζεται να εκτελεστεί στο τέλος του Reducer. Η μέθοδος η οποία καλεί τις παραπάνω μεθόδους είναι η run όπου πρώτα εκτελεί την setup, έπειτα για κάθε key-List<value> εκτελεί την reduce και αφού δεν έχει άλλα στοιχεία καλεί την cleanup πριν το τέλος της. Από οποιαδήποτε μέθοδο από τις παραπάνω μπορούν να εξαχθούν τα επιθυμητά δεδομένα σε μορφή κλειδιού τιμής για την σύνθεση του τελικού αποτελέσματος του MapReduce Job.

Τέλος αξίζει να γίνει μια αναφορά στην DistributedCache η οποία είναι υπεύθυνη για τον διαμοιρασμό αρχείων σε όλους τους κόμβους πριν την εκτέλεση ενός MapReduce Job. Χρησιμοποιώντας την λειτουργικότητα αυτή ο χρήστης μπορεί να στείλει σε όλους τους κόμβους πληροφορία η οποία χρειάζεται να είναι παντού διαθέσιμη. Αξίζει να σημειωθεί ότι η πληροφορία αυτή είναι μόνο για ανάγνωση (5).




*Εικόνα 13: Παρουσιάζεται η ροή των δεδομένων σε μια πιο λεπτομερή απεικόνιση του MapReduce (6).*

# 2. Αποτύπωση του προβλήματος

Στο κεφάλαιο αυτό θα γίνει μια αναφορά στο πρόβλημα με το οποίο ασχολείται η παρούσα εργασία με επίσημο τρόπο και έπειτα θα αποτυπωθούν οι βασικές ιδιότητες οι οποίες θα οδηγήσουν στον σχεδιασμό ενός αποδεδειγμένα σωστού αλγορίθμου.

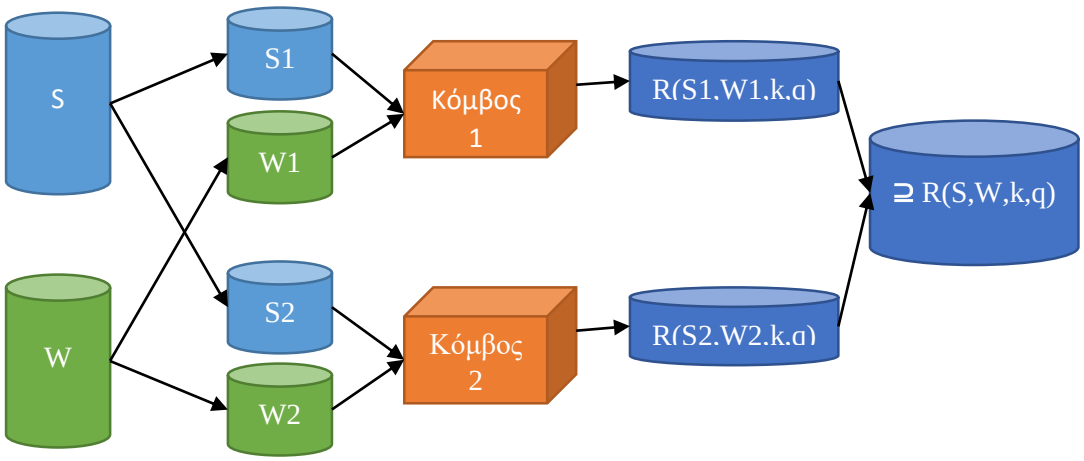
## 2.1 Προβληματική κατάσταση

Αρχικά θεωρούμε ότι υπάρχουν δύο σύνολα δεδομένων $S, W$ (όπως αναφέρθηκαν στον ορισμό του Reverse Top-k ερωτήματος) διαμοιρασμένα και κατανεμημένα σε διάφορους κόμβους (Servers) με τυχαίο τρόπο. Κάθε κόμβος έχει ένα υποσύνολο $Si \subseteq S\ (όπου: Si \cap Sj = \emptyset\ και\ S = \cup_{i=0}^{N} Si)$ και ένα υποσύνολο $Wi \subseteq W\ (όπου: Wi \cap Wj = \emptyset\ και\ W = \cup_{i=0}^{N} Wi)$ που προκύπτει από τα πλήρη σύνολα δεδομένων. Ακόμα επισημαίνεται ότι ένας κόμβος αδυνατεί να επεξεργαστεί ένα ολόκληρο σύνολο $S$ ή $W$. Ο στόχος είναι ο υπολογισμός του αποτελέσματος του Reverse Top-k ερωτήματος $R(S, W, k, q)$ μειώνοντας τον χρόνο απόκρισης μέσω παράλληλης και κατανεμημένης επεξεργασίας.

*Πρόβλημα 1: (Πρόβλημα παράλληλης κατανεμημένης επεξεργασίας Reverse TopK ερωτήματος)*

*Δοσμένων δύο συνόλων δεδομένων S και W τα οποία είναι οριζοντίως διαμοιρασμένα και κατανεμημένα σε διάφορους κόμβους, υπολόγισε το αποτέλεσμα του Reverse Top-k ερωτήματος R(S,W,k,q).*

Σε αυτό το σημείο θα αναφερθούμε σε μία απλή προσέγγιση, με την οποία υπολογίζεται το αποτέλεσμα του Reserve Top-k ερωτήματος για τα υποσύνολα $S_i$,$W_i$ και έπειτα αναφέρεται ως τελικό αποτέλεσμα η ένωση των τοπικών Reverse Top-k αποτελεσμάτων. Γενικά η προσέγγιση αυτή αποτυγχάνει να υπολογίσει σωστά τα αποτέλεσμα του Reverse Top-k ερωτήματος καθώς σε αυτά θα περιέχονται weighting vectors τα οποία δεν ανήκουν στο $R(S, W, k, q)$. Δηλαδή παρουσιάζονται λανθασμένα στοιχεία ως σωστά (false positive). Ο αλγόριθμος αυτός παρουσιάζεται στην Εικόνα 14.




*Εικόνα 14: Απεικόνιση αλγορίθμου απλής προσέγγισης για τον κατανεμημένο και παράλληλο υπολογισμό ενός Reverse Top-k ερωτήματος. Ο αλγόριθμος αυτός παρουσιάζει λανθασμένα αποτελέσματα (false positives).*

Από την Σημείωση 1 προκύπτει ότι το πρόβλημα δεν μπορεί να λυθεί με μια απλή λύση, παρουσιάζοντας τα αποτελέσματα των τοπικών Reverse Top-k ερωτημάτων ως τελικά αποτελέσματα. Η τυχαία κατανομή των υποσυνόλων $S_i$, $W_i$ δεν επιτρέπει την εξαγωγή ορθών συμπερασμάτων με μία φάση μόνο. Κάτι τέτοιο θα ήταν δυνατό αν κάθε κόμβος είχε όλο το σύνολο $S$, αλλά όπως προαναφέρθηκε κανένας κόμβος δεν μπορεί να επεξεργαστεί ένα ολόκληρο σύνολο $S$ ή $W$. Έτσι οδηγούμαστε σε μία προσέγγιση κατανεμημένης επεξεργασίας δύο φάσεων. Κατά την πρώτη φάση κάθε κόμβος παίρνει ως είσοδο ένα υποσύνολο $Si \subseteq S$ και ένα $Wi \subseteq W$ από τα σύνολα δεδομένων $S$, $W$ και υπολογίζει τα $S'i \subseteq Si$ και $W'i \subseteq Wi$ όπου τα στέλνει προς επεξεργασία στην δεύτερη φάση. Κατά την δεύτερη φάση τα σύνολα $\cup_{i=0}^{N} S'i$ , $\cup_{i=0}^{N} W'i$ πρέπει να ανακατανεμηθούν στους κόμβους προκειμένου να εκτελεστεί ορθά ο υπολογισμός του Reverse Top-k ερωτήματος με παράλληλο και κατανεμημένο τρόπο.

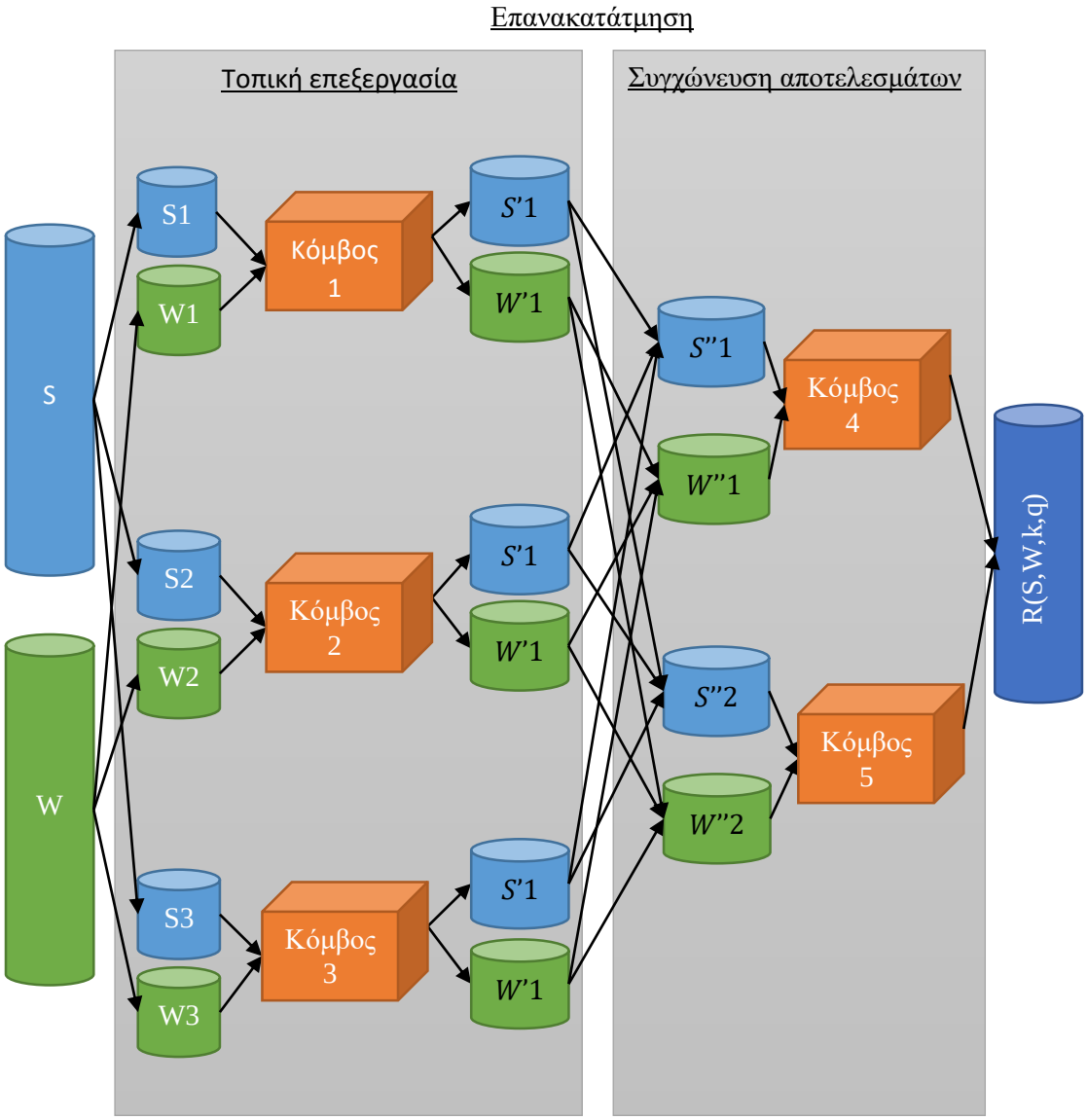
*<u>Σημείωση 1:</u>*

*Η ένωση των τοπικών Reverse Top-k αποτελεσμάτων $\cup_{i=0}^{N} R(Si, Wi, k, q)$ δεν αποτελεί πάντα σωστή λύση για τον κατανεμημένο υπολογισμό του R(S,W,k,q).*

*<u>Απόδειξη:</u> Αρχικά θεωρούμε δύο υποσύνολα για κάθε σύνολο δεδομένων S, W. Θεωρούμε επίσης ένα στοιχείο $p_1 \in S_1$, τέτοιο ώστε να κυριαρχεί του query ($p1 \prec q$). Επειδή $S_1 \cap S_2 = \emptyset$, γνωρίζουμε ότι $p_1 \notin S_2$. Ως αποτέλεσμα για οποιοδήποτε weighting vector $w \in W_2$ τέτοιο ώστε το w να ανήκει στα τοπικά Reverse Top-k αποτελέσματα $R(S_2,W_2,k,q)$, το reverse Top-k αποτέλεσμα του w δεν υπολογίζεται σωστά καθώς αμελεί το στοιχείο $p_1$ το οποίο έχει πάντα καλύτερη θέση στην κατάταξη από το q. Σε περίπτωση που υπάρχει ένα weighting vector $w \in R(S_2,W_2,k,q)$ το οποίο έχει ως k-οστό στοιχείο στην κατάταξη το q, τότε λόγω της ύπαρξης του στοιχείου $p_1$ το οποίο αμελεί, το q θα βρισκόταν τουλάχιστον στην k+1 θέση για το w, ως εκ τούτου το w λανθασμένα παρουσιάζεται στα Reverse Top-k αποτελέσματα.*

Η παράλληλη και κατανεμημένη επεξεργασία ενός Reverse Top-k ερωτήματος εμπεριέχει δύο προκλήσεις. Η πρώτη πρόκληση αφορά τον υπολογισμό της πρώτης φάσης έτσι ώστε τα σύνολα εξόδου $S'i \subseteq Si$ και $W'i \subseteq Wi$ να είναι τέτοια ώστε $R\left(\cup_{i=0}^{N} S'i, \cup_{i=0}^{N} W'i, k, q\right) = R(S, W, k, q)$. Ωστόσο ο στόχος είναι ο υπολογισμός του Reverse Top-k ερωτήματος να γίνει παράλληλα και κατανεμημένα χωρίς να υπάρξει κάποιο κεντρικό σημείο για την συγχώνευση όλων των τοπικών Reverse Top-k αποτελεσμάτων, καθώς κάτι τέτοιο θα αύξανε τον χρόνο απόκρισης και πιθανώς να μην ήταν δυνατό να υπολογιστεί λόγω υπερφόρτωσης του κόμβου στον οποίο θα πραγματοποιούταν η συγχώνευση.

Αυτό μας οδηγεί στην δεύτερη πρόκληση, η οποία είναι ο υπολογισμός του $R\left(\cup_{i=0}^{N} S'i, \cup_{i=0}^{N} W'i, k, q\right)$ με κατανεμημένο και παράλληλο τρόπο. Βάση της Σημείωση 1 δεν αρκεί να υπολογίσουμε τα τελικά αποτελέσματα με την ένωση των τοπικών Reverse Top-k αποτελεσμάτων για τυχαία κατανεμημένα υποσύνολα $S_i, W_i$. Για αυτό η δεύτερη πρόκληση είναι ο υπολογισμός των $S_i'$ και $W_i'$ για κάθε κόμβο με τέτοιο τρόπο ώστε ο υπολογισμός και η ένωση των τοπικών Reverse Top-k αποτελεσμάτων της δεύτερης φάσης να παράγουν ορθά το τελικό Reverse Top-k αποτέλεσμα, δηλαδή $\cup_{i=0}^{N} R(S''i, W''i, k, q) = R(S, W, k, q)$. Σε αυτό το σημείο αξίζει να σημειωθεί ότι η κρίσιμη διαφορά μεταξύ των συνόλων $S_i', W_i'$ σε σύγκριση με τα $S_i'', W_i''$είναι ότι τα πρώτα είναι τυχαία διαμοιρασμένα υποσύνολα των $S, W$ ενώ τα δεύτερα είναι υπολογισμένα και διαμοιρασμένα με κατάλληλο τρόπο από την πρώτη φάση, έτσι ώστε να είναι δυνατό κατά την δεύτερη φάση να εξαχθούν σωστά τα αποτελέσματα. Η Εικόνα 15 παρέχει μια γενική εικόνα υψηλού επιπέδου για την παράλληλη επεξεργασία δύο φάσεων που προτείνεται.



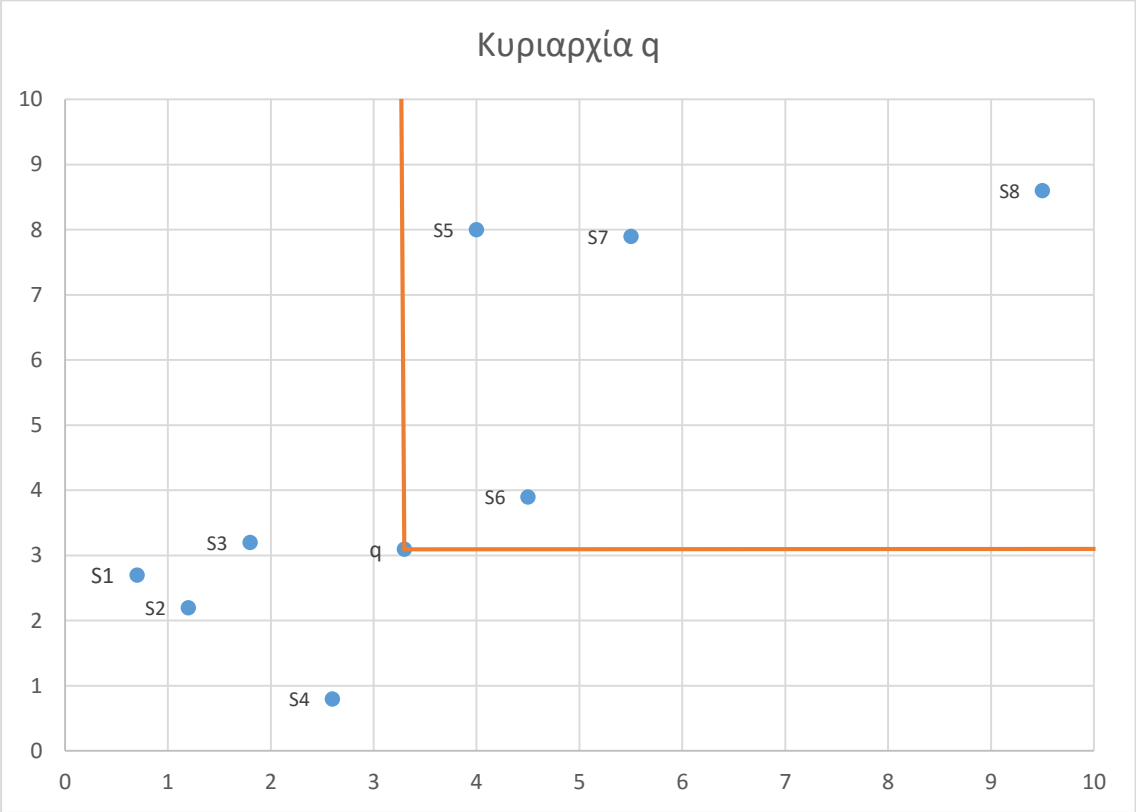

*Εικόνα 15: Απεικόνιση της παράλληλης και κατανεμημένης επεξεργασίας ενός Reverse Top-k ερωτήματος.*

## 2.2 Βασικές ιδιότητες

Οι βασικές ιδιότητες επικεντρώνονται στον περιορισμό του μεγέθους των συνόλων $S'$, $W'$ προκειμένου να βελτιώσουν την αποδοτικότητα του υπολογισμού των Reverse Top-k ερωτημάτων.

### 2.2.1 Περικοπή βάση κυριαρχίας

Σε κάθε τοπικό υποσύνολο $S_i$, τα στοιχεία $p \in S_i$ τα οποία κυριαρχούνται από το q δεν επηρεάζουν τα αποτελέσματα των Reverse Top-k ερωτημάτων και μπορούν ασφαλώς να αφαιρεθούν. Όπως αναφέρθηκε στον Ορισμός 3 ένα τέτοιο στοιχείο δεν πρόκειται να προτιμηθεί ποτέ έναντι του q, οπότε δεν θα επηρεάσει σε καμία περίπτωση την κατάταξη του στοιχείου q για οποιοδήποτε $w \in W$. Επειδή τα Reverse Top-k ερωτήματα ελέγχουν αν το q ανήκει στα Top-k στοιχεία, δεν επηρεάζεται από την απουσία τέτοιων στοιχείων καθώς θα βρίσκονταν στην κατάταξη μετά από το στοιχείο q. Έτσι με έναν απλό και αποδοτικό τρόπο παράγεται το $S_i'$ με την αφαίρεση των στοιχείων που κυριαρχούνται από το q από το σύνολο $S_i$.




*Εικόνα 16: Το στοιχείο q κυριαρχεί έναντι των στοιχείων S5,S6,S7,S8. Δηλαδή δεν πρόκειται να επιλεγούν ποτέ έναντι του q, γιατί έχουν χαμηλότερες τιμές από το q σε όλες τι διαστάσεις τους.*

### 2.2.2 Περικοπή Weighting Vector

Ένας απλός τρόπος να καθοριστεί το σύνολο $W_i'$ από το $W_i$ είναι να αφαιρεθούν όλα τα weighting vectors $w \in W_i$ τα οποία δεν ανήκουν στα αποτελέσματα του τοπικού Reverse Top-k ερωτήματος $R(Si, Wi, k, q)$. Τα στοιχεία αυτά μπορούν ασφαλώς να αφαιρεθούν καθώς δεν ανήκουν στα τελικά αποτελέσματα. Η απόδειξη βρίσκεται στην Σημείωση 2.

*Σημείωση 2:*

Οποιοδήποτε weighting vector $w \in W$, το οποίο δεν ανήκει στα τοπικά Reverse Top-k αποτελέσματα $R(Si, Wi, k, q)$ μπορεί ασφαλώς να αφαιρεθεί.

*Απόδειξη: Αρχικά θεωρούμε ότι υπάρχει ένα στοιχείο* $w \in Wi$ *τέτοιο ώστε* $w \notin R(Si, Wi, k, q)$ *και* $w \in R(S, W, k, q)$. *Τότε βάση του ορισμού του Reverse Top-k ερωτήματος υπάρχουν* $k$ *στοιχεία* $p \in TopK(w, q)$ *και* $p \in Si$ *τέτοια ώστε* $fw(p) \leq fw(q)$. *Εφόσον το* $p \in Si$ *τότε ισχύει* $p \in S$ *αφού* $Si \subseteq S$ *και αυτό οδηγεί σε αντίφαση καθώς δεν ισχύει ότι* $\exists\, p \in TopK(w, q)$ *τέτοια ώστε* $fw(q) \leq fw(p)$.

## 2.3 Απλοϊκός αλγόριθμος και ορθότητα

Οι παραπάνω ιδιότητες οδηγούν στον Απλοϊκό αλγόριθμο. Κατά την πρώτη φάση, οι κόμβοι δέχονται ως είσοδο τα $Si, Wi$ σύνολα δεδομένων και εφαρμόζουν έναν οποιοδήποτε state of the art αλγόριθμο Reverse Top-k για τον υπολογισμό των τοπικών Reverse Top-k ερωτημάτων $R(Si, Wi, k, q)$. Τα δεδομένα εξόδου της πρώτης φάσης είναι τα σύνολα $S'i = \{p | p\ Si\ και\ q \prec p\}$ και $W'i = R(S'i, Wi, k, q)$. Στην συνέχεια, κατά την δεύτερη φάση υπάρχουν δύο επιλογές. Η πρώτη είναι να υπάρχει ένας κεντρικός κόμβος ο οποίος να υπολογίζει το τελικό αποτέλεσμα $R\left(\bigcup_{i=0}^{N} S'i, \bigcup_{i=0}^{N} W'i, k, q\right) = R(S, W, k, q)$. Στην Σημείωση 3 αποδεικνύεται η ορθότητα του αλγορίθμου αυτού.

*Σημείωση 3:*

Τα υποσύνολα $\bigcup_{i=0}^{N} S'i$ και $\bigcup_{i=0}^{N} W'i$ καθορίζονται μετά από μεθόδους περικοπής που εφαρμόζονται. Όπου είναι αρκετό για να υπολογιστούν τα σωστά Reverse Top-k αποτελέσματα όταν αναθέτονται σε έναν κεντρικό κόμβο.

$$R\left(\bigcup_{i=0}^{N} S'i, \bigcup_{i=0}^{N} W'i, k, q\right) = R(S, W, k, q).$$

*Απόδειξη: Εφόσον η περικοπή των στοιχείων S γίνεται με τέτοιο τρόπο ώστε* $\mathrm{R}\left(\bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{S'i}, \bigcup_{\mathrm{i}=0}^{\mathrm{N}} Wi, \mathrm{k}, \mathrm{q}\right) = \mathrm{R}(\mathrm{S}, \mathrm{W}, \mathrm{k}, \mathrm{q})$ *και η περικοπή των στοιχείων W γίνεται με τέτοιο τρόπο ώστε* $\mathrm{R}\left(\bigcup_{\mathrm{i}=0}^{\mathrm{N}} Si, \bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{W'i}, \mathrm{k}, \mathrm{q}\right) = \mathrm{R}(\mathrm{S}, \mathrm{W}, \mathrm{k}, \mathrm{q})$, *τότε* $R\left(\bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{S'i}, \bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{W'i}, \mathrm{k}, \mathrm{q}\right) = \mathrm{R}(\mathrm{S}, \mathrm{W}, \mathrm{k}, \mathrm{q})$.

Ωστόσο, ο στόχος είναι να υπολογιστούν τα Reverse Top-k ερωτήματα με κατανεμημένο και παράλληλο τρόπο χωρίς κανένα κεντρικό σημείο όπου θα συγχωνεύονται τα τοπικά αποτελέσματα. Έτσι κατά την δεύτερη φάση πολλοί κόμβοι θα πρέπει να υπολογίσουν τμήματα του τελικού αποτελέσματος, ο καθένας παράλληλα και έπειτα να τα αναφέρουν ως τα τελικά σωστά αποτελέσματα. Στην συνέχεια αποδεικνύεται ότι αν $S''i = \bigcup_{i=0}^{N} S'i$ και $W''i \subset \bigcup_{i=0}^{N} W'i$ είναι τυχαία υποσύνολα του $\bigcup_{i=0}^{N} W'i$ τότε μπορεί να παραχθούν τα ορθά τα Reverse Top-k ερωτήματα. Αυτό οφείλεται στο γεγονός ότι ο υπολογισμός των Reverse Top-k ερωτημάτων για κάθε $w \in W$ είναι ανεξάρτητος από οποιοδήποτε $wi \in W$, αλλά ακόμα χρειάζονται όλα τα στοιχεία $s \in \bigcup_{i=0}^{N} S'i$. Η απόδειξη βρίσκεται στην Σημείωση 4.

Μπορεί να παρατηρηθεί ότι εφαρμόζοντας τον Reverse Top-k αλγόριθμο για το σύνολο $S''\mathrm{i} = \bigcup_{i=0}^{N} S'i$ σε κάθε κόμβο κατά την δεύτερη φάση, ο χρόνος εκτέλεσης θα αυξηθεί καθώς και το κόστος επικοινωνίας μεταξύ των κόμβων. Μια πιο λεπτομερής περιγραφή του Απλοϊκού αλγορίθμου θα δοθεί στο κεφάλαιο Απλοϊκός αλγόριθμος.

*Σημείωση 4:*

Τα υποσύνολα $\bigcup_{i=0}^{N} S'i$ και $\bigcup_{i=0}^{N} W'i$ καθορίζονται μετά από μεθόδους περικοπής που εφαρμόζονται, Όπου είναι αρκετό για να υπολογιστούν τα σωστά Reverse Top-k αποτελέσματα με κατανεμημένο τρόπο αν $S''i = \bigcup_{i=0}^{N} S'i$ και $W''i \subset \bigcup_{i=0}^{N} W'i$.

$$\bigcup_{i=0}^{N} R(S''i, W''i, k, q) = R(S, W, k, q).$$

*Απόδειξη: Εφόσον ισχύει από προηγούμενη απόδειξη ότι* $R\left(\bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{S'i}, \bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{W'i}, \mathrm{k}, \mathrm{q}\right) = \mathrm{R(S, W, k, q)}$. *Αν* $\mathrm{S''i} = \bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{S'i}$ *τότε ισχύει ότι* $\bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{R(S''i, W'i, k, q)} = \mathrm{R(S, W, k, q)}$. *Αν* $\mathrm{W''i} \subset \bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{W'i}$, *τότε ισχύει* $\bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{R(S''i, W'i, k, q)} = \bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{R(S''i, W''i, k, q)}$ *αν* $\bigcup_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{W'i} - \mathrm{W''i}$ *δεν ανήκουν στα Reverse Top-k αποτελέσματα, όπου αυτό μπορεί να εγγυηθεί χρησιμοποιώντας τον RTA αλγόριθμο.*

# 3. Προχωρημένες Ιδιότητες

Σε αυτό το κεφάλαιο, θα αναφερθούν πιο προχωρημένες θεωρητικές ιδιότητες οι οποίες θα οδηγήσουν στην διαμόρφωση ενός καλύτερου αλγορίθμου για την επεξεργασία των Reverse Top-k ερωτημάτων. Η πρώτη τεχνική είναι μια μέθοδος που μειώνει το πλήθος των στοιχείων που ανήκουν στο $S''i$ ομαδοποιώντας παρόμοια weighting vectors στο ίδιο $W''i$. Η δεύτερη ιδιότητα, εκμεταλλεύεται μια γενική περίληψη του συνόλου δεδομένων S προκειμένου να επιταχύνει τον υπολογισμό των Reverse Top-k ερωτημάτων και στοχεύει στην μείωση του $W''i$ απορρίπτοντας weighting vectors βάση μιας περίληψης του συνόλου δεδομένων S.

## 3.1 Ομαδοποίηση και περικοπή δεδομένων

Μέχρι τώρα, η μείωση των στοιχείων του $\bigcup_{i=0}^{N} W_i$ πραγματοποιούταν απορρίπτοντας τα weighting vectors τα οποία δεν βρίσκονταν στα τοπικά Reverse Top-k αποτελέσματα. Έχει αποδειχθεί επίσης ότι τα Reverse Top-k ερωτήματα μπορούν να απαντηθούν σωστά με παράλληλη επεξεργασία των weighting vectors, εφόσον το σύνολο $\bigcup_{i=0}^{N} S'i$ είναι διαθέσιμο σε όλους τους κόμβους.

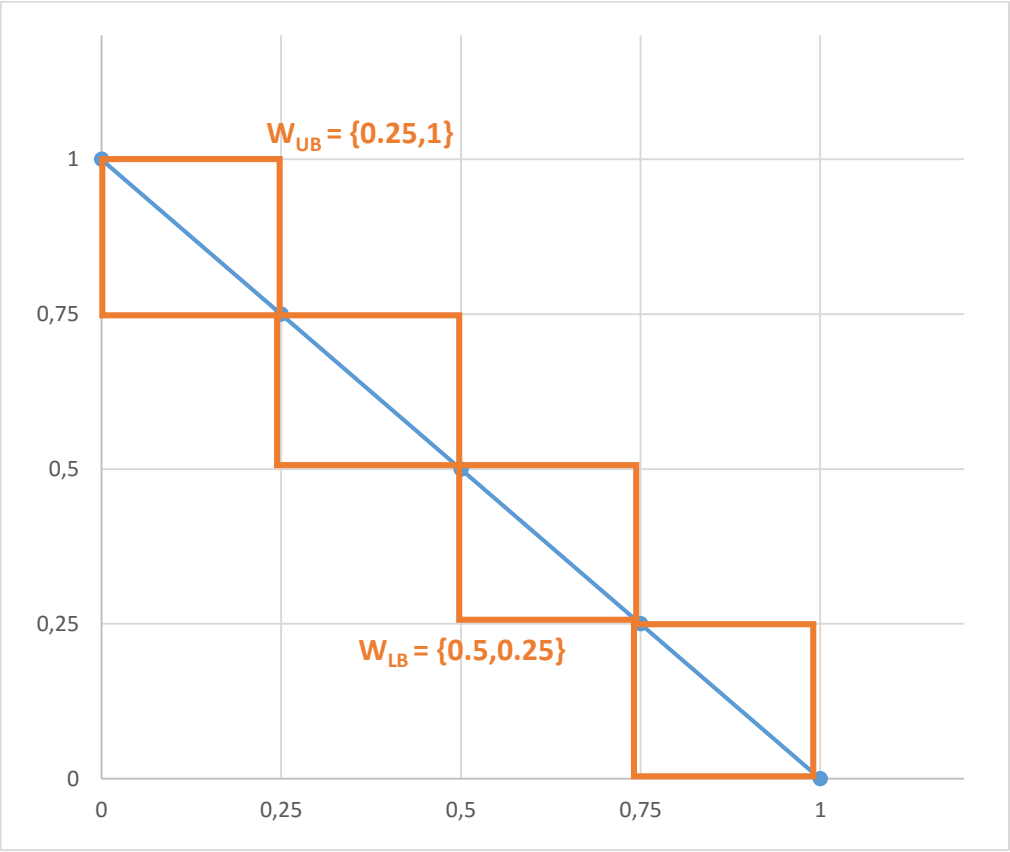
Σε αυτή την ενότητα θα παρουσιαστούν πρόσθετες ιδιότητας προκειμένου να επιτευχθούν δύο στόχοι:

- *Περικοπή στοιχείων:* μείωση του μεγέθους του $S''i$ ($\bigcup_{i=0}^{N} S'i$).
- *Παράλληλη συγχώνευση αποτελεσμάτων:* αναγνώριση υποομάδων $W''i$ και $S''i$ από τα σύνολα δεδομένων $\bigcup_{i=0}^{N} S'i$ και $\bigcup_{i=0}^{N} W'i$, έτσι ώστε να μπορεί να γίνει παράλληλη επεξεργασία και με μία απλή ένωση των αποτελεσμάτων να μπορούν να παραχθούν τα σωστά τελικά αποτελέσματα.

### 3.1.1 Ομαδοποίηση weighting vectors

Η ομαδοποίηση των weighting vectors είναι τέτοια ώστε να δημιουργούνται r ομάδες $P = \{W''1, \dots, W''r\}$ με τέτοια μορφή ώστε να μην είναι αλληλεπικαλυπτόμενες (Δηλαδή $\forall\, i \neq j => W''\text{i} \cap \text{W''j} = \emptyset$) και η ένωση των ομάδων αυτών να καλύπτει όλο το σύνολο $W''$ (Δηλαδή $\text{W''} = \bigcup_{i=0}^{N} W''i$). Πρακτικά η ομαδοποίηση αυτή μπορεί να επιτευχθεί χωρίζοντας τον χώρο των δεδομένων στον οποίο καθορίζονται τα weighting vectors (space partitioning). Δίνεται έμφαση ότι το P καθορίζεται αποκλειστικά βάση των ορίων των ομάδων και δεν περιέχονται πραγματικά weighting vectors. Επίσης αξίζει να σημειωθεί ότι η ομαδοποίηση αυτή γίνεται με μη γνωστικό για τα δεδομένα τρόπο, δηλαδή χωρίς να χρειάζεται πρόσβαση στο σύνολο δεδομένων W.

Δοσμένης μιας συγκεκριμένης ομάδας (τμήματος) $W''i$, είναι δυνατόν η ομάδα αυτή να καθοριστεί με μια οριοθέτηση τέτοια ώστε να υπάρχουν δύο σημεία τα Lower-Bound και Upper-Bound. Το Lower-Bound καθορίζεται από τις ελάχιστες τιμές της κάθε διάστασης του υπόχωρου που αντιπροσωπεύει η ομάδα αυτή, δηλαδή $W_{LB} = \{\text{Min}(W_i''_1), \dots, \text{Min}(W_i''_d)\}$. Αντίστοιχα το Upper-Bound καθορίζεται από τις μέγιστες τιμές της κάθε διάστασης του υπόχωρου που αντιπροσωπεύει η ομάδα αυτή, δηλαδή $W_{UB} = \{\text{Max}(W_i''_1), \dots, \text{Max}(W_i''_d)\}$. Στην Εικόνα 17 απεικονίζεται η ομαδοποίηση ενός χώρου W δύο διαστάσεων σε τέσσερις ομάδες, όπως γίνεται αντιληπτό τα Upper-Bound και Lower-Bound δεν θα μπορούσαν να ανήκουν στο σύνολο W καθώς για το Lower-Bound ισχύει ότι $\sum_{i=0}^{d} di < 1$ ενώ για το Upper-Bound ισχύει ότι $\sum_{i=0}^{d} di > 1$. Το score οποιουδήποτε στοιχείου $p \in S$ για οποιοδήποτε $w \in W''i$ οριοθετείται ως εξής: $f_{LB}(p) < f_w(p) < f_{UB}(p)$. Τα όρια αυτά είναι χρήσιμα προκειμένου να μπορεί να αναγνωριστεί σε ποια περίπτωση ένα σημείο $p \in S$ επηρεάζει ή όχι τα αποτελέσματα του Reverse Top-k αλγορίθμου. Η στενότητα των ορίων αυτών εξαρτάται από το μέγεθος της ομάδας (τμήματος), πχ. όσο πιο μικρή είναι τόσο πιο πολύ αυξάνεται η ομοιότητα των weighting vectors που ανήκουν στην ομάδα αυτή. Έτσι ακόμα και αν η ομαδοποίηση των weighting vectors γινόταν με τυχαίο τρόπο θα ήταν επαρκής για την περικοπή των στοιχείων $p \in S$ (δεν επηρεάζεται η ορθότητα), η αποδοτικότητα όμως ποικίλει ανάλογα με τον τύπο της ομαδοποίησης.




*Εικόνα 17: Απεικόνιση των $W_{LB}$ και $W_{UB}$ για έναν δισδιάστατο χώρο W που έχει τμηματοποιηθεί σε 4 ομάδες. Όπως παρατηρείται τα όρια δεν είναι πραγματικά στοιχεία W.*

### 3.1.2 Περικοπή στοιχείων

Τα όρια τα οποία προαναφέρθηκαν χρησιμοποιούνται προκειμένου να απορρίψουν στοιχεία $p \in S$, πχ. αναγνωρίζοντας ομάδες από weightings vectors $W''i$ για τα οποία το q θα ταξινομείται πάντα σε καλύτερη θέση από κάποια στοιχεία p. Σε αυτό το σημείο προτείνονται δύο προχωρημένες ιδιότητες οι οποίες οδηγούν στην δημιουργία δύο μεθόδων περικοπής στοιχείων. Η πρώτη μέθοδος εξετάζει ένα στοιχείο p και μία ομάδα $W''i$ του P και προσπαθεί να απορρίψει το p συγκρίνοντάς το με το q. Με αυτό τον τρόπο δεν χρειάζεται να κρατούνται στην μνήμη στοιχεία $p \in S$ τα οποία εξετάστηκαν προηγουμένως. Αντίθετα, η δεύτερη μέθοδος διατηρεί για κάθε ομάδα έναν buffer (στην κύρια μνήμη) με k στοιχεία $\in S$ όπου έχουν εξεταστεί προηγουμένως, με τα οποία μπορούν τα απορριφθούν στοιχεία $p \in S$ τα οποία δεν ήταν δυνατό να απορριφθούν από την πρώτη μέθοδο. Δηλαδή μπορεί να χρησιμοποιηθεί συνδυαστικά με την πρώτη μέθοδο.

- **ExtemeScore απόρριψη στοιχείων:**

  Δοσμένης μιας ομάδας $W''i$, εξετάζονται όλα τα στοιχεία p σε σχέση με το q, η Σημείωση 5 προσδιορίζει την συνθήκη με την οποία απορρίπτονται τα στοιχεία p για το $W''i$.

*Σημείωση 5:*

Δοσμένων ενός Reverse Top-k ερωτήματος, ενός q, ενός στοιχείου $p \in S$ και μιας ομάδας $W''i \in P$ που περιέχει weighting vectors τα οποία οριοθετούνται από το $(LB, UB)$, τότε το στοιχείο p δεν επηρεάζει τα αποτελέσματα του reverse Top-k αλγορίθμου για οποιοδήποτε $w \in W''i$ και μπορεί να απορριφθεί με ασφάλεια αν:

$$f_{UB}(q) \leq f_{LB}(p)$$

*Απόδειξη: Λόγω των ιδιοτήτων των ορίων μιας ομάδας ισχύει* $\forall\, p \in S$ *η ανισότητα* $f_{LB}(p) < f_w(p) < f_{UB}(p)$ $\forall\, w \in$ *ομάδα. Έστω δύο στοιχεία* $p, q \in S$*, τότε για αυτά ισχύουν τα εξής* $f_{LB}(p) < f_w(p) < f_{UB}(p)$*,* $f_{LB}(q) < f_w(q) < f_{UB}(q)$ $\forall\, w \in$ *ομάδα. Έστω επίσης ότι ισχύει* $f_{UB}(q) \leq f_{LB}(p)$ *τότε βάση των παραπάνω ισχύει* $f_w(q) < f_w(p)$ $\forall\, w \in$ *στην ομάδα, οπότε το q θα είναι πάντα σε ψηλότερη θέση στην κατάταξη από το p καθώς για κάθε weighting vector που ανήκει στην ομάδα έχει καλύτερη βαθμολογία. Οπότε το στοιχείο p μπορεί να αφαιρεθεί με ασφάλεια.*

- **R-Lists απόρριψη δεδομένων:**

  Για κάθε ομάδα $W''i$, διατηρείται στην κύρια μνήμη ένας buffer με k-στοιχεία $\in S$. Κάθε φορά γίνεται επεξεργασία ενός στοιχείου $p \in S$, έτσι ελέγχεται το περιεχόμενο του buffer και αν περιέχει k-στοιχεία $\{p_1, \dots, p_k\}$ τέτοια ώστε η k-οστή υψηλότερη τιμή του $f_{UB}(p_k)$ είναι καλύτερη από την καλύτερη δυνατή τιμή $f_{LB}(p)$ του στοιχείου p, τότε το στοιχείο p μπορεί να απορριφθεί με ασφάλεια καθώς δεν επηρεάζει την ταξινόμηση του q ως προς το $W''i$. Η Σημείωση 6 αποδεικνύει την ορθότητα της R-Lists απόρριψης δεδομένων.

*Σημείωση 6:*

Δοσμένων ενός Reverse Top-k ερωτήματος, ενός q, ενός στοιχείου $p \in S$ και μιας ομάδας $W''i \in P$ που περιέχει weighting vectors τα οποία οριοθετούνται από το (LB,UB), τότε το στοιχείο p δεν επηρεάζει τα αποτελέσματα του reverse Top-k αλγορίθμου για οποιοδήποτε $w \in W''i$ και μπορεί να απορριφθεί με ασφάλεια αν:

$$\exists \{p_1, .., p_k\} \in S \text{ , τέτοιο ώστε } \forall\, i \in [1, k]: f_{UB}(p_i) \leq f_{LB}(p)$$

*Απόδειξη: Έστω δύο στοιχεία p, b αν* $\mathrm{f_{UB}}(b) \leq \mathrm{f_{LB}}(\mathrm{p})$*, τότε όπως έχει ήδη δειχτεί το b θα είναι πάντα σε καλύτερη θέση στην κατάταξη από το p καθώς θα έχει υψηλότερη βαθμολογία για κάθε στοιχείο* $w \in \mathrm{W''i}$*, δηλαδή* $\mathrm{f}_w(b) < \mathrm{f_w(p)}$*. Σε ένα Top-k ερώτημα επιλέγονται τα k-στοιχεία με την υψηλότερη βαθμολογία. Οπότε από τον ορισμό του Top-k ερωτήματος αν υπάρχουν k στοιχεία με καλύτερη βαθμολογία, τότε το στοιχείο δεν ανήκει στα Top-k αποτελέσματα και αν δεν ανήκει σε αυτά δεν επηρεάζει το Reverse Top-k αποτέλεσμα. Οπότε ισχύει ότι αν* $\exists \{\mathrm{p_1}, .., \mathrm{p_k}\} \in \mathrm{S}$ , τέτοιο ώστε $\forall\, \mathrm{i} \in [1, \mathrm{k}]: \mathrm{f_{UB}}(\mathrm{p_i}) \leq \mathrm{f_{LB}}(\mathrm{p})$

Ο buffer είναι ταξινομημένος με αύξουσα σειρά βάση των τιμών του $f_{UB}(p_i)$, έτσι η υψηλότερη τιμή του $f_{UB}(p_k)$ βρίσκεται στην k-οστή θέση στην σειρά. Επίσης ο buffer ενημερώνεται όταν ένα στοιχείο p που έρχεται για έλεγχο έχει καλύτερη τιμή $f_{UB}(p)$ από την k-οστή τιμή $f_{UB}(p_k)$ του buffer. Σε αυτή την περίπτωση, το $p_k$ απομακρύνεται από τον buffer και το στοιχείο p εισάγεται στον buffer. Η παραπάνω ταξινόμηση γίνεται προκειμένου να αποφευχθεί ο έλεγχος ενός p στοιχείου με όλο τον buffer αλλά μόνο με το τελευταίο στοιχείο αυτού.

## 3.2 Απόρριψη Weighting Vectors

Θεωρώντας ένα n-διάστατο πλέγμα (grid) $G_S$ το οποίο είναι μια περίληψη του συνόλου των δεδομένων $S$, θεωρώντας επίσης ένα κελί $c = [c_l, c_u]$ του πλέγματος $G_S$, ένα στοιχείο $c_l$ το οποίο αναπαριστά την γωνία του κελιού που είναι πιο κοντά στην αρχή των αξόνων και ένα στοιχείο $c_u$ το οποίο αναπαριστά την γωνία του κελιού που είναι το πιο μακρινό από την αρχή των αξόνων (ισχύει: $c_l \prec c_u$). Κάθε κελί $c \in G_S$ κρατά πληροφορία για το πλήθος των στοιχείων $count(c)$ για τα στοιχεία $p \in S$ που εμπεριέχονται στο κελί αυτό.

*Σημείωση 7:*

Δοσμένων ενός κελιού $c = [c_l, c_u]$ ενός πλέγματος $G_S$ του συνόλου δεδομένων $S$, ισχύει ότι $\forall\, p \in c$ και για οποιοδήποτε weighting vector $w \in W$:

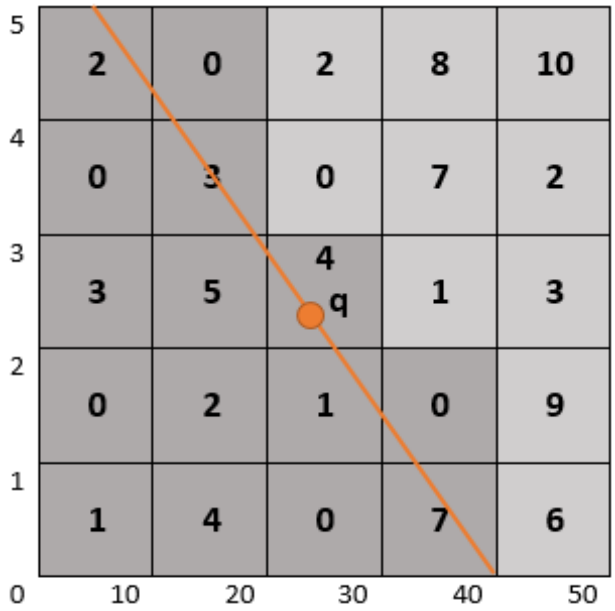
$$f_w(c_l) \leq f_w(p) \leq f_w(c_u)$$

*Απόδειξη: Από την δομή του πλέγματος γνωρίζουμε ήδη ότι $\forall\, p \in c$ ισχύει $c_l \prec p \prec c_u$ καθώς το $c_l$ αποτελείται από τον μικρότερη τιμή κάθε διάστασης στο κελί ενώ αντίθετα το $c_u$ αποτελείται από τις μεγαλύτερες τιμές της κάθε διάστασης. Έτσι αποδεικνύεται ότι ισχύει $f_w(c_l) \leq f_w(p) \leq f_w(c_u)\ \forall\, p \in c$.*

Υποθέτοντας ότι ένα $G_S$ είναι διαθέσιμο σε κάθε κόμβο που επεξεργάζεται τα υποσύνολα $Si$ και $Wi$. Τότε, για οποιοδήποτε $w \in Wi$ μπορεί να υπολογιστεί η προσεγγιστική κατάταξη (approximate rank) του q βάση του $w$, εξετάζοντας τα κελιά του $G_S$. Πιο αναλυτικά, μπορεί να καθοριστεί η ελάχιστη $m(q,w)$ και η μέγιστη $M(q,w)$ κατάταξη του q για το $w$. Αν $m(q,w) \geq k$, τότε το $w$ μπορεί ασφαλώς να απορριφθεί καθώς το q δεν βρίσκεται στο Top-k του $w$. Επιπροσθέτως αν $M(q,w) \leq k$, τότε το $w$ ανήκει σίγουρα στο Reverse Top-k αποτέλεσμα του q, δηλαδή το q βρίσκεται σίγουρα στο Top-k του w. Σε όλες τις άλλες περιπτώσεις δεν μπορούμε να γνωρίζουμε αν το w ανήκει στο Reverse Top-k αποτέλεσμα του q ή όχι.

Λεπτομερέστερα, οι τιμές $m(q,w)$ και $M(q,w)$ καθορίζονται από την score-line (γραμμή βαθμολογίας) όπου στο σημείο που διασταυρώνεται με το q αναπαριστά το w (στην πραγματικότητα, είναι η κάθετη γραμμή προς το w η οποία διασταυρώνεται με το q). Τότε, οι εξής περιπτώσεις μπορούν να αποτυπωθούν για οποιοδήποτε κελί $c = [c_l, c_u]$:

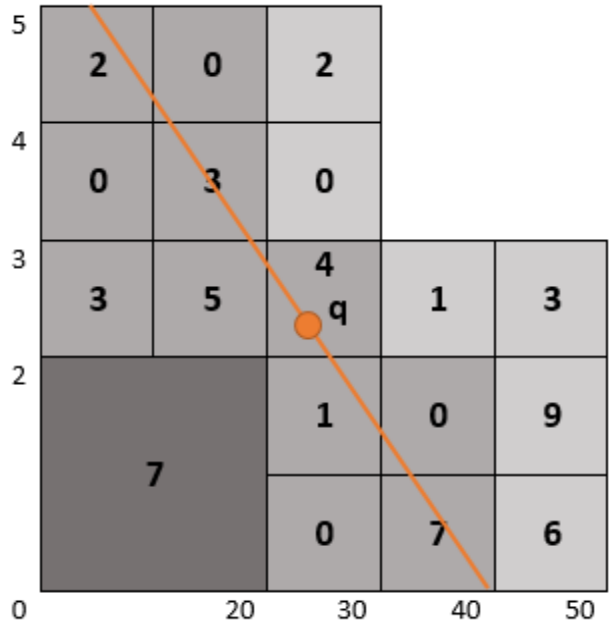
- $\mathrm{f}_w(\mathrm{q}) \leq \mathrm{f}_w(c_l)$: το κελί αυτό δεν επηρεάζει την κατάταξη του q για το w και μπορεί να παραληφθεί.
- $\mathrm{f}_w(c_l) \leq \mathrm{f}_w(q) \leq \mathrm{f}_w(c_u)$: το κελί αυτό μπορεί να επηρεάζει την κατάταξη του q (στην χειρότερη περίπτωση, όλα τα στοιχεία του κατατάσσονται σε υψηλότερη θέση από το q, ενώ στην καλύτερη περίπτωση κατατάσσονται σε χαμηλότερη θέση από το q). Για αυτό προστίθεται το $count(c)$ στο $M(q,w)$.
- $\mathrm{f}_w(c_u) \leq \mathrm{f}_w(\mathrm{q})$: όλα τα στοιχεία που εμπεριέχονται σε αυτό το κελί κατατάσσονται σε υψηλότερη θέση από το q. Έτσι, το $count(c)$ προστίθεται και στο $m(q,w)$ και στο $M(q,w)$.




*Εικόνα 18: Απεικόνιση ενός w ∈ W, σε σχέση με το q σε ένα πλέγμα $G_S$.*

Παράδειγμα 1: Έστω ότι υπάρχει ένα $G_S$ όπως απεικονίζεται στην Εικόνα 18. Σε αυτό το παράδειγμα οι τιμές είναι $m(q,w) = 10$ και $M(q,w) = 32$. Με αυτό τον τρόπο γνωρίζοντας το k είναι δυνατό να απορριφθεί ένα στοιχείο w ή να συμπεριληφθεί στα τελικά αποτελέσματα χωρίς να υπολογιστεί το Reverse Top-k ερώτημα.

Το πλέγμα στην Εικόνα 19 είναι μια περιληπτική έκδοση του $G_S$, όπου τα κελιά τα οποία κυριαρχούν το q (δηλαδή $c_u \prec q$ ) συγχωνεύονται και κρατείται το συνολικό τους άθροισμα καθώς πάντα θα υπάρχει τουλάχιστον αυτό το πλήθος των στοιχείων σε υψηλότερη θέση από το q στην κατάταξη για οποιοδήποτε $w \in Wi$. Ακόμα τα κελιά τα οποία κυριαρχούνται από το q (δηλαδή $q \prec c_l$ ) αφαιρούνται από το grid καθώς τα στοιχεία τα οποία περιέχουν θα είναι πάντα χαμηλότερα στην κατάταξη από το q, οπότε δεν επηρεάζουν το αποτέλεσμα του Reverse Top-k αλγορίθμου.




*Εικόνα 19: Απεικόνιση του παραδείγματος της Εικόνα 18 σε ένα περιληπτικό πλέγμα $G_S$.*

# 4. Reverse Top-k αλγόριθμοι

Σε αυτό το κεφάλαιο παρουσιάζεται ο βασικός Reverse Top-k αλγόριθμος οι οποίος χρησιμοποιείται ευρέως και θα χρησιμοποιηθεί και στα πλαίσια της εργασίας αυτής. Πιο αναλυτικά θα παρουσιαστεί ο RTA αλγόριθμος που προτάθηκε από στην δημοσίευση (1) και αποτελεί τον αλγόριθμο που χρησιμοποιείται και από τους δύο αλγορίθμους που προτείνονται για την αντιμετώπιση του υπολογισμού των Reverse Top-k ερωτημάτων σε κατανεμημένο περιβάλλον.

## 4.1 RTA αλγόριθμος

Στην πιο απλή περίπτωση για την απάντηση ενός Reverse Top-k ερωτήματος υπολογίζονται για όλα τα weighting vectors τα Top-k ερωτήματα και έπειτα ελέγχεται σε ποια Top-k αποτελέσματα υπάρχει το q. Όσα weighting vectors έχουν το q στα Top-k αποτελέσματά τους συμπεριλαμβάνονται στα αποτελέσματα του Reverse Top-k ερωτήματος. Όπως είναι λογικό ένας τέτοιος αλγόριθμος συνοδεύεται από υψηλούς χρόνους εκτέλεσης καθώς ο υπολογισμός ενός Top-k ερωτήματος είναι μια ακριβή πράξη.

Για τον λόγο αυτό προτάθηκε ο RTA (Reverse Top-k Threshold Algorithm) αλγόριθμος (1). Ο αλγόριθμος αυτός δεν εξετάζει τα Top-k ερωτήματα για κάθε weighting vector καθώς με την χρησιμοποίηση ενός κατωφλιού μπορεί να υπολογίσει αν ένα weighting vector δεν έχει στα Top-k αποτελέσματά του το q. Αυτό συμβαίνει για μερικές περιπτώσεις όπως θα παρουσιαστεί παρακάτω. Επιπλέον αξίζει να αναφερθεί ότι η αποδοτικότητα του RTA αλγορίθμου αυξάνεται όταν δέχεται τα weighting vectors ταξινομημένα. Πιο αναλυτικά ο αλγόριθμος αυτός κρατά τα αποτελέσματα του τελευταίου Top-k ερωτήματος που υπολογίστηκε σε έναν buffer. Έπειτα για ένα weighting vector υπολογίζει την βαθμολογία των k στοιχείων του buffer και θέτει ως κατώφλι το στοιχείο με την χειρότερη βαθμολογία. Αν το q έχει χειρότερη βαθμολογία από την βαθμολογία του κατωφλιού, τότε το weighting vector αυτό μπορεί με ασφάλεια να αφαιρεθεί από τα αποτελέσματα καθώς δεν ανήκει σε αυτά. Αυτό συμβαίνει διότι υπάρχουν ήδη k-στοιχεία με καλύτερη βαθμολογία από το q για αυτό το weighing vector. Παρακάτω παρουσιάζεται ο ψευδοκώδικας του RTA αλγορίθμου.

| **RTA Αλγόριθμος** | |
|---|---|
| **1.** | $Στοιχεία\ εισόδου: S, W, q, k$ |
| **2.** | $Στοιχεία\ εξόδου: ReverseTopk\ (q)$ |
| **3.** | $W' = \{\emptyset\}$ |
| **4.** | $buffer = \{\emptyset\}$ |
| **5.** | $threshold = \infty$ |
| **6.** | $for\ (W_i \in W\ )\ \{$ |
| **7.** | $\quad if\ (f_{w_i}(q) \leq threshold\ )\ \{$ |
| **8.** | $\quad\quad buffer = TOPk(w_i)$ |
| **9.** | $\quad\quad if\ (f_{w_i}(q) \leq \max\{f_{w_i}(buffer)\}\ )\ \{$ |
| **10.** | $\quad\quad\quad W' += w_i$ |
| **11.** | $\quad\quad \}$ |
| **12.** | $\quad \}$ |
| **13.** | $\quad threshold = \max\{f_{w_i}(buffer)\}$ |
| **14.** | $\}$ |
| **15.** | $return\ W'$ |

*Αλγόριθμος 1: Παρουσιάζεται ο RTA αλγόριθμος όπως περιγράφεται στην δημοσίευση (1).*

Σε αυτό το σημείο κρίνεται αναγκαίο να δοθεί ένα παράδειγμα προκειμένου να γίνει πιο εύκολα κατανοητή η λειτουργία του RTA αλγορίθμου. Συνεχίζοντας το παράδειγμα με το σύστημα συστάσεων των ξενοδοχείων που παρουσιάστηκε στην εισαγωγή, θεωρούμε ότι υπάρχουν 5 ξενοδοχεία και 5 χρήστες, όπως παρουσιάζεται στην Εικόνα 20 και στην Εικόνα 21 αντίστοιχα. Σημειώνεται ότι τα weighting vectors των χρηστών δίνονται ταξινομημένα. Έστω επίσης το ξενοδοχείο Πέτρινο το οποίο είναι το q και έχει τιμή 20 ευρώ ενώ η απόστασή του από την θάλασσά αγγίζει τα 600 μέτρα. Με τα παρακάτω στοιχεία θα υπολογιστεί το Reverse Top-2 ερώτημα με την χρήση του RTA αλγορίθμου.

Αρχικά θα υπολογιστεί το Top-2 του χρήστη Θανάση το οποίο είναι το εξής [Ανεμόμυλος, Βυθός]. Το Top-2 του Θανάση αποθηκεύεται στον buffer. Έπειτα τίθεται ως κατώφλι η χειρότερη βαθμολογία των στοιχείων του buffer βάση με τις προτιμήσεις του χρήστη Παναγιώτη. Η χειρότερη βαθμολογία είναι αυτή του ξενοδοχείου Βυθού και είναι η εξής $f_{Παναγιώτης}(Βυθός) = 200$. Έπειτα υπολογίζεται η βαθμολογία του q για τις προτιμήσεις του χρήστη Παναγιώτη η οποία είναι $f_{Παναγιώτης}(q) = 310$. Αφού η βαθμολογία του q είναι χειρότερη από το κατώφλι, δεν χρειάζεται να εξεταστεί το Top-2 του χρήστη Παναγιώτη καθώς υπάρχουν 2 καλύτερα στοιχεία από το q, οπότε ο χρήστης αυτός δεν ανήκει στα Reverse Top-2 αποτέλεσμα. Έπειτα διατηρώντας τον ίδιο buffer τίθεται ως κατώφλι η χειρότερη βαθμολογία των στοιχείων του buffer για τις προτιμήσεις του χρήστη Μάρκου η οποία είναι $f_{Μάρκος}(Βυθός) = 180$. Αντίστοιχα υπολογίζεται η βαθμολογία του q για τον Μάρκο $f_{Μάρκος}(q) = 368$. Αφού η βαθμολογία του q είναι χειρότερη από αυτή του κατωφλιού δεν υπολογίζεται το Top-2 του Μάρκου. Αντίστοιχα δεν υπολογίζεται το Top-2 του χρήστη Έλενα καθώς η βαθμολογία του q είναι χειρότερη από αυτή του κατωφλιού. Στην συνέχεια ανανεώνεται η τιμή του κατωφλιού βάση τις προτιμήσεις του χρήστη Αποστόλη όπου είναι η εξής τιμή $f_{Αποστόλη}(κατώφλι) = 280$. Αντίστοιχα υπολογίζεται η βαθμολογία του q για τις προτιμήσεις του χρήστη Αποστόλη η οποία είναι η εξής $f_{Αποστόλη}(q) = 78$. Αφού η τιμή του q είναι καλύτερη από αυτή του κατωφλιού υπολογίζεται το Top-2 του Αποστόλη που είναι το εξής [Ανεμόμυλος, q]. Ως αποτέλεσμα το Reverse Top-2 ερώτημα θα επιστρέψει τον χρήστη Αποστόλη. Με αυτό τον τρόπο υπολογίστηκαν μόνο 2 Top-k ερωτήματα από τα 5 που θα έπρεπε να υπολογιστούν χωρίς την χρήση του RTA αλγορίθμου.

| Ξενοδοχείο | Τιμή | Απόσταση από θάλασσα |
|---|---|---|
| Αστερίας | 50 € | 800 m. |
| Βυθός | 300 € | 100 m. |
| Πανόραμα | 70 € | 700 m. |
| Ανεμόμυλος | 40 € | 250 m. |
| Ακρογιαλιά | 50 € | 500 m. |

*Εικόνα 20: Στην εικόνα αυτή παρουσιάζονται 5 ξενοδοχεία. Οι διαστάσεις είναι 2, η τιμή και η απόσταση από την θάλασσα.*

| Ταξιδιώτης | Βάρος τιμής | Βάρος απόστασης από την θάλασσα |
|---|---|---|
| Θανάσης | 0,5 | 0,5 |
| Παναγιώτης | 0,5 | 0,5 |
| Μάρκος | 0,4 | 0,6 |
| Έλενα | 0,6 | 0,4 |
| Αποστόλης | 0,9 | 0,1 |

*Εικόνα 21: Παρουσιάζονται τα weighting vectors 5 χρηστών. Η πρώτη διάσταση αφορά το βάρος της τιμής ενώ η δεύτερη διάσταση το βάρος της απόστασης από την θάλασσα.*

# 5. Απλοϊκός αλγόριθμος

Σε αυτό το κεφάλαιο, παρουσιάζεται μια πρώτη λύση για το πρόβλημα της παράλληλης και κατανεμημένης επεξεργασίας των Reverse Top-k ερωτημάτων, το οποίο βασίζεται στις βασικές ιδιότητες που παρουσιάστηκαν προηγουμένως. Ο προτεινόμενος αλγόριθμος ονομάζεται Απλοϊκός αλγόριθμος και είναι σχεδιασμένος για το δημοφιλές MapReduce Framework το οποίο έχει καθιερωθεί για την παράλληλη και κατανεμημένη επεξεργασία δεδομένων τα τελευταία χρόνια.

Η επεξεργασία των υποσυνόλων των $S$, $W$ γίνεται όπως προαναφέρθηκε στα προηγούμενα κεφάλαια σε δύο φάσεις, την Map και την Reduce φάση για το framework αυτό. Κατά την Map φάση πραγματοποιείται η περικοπή στοιχείων $S$, $W$ και κατά την Reduce φάση πραγματοποιείται ο υπολογισμός και η ένωση των τελικών αποτελεσμάτων.

Τέλος γίνεται μια αναφορά στην υλοποίηση καθώς και σε τεχνικές λεπτομέρειες. Η πειραματική αξιολόγηση του αλγορίθμου αυτού θα γίνει σε επόμενο κεφάλαιο όπου θα παρουσιαστούν και θα αιτιολογηθούν κάποια πειράματα που εκτελέστηκαν.

## 5.1 Map φάση

Κατά την Map φάση, η Map διαδικασία εκτελείται σε κάθε Mapper και παίρνει ως είσοδο δύο υποσύνολα τα $S_i \subset S$ και $W_i \subset W$. Επίσης είναι διαθέσιμο το πλήθος των Reduces που έχει καθοριστεί από τον χρήστη. Σε πιο τεχνικούς όρους, εφαρμόζεται ένα εξατομικευμένο InputFormat το οποίο δημιουργεί InputSplits όπου αυτά περιέχουν στοιχεία και από τα δύο σύνολα δεδομένων $S, W$. Επιπλέον τα στοιχεία αυτά είναι ταξινομημένα με τέτοιο τρόπο έτσι ώστε να διαβάζονται από τον Mapper πρώτα τα στοιχεία $p \in Si$ και μετά τα στοιχεία $w \in Wi$. Περαιτέρω λεπτομέρειες για την τεχνική υλοποίηση θα δοθούν στην συνέχεια του κεφαλαίου.

Ο Αλγόριθμος 2 περιγράφει την τοπική επεξεργασία και την μέθοδο περικοπής που εφαρμόζεται κατά την Map φάση του Απλοϊκού αλγορίθμου. Όσο η είσοδος αφορά στοιχεία $p \in Si$ (γραμμή: 4-10), εφαρμόζεται η περικοπή βάση κυριαρχίας σε σχέση με το q (γραμμή: 5). Με αυτό τον τρόπο απορρίπτονται όλα τα στοιχεία τα οποία κυριαρχούνται από το q, όπως αναφέρθηκε στο κεφάλαιο Περικοπή βάση κυριαρχίας. Τα στοιχεία τα οποία επιβίωσαν από την περικοπή βάση κυριαρχίας αποτελούν το σύνολο $S'i$ και αποθηκεύονται στην κύρια μνήμη για τον υπολογισμό του αποτελέσματος του τοπικού Reverse Top-k αλγορίθμου. Έπειτα κατά την διάρκεια της ανάγνωσης των στοιχείων $w \in Wi$ (γραμμή: 11-13), αποθηκεύονται τα στοιχεία αυτά στην κύρια μνήμη (γραμμή: 12). Υπενθυμίζεται ότι τα στοιχεία $w \in Wi$ διαβάζονται μετά από τα στοιχεία $p \in Si$. Αφού αναγνωστούν όλα τα δεδομένα που δόθηκαν ως είσοδο, πραγματοποιείται η περικοπή των στοιχείων $w \in Wi$ βάση του υπολογισμού του τοπικού Reverse Top-k αλγορίθμου (γραμμή: 15-28). Αρχικά ελέγχεται (γραμμή: 16-18) αν το πλήθος των στοιχείων $p \in S'i$ τα οποία επιβίωσαν από την περικοπή βάση κυριαρχίας είναι λιγότερα από k, αν ναι τότε δεν υπολογίζεται ο τοπικός Reverse Top-k αλγόριθμος καθώς το q θα βρίσκεται πάντα μέσα στα αποτελέσματα. Έτσι σε αυτή την περίπτωση δεν γίνεται περικοπή κανενός στοιχείου $w \in Wi$. Σε αντίθετη περίπτωση (γραμμή: 19-22), πραγματοποιείται ταξινόμηση των στοιχείων $w \in Wi$ καθώς προτείνεται προκειμένου να εκτελεστεί αποδοτικά ο Reverse Top-k αλγόριθμος RTA (1) και έπειτα εκτελείται ο αλγόριθμος αυτός όπου τα αποτελέσματά του δίνονται για επεξεργασία στην επόμενη φάση.

Παρατηρείται ότι το σύνολο $S'i$ στέλνεται σε όλους τους Reducers (γραμμή: 7-8), ενώ τα στοιχεία $w \in W'i$ κατανέμονται με τυχαίο τρόπο σε όλους τους Reducers (γραμμή: 23-27). Οπότε κάθε στοιχείο $p \in S'i$ θα σταλεί σε όλους τους Reducers, ενώ κάθε στοιχείο $w \in W'i$ θα σταλεί σε έναν μόνο Reducer. Στην περίπτωση που ο χρήστης θέσει το πλήθος των Reducers ίσο με 1, τότε θα σταλούν όλα τα στοιχεία σε έναν Reducer.

| **Απλοϊκός Αλγόριθμος: Map φάση** | |
|---|---|
| **1.** | *Στοιχεία εισόδου:* $Si, Wi, q, k, reducersNumber$ |
| **2.** | *Στοιχεία εξόδου:* $S'i, W'i$ |
| **3.** | *MAP (Element x) {* |
| **4.** | *if (* $x \in Si$ *) {* |
| **5.** | *if (* $q \prec x$ *) {* |
| **6.** | $S'i \mathrel{+}= x$ |
| **7.** | *for (* $j = 0 \ldots reducersNumber$ *)* |
| **8.** | $Output(j, x)$ |
| **9.** | *}* |
| **10.** | *}* |
| **11.** | *else if (* $x \in Wi$ *) {* |
| **12.** | $Wi \mathrel{+}= x$ |
| **13.** | *}* |
| **14.** | *}* |
| **15.** | *CLEANUP () {* |
| **16.** | *if (* $Wi.size < k$ *) {* |
| **17.** | $W'i = Wi$ |
| **18.** | *}* |
| **19.** | *else {* |
| **20.** | $Sort(Wi)$ |
| **21.** | $W'i = RTA(S'i, Wi, q, k)$ |
| **22.** | *}* |
| **23.** | $j = 0$ |
| **24.** | *for (* $z = 0 \ldots W'i.size$ *) {* |
| **25.** | $j = j + 1$ |
| **26.** | $Output(j\%reducersNumber, W'i[z])$ |
| **27.** | *}* |
| **28.** | *}* |

*Αλγόριθμος 2: Παρουσιάζεται ο ψευδοκώδικας του Απλοϊκού αλγορίθμου. Διευκρινίζεται ότι αφού κληθεί η MAP μέθοδος για κάθε στοιχείο x που δόθηκε ως είσοδος έπειτα καλείται η CLEANUP. Επιπλέον η πρώτη παράμετρος της output μεθόδου ορίζει σε ποιον Reducer θα σταλεί το στοιχείο της δεύτερης παραμέτρου της μεθόδου αυτής.*

## 5.2 Reduce φάση

Τα τοπικά σύνολα δεδομένων $\bigcup_{i=0}^{N} S'i$ και $\bigcup_{i=0}^{N} W'i$ που δόθηκαν ως έξοδο από την Map φάση πρέπει να επεξεργαστούνε προκειμένου να εφαρμοστεί η ένωση των τελικών αποτελεσμάτων. Μια απλή λύση είναι να χρησιμοποιηθεί ένας μοναδικός Reducer ο οποίος θα δέχεται όλα τα σύνολα $\bigcup_{i=0}^{N} S'i$ και $\bigcup_{i=0}^{N} W'i$ και θα εκτελεί έναν Reverse Top-k αλγόριθμο στην κύρια μνήμη προκειμένου να εξάγει τα τελικά αποτελέσματα. Προφανώς η λύση αυτή έχει περιορισμούς στην δυνατότητα κλιμάκωσης λόγω του μεγέθους των συνόλων $\bigcup_{i=0}^{N} S'i$ και $\bigcup_{i=0}^{N} W'i$, καθώς υπάρχει ένα όριο στην κύρια μνήμη του κόμβου που εκτελείται ο Reducer. Από την άλλη, μπορεί να διασπαστεί το σύνολο $\bigcup_{i=0}^{N} W'i$ και τα τμήματα που θα προκύψουνε να μοιραστούν στους διαθέσιμους Reducers. Παρόλα αυτά, το σύνολο $\bigcup_{i=0}^{N} S'i$ θα πρέπει να σταλεί σε όλους τους Reducers. Σε αυτή την περίπτωση κάθε Reducer θα παράγει τα τοπικά αποτελέσματα του Reverse Top-k αλγορίθμου και τα τελικά αποτελέσματα θα προκύψουν από την ένωση των τοπικών αυτών αποτελεσμάτων. Ως εκ τούτου, η λύση αυτή συνοδεύεται από μία αδυναμία η οποία είναι ότι χρειάζεται να μεταφερθεί όλο το σύνολο $\bigcup_{i=0}^{N} S'i$ πολλαπλές φορές στο δίκτυο, το οποίο έχει υψηλό κόστος επικοινωνίας και επίσης θα σπαταληθεί περισσότερος χρόνος στο shuffling. Πιο συγκεκριμένα το σύνολο $\bigcup_{i=0}^{N} S'i$ θα χρειαστεί να μεταφερθεί r φορές (όσο και το πλήθος των Reducers). Τέλος μια ακόμα αδυναμία της λύσης αυτής είναι ότι το σύνολο $\bigcup_{i=0}^{N} S'i$ θα πρέπει να αποθηκεύεται στην κύρια μνήμη του κάθε Reducer, κάτι που μπορεί να οδηγήσει σε κατάρρευση του κόμβου που εκτελείται ο Reducer λόγω του περιορισμού της κύριας μνήμης. Ο Αλγόριθμος 3 περιγράφει τον ψευδοκώδικα της Reduce φάσης.

| **Απλοϊκός Αλγόριθμος: Reduce φάση** | |
|---|---|
| **1.** | *Στοιχεία εισόδου:* $S''i, W''i, q, k$ |
| **2.** | *Στοιχεία εξόδου:* $R(S''i, W''i, k, q)$ |
| **3.** | *REDUCE (key, List<Element> List) {* |
| **4.** | *previous = empty* |
| **5.** | *for (Element x in List) {* |
| **6.** | *if (* $x \in S''i$ *) {* |
| **7.** | $S''i \mathrel{+}= x$ |
| **8.** | *}* |
| **9.** | *else if (* $x \in W''i$ *) {* |
| **10.** | *if(*$RTA.belongsToResults(x, S''i, k, q)$*) {* |
| **11.** | *output (x);* |
| **12.** | *}* |
| **13.** | *}* |
| **14.** | *}* |
| **15.** | *}* |

*Αλγόριθμος 3: Παρουσιάζεται ο ψευδοκώδικας του Απλοϊκού αλγορίθμου για την Reduce φάση. Διευκρινίζεται ότι τα στοιχεία* $p \in S$ *έρχονται πρώτα και έπειτα ακολουθούν τα στοιχεία* $w \in W$.

## 5.3 Ανάλυση Πολυπλοκότητας

Σε όρους χωρικής πολυπλοκότητας, η Map φάση απαιτεί $O(|S_i'| * |W_i|)$ κατανάλωση μνήμης. Επίσης το πλήθος των δεδομένων κατά το shuffling είναι $O(|\cup_{i=0}^{N} S_i'| + |\cup_{i=0}^{N} W_i'|)$ όταν χρησιμοποιείται ένας Reducer, ενώ είναι $O(r * |\cup_{i=0}^{N} S_i'| + |\cup_{i=0}^{N} W_i'|)$ όταν χρησιμοποιούνται r στο πλήθος Reducers. Κατά την Reduce φάση, η χωρική πολυπλοκότητα είναι $O(|\cup_{i=0}^{N} S_i'| + |\cup_{i=0}^{N} W_i'|)$ όταν χρησιμοποιείται ένας Reducer, ενώ είναι $O(|\cup_{i=0}^{N} S_i'| + \frac{|\cup_{i=0}^{N} W_i'|}{r})$ όταν χρησιμοποιούνται r στο πλήθος Reducers. Περιληπτικά το κύριο πρόβλημα είναι το μέγεθος του $\cup_{i=0}^{N} S_i'$, το οποίο βρίσκεται σε κάθε Reducer και δεν πρέπει να είναι μεγαλύτερο της διαθέσιμης κύριας μνήμης του.

Σχετικά με την χρονική πολυπλοκότητα, κάθε Mapper χρειάζεται να επεξεργαστεί στην χειρότερη περίπτωση $O(|W_i|)$ Top-k ερωτήματα, λόγω της χρήσης του RTA αλγορίθμου (1). Κατά την Reduce φάση, όταν χρησιμοποιείται ένας Reducer στην χειρότερη περίπτωση θα γίνει επεξεργασία $O(|\cup_{i=0}^{N} W_i'|)$ Top-k ερωτημάτων ενώ για r στο πλήθος Reducers ο αριθμός αυτός ανέρχεται στα $O(\frac{|\cup_{i=0}^{N} W_i'|}{r})$.

## 5.4 Περιγραφή υλοποίησης

Στην ενότητα αυτή θα παρουσιαστεί η υλοποίηση του αλγορίθμου. Αρχικά θα γίνει μια αναφορά στα στοιχεία που χρησιμοποιήθηκαν για την εκτέλεση του αλγορίθμου. Έπειτα θα παρουσιαστούν τεχνικά στοιχεία σχετικά με το Hadoop, σημειώνεται ότι η σειρά παρουσίασης θα είναι αντίστοιχη με την ροή των φάσεων που εκτελεί το Hadoop. Τέλος θα παρουσιαστούν λεπτομέρειες σχετικά με την δομή του κώδικα και θα αναλυθεί περαιτέρω ότι κρίνεται αναγκαίο αποφεύγοντας αχρείαστες λεπτομέρειες καθώς η υλοποίηση συνοδεύεται από Javadoc το οποίο περιγράφει όλες τις λεπτομέρειες σχετικά με την υλοποίηση.

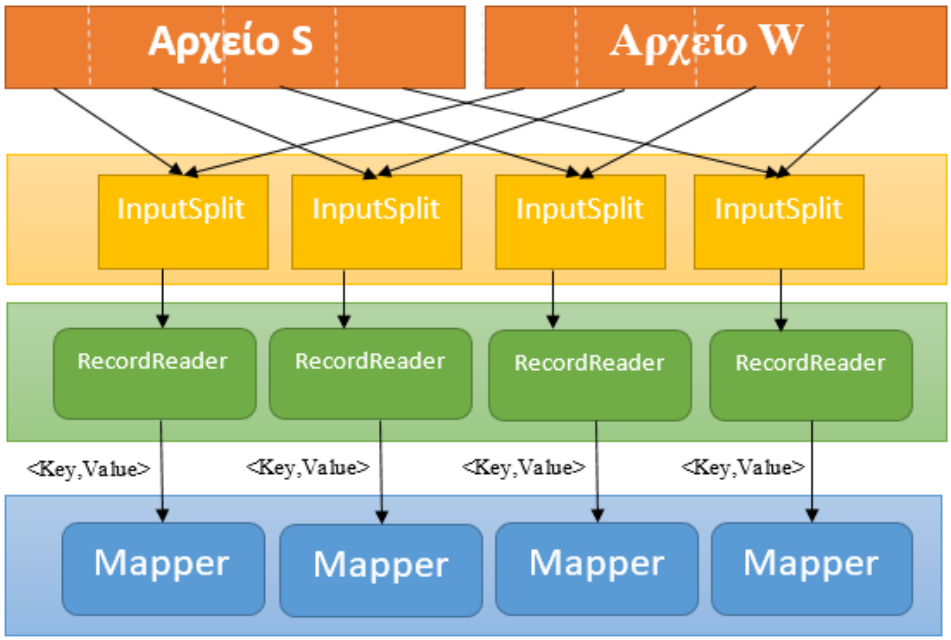
### 5.4.1 Βασικά χαρακτηριστικά

Η γλώσσα προγραμματισμού που επιλέχτηκε για την υλοποίηση του αλγορίθμου είναι η Java και πιο συγκεκριμένα η έκδοση στην οποία βασίζεται η υλοποίηση είναι η 1.7, όσο αναφορά την έκδοση του Hadoop API που χρησιμοποιήθηκε είναι η 2.6.0. Αποφασίστηκαν οι παραπάνω εκδόσεις προκειμένου να είναι συμβατές με το περιβάλλον εκτέλεσης των πειραμάτων. Όσο αναφορά την υλοποίηση συνοδεύεται από Javadoc για την περιγραφή της λειτουργικότητας των μεθόδων και των κλάσεων, επιπλέον δόθηκε έμφαση προκειμένου ο κώδικας να είναι αυτό-περιγραφικός (self-documented) έτσι ώστε να μπορεί να μελετηθεί εύκολα.

### 5.4.2 Περιγραφή υλοποίησης των στοιχείων του Hadoop

Αρχικά πρέπει να καθοριστεί ο τύπος δεδομένων που δέχονται ως είσοδο οι Mappers. Όπως αναφέρθηκε κατά την περιγραφή του αλγορίθμου κάθε Mapper πρέπει να έχει δεδομένα και από τα δύο σύνολα δεδομένων $S, W$. Όντας ανάγκη λοιπόν να διαβάζονται πάντα και τα δύο αρχεία $S, W$ από κάθε Mapper γίνεται αντιληπτό ότι πρέπει να δημιουργούνται InputSplits τα οποία θα περιέχουν δείκτες προς τμήματα και των δύο αρχείων $S, W$. Η διαίρεση των αρχείων σε InputSplits καθορίζεται από το InputFormat (5), το οποίο διαβάζει τα αρχεία που έδωσε ο χρήστης ως είσοδο στην MapReduce διαδικασία και δημιουργεί τα InputSplits. Έτσι δημιουργήθηκε ένα προσαρμοσμένο σε αυτή την περίπτωση InputFormat το οποίο παίρνει το συνολικό μέγεθος των δύο αρχείων $S, W$, έπειτα διαβάζει το μέγεθος του InputSplit που καθόρισε ο χρήστης και υπολογίζει ποιο είναι το ποσοστό του μεγέθους του InputSplit που αντιστοιχεί στο S και στο W. Για παράδειγμα έστω ότι ο χρήστης όρισε ως μέγεθος InputSplit 10MB και δόθηκαν ως είσοδος τα αρχεία S,W με μέγεθος 90MB και 10MB αντίστοιχα. Προκειμένου κάθε InputSplit να περιέχει τμήμα και από τα δύο αρχεία θα περιέχει 9MB από το αρχείο S και 1MB από το αρχείο W. Ο τρόπος δημιουργίας των InputSplits απεικονίζεται στην Εικόνα 22.

Δημιουργώντας ένα InputFormat είναι εφικτό ο κάθε Mapper να διαβάζει στοιχεία και από τα δύο σύνολα δεδομένων αλλά χωρίς να γνωρίζει από ποιο σύνολο δεδομένων προήλθε ένα στοιχείο, κάτι που κρίνεται απαραίτητο. Όπως αναφέρθηκε στην εισαγωγή ο RecordReader είναι υπεύθυνος για την ανάγνωση των InputSplits και την μετατροπή αυτών σε ζευγάρια κλειδιών τιμής (δηλ. <key,value> pairs) που δέχονται ως είσοδο οι Mappers. Έτσι δημιουργήθηκε ένας προσαρμοσμένος στην περίπτωση RecordReader και ένας νέος τύπος τιμής (value) όπου όταν ο RecordReader διαβάζει μια γραμμή από ένα InputSplit στέλνει ως κλειδί (key) τον αριθμό της γραμμής που διάβασε και ως τιμή (value) στέλνει την γραμμή του αρχείου μαζί με το όνομα το αρχείου. Με αυτό τον τρόπο έχοντας ορίσει ο χρήστης τα αρχεία που βρίσκονται τα σύνολα δεδομένων S,W καθώς και ποιο αρχείο αντιστοιχεί σε ποιο σύνολο είναι δυνατόν οι Mappers να γνωρίζουν την προέλευση όλων των στοιχείων.



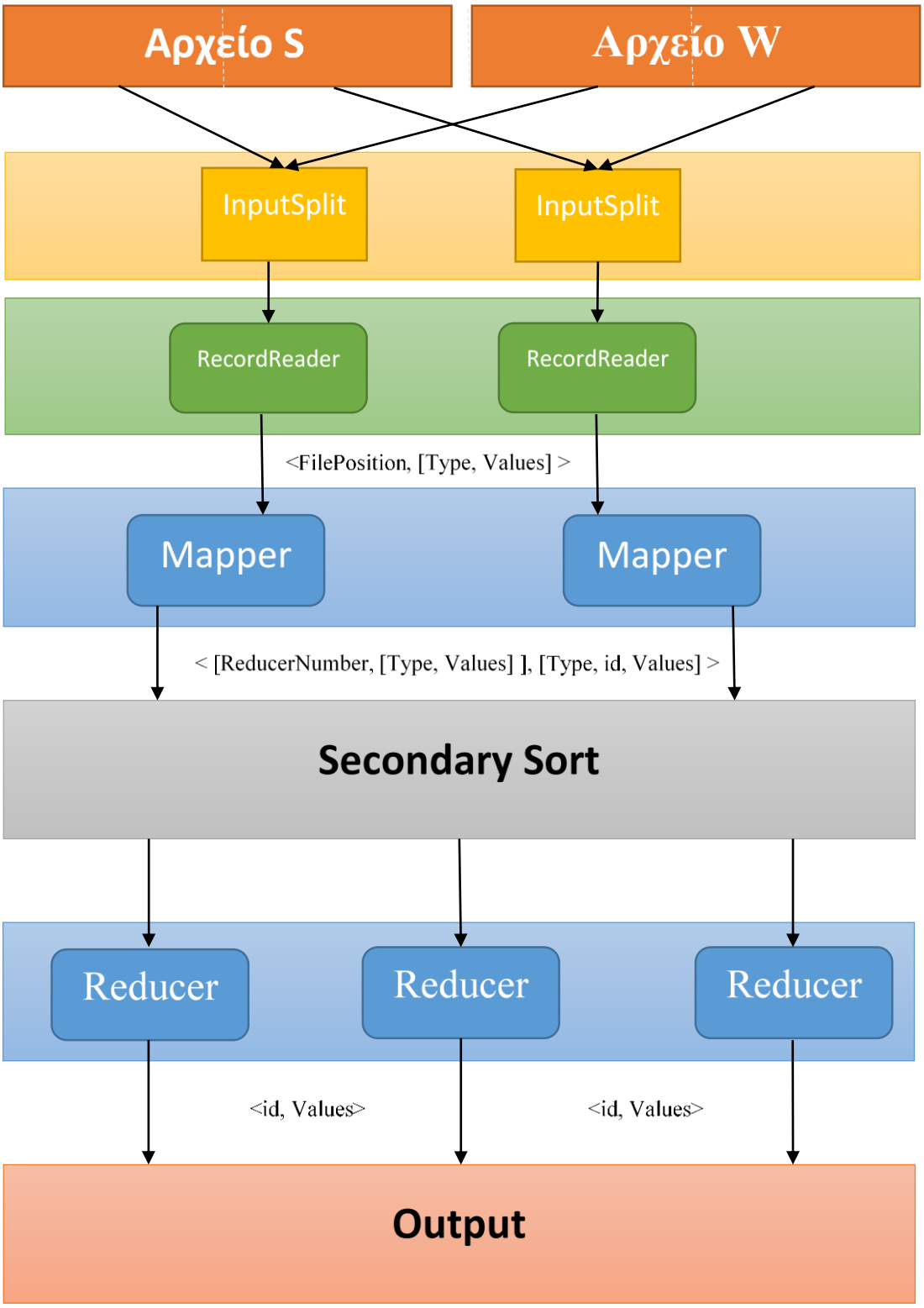

*Εικόνα 22: Απεικόνιση του τρόπου δημιουργίας των InputSplits προκειμένου να περιέχουν δεδομένα και από τα αρχεία S,W.*

Πλέον κάθε Mapper δέχεται ως είσοδο στοιχεία και από τα δύο σύνολα δεδομένων και γνωρίζει την προέλευση του κάθε στοιχείου. Η δουλειά των Mappers είναι να στείλουν προς την Reduce φάση όσο λιγότερα δεδομένα γίνεται, ο αλγόριθμος για την περικοπή των στοιχείων και την αποστολή των δεδομένων στην Reduce φάση παρουσιάζεται στο πεδίο Αλγόριθμος 2. Τα δεδομένα τα οποία στέλνει προς τους Reducers είναι στοιχεία τα οποία ανήκουν στα σύνολα δεδομένων $S, W$ και έχουν ως πληροφορία έναν μοναδικό αριθμό (id) ανά σύνολο δεδομένων και μία λίστα με δεκαδικές τιμές (το μέγεθος της λίστας είναι ίσο με το πλήθος των διαστάσεων). Προκειμένου οι Reducers να γνωρίζουν το σύνολο των δεδομένων όπου προέρχεται το κάθε στοιχείο δημιουργείται ένας νέος τύπος τιμής (value) ο οποίος κρατά τα δεδομένα του στοιχείου καθώς και τον τύπο τους (δηλ. αν ανήκουν στο σύνολο S ή W). Το κλειδί το οποίο χρησιμοποιείται είναι ένα σύνθετο κλειδί το οποίο αποτελείται από την τιμή (value) και τον αριθμό του Reducer στον οποίο πρέπει να σταλεί το στοιχείο (reducerKey), προκειμένου να φτάσουν τα στοιχεία ταξινομημένα στους Reducers (Secondary Sorting) (4). Δηλαδή ο τύπος εξόδου της Map φάσης έχει την εξής μορφή <(reducerKey,value),value>. Σε αυτό το σημείο πρέπει να σημειωθεί ότι ο σύνθετος αυτός τύπος του κλειδιού είναι μία κλάση η οποία οφείλει να υλοποιεί (implements) την διεπαφή (interface) WritableComparable του Hadoop API (5). Έτσι στην κλάση ορίζεται η μέθοδος ταξινόμησης των στοιχείων η οποία υλοποιήθηκε ώστε να ταξινομεί τα στοιχεία ως εξής: πρώτα έρχονται τα στοιχεία που ανήκουν στο σύνολο δεδομένων S και μετά αυτά του συνόλου δεδομένων W, τα στοιχεία του συνόλου δεδομένων W ταξινομούνται μεταξύ τους όπως αναφέρεται στο (1).

Τα δεδομένα που στέλνουν οι Mappers προς τους Reducer διαβάζονται πρώτα από έναν Partitioner ο οποίος είναι υπεύθυνος για τον διαμοιρασμό των στοιχείων στους Reducers. Επειδή μόνο ένα τμήμα του κλειδιού πρέπει να ορίζει που θα σταλεί το στοιχείο και όχι όλο το κλειδί δημιουργείται ένας νέος Partitioner. Ο Partitioner αυτός λαμβάνει υπόψη του μόνο τον αριθμό του Reducer (reducerKey) από το κλειδί και δεν κοιτά την τιμή που κουβαλά το κλειδί έτσι ώστε να σταλεί το στοιχείο στον σωστό Reducer.

Αφού ο Partitioner στείλει τα στοιχεία στους αντίστοιχους Reducers, διαβάζει τα στοιχεία ένας GroupComparator ο οποίος είναι υπεύθυνος να αναγνωρίσει πλειάδες με το ίδιο κλειδί και να τις βάλει σε μία λίστα, να δημιουργήσει ομάδες δηλαδή. Λόγω αυτού δημιουργείται ένας νέος GroupComparator ο οποίος συγκρίνει τις πλειάδες όχι βάση όλου του κλειδιού τους αλλά βάση του αριθμού του Reducer (reducerKey) προκειμένου να μπουν τα στοιχεία με τον ίδιο αριθμό Reducer (reducerKey) στην ίδια ομάδα.

Αφού υλοποιήθηκε η δευτερεύουσα ταξινόμηση (Secondary Sorting) και τα στοιχεία έρχονται ταξινομημένα στον Reducer με την επιθυμητή σειρά, οι Reducers καλούνται να υπολογίσουν τα τελικά αποτελέσματα. Ο αλγόριθμος ο οποίος εκτελεί κάθε Reducer περιγράφεται στο πεδίο Αλγόριθμος 3. Ως έξοδος της Reduce φάση είναι τα στοιχεία του συνόλου δεδομένων W που αποτελούν τα Reverse Top-k αποτελέσματα. Ο τύπος των εξαγόμενων αποτελεσμάτων είναι πάλι ένα ζευγάρι κλειδιού, τιμής (δηλ. <key,value> pair) τόσο ως τιμή όσο και ως κλειδί επιλέχτηκε ο αλφαριθμητικός τύπος Text του Hadoop API και ως κλειδί γράφεται το μοναδικό αναγνωριστικό (id) του εκάστοτε στοιχείου ενώ ως τιμή γράφεται οι τιμές των διαστάσεων του στοιχείου χωρισμένες με κόμμα (,).




*Εικόνα 23: Στην εικόνα αυτή παρουσιάζεται σχηματικά ο τρόπος λειτουργίας του Απλού αλγορίθμου καθώς και τα κλειδιά τα οποία μεταφέρονται σε κάθε φάση.*

### 5.4.3 Στοιχεία μέτρησης

Προκειμένου να μελετηθεί η συμπεριφορά του αλγορίθμου χρησιμοποιήθηκε η λειτουργικότητα των μετρητών (Counters) που παρέχει το Hadoop API (5). Πιο συγκεκριμένα δημιουργήθηκαν προσαρμοσμένοι μετρητές (Counters) έτσι ώστε να μετρούνται σημαντικά στοιχεία κατά την εκτέλεση ενός Map Reduce Job. Οι μετρητές αυτοί είναι διαθέσιμοι μετά την εκτέλεση του αλγορίθμου σε μορφή αναφοράς και αποτελούν τις βασικές πηγές μετρήσεων για την πειραματικά αξιολόγηση.

| **Μετρητές** | |
|---|---|
| **TOTAL_S** | Το πλήθος των στοιχείων p ∈ S που δόθηκαν ως είσοδο στο Map Reduce Job |
| **TOTAL_W** | Το πλήθος των στοιχείων w ∈ W που δόθηκαν ως είσοδο στο Map Reduce Job |
| **DOMINACE_S** | Το πλήθος των στοιχείων p ∈ S τα οποία επιβίωσαν από την περικοπή βάση της κυριαρχίας του q (δεν κυριαρχούνται από το q) |
| **RTOPk_W_Map** | Το πλήθος των στοιχείων w ∈ W που ανήκουν στα τοπικά αποτελέσματα των Reverse Top-k ερωτημάτων στους Mappers και θα σταλούν προς τους Reducers γιατί ενδεχομένως να ανήκουν και στα τελικά αποτελέσματα. |
| **Examine_WTopK_Map** | Το πλήθος των στοιχείων w ∈ W των οποίων εξετάστηκε το Top-k τους κατά την εκτέλεση του RTA αλγορίθμου στην Map φάση. |
| **W_output_Map** | Το πλήθος των στοιχείων w ∈ W που στέλνονται από την Map φάση προς τους Reducers. |
| **S_output_Map** | Το πλήθος των στοιχείων p ∈ S που στέλνονται από την Map φάση προς τους Reducers. Προσοχή γιατί δεν αντιπροσωπεύει την κίνηση στο δίκτυο καθώς αυτή θα πρέπει να υπολογιστεί ως $reducerNumber * S\_output\_Map$. |

| Μετρητές | |
|---|---|
| **RTOPk_W_Reduce** | Το πλήθος των στοιχείων w ∈ W που είναι αποτελέσματα του Reverse Top-k αλγορίθμου στους Reducers. |
| **Examine_WTopK_Reduce** | Το πλήθος των στοιχείων w ∈ W των οποίων εξετάστηκε το Top-k τους κατά την εκτέλεση του RTA αλγορίθμου στην Reduce φάση. |
| **S_Input_Reducer** | Το πλήθος των στοιχείων p ∈ S που στάλθηκαν από την Map φάση προς τους Reducers. Αντιπροσωπεύει την κίνηση στο δίκτυο καθώς αυτή αντιστοιχεί στο $reducerNumber * S\_output\_Map$. |
| **W_output_Reducer** | Το πλήθος των στοιχείων που ανήκουν στα τελικά Reverse Top-k αποτελέσματα. |
| **AntidominateArea** | Το πλήθος των στοιχείων p ∈ S που ανήκουν στην περιοχή κυριάρχων του q (anti-dominate area). |
| **SortingTime_Seconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Mapper για την ταξινόμηση των στοιχείων w ∈ W πριν την εκτέλεση του RTA αλγορίθμου. Η μονάδα του χρόνου που χρησιμοποιείται είναι τα δευτερόλεπτα (seconds). |
| **TopkTime_Seconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Mapper και Reducer για τον υπολογισμό Top-k ερωτημάτων για τον RTA αλγόριθμο. Η μονάδα του χρόνου που χρησιμοποιείται είναι τα δευτερόλεπτα (seconds). |
| **InputTime_Seconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Mapper για την αποθήκευση όλων των στοιχείων στην μνήμη. Η μονάδα του χρόνου που χρησιμοποιείται είναι τα δευτερόλεπτα (seconds). |

# 6. Σύνθετος Αλγόριθμος

Σε αυτό το κεφάλαιο, προτείνεται ένας νέος αλγόριθμος ο οποίος ονομάζεται Σύνθετος αλγόριθμος. Ο αλγόριθμος αυτός αξιοποιεί τις προχωρημένες ιδιότητες που περιεγράφηκαν σε προηγούμενο κεφάλαιο, προκειμένου να επεξεργάζεται αποδοτικά τα Reverse Top-k ερωτήματα με παράλληλο και κατανεμημένο τρόπο. Ο αλγόριθμος αυτός είναι σχεδιασμένος για το δημοφιλές MapReduce Framework το οποίο έχει καθιερωθεί για την παράλληλη και κατανεμημένη επεξεργασία δεδομένων τα τελευταία χρόνια.

Η επεξεργασία των υποσυνόλων των $S, W$ γίνεται σε δύο φάσεις, την Map και την Reduce φάση. Κατά την Map φάση πραγματοποιείται η περικοπή στοιχείων $S, W$ και κατά την Reduce φάση πραγματοποιείται ο υπολογισμός και η ένωση των τελικών αποτελεσμάτων. Αξίζει να σημειωθεί ότι και οι δύο φάσεις εκτελούνται παράλληλα, δηλαδή εκτελούνται παράλληλα πολλοί Mappers κατά την Map φάση και πολλοί Reducers κατά την Reduce φάση.

Τέλος γίνεται μια αναφορά στην υλοποίηση καθώς και σε τεχνικές λεπτομέρειες. Η πειραματική αξιολόγηση του αλγορίθμου αυτού θα γίνει σε επόμενο κεφάλαιο όπου θα παρουσιαστούν και θα αιτιολογηθούν τα πειράματα τα οποία εκτελέστηκαν.

## 6.1 Ομαδοποίηση weighting vectors

Προκειμένου να μειωθεί η κίνηση δεδομένων στο δίκτυο καθώς και άσκοποι υπολογισμοί για την απάντηση ενός Reverse Top-k ερωτήματος χρησιμοποιείται από τις προχωρημένες ιδιότητες η ομαδοποίηση των weighting vectors όπου αξιοποιείται για την περικοπή των στοιχείων $p \in S$. Όπως αναφέρθηκε στο κεφάλαιο Προχωρημένες Ιδιότητες η ομαδοποίηση των στοιχείων αυτών είναι εφικτή χωρίζοντας τον χώρο των δεδομένων σε τμήματα μη-αλληλεπικαλυπτόμενα μεταξύ τους (space partitioning). Η δομή αυτή απαιτείται να υπάρχει πριν την εκτέλεση του αλγορίθμου και να έχει μοιραστεί σε όλους τους κόμβους (Mappers) προκειμένου να είναι διαθέσιμη κατά την τοπική επεξεργασία. Απαιτείται δηλαδή μια μορφή προ-επεξεργασίας. Επειδή όμως η δημιουργία των ομάδων βασίζεται στον χώρο και όχι στα εκάστοτε δεδομένα, μπορεί να γίνει μία φορά και να εφαρμοστεί σε οποιοδήποτε σύνολο δεδομένων W.

Στην συνέχεια θα παρουσιαστεί ο τρόπος με τον οποίο δημιουργούνται τα τμήματα του συνόλου δεδομένων W καθώς και ο τρόπος αποθήκευσής του σε μία δομή στην κύρια μνήμη του κάθε κόμβου (Mapper).

### 6.1.1 Κατασκευή

Η κατασκευή των ομάδων αυτών είναι σχετικά απλή. Ο χρήστης δίνει το πλήθος των τμημάτων που επιθυμεί να χωριστεί η κάθε διάσταση και έτσι γνωρίζοντας τον χώρο καθώς και ότι οι τιμές της κάθε διάστασης κυμαίνονται από 0 μέχρι 1. Διασπάται η κάθε διάσταση στο πλήθος των τμημάτων και δημιουργείται μια δομή που μοιάζει πολύ με το πλέγμα για τα στοιχεία S, η μόνη διαφορά είναι ότι τα στοιχεία της δομής αυτής έχουν συγκεκριμένες ιδιότητες για τις τιμές της κάθε διάστασης.

### 6.1.2 Πλέγμα μη υπαρκτών ορίων (Not Real Bounds Grid)

Αφού έχουν δημιουργηθεί οι ομάδες με τον παραπάνω τρόπο, τώρα πρέπει να αποφασιστεί σε ποια δομή θα αποθηκευτούν, επιπλέον πρέπει να υλοποιηθεί ο αλγόριθμος που αποφασίζει σε ποια ομάδα ανήκει ένα weighting vector. Σημειώνεται επίσης σε αυτό το σημείο ότι θα πρέπει να διατρέχονται όλες οι ομάδες κατά την περικοπή των δεδομένων $p \in S$. Η δομή η οποία αποφασίστηκε να αποθηκευτούν οι ομάδες των weighting vectors είναι σε μία λίστα από ομάδες. Όπως αναφέρθηκε προηγουμένως κάθε ομάδα έχει την εξής πληροφορία, ένα μοναδικό αριθμό (id), και δύο όρια ένα κατώτατο (Lower Bound) και ένα ανώτατο (Upper Bound) όριο. Υπενθυμίζεται σε αυτό το σημείο ότι το πλήθος των ομάδων αυτών ισούται με το πλήθος των κόμβων (Reducers) κατά την δεύτερη φάση (Reduce φάση), ώστε ο κάθε κόμβος (Reducer) να επεξεργάζεται μία και μόνο μία ομάδα. Προκειμένου να αποφασιστεί κατά την πρώτη φάση (Map φάση) σε ποιον κόμβο (Reducer) της δεύτερης φάσης (Reduce φάση) θα πρέπει να σταλεί ένα στοιχείο $w \in W$ υλοποιήθηκε ο αλγόριθμος ο οποίος περιγράφεται στο πεδίο Αλγόριθμος 4.

| **Not Real Bounds Grid: Αλγόριθμος αντιστοίχισης σε ομάδα** | |
|---|---|
| **1.** | Στοιχεία εισόδου: $G_w$, W’ |
| **2.** | Στοιχεία εξόδου: W’’ |
| **3.** | SEND_TO_REDUCER( for each w ∈ W’ ) { |
| **4.** | for ( currentGroup in $G_w$ ) { |
| **5.** | belongsToGroup = true |
| **6.** | for ( currentDimension = 0… w.dimensions ) { |
| **7.** | if ( w[currentDimension] < currentGroup.LB[currentDimension] ) { |
| **8.** | belongsToGroup = false |
| **9.** | } |
| **10.** | else if (currentGroup.UB[currentDimension] < w[currentDimension] ) { |
| **11.** | belongsToGroup = false |
| **12.** | } |
| **13.** | } |
| **14.** | if (belongsToGroup = true ) { |
| **15.** | output< currentGroup.id , w > |
| **16.** | break line:4 |
| **17.** | } |
| **18.** | } |
| **19.** | } |

*Αλγόριθμος 4: Ο αλγόριθμος αυτός παρουσιάζει τον τρόπο με τον οποίο βρίσκεται η ομάδα που ανήκει ένα weighting vector και αποστέλλεται στον αντίστοιχο κόμβο (Reducer) της δεύτερης φάσης (Reduce φάση) που επεξεργάζεται την ομάδα αυτή.*

## 6.2 Αλγόριθμος περικοπής στοιχείων S

Έχοντας δημιουργήσει μια δομή όπου αποθηκεύονται τα τμήματα του χώρου W και μπορεί να απαντηθεί σε ποιο τμήμα ανήκει ένα στοιχείο w ∈ W, καθώς και ποιος κόμβος (Reducer) θα αναλάβει την επεξεργασία του αντίστοιχου τμήματος κατά την δεύτερη φάση (Reduce φάση), μπορεί να πραγματοποιηθεί περικοπή των στοιχείων που στέλνονται προς επεξεργασία στην δεύτερη φάση (Reduce φάση). Για την περικοπή των δεδομένων αξιοποιούνται οι προχωρημένες ιδιότητες όπως αναφέρθηκαν σε προηγούμενο κεφάλαιο.

Στην συνέχεια θα περιγραφούν αναλυτικά οι 3 αλγόριθμοι που προτείνονται για την περικοπή των δεδομένων αυτών. Όσο αναφορά την πειραματική αξιολόγηση και την σύγκριση μεταξύ των αλγορίθμων αυτών θα ακολουθήσει σε επόμενο κεφάλαιο.

### 6.2.1 ExtremeScore

Σε αυτό το σημείο προτείνεται ο αλγόριθμος ExtremeScore για την περικοπή των δεδομένων. Ο αλγόριθμος αυτός αξιοποιεί το γεγονός ότι κάθε κόμβος (Reducer) κατά την δεύτερη φάση (Reduce φάση) επεξεργάζεται μία και μόνο ομάδα weighting vectors. Όπως αναφέρθηκε στο κεφάλαιο Προχωρημένες Ιδιότητες κάθε ομάδα καθορίζεται από τα Lower και Upper Bounds (κατώτερο και ανώτερο όριο) όπου βάση της βαθμολογίας του q και ενός στοιχείου p ∈ S για τα παραπάνω όρια είναι δυνατόν να προβλεφθεί αν το στοιχείο p είναι πιο πάνω ή πιο κάτω στην κατάταξη από το q υπό συγκεκριμένες περιπτώσεις για όλα τα στοιχεία μιας ομάδας. Πιο αναλυτικά:

- Αν $f_{UB}(p) \leq f_{LB}(q)$ τότε το p βρίσκεται σε υψηλότερη θέση από το q στην κατάταξη για οποιοδήποτε weighting vector ανήκει στην ομάδα αυτή.
- Αν $f_{UB}(q) \leq f_{LB}(p)$ τότε το p βρίσκεται σε χαμηλότερη θέση από το q στην κατάταξη για οποιοδήποτε weighting vector ανήκει στην ομάδα αυτή.
- Διαφορετικά δεν μπορεί να προσδιοριστεί αν το p βρίσκεται σε υψηλότερη ή σε χαμηλότερη θέση στην κατάταξη σχετικά με το q για την ομάδα αυτή.

Αξιοποιώντας τις παραπάνω ιδιότητες ο αλγόριθμος αυτός μπορεί κατά την πρώτη φάση (Map φάση) να πραγματοποιεί περικοπή των δεδομένων που αποστέλλονται προς επεξεργασία κατά την δεύτερη φάση (Reduce φάση). Ενώ κατά την δεύτερη φάση (Reduce φάση) μπορεί να ελέγχεται αν για μια ομάδα weighting vectors υπάρχουν περισσότερα από k στοιχεία τα οποία είναι σε υψηλότερη θέση στην κατάταξη από το q για όλα τα μέλη της ομάδας. Σε αυτή την περίπτωση κανένα στοιχείο της ομάδας δεν θα βρίσκεται στα τελικά Reverse top-k αποτελέσματα οπότε δεν χρειάζεται να εξεταστούν τα στοιχεία της ομάδας αυτής. Αυτό μπορεί να επιτευχθεί σταματώντας τον κόμβο (Reducer) που διαχειρίζεται την ομάδα αυτή. Έτσι βάση των όσων αναφέρθηκαν παραπάνω, διαμορφώνεται ο αλγόριθμος ExtremeScore ο οποίος δεν χρησιμοποιεί την κύρια μνήμη για την αποθήκευση στοιχείων που ελέγχτηκαν προηγουμένως. Στο πεδίο Αλγόριθμος 5 περιγράφεται η λειτουργία για την περικοπή των στοιχείων που αποστέλλονται προς την δεύτερη φάση ενώ στον πεδίο Αλγόριθμος 6 περιγράφεται η λειτουργία για τον μη υπολογισμό του Reverse Top-k ερωτήματος για μια ομάδα weighting vectors κατά την δεύτερη φάση (Reduce φάση).

| ExtremeScore: Αλγόριθμος περικοπής στοιχείων | |
|---|---|
| 1. | Στοιχεία εισόδου: $G_w$, S', q |
| 2. | Στοιχεία εξόδου: S'' |
| 3. | SEND_TO_REDUCER( for each p ∈ S ) { |
| 4. | for ( currentGroup in $G_w$ ) { |
| 5. | if ($f_{\text{currentGroup}.UB}(q) \leq f_{\text{currentGroup.LB}}(p)$ ) { |
| 6. | // Do nothing |
| 7. | } |
| 8. | else { |
| 9. | output< currentGroup.id , p > |
| 10. | } |
| 11. | } |
| 12. | } |

*Αλγόριθμος 5: Παρουσιάζεται ο αλγόριθμος περικοπής στοιχείων που εφαρμόζεται από τον ExtremeScore σε κάθε κόμβο (Mapper) κατά την πρώτη φάση (Map φάση).*

| ExtremeScore: Αλγόριθμος παράλειψης υπολογισμού | |
|---|---|
| 1. | Στοιχεία εισόδου: currentGroup in $G_w$, S'', q, k |
| 2. | Στοιχεία εξόδου: true-false (in order to stop the reducer) |
| 3. | STOP_REDUCER( for each p ∈ S ) { |
| 4. | localAntidominateAreaSum = 0 |
| 5. | if ($f_{\text{currentGroup}.UB}(p) \leq f_{\text{currentGroup.LB}}(q)$ ) { |
| 6. | localAntidominateAreaSum += 1 |
| 7. | if (localAntidominateAreaSum >= k ) { |
| 8. | return true |
| 9. | } |
| 10. | } |
| 11. | return false |
| 12. | } |

*Αλγόριθμος 6: Παρουσιάζεται ο αλγόριθμος παράλειψης υπολογισμού του Reverse Top-k ερωτήματος για όλα τα στοιχεία μιας ομάδας weighting vectors που εφαρμόζεται από τον ExtremeScore σε κάθε κόμβο (Reducer) κατά την δεύτερη φάση (Reduce φάση).*

### 6.2.2 R-Lists

Αξιοποιώντας τις προχωρημένες ιδιότητες που περιεγράφηκαν σε προηγούμενο κεφάλαιο προτείνεται ο αλγόριθμος R-Lists. Ο αλγόριθμος αυτός χρησιμοποιείται για την περικοπή των στοιχείων p ∈ S, τα οποία στέλνονται προς την δεύτερη φάση (Reduce φάση). Εκμεταλλευόμενος την Σημείωση 6 ο αλγόριθμος αυτός κρατά στην κύρια μνήμη k ταξινομημένα στοιχεία για κάθε ομάδα weighting vectors και βάση αυτών των k-στοιχείων πραγματοποιεί περικοπή των δεδομένων που στέλνονται προς την δεύτερη φάση (Reduce φάση) υπό συγκεκριμένες συνθήκες. Πιο αναλυτικά:

- Αν $f_{UB}(p_k) \leq f_{LB}(p)$ τότε το στοιχείο p δεν επηρεάζει τα Reverse Top-k αποτελέσματα για την ομάδα αυτή καθώς υπάρχουν ήδη k στοιχεία σε υψηλότερη θέση στην κατάταξη από το p.
- Διαφορετικά δεν είναι δυνατόν να διαπιστωθεί αν ένα στοιχείο p επηρεάζει ή όχι τα τελικά Reverse Top-k αποτελέσματα.

Με αυτό τον τρόπο διαμορφώνεται ο αλγόριθμος αυτός ο οποίος παρουσιάζεται στο πεδίο Αλγόριθμος 7. Η μνήμη η οποία καταλαμβάνει ο αλγόριθμος αυτός είναι $k * G_w.size$ διότι κρατά k στοιχεία για κάθε ομάδα weighting vectors.

| **R-Lists: Αλγόριθμος περικοπής στοιχείων** | |
|---|---|
| **1.** | Στοιχεία εισόδου: $G_w$, S', q, k |
| **2.** | Στοιχεία εξόδου: S'' |
| **3.** | SEND_TO_REDUCER( for each p ∈ S ) { |
| **4.** | for ( currentGroup in $G_w$ ) { |
| **5.** | currentList = getList ( currentGroup ) |
| **6.** | if ( currentList.size < k ) { |
| **7.** | currentList += p (sort by currentGroup.UB) |
| **8.** | output< currentGroup.id , p > |
| **9.** | } |
| **10.** | else { |
| **11.** | if ( $f_{UB}$( currentList[k] ) ≤ $f_{LB}(p)$ ) { |
| **12.** | // Do nothing |
| **13.** | } |
| **14.** | else { |
| **15.** | if ( $f_{UB}(p) < f_{UB}$( currentList[k] ) ) { |
| **16.** | currentList[k] = empty |
| **17.** | currentList += p (sort by currentGroup.UB) |
| **18.** | } |
| **19.** | output< currentGroup.id , p > |
| **20.** | } |
| **21.** | } |
| **22.** | } |
| **23.** | } |

*Αλγόριθμος 7: Παρουσιάζεται ο αλγόριθμος περικοπής στοιχείων που εφαρμόζεται από τον R-Lists σε κάθε κόμβο (Mapper) κατά την πρώτη φάση (Map φάση).*

### 6.2.3 ExtremeScore & R-Lists

Προηγουμένως παρουσιάστηκαν οι αλγόριθμοι ExtremeScore και R-Lists. Ο πρώτος αλγόριθμος εκμεταλλεύεται το γεγονός ότι μια ομάδα weighting vectors έχει κάποια κοινά χαρακτηριστικά σχετικά με την βαθμολόγηση των στοιχείων ενώ ο δεύτερος αλγόριθμος αξιοποιεί το γεγονός της ύπαρξης k στοιχείων τα οποία είναι πάντα πιο ψηλά στην κατάταξη από κάποια άλλα στοιχεία για όλα τα weighting vectors μιας ομάδας. Και οι δύο αλγόριθμοι μειώνουν το πλήθος των στοιχείων που στέλνονται προς την δεύτερη φάση (Reduce φάση), έτσι προκύπτει η ιδέα για τον συνδυασμό των αλγορίθμων αυτών σε έναν αλγόριθμο προκειμένου να γίνεται καλύτερη περικοπή των στοιχείων. Σε αυτό το σημείο αξίζει να σημειωθεί ότι ο πρώτος αλγόριθμος αποτελεί μια φθηνή λύση καθώς δεν χρησιμοποιεί μνήμη και κάνει ελάχιστους υπολογισμούς ενώ ο δεύτερος αλγόριθμος είναι σχετικά με τον πρώτο πιο ακριβός καθώς δεσμεύει κύρια μνήμη και κάνει πιο πολύπλοκους υπολογισμούς. Λόγω των ιδιοτήτων αυτών καθώς και της ιδιότητας ότι η ποιότητα της περικοπής των στοιχείων του R-Lists αλγόριθμου εξαρτάται από την σειρά των στοιχείων με την οποία τα επεξεργάζεται, θα εκτελείται πρώτα ο αλγόριθμος ExtremeScore και έπειτα όσα στοιχεία επιβιώνουν από αυτόν θα επεξεργάζονται από τον αλγόριθμο R-Lists. Έτσι προκύπτει ο τελικός αλγόριθμος του οποίου η λειτουργικότητα κατά την πρώτη φάση (Map φάση) περιγράφεται στο πεδίο Αλγόριθμος 9 ενώ η λειτουργικότητα κατά την δεύτερη φάση (Reduce φάση) περιγράφεται στο πεδίο Αλγόριθμος 8. Τα θετικά του αλγορίθμου αυτού είναι ότι πραγματοποιεί καλύτερη περικοπή δεδομένων αλλά αυτό συνοδεύεται από ένα αρνητικό στοιχείο καθώς είναι ακριβότερος σε πόρους αφού συνδυάζει δύο αλγορίθμους.

| **ExtremeScore & RLists: Αλγόριθμος παράλειψης υπολογισμού** | |
|---|---|
| **1.** | Στοιχεία εισόδου: currentGroup in $G_w$, S'', q, k |
| **2.** | Στοιχεία εξόδου: true-false (in order to stop the reducer) |
| **3.** | STOP_REDUCER( for each p ∈ S ) { |
| **4.** | localAntidominateAreaSum = 0 |
| **5.** | if ($f_{\text{currentGroup}.UB}(p) \leq f_{\text{currentGroup.LB}}(q)$ ) { |
| **6.** | localAntidominateAreaSum += 1 |
| **7.** | if (localAntidominateAreaSum >= k ) { |
| **8.** | return true |
| **9.** | } |
| **10.** | } |
| **11.** | return false |
| **12.** | } |

*Αλγόριθμος 8: Παρουσιάζεται ο αλγόριθμος παράλειψης υπολογισμού του Reverse Top-k ερωτήματος για όλα τα στοιχεία μιας ομάδας weighting vectors που εφαρμόζεται από τον ExtremeScore & R-Lists σε κάθε κόμβο (Reducer) κατά την δεύτερη φάση (Reduce φάση).*

| ExtremeScore & RLists: Αλγόριθμος περικοπής στοιχείων | |
|---|---|
| 1. | Στοιχεία εισόδου: $G_w$, S', q, k |
| 2. | Στοιχεία εξόδου: S'' |
| 3. | SEND_TO_REDUCER( for each p ∈ S ) { |
| 4. | for ( currentGroup in $G_w$ ) { |
| 5. | if ($f_{\text{currentGroup}.UB}(q) \leq f_{\text{currentGroup.LB}}(p)$ ) { |
| 6. | continue line:4 |
| 7. | } |
| 8. | currentList = getList ( currentGroup ) |
| 9. | if ( currentList.size < k ) { |
| 10. | currentList += p (sort by currentGroup.UB) |
| 11. | output< currentGroup.id , p > |
| 12. | } |
| 13. | else { |
| 14. | if ( $f_{UB}$( currentList[k] ) ≤ $f_{LB}$(p) ) { |
| 15. | // Do nothing |
| 16. | } |
| 17. | else { |
| 18. | if ( $f_{UB}$(p) < $f_{UB}$( currentList[k] ) ) { |
| 19. | currentList[k] = empty |
| 20. | currentList += p (sort by currentGroup.UB) |
| 21. | } |
| 22. | output< currentGroup.id , p > |
| 23. | } |
| 24. | } |
| 25. | } |
| 26. | } |

*Αλγόριθμος 9: Παρουσιάζεται ο αλγόριθμος περικοπής στοιχείων που εφαρμόζεται από τον ExtremeScore & R-Lists σε κάθε κόμβο (Mapper) κατά την πρώτη φάση (Map φάση).*

## 6.3 Πλέγμα S

Προκειμένου να υπολογιστούν κατά την πρώτη φάση (Map φάση) τα στοιχεία $w \in W'i$ τα οποία ανήκουν στα τελικά Reverse Top-k αποτελέσματα ή όχι προτάθηκε η δημιουργία μιας μορφής πλέγματος $G_s$ η οποία αποτελεί μια περίληψη του συνόλου δεδομένων S. Όπως σημειώθηκε στο κεφάλαιο Προχωρημένες Ιδιότητες μπορεί να υπολογιστεί προσεγγιστικά η θέση του q στην κατάταξη ενός weighting vector. Έτσι μπορούν να απορριφθούν weighting vectors τα οποία είναι γνωστό από την πρώτη φάση ότι δεν ανήκουν στα τελικά Reverse Top-k αποτελέσματα, και όχι μόνο. Είναι επίσης δυνατό να μην δοθεί προς επεξεργασία ένα weighting vector κατά την δεύτερη φάση (Reduce φάση) προκειμένου να γίνει έλεγχος αν ανήκει στα τελικά αποτελέσματα, στην περίπτωση που από την πρώτη φάση είναι σίγουρο ότι ανήκει σε αυτά. Η δομή αυτή απαιτείται να υπάρχει πριν την εκτέλεση του αλγορίθμου και να έχει μοιραστεί σε όλους τους κόμβους (Mappers) προκειμένου να είναι διαθέσιμη κατά την τοπική επεξεργασία. Απαιτείται δηλαδή μια μορφή προ-επεξεργασίας.

Στην συνέχεια θα παρουσιαστεί ο τρόπος με τον οποίο δημιουργείται η δομή αυτή κατά την φάση της προ-επεξεργασίας. Έπειτα θα ακολουθήσει μια ανάλυση των αλγορίθμων που υλοποιήθηκαν, προκειμένου να βελτιστοποιηθεί η απόδοση της λειτουργίας της περικοπής των δεδομένων. Τέλος όσο αναφορά την πειραματική αξιολόγηση των αλγορίθμων που υλοποιήθηκαν θα πραγματοποιηθεί σε επόμενο κεφάλαιο.

### 6.3.1 Κατασκευή

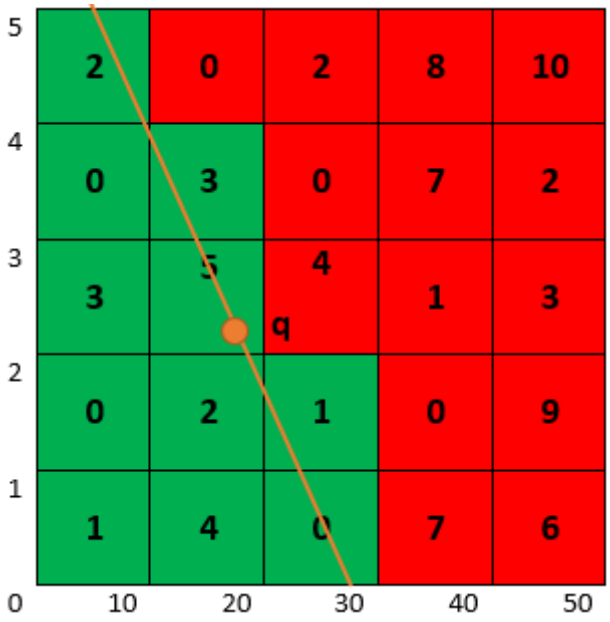
Η κατασκευή του πλέγματος $G_s$ γίνεται κατά την φάση της προ-επεξεργασίας των δεδομένων. Στο Hadoop η διαδικασία αυτή θα μπορούσε να γίνει κατά την μεταφόρτωση του συνόλου δεδομένων S στο HDFS. Όπου κατά την μεταφόρτωση θα γινόταν ανάγνωση κάθε πλειάδας του συνόλου και παράλληλα με την διαδικασία αυτή μπορεί να δημιουργηθεί η δομή $G_s$. Με αυτό τον τρόπο δημιουργείται το $G_s$ με ελάχιστη επιβάρυνση και αποθηκεύεται σε ένα ξεχωριστό αρχείο στο HDFS. Επίσης μπορεί να δημιουργηθεί ένα MapReduce Job το οποίο θα δέχεται ως είσοδο ένα σύνολο δεδομένων S που υπάρχει ήδη στο HDFS και θα παράγει την δομή $G_s$. Σε αυτή την περίπτωση αυτό το MapReduce Job θα πρέπει να εκτελεστεί μια φορά πριν τον αλγόριθμο που προτείνεται.

Όσο αναφορά την διαδικασία της δημιουργίας της δομής $G_s$, αρχικά πρέπει να επιλεγούν τα όρια των τμημάτων της κάθε διάστασης, τα οποία ουσιαστικά καθορίζουν ένα κελί. Κάθε κελί διατηρεί πληροφορία σχετικά με το πλήθος των στοιχείων $p \in S$ που περιέχονται σε αυτό, κρατά δηλαδή έναν μετρητή (counter). Έπειτα το $G_s$ μπορεί να δημιουργηθεί πραγματοποιώντας μια φορά ανάγνωση του συνόλου δεδομένων S και αυξάνοντας τον counter του κελιού το οποίο περιέχει το τρέχον στοιχείο $p \in S$ κατά 1. Έπειτα η δομή αυτή μπορεί να αποθηκευτεί στο HDFS με τέτοιο τρόπο ώστε να περιγράφονται τα κελιά καθώς και οι counters αυτών. Σε αυτό το σημείο αξίζει να σημειωθεί ότι όσο πιο σφιχτά είναι τα όρια τόσο πιο αναλυτικό θα είναι το $G_s$, με συνέπεια να καταλαμβάνει περισσότερο χώρο ενώ όσο πιο χαλαρά είναι τα όρια τόσο πιο περιγραφικό θα είναι το $G_s$ αλλά θα καταλαμβάνει λιγότερο χώρο. Σχετικά με τον αλγόριθμο ένα αναλυτικό $G_s$ θα συμβάλει σε πιο ακριβείς εκτιμήσεις σε περισσότερο χρόνο ενώ με ένα περιγραφικό $G_s$ θα παράγονται λιγότερο ακριβείς εκτιμήσεις γρηγορότερα.

Ο τρόπος επιλογής των ορίων της κάθε διάστασης μπορεί να επηρεάσει την αποδοτικότητα του αλγορίθμου αυτού. Για τον λόγο αυτό προτείνεται η δημιουργία των ορίων αυτών να γίνεται με ανομοιόμορφο τρόπο. Διαισθητικά, χρειάζεται μεγαλύτερη ακρίβεια στις μικρότερες τιμές κάθε διάστασης (δηλαδή περισσότερα κελιά), καθώς τα Top-k δεδομένα αναμένεται να βρίσκονται σε αυτή την περιοχή. Έτσι εφαρμόζεται λογαριθμικό βήμα για τον καθορισμό των ορίων αυτών, προκειμένου να υπάρχουν περισσότερα κελιά κοντά στην αρχή των αξόνων. Οπότε ο χρήστης επιλέγει σε πόσα τμήματα θέλει να χωριστεί η κάθε διάσταση και τα κελιά δημιουργούνται με τον τρόπο που προαναφέρθηκε. Σε αυτό το σημείο επισημαίνεται η εξής σχέση $cells = p^d$ όπου: cells: το πλήθος των κελιών, p: το πλήθος των τμημάτων κάθε διάστασης, d: το πλήθος των διαστάσεων.

### 6.3.2 Simple

Αρχικά υλοποιήθηκε ένας απλός τύπος πλέγματος που ονομάζεται Simple. Ο τύπος αυτός σχετίζεται τόσο με την αποθήκευση του $G_s$ όσο και με τον αλγόριθμο ο οποίος υπολογίζει την προσεγγιστική κατάταξη του q σχετικά με ένα weighting vector βάση του $G_s$. Η αποθήκευση του $G_s$ στην κύρια μνήμη του κάθε κόμβου (Mapper) υλοποιείται χρησιμοποιώντας μια λίστα η οποία περιέχει όλα τα κελιά του πλέγματος αυτού. Κάθε κελί περιέχει την εξής πληροφορία: $c_l$, $c_u$, count. Ο υπολογισμός της προσεγγιστικής κατάταξης του q βάση ενός weighting vector πραγματοποιείται ελέγχοντας όλα τα κελιά του πλέγματος, ο Αλγόριθμος 10 περιγράφει τον υπολογισμό αυτό. Η επιλογή αυτή έχει το πλεονέκτημα ότι είναι ανεξάρτητη της δημιουργίας του πλέγματος, για παράδειγμα αν αποφασιζόταν για κάποιο λόγο να δημιουργείται το πλέγμα με τέτοιο τρόπο ώστε κάθε κελί να περιέχει ακριβώς N στοιχεία, τότε η υλοποίηση αυτή θα λειτουργούσε ορθά. Από την άλλη ο αλγόριθμος αυτός ελέγχει κελιά τα οποία δεν επηρεάζουν το αποτέλεσμα, κάτι που έχει κόστος τόσο χρονικό όσο και μνήμης. Για παράδειγμα ελέγχονται κελιά τα οποία βρίσκονται στην dominate-area του q τα οποία ποτέ δεν θα επηρεάσουν τα αποτελέσματα, έτσι χάνεται χρόνος για τον έλεγχό τους και μνήμη καθώς δεν χρειάζονται. Στην Εικόνα 24 παρουσιάζεται ένα παράδειγμα στο οποίο τα κελιά που επηρεάσουν το αποτέλεσμα είναι 11 ενώ ελέγχονται 25 κελιά.




*Εικόνα 24: Στην εικόνα αυτή παρουσιάζονται με πράσινο χρώμα τα κελιά του πλέγματος τα οποία επηρεάζουν το αποτέλεσμα της προσεγγιστικής κατάταξης του q για ένα weighting vector, ενώ με κόκκινο χρώμα αυτά που δεν επηρεάζουν το αποτέλεσμα.*

| **Simple $G_s$: Αλγόριθμος υπολογισμού προσεγγιστικής κατάταξης q** | |
|---|---|
| **1.** | Στοιχεία εισόδου: $G_s$ as cellsList, q |
| **2.** | Στοιχεία εξόδου: min, max |
| **3.** | GET_COUNT() { |
| **4.** | min = 0 |
| **5.** | max = 0 |
| **6.** | queryScore = $f_w(q)$ |
| **7.** | for ( cell in cellsList ) { |
| **8.** | if ( $f_w(cell.c_u) <$ queryScore ) { |
| **9.** | min += cell.count |
| **10.** | } |
| **11.** | else if ( $f_w(cell.c_l) <$ queryScore ) { |
| **12.** | max += cell.count |
| **13.** | } |
| **14.** | } |
| **15.** | max += min |
| **16.** | return [max,min] |
| **17.** | } |

*Αλγόριθμος 10: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο υπολογίζεται η προσεγγιστική κατάταξη του q για ένα weighting vector βάση του πλέγματος $G_s$ με τύπο Simple.*

### 6.3.3 Summarized

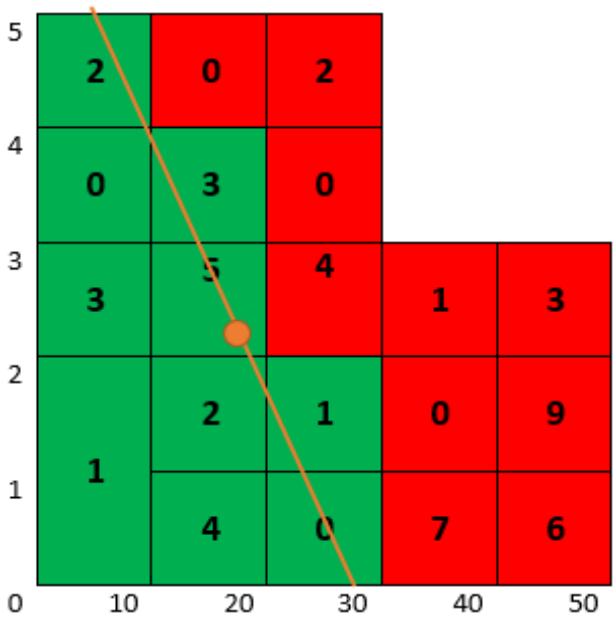
Προκειμένου να διατηρηθούν τα πλεονεκτήματα του Simple τύπου και να υπάρχει καλύτερη απόδοση δημιουργήθηκε ένας νέος τύπος για την αποθήκευση του πλέγματος και τον υπολογισμό της προσεγγιστικής κατάταξης του q, ο Summarized. Ο τύπος αυτός βασίζεται στον Simple τύπο μόνο που διατηρεί μια περίληψη του ευρετηρίου και όχι όλο, βασικά διαγράφει και συμπιέζει όποια πληροφορία δεν επηρεάζει το αποτέλεσμα για το εκάστοτε q. Πιο αναλυτικά, εκμεταλλεύεται την ιδιότητα της κυριαρχίας και όπως αναφέρθηκε στο κεφάλαιο Προχωρημένες Ιδιότητες, συγχωνεύει τα κελιά που βρίσκονται στην περιοχή κυριάρχων του q (anti-dominate area) σε ένα που περιέχει το άθροισμα των μετρητών των συγχωνευμένων κελιών και διαγράφει όλα τα κελιά που βρίσκονται στην περιοχή κυριαρχίας του q. Με αυτό τον τρόπο γίνεται καλύτερη αξιοποίηση της κύριας μνήμης. Επιπλέον ελέγχονται λιγότερα κελιά προκειμένου να υπολογιστεί η προσεγγιστική κατάταξη του q, οπότε υπάρχει σημαντικό όφελος στην απόδοση του αλγορίθμου (περισσότερες λεπτομέρειες στο κεφάλαιο Πειραματική Αξιολόγηση). Ο τύπος αποθήκευσης του πλέγματος είναι όπως και στον Simple τύπο, μία λίστα και για τον υπολογισμό της προσεγγιστικής κατάταξης του q ελέγχονται όλα τα πεδία στην λίστα αυτή. Στον Αλγόριθμος 11 περιγράφεται ο τρόπος με τον οποίο περικόπτονται και συμπιέζονται τα κελιά του πλέγματος, ενώ στην Εικόνα 25 απεικονίζεται το πλέγμα της Εικόνα 24 ως Summarized. Ακόμα στον Αλγόριθμος 12 περιγράφεται ο τρόπος με τον οποίο υπολογίζεται η προσεγγιστική κατάταξη του q. Τέλος στην Εικόνα 25 παρατηρείται ότι ελέγχονται 19 κελιά αντί για 25 κελιά όπου ελέγχονταν στον Simple τύπο, ενώ τα κελιά που επηρεάζουν το αποτέλεσμα είναι 9 αντί για 11.

| **Summarized $G_s$: Αλγόριθμος δημιουργίας του πλέγματος** | |
|---|---|
| **1.** | Στοιχεία εισόδου: $G_s$ as cellsList, q |
| **2.** | Στοιχεία εξόδου: Summarized $G_s$ as cellsList, antiDominateAreaCount |
| **3.** | CREATE_GRID() { |
| **4.** | antiDominateAreaCount = 0 |
| **5.** | for ( cell in cellsList ) { |
| **6.** | if ( $q \prec cell.c_l$ ) { |
| **7.** | cellsList.remove( cell ) |
| **8.** | } |
| **9.** | else if ( $cell.c_u \prec q$ ) { |
| **10.** | antiDominateAreaCount += cell.count |
| **11.** | cellsList.remove( cell ) |
| **12.** | } |
| **13.** | } |
| **14.** | return [cellsList, antiDominateAreaCount] |
| **15.** | } |

*Αλγόριθμος 11: Παρουσιάζεται ο αλγόριθμος με τον οποίο δημιουργείται ο Summarized τύπος πλέγματος.*

| Summarized $G_s$: Αλγόριθμος υπολογισμού προσεγγιστικής κατάταξης q | |
|---|---|
| 1. | Στοιχεία εισόδου: Summarized $G_s$ as cellsList, antiDominateAreaCount, q |
| 2. | Στοιχεία εξόδου: min, max |
| 3. | GET_COUNT() { |
| 4. | min = antiDominateAreaCount |
| 5. | max = 0 |
| 6. | queryScore = $f_w(q)$ |
| 7. | for ( cell in cellsList ) { |
| 8. | if ( $f_w(cell.c_u) < queryScore$ ) { |
| 9. | min += cell.count |
| 10. | } |
| 11. | else if ( $f_w(cell.c_l) < queryScore$ ) { |
| 12. | max += cell.count |
| 13. | } |
| 14. | } |
| 15. | max += min |
| 16. | return [max,min] |
| 17. | } |

*Αλγόριθμος 12: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο υπολογίζεται η προσεγγιστική κατάταξη του q για ένα weighting vector βάση του πλέγματος $G_s$ με τύπο Summarized.*




*Εικόνα 25: Στην εικόνα αυτή παρουσιάζονται με πράσινο χρώμα τα κελιά του πλέγματος τα οποία επηρεάζουν το αποτέλεσμα της προσεγγιστικής κατάταξης του q για ένα weighting vector, ενώ με κόκκινο χρώμα αυτά που δεν επηρεάζουν το αποτέλεσμα. Σημειώνεται ότι το κελί που βρίσκεται στην περιοχή κυριάρχων του q είναι εικονικό, κρατείται ένας μετρητής για αυτή την πληροφορία.*

### 6.3.4 Tree

Προηγουμένως παρουσιάστηκαν οι δύο τύποι πλέγματος Simple και Summarized, οι αλγόριθμοι για την προσεγγιστική κατάταξη του q που συνοδεύουν τους τύπους αυτούς δεν έχουν ευφυΐα, καθώς ελέγχουν όλα τα κελιά που έχουν διαθέσιμα. Από την πειραματική αξιολόγηση είναι εμφανές ότι επιβαρύνεται ο χρόνος απόκρισης του αλγορίθμου. Οπότε προκύπτει η ανάγκη για την δημιουργία μιας πιο «έξυπνης» δομής η οποία θα περιορίζει όσο λιγότερο γίνεται το πλήθος των κελιών που ελέγχονται προκειμένου να υπολογιστεί η προσεγγιστική κατάταξη του q. Αξίζει να επισημανθεί σε αυτό το σημείο ότι η προσεγγιστική κατάταξη του q θα υπολογιστεί για κάθε w ∈ W, έτσι μια μικρή καθυστέρηση σε κάθε υπολογισμό μπορεί να συμβάλει στην δημιουργία μιας μεγάλης καθυστέρησης για όλο τον αλγόριθμο.

Διατρέχοντας τα κελιά βάση την απόστασή τους από την αρχή των αξόνων είναι εφικτό να μπορεί να προβλεφθεί αν το επόμενο κελί επηρεάζει ή όχι την προσεγγιστική κατάταξη του q. Πιο συγκεκριμένα αξιοποιώντας την Σημείωση 8 αν διατρεχτούν τα δεδομένα με τέτοιο τρόπο ώστε το lower-bound του προηγούμενου κελιού να κυριαρχεί αυτό του επόμενου, τότε μπορεί να μην ελεγχθούν τα κελιά τα οποία ακολουθούν έπειτα από το κελί που ισχύει ότι $f_w(c_l) \leq f_w(q)$. Ένας τρόπος προκειμένου να είναι σίγουρο ότι το lower-bound ενός κελιού κυριαρχεί του επόμενου είναι το τρέχον κελί να έχει σε τουλάχιστον μία διάσταση μικρότερη τιμή και σε καμία άλλη μεγαλύτερη από τα κελιά που ακολουθούν. Αξιοποιώντας αυτό δημιουργήθηκε μια δομή όπου επιτρέπει να διατρεχτούν τα κελιά έτσι ώστε το τρέχον στοιχείο να κυριαρχεί το επόμενο για κάθε τιμή των N-1 διαστάσεων. Ο Αλγόριθμος 13 περιγράφει τον τρόπο με τον οποίο δημιουργείται αυτή η δενδροειδής μορφή. Έπειτα ο Αλγόριθμος 14 περιγράφει τον τρόπο με τον οποίο υπολογίζεται η προσεγγιστική κατάταξη του q. Όπως φαίνεται στην Εικόνα 26 ελέγχονται 15 στοιχεία ενώ 11 στοιχεία επηρεάζουν το αποτέλεσμα. Συγκριτικά με τον Simple τύπο πλέγματος ελέγχονται 10 κελιά λιγότερα ενώ με τον τύπο Summarized ελέγχονται 4 κελιά λιγότερα για το ίδιο παράδειγμα. Σε αυτό το σημείο αξίζει να σημειωθεί ότι το δέντρο έχει ύψος ίσο με το πλήθος των διαστάσεων και κάθε κόμβος έχει τόσα παιδιά όσο και το πλήθος των τμημάτων της κάθε διάστασης.

*Σημείωση 8:*

Δοσμένου ενός πλέγματος $G_s$ όπου τα κελιά δημιουργούνται χωρίζοντας μια διάσταση σε τμήματα ανεξαρτήτως βήματος και θεωρώντας ένα κελί c. Αν ισχύει $f_w(q) \leq f_w(c_l)$ *τότε το c δεν επηρεάζει την προσεγγιστική κατάταξη του q* και κάθε c' όπου $c \prec c'_l$ δεν επηρεάζει την κατάταξη του q οπότε δεν χρειάζεται να ελεγχθεί.

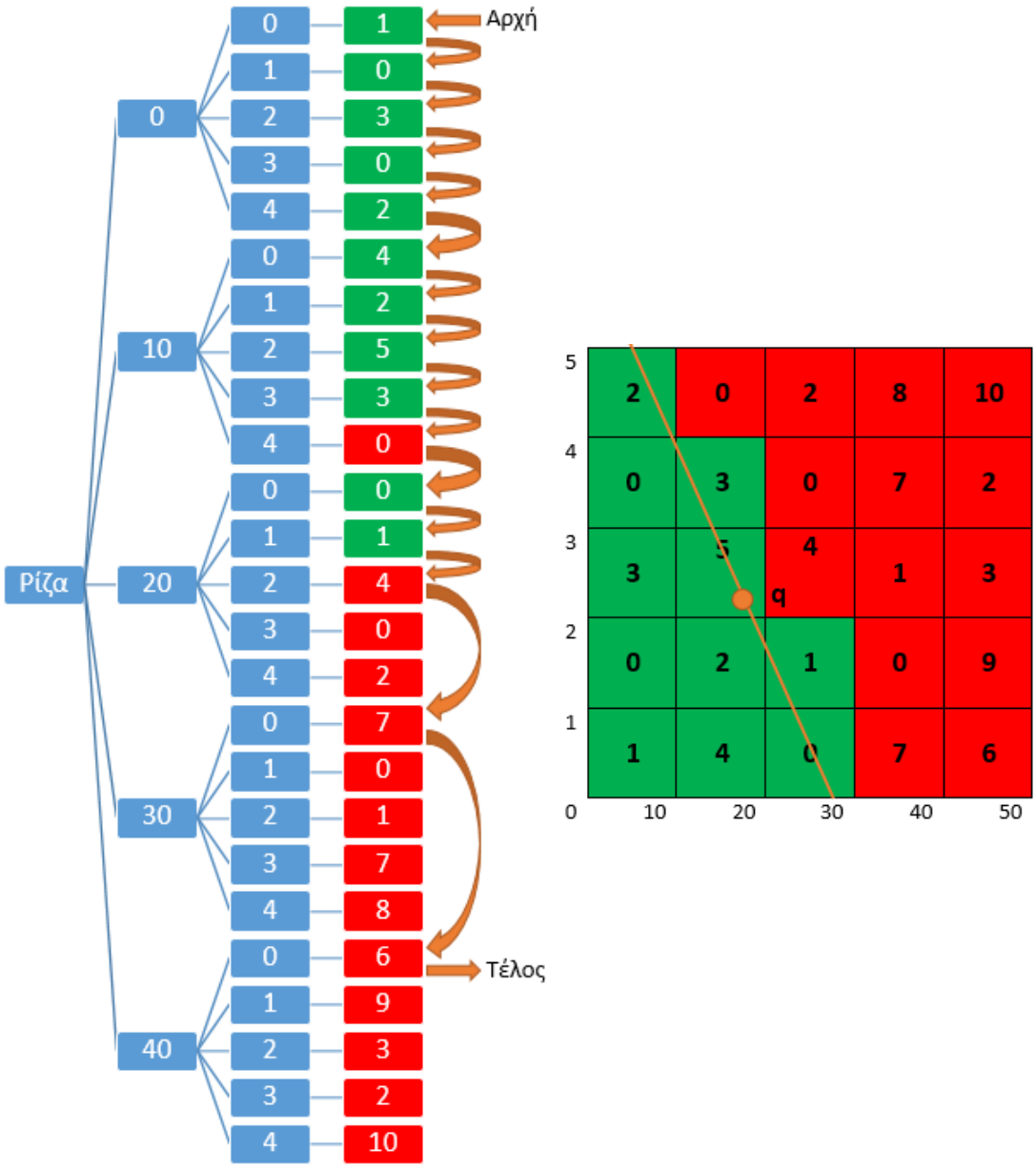
*Απόδειξη: Αρχικά ισχύει* $\forall\, p \in c$ *ότι* $c_l \prec p$, *αυτό σημαίνει ότι* $f_w(c_l) < f_w(p)$. *Αν* $f_w(q) \leq f_w(c_l)$ *τότε* $f_w(q) < f_w(p)$, *άρα όλα τα στοιχεία που ανήκουν στο κελί αυτό βρίσκονται μετά το q στην κατάταξη οπότε δεν επηρεάζουν την κατάταξη του q. Ακόμα για οποιοδήποτε c' όπου* $c \prec c'_l$ *ισχύει ότι* $f_w(c_l) < f_w(c'_l)$ *οπότε* $f_w(q) < f_w(p')\ \forall\, p' \in c'$, *άρα δεν επηρεάζει την προσεγγιστική κατάταξη του q.*

| **Tree $G_s$: Αλγόριθμος δημιουργίας του πλέγματος** | |
|---|---|
| **1.** | Στοιχεία εισόδου: $G_s$ as cellsList |
| **2.** | Στοιχεία εξόδου: Tree $G_s$ as cellsTree |
| **3.** | CREATE_GRID() { |
| **4.** | for ( cell in cellsList ) { |
| **5.** | currentNode = Tree.root |
| **6.** | for ( i = 0…cell.LowerBoundDim.lenght ) { |
| **7.** | for ( child in currentNode.children) { |
| **8.** | if ( child.value = cell.getDimensionValue(i) ) { |
| **9.** | currentNode = child |
| **10.** | continue line:6 |
| **11.** | } |
| **12.** | } |
| **13.** | nextNode.parent = current |
| **14.** | nextNode.value = cell.getDimensionValue(i) |
| **15.** | currentNode.children += nextNode |
| **16.** | currentNode.children.sort |
| **17.** | if ( i + 1 = cell.LowerBoundDim.lenght ) { |
| **18.** | nextNode.leaf = true |
| **19.** | nextNode.cell = cell |
| **20.** | } |
| **21.** | currentNode = nextNode |
| **22.** | } |
| **23.** | } |
| **24.** | return [Tree] |
| **25.** | } |

*Αλγόριθμος 13: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο δημιουργείται η δενδροειδής μορφή του πλέγματος $G_s$.*

| Tree $G_s$: Αλγόριθμος υπολογισμού προσεγγιστικής κατάταξης q | |
|---|---|
| 1. | Στοιχεία εισόδου: $G_s$ as cellsTree, q |
| 2. | Στοιχεία εξόδου: min, max |
| 3. | GET_COUNT() { |
| 4. | min = 0 |
| 5. | max = 0 |
| 6. | queryScore = $f_w(q)$ |
| 7. | currentNode = cellsTree.root |
| 8. | positions[] = table[dimensionsNumber] |
| 9. | currentDepth = 0 |
| 10. | while ( current not Null ) { |
| 11. | for ( i = positions[currentDepth]…currentNode.children ) { |
| 12. | positions[currentDepth] = i |
| 13. | if ( currentNode.children[i].leaf = true ) { |
| 14. | currentChild = currentNode.children[i] |
| 15. | if ( $f_w(currentChild.LowerBound) <$ queryScore ) { |
| 16. | min += currentChild .cell.count |
| 17. | } |
| 18. | else if ( $f_w(currentChild.UpperBound) <$ queryScore ) { |
| 19. | max += currentChild .cell.count |
| 20. | } |
| 21. | else { |
| 22. | currentNode = currentNode.parent |
| 23. | position[currentDepth] = 0 |
| 24. | currentDepth -= 1 |
| 25. | if ( currentDepth >= 0 ) { |
| 26. | positions[currentDepth] += 1 |
| 27. | } |
| 28. | continue line:10 |
| 29. | } |
| 30. | } |
| 31. | else { |
| 32. | currentNode = currentNode.children[i] |
| 33. | currentDepth += 1 |
| 34. | continue line:10 |
| 35. | } |
| 36. | } |
| 37. | currentNode = currentNode.parent |
| 38. | position[currentDepth] = 0 |
| 39. | currentDepth -= 1 |
| 40. | if ( currentDepth >= 0 ) { |
| 41. | positions[currentDepth] += 1 |
| 42. | } |
| 43. | } |
| 44. | max += min |
| 45. | return [min,max] |
| 46. | } |

*Αλγόριθμος 14: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο υπολογίζεται η προσεγγιστική κατάταξη του q για ένα weighting vector βάση του πλέγματος $G_s$ με τύπο Tree.*




*Εικόνα 26: Στην εικόνα αυτή παρουσιάζονται στα δεξιά το πλέγμα σε μορφή πίνακα, ενώ αριστερά παρουσιάζεται το πλέγμα σε δενδροειδής μορφή. Με πράσινο χρώμα απεικονίζονται τα κελιά τα οποία επηρεάζουν την προσεγγιστική κατάταξη του q, ενώ με κόκκινο χρώμα σημειώνονται τα κελιά τα οποία δεν επηρεάζουν την κατάταξη αυτή. Επιπλέον τα βέλη περιγράφουν τον τρόπο με τον οποίο διατρέχονται τα κελιά, βάση των αλγορίθμων που περιεγράφηκαν.*

### 6.3.5 Summarized Tree

Προηγουμένως παρουσιάστηκαν οι τύποι πλέγματος Summarized και Tree, όπου ο πρώτος τύπος αφαιρεί και συμπιέζει πληροφορία ενώ ο δεύτερος παρέχει μια «εξυπνότερη» μορφή προκειμένου να ελέγχονται λιγότερα κελιά κατά τον υπολογισμό της προσεγγιστικής κατάταξης του q. Είναι εμφανές ότι είναι δυνατό να συνδυαστούν οι δύο τύποι πλέγματος που προαναφέρθηκαν προκειμένου να προκύψει ένας νέος ταχύτερος τύπος. Ο νέος τύπος πλέγματος ονομάζεται Summarized Tree και αποτελεί μια περίληψη του πλέγματος σε δενδροειδής μορφή.

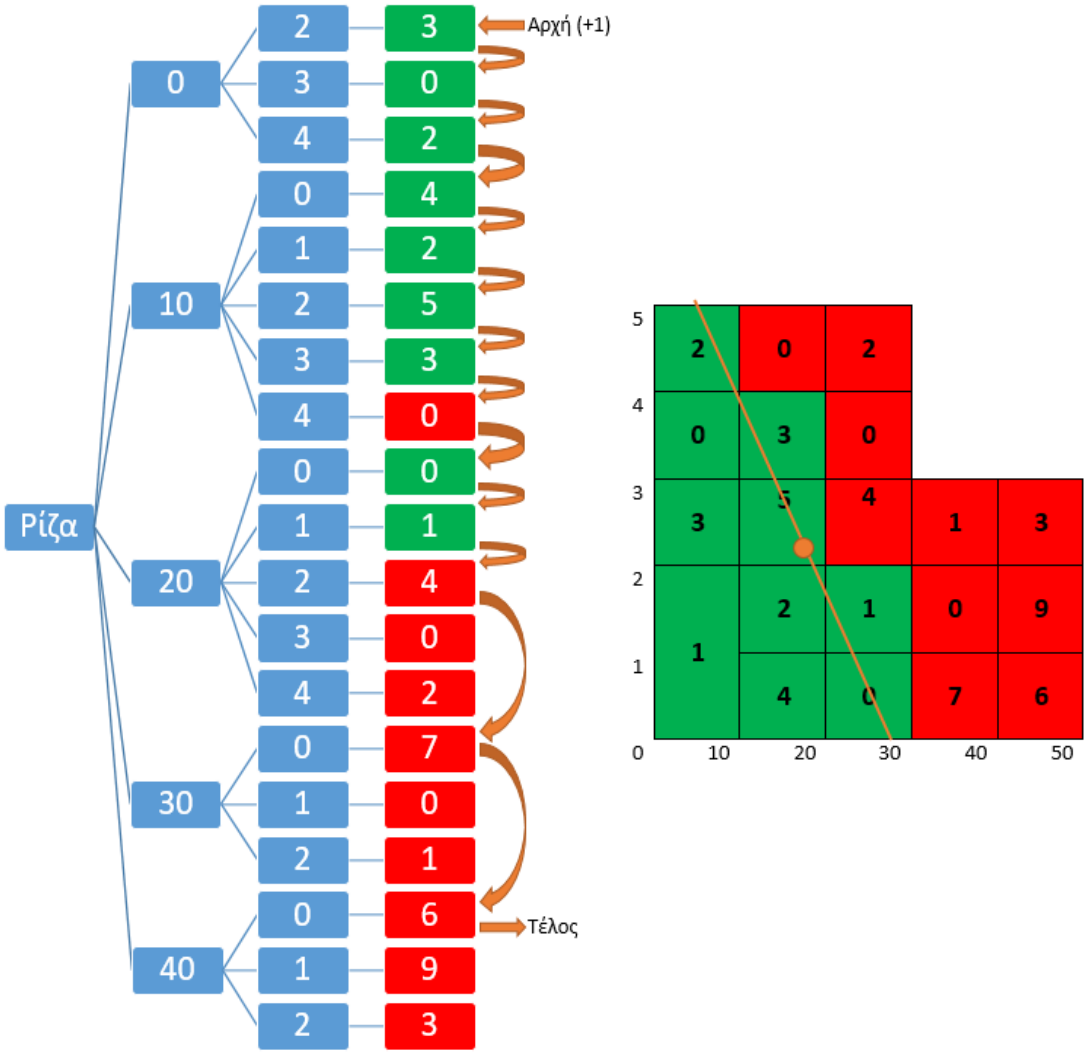
Ο τρόπος με τον οποίο αποθηκεύεται η δομή αυτή καθώς και ο τρόπος με τον οποίο διατρέχονται τα κελιά είναι παρόμοιος με τον τύπο Tree. Η μόνη διαφορά είναι ότι δεν αποθηκεύονται στην δενδροειδή αυτή μορφή τα κελιά που κυριαρχούνται από το q ούτε αυτά που κυριαρχούν το q, μόνο που για τα δεύτερα κρατείται ένας μετρητής με το άθροισμα του πλήθους των στοιχείων που περιέχονται σε αυτά τα κελιά. Δηλαδή ο τρόπος με τον οποίο συμπιέζεται και διαγράφεται άχρηστη πληροφορία είναι παρόμοιος με αυτόν του τύπου Summarized. Ο Αλγόριθμος 15 περιγράφει τον τρόπο που δημιουργείται η δομή αυτή, ενώ ο Αλγόριθμος 16 αποτυπώνει την διαδικασία με την οποία διατρέχονται τα κελιά. Στην Εικόνα 27 παρουσιάζεται η δομή αυτή με ένα παράδειγμα στο οποίο ελέγχονται 13 κελιά ενώ 11 κελιά επηρεάζουν την σχετική κατάταξη του q. Συγκριτικά με τους προηγούμενους τύπους ελέγχονται 12 κελιά λιγότερα σε σχέση με τον τύπο Simple, 8 κελιά λιγότερα σε σχέση με τον τύπο Summarized και 2 κελιά λιγότερα σε σχέση με τον τύπο Tree. Οπότε συμπεραίνεται ότι ο τύπος αυτός είναι ο πιο αποδοτικός από όσους προτάθηκαν καθώς πραγματοποιεί τους λιγότερους ελέγχους και διαχειρίζεται καλά την μνήμη διαγράφοντας άχρηστη πληροφορία.

| Summarized Tree $G_s$: Αλγόριθμος δημιουργίας του πλέγματος | |
|---|---|
| 1. | Στοιχεία εισόδου: $G_s$ as cellsList, q |
| 2. | Στοιχεία εξόδου: Tree $G_s$ as cellsTree, antiDominateAreaCount |
| 3. | CREATE_GRID() { |
| 4. | antiDominateAreaCount = 0 |
| 5. | for ( cell in cellsList ) { |
| 6. | if ( $q \prec cell.c_l$ ) { |
| 7. | continue line:5 |
| 8. | } |
| 9. | else if ( $cell.c_u \prec q$ ) { |
| 10. | antiDominateAreaCount += cell.count |
| 11. | continue line:5 |
| 12. | } |
| 13. | currentNode = Tree.root |
| 14. | for ( i = 0…cell.LowerBoundDim.lenght ) { |
| 15. | for ( child in currentNode.children) { |
| 16. | if ( child.value = cell.getDimensionValue(i) ) { |
| 17. | currentNode = child |
| 18. | continue line:14 |
| 19. | } |
| 20. | } |
| 21. | nextNode.parent = current |
| 22. | nextNode.value = cell.getDimensionValue(i) |
| 23. | currentNode.children += nextNode |
| 24. | currentNode.children.sort |
| 25. | if ( i + 1 = cell.LowerBoundDim.lenght ) { |
| 26. | nextNode.leaf = true |
| 27. | nextNode.cell = cell |
| 28. | } |
| 29. | currentNode = nextNode |
| 30. | } |
| 31. | } |
| 32. | return [Tree, antiDominateAreaCount] |
| 33. | } |

*Αλγόριθμος 15: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο δημιουργείται η περιγραφική δενδροειδής μορφή του πλέγματος $G_s$.*

| Summarized Tree $G_s$: Αλγόριθμος υπολογισμού προσεγγιστικής κατάταξης q | |
|---|---|
| 1. | Στοιχεία εισόδου: $G_s$ as cellsTree, q, antiDominateAreaCount |
| 2. | Στοιχεία εξόδου: min, max |
| 3. | GET_COUNT() { |
| 4. | min = antiDominateAreaCount |
| 5. | max = 0 |
| 6. | queryScore = $f_w(q)$ |
| 7. | currentNode = cellsTree.root |
| 8. | positions[] = table[dimensionsNumber] |
| 9. | currentDepth = 0 |
| 10. | while ( current not Null ) { |
| 11. | for ( i = positions[currentDepth]…currentNode.children ) { |
| 12. | positions[currentDepth] = i |
| 13. | if ( currentNode.children[i].leaf = true ) { |
| 14. | currentChild = currentNode.children[i] |
| 15. | if ( $f_w(currentChild.LowerBound) <$ queryScore ) { |
| 16. | min += currentChild .cell.count |
| 17. | } |
| 18. | else if ( $f_w(currentChild.UpperBound) <$ queryScore ) { |
| 19. | max += currentChild .cell.count |
| 20. | } |
| 21. | else { |
| 22. | currentNode = currentNode.parent |
| 23. | position[currentDepth] = 0 |
| 24. | currentDepth -= 1 |
| 25. | if ( currentDepth >= 0 ) { |
| 26. | positions[currentDepth] += 1 |
| 27. | } |
| 28. | continue line:10 |
| 29. | } |
| 30. | } |
| 31. | else { |
| 32. | currentNode = currentNode.children[i] |
| 33. | currentDepth += 1 |
| 34. | continue line:10 |
| 35. | } |
| 36. | } |
| 37. | currentNode = currentNode.parent |
| 38. | position[currentDepth] = 0 |
| 39. | currentDepth -= 1 |
| 40. | if ( currentDepth >= 0 ) { |
| 41. | positions[currentDepth] += 1 |
| 42. | } |
| 43. | } |
| 44. | max += min |
| 45. | return [min,max] |
| 46. | } |

*Αλγόριθμος 16: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο υπολογίζεται η προσεγγιστική κατάταξη του q για ένα weighting vector βάση του πλέγματος $G_s$ με τύπο Summarized Tree.*




*Εικόνα 27: Στην εικόνα αυτή παρουσιάζονται στα δεξιά το Summarized πλέγμα σε μορφή πίνακα, ενώ αριστερά παρουσιάζεται το πλέγμα σε δενδροειδής περιληπτική μορφή. Με πράσινο χρώμα απεικονίζονται τα κελιά τα οποία επηρεάζουν την προσεγγιστική κατάταξη του q, ενώ με κόκκινο χρώμα σημειώνονται τα κελιά τα οποία δεν επηρεάζουν την κατάταξη αυτή. Επιπλέον τα βέλη περιγράφουν τον τρόπο με τον οποίο διατρέχονται τα κελιά, βάση των αλγορίθμων που περιεγράφηκαν. (Υπενθυμίζεται ότι τα κελιά που κυριαρχούν το q δεν αποθηκεύονται και κρατείται ένας μετρητής για το πλήθος των στοιχείων τα οποία περιέχουν, ο μετρητής αυτός παρουσιάζεται σαν κελί για κατανόηση).*

## 6.4 Αλγόριθμος περικοπής W

Χρησιμοποιώντας ένα πλέγμα $G_s$ όπως προαναφέρθηκε είναι δυνατόν να υπολογιστεί η προσεγγιστική κατάταξη του q για οποιοδήποτε w ∈ W. Αξιοποιώντας το αποτέλεσμα αυτό μπορεί να βελτιωθεί σημαντικά η απόδοση του αλγορίθμου αποφεύγοντας αχρείαστους υπολογισμούς και άσκοπη μεταφορά δεδομένων. Το πλέγμα αυτό είναι διαθέσιμο σε κάθε κόμβο (Mapper) κατά την πρώτη φάση (Map φάση), επίσης σε κάθε κόμβο είναι διαθέσιμο ένα υποσύνολο w ∈ W'. Στόχος είναι να σταλούν προς την δεύτερη φάση (Reduce φάση) τόσα w ώστε να προσεγγίζουν το πλήθος αυτών που ανήκουν στα τελικά αποτελέσματα του Reverse Top-k αλγορίθμου. Όπως αναφέρθηκε στο κεφάλαιο Προχωρημένες Ιδιότητες ισχύουν τα εξής:

- Αν $M(q,w) < k$ τότε: Το w ανήκει στα τελικά Reverse Top-k αποτελέσματα του q.
- Αν $k < m(q,w)$ τότε: Το w δεν ανήκει στα τελικά Reverse Top-k αποτελέσματα του q.
- Αν $m(q,w) \leq k \leq M(q,w)$ τότε: Το w δεν μπορεί να προσδιοριστεί αν ανήκει ή όχι στα τελικά Reverse Top-k αποτελέσματα του q.

Με την χρήση των ιδιοτήτων αυτών προκύπτει ένας αλγόριθμος για την περικοπή των weighting vectors. Κάθε στοιχείο w ∈ W' που στέλνεται για επεξεργασία προς την δεύτερη φάση (Reduce φάση) συνοδεύεται με μια ετικέτα που χαρακτηρίζει την αξιολόγησή του βάση τις παραπάνω ιδιότητες. Για παράδειγμα τα στοιχεία τα οποία ανήκουν στα τελικά Reverse Top-k αποτελέσματα συνοδεύονται από τον αριθμό 1, έτσι ώστε ο κόμβος της δεύτερης φάσης που θα το παραλάβει να μην υπολογίσει αν το στοιχείο ανήκει στα Reverse Top-k αποτελέσματα και να το εξάγει απευθείας στα τελικά αποτελέσματα. Ο Αλγόριθμος 17 παρουσιάζει τον τρόπο με τον οποίο γίνεται η περικοπή των weighting vectors.

| **Αλγόριθμος περικοπής στοιχείων W** | |
|---|---|
| **1.** | Στοιχεία εισόδου: W', q, k, $G_s$ |
| **2.** | Στοιχεία εξόδου: W'' |
| **3.** | PruningW { |
| **4.** | for ( w in W' ) { |
| **5.** | min, max = $G_s$.getCount(q, w) |
| **6.** | if ( max < $k$ ) { |
| **7.** | R(S,W,k,q) += w |
| **8.** | } |
| **9.** | else if ( $k < min$ ) { |
| **10.** | //Do nothing |
| **11.** | } |
| **12.** | else { |
| **13.** | W'' += w |
| **14.** | } |
| **15.** | } |
| **16.** | } |

*Αλγόριθμος 17: Ο αλγόριθμος αυτός περιγράφει τον τρόπο με τον οποίο γίνεται η περικοπή weighting vectors τα οποία δεν επηρεάζουν τα τελικά αποτελέσματα και η αποστολή στοιχείων απευθείας στα τελικά αποτελέσματα όταν ανήκουν σε αυτά και είναι γνωστό από το $G_s$.*

## 6.5 Map φάση

Κατά την Map φάση, η Map μέθοδος εκτελείται σε κάθε Mapper και παίρνει ως είσοδο δύο υποσύνολα τα $S_i \subset S$ και $W_i \subset W$. Επίσης είναι διαθέσιμο το πλέγμα $G_s$ καθώς και το $G_w$ (οι ομάδες των weighting vectors). Στόχος της Map φάσης είναι να στείλει προς την Reduce φάση τα λιγότερα δυνατά δεδομένα προκειμένου να παραχθούν σωστά τα τελικά αποτελέσματα του Reverse Top-k αλγορίθμου. Αλγόριθμοι για την περικοπή των στοιχείων p ∈ S όσο και των w ∈ W, περιεγράφηκαν προηγουμένως. Όμως πρέπει να επιλεγούν οι πιο αποδοτικοί από αυτούς τους αλγορίθμους προκειμένου να υπάρξει το καλύτερο δυνατό αποτέλεσμα βάση αυτών. Όπως φαίνεται στην πειραματική αξιολόγηση ο πιο αποδοτικός αλγόριθμος για την περικοπή των στοιχείων w ∈ W είναι ο ExtremeScore & R-Lists ενώ για την περικοπή των στοιχείων p ∈ S η πιο αποδοτική δομή του $G_s$ είναι η Summarized Tree. Έτσι ο συνδυασμός των αλγορίθμων αυτών θα αποτελέσει τον αλγόριθμο της Map φάσης ο οποίος περιγράφεται στο πεδίο Αλγόριθμος 18. Όπως αναφέρθηκε στον αλγόριθμο περικοπής των στοιχείων p ∈ S σε προηγούμενο κεφάλαιο τα στοιχεία που συνοδεύονται με τον αριθμό 1 στο κλειδί τους ανήκουν στα τελικά Reverse Top-k αποτελέσματα ενώ αυτά που συνοδεύονται με τον αριθμό 0 δεν είναι γνωστό αν ανήκουν ή όχι στα τελικά Reverse Top-k αποτελέσματα. Όσο αναφορά την διαχείριση των σύνθετων κλειδιών περισσότερες λεπτομέρειες θα δοθούν στην συνέχεια. Τέλος σε αυτό το σημείο αξίζει να σημειωθεί ότι βάση του πλέγματος $G_s$ πραγματοποιείται ένας έλεγχος και αν διαπιστωθεί ότι το πλήθος των κυριάρχων του q (anti-dominate area) είναι μεγαλύτερο ή ίσο με το k τότε δεν εκτελείται ο αλγόριθμος καθώς θα έχει κενό πλήθος αποτελεσμάτων.

| **Σύνθετος Αλγόριθμος: Map φάση** | |
|---|---|
| **1.** | Στοιχεία εισόδου: Si, Wi, q, k, $G_s$, $G_w$ |
| **2.** | Στοιχεία εξόδου: S'i, W'i |
| **3.** | MAP (Element x) { |
| **4.** | if ( x ∈ Si ) { |
| **5.** | if ( q ≺ x ) { |
| **6.** | NOT_REAL_BOUND_AND_RLISTS.SEND_TO_REDUCER(x) |
| **7.** | } |
| **8.** | } |
| **9.** | else if ( x ∈ Wi ) { |
| **10.** | min, max = $G_s$.getCount(q, w) |
| **11.** | if ( max < $k$ ) { |
| **12.** | relativeGroup = $G_w$.getRelativeGroup(w) |
| **13.** | output< (relativeGroup.id,1) , w > |
| **14.** | } |
| **15.** | else if ( $k$ < $min$ ) { |
| **16.** | //Do nothing |
| **17.** | } |
| **18.** | else { |
| **19.** | relativeGroup = $G_w$.getRelativeGroup(w) |
| **20.** | output< (relativeGroup.id,0) , w > |
| **21.** | } |
| **22.** | } |
| **23.** | } |

*Αλγόριθμος 18: Παρουσιάζεται ο Ψευδοκώδικας του Σύνθετου αλγορίθμου. Διευκρινίζεται ότι η γραμμή 6 είναι παραπομπή στον Αλγόριθμος 9 για συντομία.*

## 6.6 Reduce φάση

Κατά την Reduce φάση υπάρχουν πολλοί Reducers οι οποίοι εκτελούνται παράλληλα προκειμένου να υπολογίσουν το τελικό αποτέλεσμα του Reverse Top-k αλγορίθμου. Κάθε Reducer επεξεργάζεται μία και μόνο μία ομάδα weighting vectors. Για τον υπολογισμό των τοπικών Reverse Top-k αποτελεσμάτων χρησιμοποιείται ο RTA αλγόριθμος. Υπενθυμίζεται ότι η ένωση των τελικών Reverse Top-k αποτελεσμάτων των Reducer παράγει τα τελικά Reverser Top-k αποτελέσματα. Βάση του αλγορίθμου ExtremeScore & R-Lists είναι δυνατόν να προβλεφθεί με χαμηλό κόστος η περίπτωση όπου κανένα weighting vector σε μια ομάδα δεν ανήκει στα τελικά Reverse Top-k αποτελέσματα. Έτσι μπορεί να σταματήσει ο Reducer όπως φαίνεται στην γραμμή:10-12. Αξίζει να σημειωθεί ότι τα στοιχεία έρχονται ταξινομημένα στον κάθε Reducer με την εξής σειρά, πρώτα έρχονται τα weighting vectors που ανήκουν στα τελικά Reverse Top-k αποτελέσματα (αυτά που συνοδεύονται από τον αριθμό 0 στο κλειδί τους), έπειτα έρχονται τα στοιχεία p ∈ S και τέλος τα υπόλοιπα weighting vectors (αυτά που συνοδεύονται από τον αριθμό 1 στο κλειδί τους). Στην γραμμή:16-18 φαίνεται η απευθείας εγγραφή στα αποτελέσματα των weighting vectors που από την Map φάση ήταν γνωστά ότι ανήκουν στα τελικά αποτελέσματα. Ο αλγόριθμος για την Reduce φάση παρουσιάζεται στο πεδίο Αλγόριθμος 19.

| **Σύνθετος Αλγόριθμος: Reduce φάση** | |
|---|---|
| **1.** | Στοιχεία εισόδου: S''i, W''i, q, k, $G_w$ |
| **2.** | Στοιχεία εξόδου: R(S''i, W''i, k, q) |
| **3.** | REDUCE (key, List<Element> List) { |
| **4.** | localAntidominateAreaSum = 0 |
| **5.** | currentGroup = $G_w$.get(key) |
| **6.** | for (Element x in List) { |
| **7.** | if ( x ∈ S''i ) { |
| **8.** | S''i += x |
| **9.** | if ($f_{\text{currentGroup}.UB}(p) \leq f_{\text{currentGroup.LB}}(q)$ ) { |
| **10.** | localAntidominateAreaSum += 1 |
| **11.** | if (localAntidominateAreaSum >= k ) { |
| **12.** | STOP_REDUCER |
| **13.** | } |
| **14.** | } |
| **15.** | } |
| **16.** | else if ( x ∈ W''i ) { |
| **17.** | if ( x ∈ R(Si, Wi, k, q) ) { |
| **18.** | output ( x ) |
| **19.** | } |
| **20.** | else if ( RTA.belongsToResults(x, S''i, k, q) ) { |
| **21.** | output ( x ) |
| **22.** | } |
| **23.** | } |
| **24.** | } |
| **25.** | } |

*Αλγόριθμος 19: Παρουσιάζεται ο ψευδοκώδικας του Σύνθετου αλγορίθμου για την Reduce φάση. Διευκρινίζεται ότι τα στοιχεία w∈W που ανήκουν στα τελικά αποτελέσματα έρχονται πρώτα, έπειτα ακολουθούν τα στοιχεία p∈S και τέλος φτάνουν τα στοιχεία w∈W.*

## 6.7 Περιγραφή υλοποίησης

Στην ενότητα αυτή θα παρουσιαστεί η υλοποίηση του αλγορίθμου. Αρχικά θα γίνει μια αναφορά στα στοιχεία που χρησιμοποιήθηκαν για την εκτέλεση του αλγορίθμου. Έπειτα θα παρουσιαστούν τεχνικά στοιχεία σχετικά με το Hadoop, σημειώνεται ότι η σειρά παρουσίασης θα είναι αντίστοιχη με την ροή των εργασιών που εκτελεί το Hadoop. Τέλος θα παρουσιαστούν λεπτομέρειες σχετικά με την δομή του κώδικα και θα αναλυθεί περαιτέρω ότι κρίνεται αναγκαίο αποφεύγοντας αχρείαστες λεπτομέρειες καθώς η υλοποίηση συνοδεύεται από Javadoc το οποίο περιγράφει όλες τις λεπτομέρειες σχετικά με την υλοποίηση.

### 6.7.1 Βασικά χαρακτηριστικά

Η γλώσσα προγραμματισμού που επιλέχτηκε για την υλοποίηση του αλγορίθμου είναι η Java και πιο συγκεκριμένα η έκδοση στην οποία βασίζεται η υλοποίηση είναι η 1.7, όσο αναφορά την έκδοση του Hadoop API που χρησιμοποιήθηκε είναι η 2.6.0. Αποφασίστηκαν οι παραπάνω εκδόσεις προκειμένου να είναι συμβατές με το περιβάλλον εκτέλεσης των πειραμάτων. Όσο αναφορά την υλοποίηση συνοδεύεται από Javadoc για την περιγραφή της λειτουργικότητας των μεθόδων και των κλάσεων, επιπλέον δόθηκε έμφαση προκειμένου ο κώδικας να είναι αυτό-περιγραφικός (self-documented) έτσι ώστε να μπορεί να μελετηθεί εύκολα. Ακόμα αξίζει να αναφερθεί ότι υλοποιήθηκαν Unit Tests για τον έλεγχο κρίσιμων τμημάτων του αλγορίθμου, για την υλοποίηση των Tests αυτών χρησιμοποιήθηκε το JUnit και πιο συγκεκριμένα η έκδοση 4.11.

### 6.7.2 Περιγραφή υλοποίησης των στοιχείων του Hadoop

Αρχικά όπως αναφέρθηκε στην περιγραφή του αλγορίθμου κάθε κόμβος (Mapper και Reducer) πρέπει να έχει το πλέγμα $G_s$ καθώς και το $G_w$ (ομάδες weighting vectors). Προκειμένου να δοθούν τα δύο αυτά στοιχεία προς όλους τους κόμβους χρησιμοποιήθηκε η DistributedCache όπου ένα αρχείο στέλνεται και είναι διαθέσιμο τοπικά πριν την εκκίνηση του Job σε όλους τους κόμβους (5). Με αυτό τον τρόπο ο χρήστης ορίζει την τοποθεσία των αρχείων που περιέχουν την πληροφορία αυτή και έπειτα τα αρχεία αυτά μοιράζονται προς όλους τους κόμβους.

Έπειτα πρέπει να καθοριστεί ο τύπος δεδομένων που δέχονται ως είσοδο οι Mappers. Όπως αναφέρθηκε κατά την περιγραφή του αλγορίθμου οι Mappers δέχονται ως είσοδο στοιχεία από τα δύο σύνολα δεδομένων S, W. Οπότε μπορεί να χρησιμοποιηθεί το FileInputFormat που παρέχεται από το Hadoop API (5) ή το InputFormat που δημιουργήθηκε στον Απλοϊκό αλγόριθμο. Σε αυτό το σημείο επιλέχθηκε η χρήση του δεύτερου InputFormat.

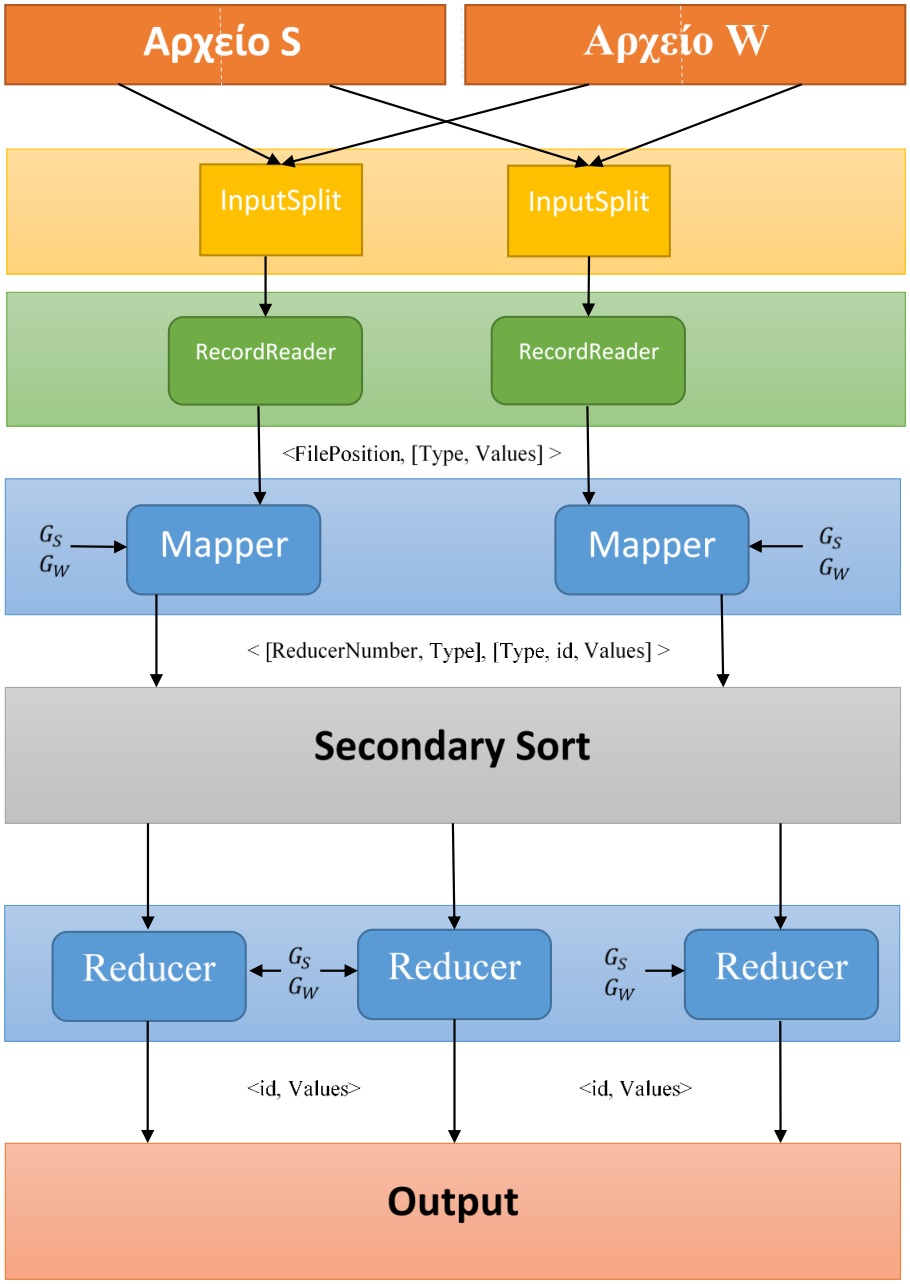
Δημιουργώντας ένα InputFormat είναι εφικτό ο κάθε Mapper να διαβάζει στοιχεία από τα σύνολα δεδομένων S,W αλλά χωρίς να γνωρίζει από πιο σύνολο δεδομένων προήλθε ένα στοιχείο, κάτι που κρίνεται απαραίτητο. Όπως αναφέρθηκε στην εισαγωγή ο RecordReader είναι υπεύθυνος για την ανάγνωση των InputSplits και την μετατροπή αυτών σε ζευγάρια κλειδιών τιμής (δηλ. <key,value> pairs) που δέχονται ως είσοδο οι Mappers. Έτσι δημιουργήθηκε ένας προσαρμοσμένος στην περίπτωση RecordReader και ένας νέος τύπος τιμής (value) όπου όταν ο RecordReader διαβάζει μια γραμμή από ένα InputSplit στέλνει ως κλειδί (key) τον αριθμό της γραμμής που διάβασε και ως τιμή (value) στέλνει την γραμμή του αρχείου μαζί με το όνομα το αρχείου. Με αυτό τον τρόπο έχοντας ορίσει ο χρήστης τα αρχεία που βρίσκονται τα σύνολα δεδομένων S,W καθώς και ποιο αρχείο αντιστοιχεί σε ποιο σύνολο είναι δυνατόν οι Mappers να γνωρίζουν την προέλευση όλων των στοιχείων.

Πλέον κάθε Mapper δέχεται ως είσοδο στοιχεία από τα σύνολα δεδομένων S,W και γνωρίζει την προέλευση του κάθε στοιχείου. Η δουλειά των Mappers είναι να στείλουν προς την Reduce φάση όσο λιγότερα δεδομένα γίνεται, ο αλγόριθμος για την περικοπή των στοιχείων και την αποστολή των δεδομένων στην Reduce φάση παρουσιάζεται στο πεδίο Αλγόριθμος 18. Τα δεδομένα τα οποία στέλνει προς τους Reducers είναι στοιχεία τα οποία ανήκουν στα σύνολα δεδομένων S, W και έχουν ως πληροφορία έναν μοναδικό αριθμό (id) ανά σύνολο δεδομένων και μία λίστα με δεκαδικές τιμές (το μέγεθος της λίστας είναι ίσο με το πλήθος των διαστάσεων). Προκειμένου οι Reducers να γνωρίζουν το σύνολο των δεδομένων όπου προέρχεται το κάθε στοιχείο δημιουργείται ένας νέος τύπος τιμής (value) ο οποίος κρατά τα δεδομένα του στοιχείου καθώς και τον τύπο τους (δηλ. αν ανήκουν στο σύνολο S ή W). Σε αυτό το σημείο αξίζει να σημειωθεί ότι δημιουργήθηκε και ένας άλλος εικονικός τύπος δεδομένων ο W_InRTopK όπου αφορά στοιχεία που ανήκουν στο σύνολο δεδομένων W αλλά από την Map φάση είναι γνωστό ότι ανήκουν στα τελικά Reverse Top-k αποτελέσματα. Το κλειδί το οποίο χρησιμοποιείται είναι ένα σύνθετο κλειδί το οποίο αποτελείται από τον τύπο του στοιχείου (S, W, W_InRTopK) και τον αριθμό του Reducer στον οποίο πρέπει να σταλεί το στοιχείο (reducerKey), προκειμένου να φτάσουν τα στοιχεία ταξινομημένα στους Reducers (Secondary Sorting) (4). Έγινε η επιλογή να στέλνεται μόνο ο τύπος του στοιχείου και όχι όλη τιμή (value) ώστε να μην υπερφορτώνεται το δίκτυο και παράλληλα να είναι δυνατή η δευτερεύουσα ταξινόμηση (Secondary Sorting). Δηλαδή ο τύπος εξόδου της Map φάση έχει την εξής μορφή <(reducerKey,itemType),value>. Σε αυτό το σημείο πρέπει να σημειωθεί ότι ο σύνθετος αυτός τύπος του κλειδιού είναι μία κλάση η οποία οφείλει να υλοποιεί (implements) την διεπαφή (interface) WritableComparable του Hadoop API (5). Έτσι στην κλάση ορίζεται η μέθοδος ταξινόμησης των στοιχείων η οποία υλοποιήθηκε ώστε να ταξινομεί τα στοιχεία ως εξής: πρώτα έρχονται τα στοιχεία που έχουν χαρακτηριστεί ως W_InRTopK, έπειτα όσα ανήκουν στο σύνολο δεδομένων S και τέλος αυτά του συνόλου δεδομένων W.

Τα δεδομένα που στέλνουν οι Mappers προς τους Reducer διαβάζονται πρώτα από έναν Partitioner ο οποίος είναι υπεύθυνος για τον διαμοιρασμό των στοιχείων στους Reducers. Επειδή μόνο ένα τμήμα του κλειδιού πρέπει να ορίζει που θα σταλεί το στοιχείο και όχι όλο το κλειδί δημιουργείται ένας νέος Partitioner. Ο Partitioner αυτός λαμβάνει υπόψη του μόνο τον αριθμό του Reducer (reducerKey) από το κλειδί και δεν κοιτά τον τύπο του στοιχείου που κουβαλά το κλειδί έτσι ώστε να σταλεί το στοιχείο στον σωστό Reducer.

Αφού ο Partitioner στείλει τα στοιχεία στους αντίστοιχους Reducers, διαβάζει τα στοιχεία ένας GroupComparator ο οποίος είναι υπεύθυνος να αναγνωρίσει πλειάδες με το ίδιο κλειδί και να τις βάλει σε μία λίστα, να δημιουργήσει ομάδες δηλαδή. Λόγω αυτού δημιουργείται ένας νέος GroupComparator ο οποίος συγκρίνει τις πλειάδες όχι βάση όλου του κλειδιού τους αλλά βάση του αριθμού του Reducer (reducerKey) προκειμένου να μπουν τα στοιχεία με τον ίδιο αριθμό Reducer (reducerKey) στην ίδια ομάδα.

Αφού υλοποιήθηκε η δευτερεύουσα ταξινόμηση (Secondary Sorting) και τα στοιχεία έρχονται ταξινομημένα στον Reducer με την επιθυμητή σειρά, οι Reducers καλούνται να υπολογίσουν τα τελικά αποτελέσματα. Ο αλγόριθμος ο οποίος εκτελεί κάθε Reducer περιγράφεται στο πεδίο Αλγόριθμος 19. Επίσης αξίζει να σημειωθεί ότι έχει υλοποιηθεί εκτός από τον RTA και ο BRS αλγόριθμος για τον υπολογισμό των Reverse Top-k αποτελεσμάτων. Ως έξοδος της Reduce φάση είναι τα στοιχεία του συνόλου δεδομένων W που αποτελούν τα Reverse Top-k αποτελέσματα. Ο τύπος των εξαγόμενων αποτελεσμάτων είναι πάλι ένα ζευγάρι κλειδιού, τιμής (δηλ. <key,value> pair) τόσο ως τιμή όσο και ως κλειδί επιλέχτηκε ο αλφαριθμητικός τύπος Text του Hadoop API και ως κλειδί γράφεται το μοναδικό αναγνωριστικό (id) του εκάστοτε στοιχείου ενώ ως τιμή γράφεται οι τιμές των διαστάσεων του στοιχείου χωρισμένες με κόμμα (,).




*Εικόνα 28: Στην εικόνα αυτή παρουσιάζεται σχηματικά ο τρόπος λειτουργίας του Σύνθετου αλγορίθμου καθώς και τα κλειδιά τα οποία μεταφέρονται σε κάθε φάση.*

### 6.7.3 Στοιχεία μέτρησης

Προκειμένου να μελετηθεί η συμπεριφορά του αλγορίθμου χρησιμοποιήθηκε η λειτουργικότητα των μετρητών (Counters) που παρέχει το Hadoop API (5). Πιο συγκεκριμένα δημιουργήθηκαν προσαρμοσμένοι μετρητές (Counters) έτσι ώστε να μετρούνται σημαντικά στοιχεία κατά την εκτέλεση ενός Map Reduce Job. Οι μετρητές αυτοί είναι διαθέσιμοι μετά την εκτέλεση του αλγορίθμου σε μορφή αναφοράς και αποτελούν τις βασικές πηγές μετρήσεων για την πειραματικά αξιολόγηση.

| Μετρητές | Περιγραφή |
|---|---|
| **S** | Το πλήθος των στοιχείων p ∈ S που δόθηκαν ως είσοδο στο Map Reduce Job |
| **W** | Το πλήθος των στοιχείων w ∈ W που δόθηκαν ως είσοδο στο Map Reduce Job |
| **S1** | Το πλήθος των στοιχείων p ∈ S τα οποία επιβίωσαν από την περικοπή βάση της κυριαρχίας του q (δεν κυριαρχούνται από το q) |
| **S_pruned_by_domination** | Το πλήθος των στοιχείων p ∈ S τα οποία δεν επιβίωσαν από την περικοπή βάση της κυριαρχίας του q (κυριαρχούνται από το q) |
| **W_in_RTOPk** | Το πλήθος των στοιχείων w ∈ W τα οποία γνωρίζουμε ότι βρίσκονται στα τελικά αποτελέσματα από την Map φάση βάση του πλέγματος $G_s$. |
| **W_pruned_by_GridS** | Το πλήθος των στοιχείων w ∈ W τα οποία γνωρίζουμε ότι δεν βρίσκονται στα τελικά αποτελέσματα από την Map φάση βάση του πλέγματος $G_s$. Δηλαδή απορρίπτονται. |
| **W1** | Το πλήθος των στοιχείων w ∈ W τα οποία δεν γνωρίζουμε αν βρίσκονται ή όχι στα τελικά αποτελέσματα από την Map φάση βάση του πλέγματος $G_s$. Δηλαδή στέλνονται για επεξεργασία στην Reduce φάση. |

| Μετρητές | Περιγραφή |
|---|---|
| **S2_pruned_by_GridW** | Το πλήθος των στοιχείων p ∈ S τα οποία δεν στάλθηκαν στην Reduce φάση χάρη στον αλγόριθμο περικοπής των στοιχείων S ExtremeScore όπου αυτός χρησιμοποιείται. Υπενθυμίζεται ότι στην χειρότερη περίπτωση μπορούν να σταλούν $reducerNumber * S1$ στοιχεία στους Reducers. |
| **S2_pruned_by_RLists** | Το πλήθος των στοιχείων p ∈ S τα οποία δεν στάλθηκαν στην Reduce φάση χάρη στον αλγόριθμο περικοπής των στοιχείων S R-Lists όπου αυτός χρησιμοποιείται. Υπενθυμίζεται ότι στην χειρότερη περίπτωση μπορούν να σταλούν $reducerNumber * S1$ στοιχεία στους Reducers. |
| **Total_effort_for_pruning_S_in_MilliSeconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Mapper για την περικοπή των στοιχείων p ∈ S. Η μονάδα χρόνου που χρησιμοποιείται είναι Milliseconds. |
| **Total_effort_for_pruning_W_in_MilliSeconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Mapper για την περικοπή των στοιχείων w ∈ W. Η μονάδα χρόνου που χρησιμοποιείται είναι Milliseconds. |
| **S2** | Το πλήθος των στοιχείων p ∈ S που στέλνονται από την Map φάση προς τους Reducers. Υπενθυμίζεται ότι στην χειρότερη περίπτωση μπορούν να σταλούν $reducerNumber * S1$ στοιχεία στους Reducers. |
| **W2** | Το πλήθος των στοιχείων w ∈ W που στέλνονται προς τους Reducers προκειμένου να εξακριβωθεί αν ανήκουν ή όχι στα τελικά Reverse Top-k αποτελέσματα. |
| **RTOPk_Output** | Το πλήθος των στοιχείων που ανήκουν στα τελικά Reverse Top-k αποτελέσματα. |
| **S_Elements_In_Antidominance_Area_Of_GridS** | Το πλήθος των στοιχείων που βρίσκονται στην περιοχή κυριάρχων του q (anti-dominate area) βάση του πλέγματος $G_s$. |
| **Reducers_Early_Terminated** | Το πλήθος των Reducer που σταμάτησαν πρόωρα χάρη στον αλγόριθμο ExtremeScore. |
| **Total_effort_to_create_rtree_in_seconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Reducer για την αποθήκευση των στοιχείων p ∈ S στην μνήμη (σε R-Tree στον BRS, σε λίστα στον RTA). Η μονάδα χρόνου που χρησιμοποιείται είναι Milliseconds. |
| **Total_effort_for_rtopk_algorithm_in_seconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Reducer για τον υπολογισμό των Reverse Top-k αποτελεσμάτων. Η μονάδα χρόνου που χρησιμοποιείται είναι Milliseconds. |
| **Total_effort_for_processing_w_in_rtopk_in_seconds** | Ο χρόνος που σπαταλήθηκε αθροιστικά από κάθε Reducer για την απευθείας εγγραφή στα τελικά αποτελέσματα των στοιχείων W_InRTopK. Η μονάδα χρόνου που χρησιμοποιείται είναι Milliseconds. |

# 7. Πειραματική Αξιολόγηση

Σε αυτό το κεφάλαιο πραγματοποιείται μια αξιολόγηση των αλγορίθμων που υλοποιήθηκαν βάση των επιδόσεών τους σε διάφορα πειράματα. Αρχικά θα γίνει μια αναφορά στο περιβάλλον που έτρεξαν τα πειράματα, στα σύνολα δεδομένων που χρησιμοποιήθηκαν και στον τρόπο αξιολόγησης. Έπειτα θα ακολουθήσει μια σύγκριση μεταξύ των αλγορίθμων για την περικοπή των στοιχείων S καθώς και των πλεγμάτων $G_s$ που προτάθηκαν στον Σύνθετο Αλγόριθμο. Τέλος θα συγκριθούν ο Απλοϊκός Αλγόριθμος με τον Σύνθετο αλγόριθμο προκειμένου να διαπιστωθούν οι όποιες διαφορές τους.

## 7.1 Περιβάλλον

### 7.1.1 Υπολογιστικό περιβάλλον

Το περιβάλλον στο οποίο εκτελέστηκαν τα πειράματα είναι ένα κατανεμημένο περιβάλλον (CHD cluster) το οποίο αποτελείται από 12 κόμβους. Σε αυτό το περιβάλλον υπάρχουν δύο τύποι κόμβων, οι κόμβοι d1-d8 είναι του πρώτου τύπου ενώ οι κόμβοι d9-d12 είναι του δεύτερου τύπου. Πιο αναλυτικά:

- Τύπος 1 (d1-d8)
    - Κύρια μνήμη (RAM): 32 GB
    - Σκληρός δίσκος (Hard disk): 5 TB
    - Επεξεργαστής (CPU): 2 CPUs με 8 πυρήνες (cores) ο καθένας και συχνότητα 2.6 GHz.
- Τύπος 2 (d9-d12)
    - Κύρια μνήμη (RAM): 128 GB
    - Σκληρός δίσκος (Hard disk): 8 TB
    - Επεξεργαστής (CPU): 2 CPUs με 12 πυρήνες (cores) ο καθένας και συχνότητα 2.6 GHz.

Ένας κόμβος από αυτούς έχει οριστεί ως κύριος κόμβος και έχει τον ρόλο του Job Tracker και του Namenode. Επιπλέον κάθε κόμβος τρέχει σε λειτουργικό σύστημα Ubuntu Server 12.04 LTS. Χρησιμοποιείται επίσης η έκδοση CHD 5.4.8 του Cloudera, όσο αναφορά την έκδοση της Java χρησιμοποιείται η έκδοση 1.7. Σχετικά με τις ρυθμίσεις του HDFS έχει οριστεί το μέγεθος του block σε 64 MB και το πλήθος αντιγράφων σε 3 (replication factor). Τέλος η έκδοση του Hadoop που χρησιμοποιείται είναι η 2.6.0.

### 7.1.2 Σύνολα δεδομένων

Προκειμένου να χρησιμοποιηθούν σύνολα δεδομένων τα οποία να δικαιολογούν την εκτέλεση των πειραμάτων στο Hadoop, δημιουργήθηκαν συνθετικά δεδομένα έτσι ώστε να αξιολογηθεί η απόδοση των αλγορίθμων που προτάθηκαν. Κάθε σύνολο δεδομένων S,W δημιουργείται πριν την εκτέλεση των πειραμάτων και αποθηκεύεται στο HDFS ως αρχείο κειμένου. Κάθε γραμμή αναπαριστά μια πλειάδα όπου τα στοιχεία της χωρίζονται με τον χαρακτήρα «Tab». Κάθε πλειάδα περιέχει στην πρώτη στήλη έναν μοναδικό αριθμό (id) και στις επόμενες στήλες τις τιμές της κάθε διάστασης στον χώρο.

Το μεγαλύτερο σύνολο δεδομένων αποτελείται από 218,5 εκατομμύρια πλειάδες και καταλαμβάνει 10 GB χώρο στον δίσκο. Το σύνολο δεδομένων αυτό είναι δύο τάξεις μεγέθους μεγαλύτερο από το μεγαλύτερο σύνολο δεδομένων που χρησιμοποιήθηκε μέχρι τώρα στην βιβλιογραφία, όπου δεν μπορεί να επεξεργαστεί σε λογικό χρόνο από έναν μόνο κόμβο. Ακόμα αξίζει να σημειωθεί σε αυτό το σημείο ότι μια απλή λύση υπολογισμού Reverse Top-k ερωτήματος θα πρέπει να πραγματοποιήσει τον υπολογισμό 218,5 εκατομμύριων Top-k ερωτημάτων, κάτι που τονίζει την ανάγκη χρησιμοποίησης του Hadoop.

Όσο αναφορά το σύνολο δεδομένων S, δημιουργήθηκαν σύνολα δεδομένων με τις εξής κατανομές: uniform (UN), correlated (CO) και anti-correlated (AC). Το σύνολο δεδομένων W δημιουργείται έτσι ώστε $\sum_{i=0}^{d} w[i] = 1$.

| Τύπος συνόλου δεδομένων | Μέγεθος (GB) | Πλήθος διαστάσεων | Κατανομή |
|---|---|---|---|
| **S** | 1 | 4 | UNIFORM |
| **S** | 5 | 4 | UNIFORM |
| **S** | 5 | 2 | UNIFORM |
| **S** | 5 | 6 | UNIFORM |
| **S** | 5 | 4 | CORRELATED |
| **S** | 5 | 4 | ANTI-CORRELATED |
| **S** | 10 | 4 | UNIFORM |
| **W** | 1 | 4 | UNIFORM |
| **W** | 5 | 4 | UNIFORM |
| **W** | 5 | 2 | UNIFORM |
| **W** | 5 | 6 | UNIFORM |
| **W** | 10 | 4 | UNIFORM |

## 7.2 Σύγκριση πλεγμάτων Gs

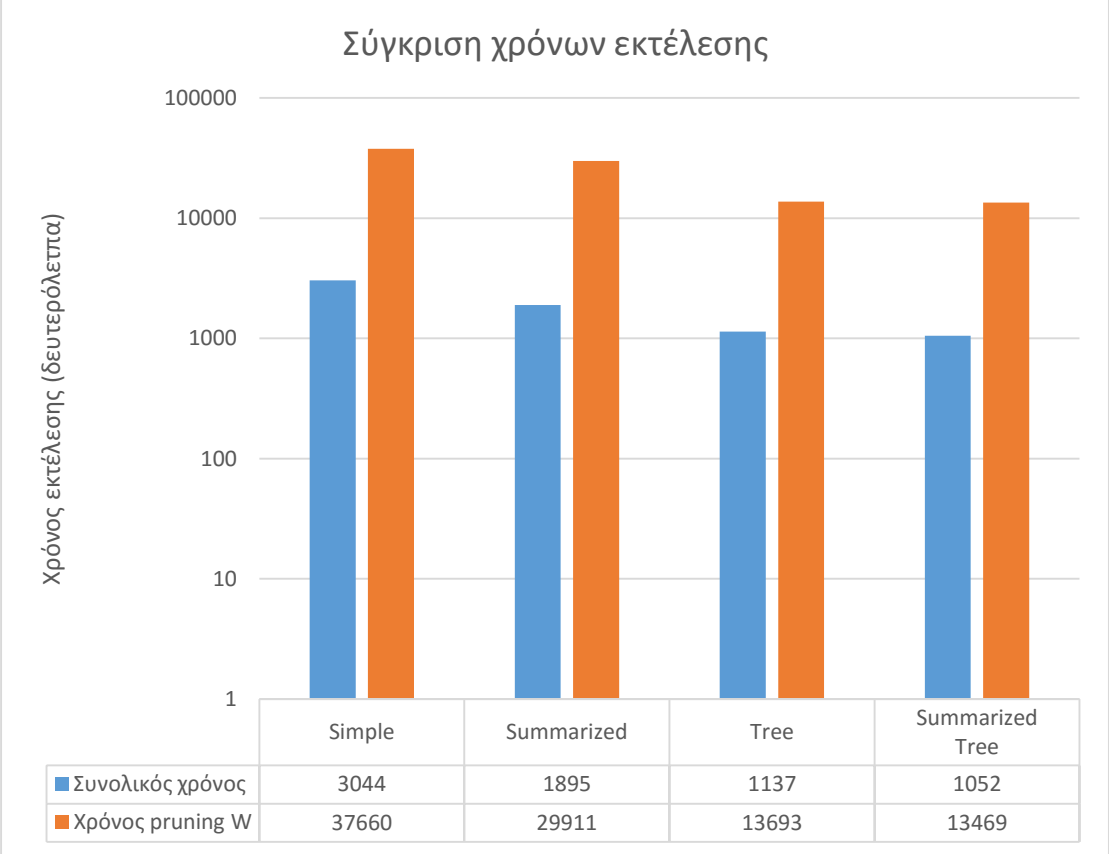
Στο κεφάλαιο Σύνθετος Αλγόριθμος προτάθηκαν 4 τύποι πλεγμάτων $G_s$, ο Simple, ο Summarized, ο Tree και τέλος ο Summarized Tree. Προκειμένου να δημιουργηθεί ο αποδοτικότερος αλγόριθμος για το Reverse Top-k ερώτημα με την τρέχουσα γνώση, πρέπει να επιλεγεί ο καλύτερος τύπος πλέγματος. Από το παράδειγμα που δόθηκε κατά την παρουσίαση των τύπων αυτών καθώς και από την «ευφυΐα» του κάθε τύπου αναμένεται να αναδειχθεί ως καλύτερος τύπος πλέγματος ο τύπος Summarized Tree. Παρόλα αυτά κρίνεται αναγκαία η πειραματική αξιολόγηση προκειμένου να παρατηρηθεί και σε πραγματικό περιβάλλον ποιος είναι όντως ο καλύτερος τύπος πλέγματος.

Εφόσον κρίνεται αναγκαία η σύγκριση των τύπων του πλέγματος $G_s$, πρέπει να οριστούν τα μέτρα σύγκρισης τα οποία θα αναδείξουν τον καλύτερο τύπο. Ο χρόνος εκτέλεσης του αλγορίθμου περικοπής στοιχείων W κατά την Map φάση είναι το βασικό κριτήριο για την αξιολόγηση των τύπων των πλεγμάτων καθώς υπολογίζει την προσεγγιστική κατάταξη του q με την χρήση του πλέγματος. Αναμένεται να αποτελεί σημαντικό παράγοντα επιβάρυνσης της Map φάσης για αυτό οι επιδόσεις του πλέγματος θα επηρεάζουν σημαντικά τον συνολικό χρόνο εκτέλεσης, ο οποίος θα αποτελέσει το δεύτερο κριτήριο αξιολόγησης.

Σε αυτό το σημείο πρέπει να καθοριστούν τα πειράματα τα οποία θα εκτελεστούν για την ανάδειξη του αποδοτικότερου τύπου πλέγματος. Οι παράγοντες οι οποίοι επηρεάζουν την απόδοση του πλέγματος είναι το μέγεθος του πλέγματος, το πλήθος των διαστάσεων και το ερώτημα καθώς σε αυτό βασίζεται η λειτουργία των περιληπτικών πλεγμάτων, όπου περικόπτει τα κελιά που κυριαρχούνται από το q και κρατά μόνο έναν μετρητή για τα κελιά που κυριαρχούν του q. Παρόλα αυτά επειδή οι διαφορές είναι μεγάλες από το πρώτο πείραμα καθώς και επειδή θεωρητικά  ο Summarized Tree είναι ο καλύτερος αλγόριθμος καθώς εξετάζει τα λιγότερα δυνατά κελιά. Για τον λόγο αυτό θα παρουσιαστεί ένα μόνο πείραμα.

Για το πείραμα το οποίο εκτελέστηκε χρησιμοποιήθηκαν σύνολα δεδομένων 4-διαστάσεων. Όπου το σύνολο δεδομένων S ακολουθεί uniform (UN) κατανομή, το μέγεθος του είναι 5 GB και το πλήθος των στοιχείων του είναι 109,2 εκατομμύρια. Επιπλέον το πλέγμα $G_s$ δημιουργείται χωρίζοντας την κάθε διάσταση σε 10 τμήματα. Όσο αναφορά το σύνολο δεδομένων W ακολουθεί uniform (UN) κατανομή, το μέγεθος του είναι 5 GB και το πλήθος των στοιχείων του είναι 143,2 εκατομμύρια. Επιπλέον δημιουργήθηκαν 867 ομάδες από την τμηματοποίηση του χώρου W ώστε να χρησιμοποιηθούν από τον Σύνθετο αλγόριθμο. Τέλος το k είναι 10 ενώ ο q επιλέχτηκε από το k=3 k-skyband.

Στην Εικόνα 29 παρουσιάζονται οι χρόνοι εκτέλεσης. Όπως παρατηρείται η αποδοτικότητα του πλέγματος παίζει καθοριστικό ρόλο για τον συνολικό χρόνο εκτέλεσης, αυτό φαίνεται από το γεγονός ότι στην καλύτερη περίπτωση ο αλγόριθμος εκτελείται σχεδόν 3 φορές γρηγορότερα. Πιο αναλυτικά παρατηρείται ότι πιο αποδοτικός τύπος πλέγματος είναι ο Summarize Tree, έπειτα ο Tree, ο Summarized και ο Simple. Οι τύποι παρουσιάζουν μεγάλες διαφορές μεταξύ τους ειδικά ο τύπος Simple με τους υπόλοιπους. Η διαφορά αυτή οφείλεται στο πλήθος των κελιών που επισκέπτεται ο αλγόριθμος, καθώς πραγματοποιείται επίσκεψη όλων των κελιών για 143.249.997 στοιχεία W είναι κατανοητό ότι θα δημιουργηθούν καθυστερήσεις. Επιπλέον όσο αναφορά τους χρόνους εκτέλεσης του αλγορίθμου περικοπής στοιχείων W παρατηρούνται αντίστοιχες διαφορές με αυτές των συνολικών χρόνων εκτέλεσης. Από τα παραπάνω προκύπτει ότι ο αποδοτικότερος τύπος πλέγματος είναι ο Summarized Tree ο οποίος θα χρησιμοποιηθεί στην πειραματική αξιολόγηση στην συνέχεια του κεφαλαίου.




*Εικόνα 29: Στην εικόνα αυτή παρουσιάζονται οι χρόνοι εκτέλεσης των Map φάσεων καθώς και οι συνολικοί χρόνοι εκτέλεσης προκειμένου να συγκριθεί η αποδοτικότητα των τύπων των πλεγμάτων που υπάρχουν. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα των χρόνων εκτέλεσης είναι λογαριθμική.*

## 7.3 Σύγκριση αλγορίθμων Περικοπής στοιχείων S

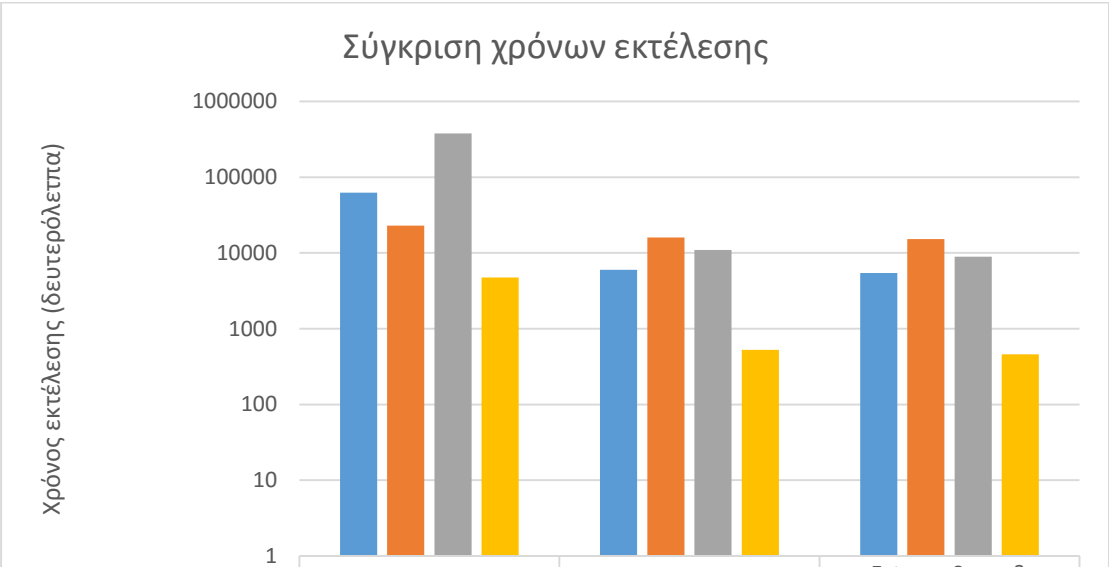
Στο κεφάλαιο Σύνθετος Αλγόριθμος προτάθηκαν 3 αλγόριθμοι για την περικοπή των στοιχείων p ∈ S, ο ExtremeScore, ο R-Lists και τέλος ο ExtremeScore & R-Lists. Προκειμένου να δημιουργηθεί ο αποδοτικότερος αλγόριθμος για το Reverse Top-k ερώτημα με την τρέχουσα γνώση, πρέπει να επιλεγεί ο αποδοτικότερος αλγόριθμος για την περικοπή των στοιχείων p ∈ S. Από την περιγραφή των αλγορίθμων αναμένεται να επιλεγεί ο ExtremeScore & R-Lists καθώς περικόπτει τα περισσότερα στοιχεία αφού αποτελεί συνδυασμό των άλλων δύο. Παρόλα αυτά κρίνεται αναγκαία η πειραματική αξιολόγηση προκειμένου να παρατηρηθεί και σε πραγματικό περιβάλλον ποιος είναι όντως ο αποδοτικότερος αλγόριθμος.

Εφόσον κρίνεται αναγκαία η σύγκριση των αλγορίθμων αυτών, πρέπει να οριστούν τα μέτρα σύγκρισης τα οποία θα αναδείξουν τον καλύτερο αλγόριθμο. Ο χρόνος εκτέλεσης του αλγορίθμου περικοπής των στοιχείων S κατά την Map φάση είναι το βασικό κριτήριο για την αξιολόγηση των αλγορίθμων καθώς αντικατοπτρίζει τις επιδόσεις αυτών. Το πλήθος των στοιχείων p ∈ S που στέλνονται προς την Reduce φάση είναι ακόμα ένα κύριο κριτήριο όπως και το πλήθος των Reducer που σταματούν λόγω των αλγορίθμων αυτών. Ακόμα ο χρόνος εκτέλεσης της Reduce φάσης έχει σημαντικό ρόλο καθώς όσα περισσότερα στοιχεία S στέλνονται στους Reducers τόσο θα αυξάνεται ο χρόνος εκτέλεσής τους. Τέλος δεν μπορεί να παραληφθεί ο συνολικός χρόνος του Reverse Top-k αλγορίθμου από τα βασικά κριτήρια.

Σε αυτό το σημείο πρέπει να καθοριστούν τα πειράματα τα οποία θα εκτελεστούν για την ανάδειξη του αποδοτικότερου αλγορίθμου. Οι παράγοντες οι οποίοι επηρεάζουν την απόδοση του αλγορίθμου είναι το πλήθος των ομάδων των weighting vectors, το πλήθος των διαστάσεων και το k. Οι διαφορές είναι μεγάλες από το πρώτο πείραμα και επειδή θεωρητικά ο ExtremeScore & R-Lists είναι ο καλύτερος αλγόριθμος καθώς περικόπτει τα περισσότερα στοιχεία θα παρουσιαστεί ένα μόνο πείραμα.

Για το πείραμα το οποίο εκτελέστηκε χρησιμοποιήθηκαν σύνολα δεδομένων 4-διαστάσεων. Όπου το σύνολο δεδομένων S ακολουθεί uniform (UN) κατανομή, το μέγεθος του είναι 5 GB και το πλήθος των στοιχείων του είναι 109,2 εκατομμύρια. Επιπλέον το πλέγμα $G_S$ δημιουργείται χωρίζοντας την κάθε διάσταση σε 10 τμήματα. Όσο αναφορά το σύνολο δεδομένων W ακολουθεί uniform (UN) κατανομή, το μέγεθος του είναι 5 GB και το πλήθος των στοιχείων του είναι 143,2 εκατομμύρια. Επιπλέον δημιουργήθηκαν 867 ομάδες από την τμηματοποίηση του χώρου W ώστε να χρησιμοποιηθούν από τον Σύνθετο αλγόριθμο. Τέλος το k είναι 10 ενώ ο q επιλέχτηκε από το k=3 k-skyband.

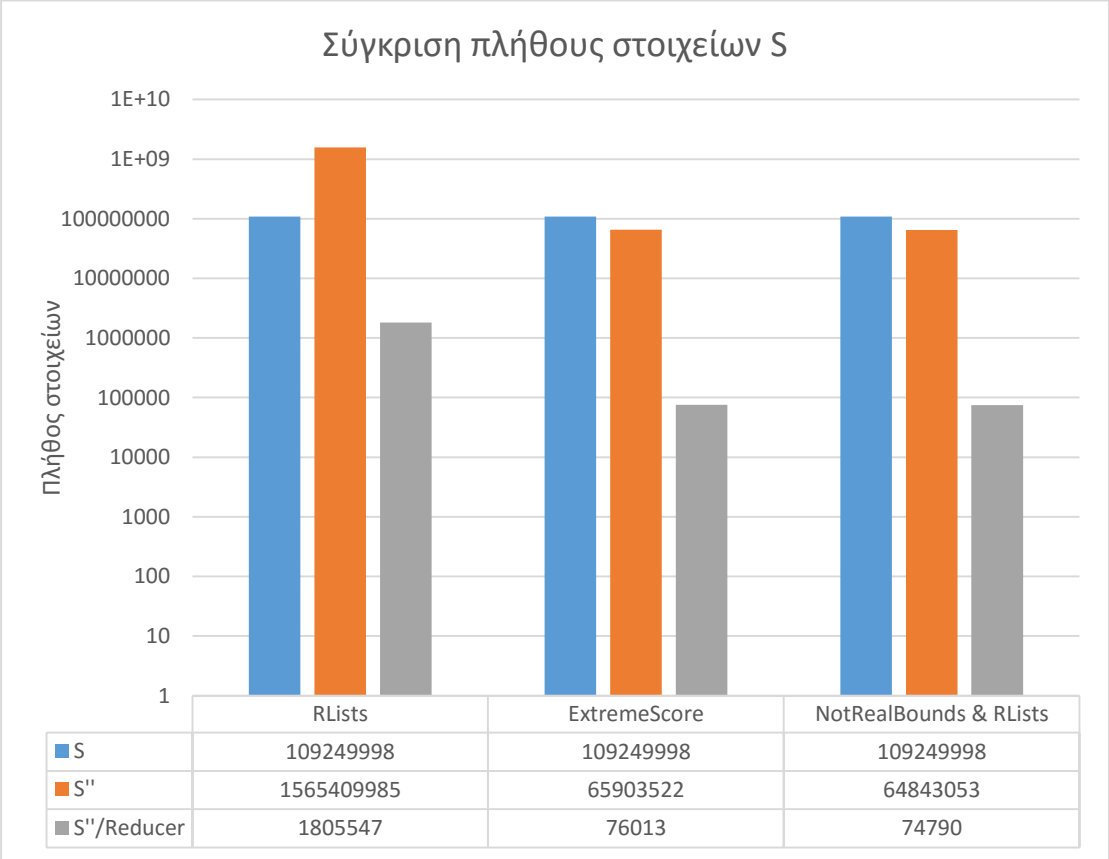
Στην Εικόνα 30 παρουσιάζονται οι χρόνοι εκτέλεσης. Όπως παρατηρείται η αποδοτικότητα των αλγορίθμων περικοπής των στοιχείων S παίζει καθοριστικό ρόλο για τον συνολικό χρόνο εκτέλεσης, αυτό φαίνεται από το γεγονός ότι στην καλύτερη περίπτωση ο αλγόριθμος εκτελείται σχεδόν 12 φορές γρηγορότερα. Πιο αναλυτικά παρατηρείται ότι ο πιο αποδοτικός αλγόριθμος περικοπής στοιχείων S είναι ο ExtremeScore & R-Lists, έπειτα ο ExtremeScore και τέλος ο R-Lists. Από τους συνολικούς χρόνους εκτέλεσης μπορεί να συμπεράνει κανείς ότι ο αλγόριθμος ExtremeScore & R-Lists εκτελείται ταχύτερα, ενώ ο αλγόριθμος R-Lists έχει σοβαρό πρόβλημα στην αποδοτικότητά του. Δηλαδή ο συνδυασμός των δύο αλγορίθμων ExtremeScore & R-Lists αποδίδει το καλύτερο αποτέλεσμα. Όσο αναφορά τον χρόνο εκτέλεσης της Map φάσης παρατηρείται ότι ο R-Lists είναι ο ακριβότερος αλγόριθμος καθώς απαιτεί την εξέταση αρκετών στοιχείων κάθε φορά, αντίθετα ο αλγόριθμος ExtremeScore & R-Lists είναι ο αποδοτικότερος με μικρές διαφορές σχετικά με τον αλγόριθμο ExtremeScore. Ομοίως οι χρόνοι εκτέλεσης της Reduce φάσης παρουσιάζουν μεγάλες διαφορές καθώς η Reduce φάση του αλγορίθμου R-Lists καθυστερεί 42 φορές περισσότερο από αυτή του ExtremeScore & R-Lists. Επιπλέον παρατηρείται ότι η Reduce φάση του ExtremeScore & R-Lists είναι 20% ταχύτερη από αυτή του αλγορίθμου ExtremeScore. Αυτό συμβαίνει γιατί όσο καλύτερη περικοπή δεδομένων κάνει ένας αλγόριθμος κατά την Map φάση τόσο λιγότερα Top-k ερωτήματα θα εκτελεστούν στην Reduce φάση, κάτι που εξοικονομεί όπως φαίνεται πολύτιμο χρόνο. Τέλος όσο αναφορά τους χρόνους εκτέλεσης των αλγορίθμων περικοπής στοιχείων S είναι εμφανές ότι ο αλγόριθμος ExtremeScore & R-Lists έχει τους μικρότερους χρόνους απόκρισης, έπειτα ακολουθεί ο ExtremeScore και τέλος ο R-Lists με πολύ μεγάλη διαφορά, λόγο των πολλαπλών ελέγχων που ασκεί.




| | RLists | ExtremeScore | ExtremeScore & RLists |
|---|---|---|---|
| Συνολικός χρόνος | 62648 | 6018 | 5416 |
| Χρόνος Map φάσης | 23061 | 15966 | 15240 |
| Χρόνος Reduce φάσης | 379435 | 10920 | 8967 |
| Χρόνος pruning W | 4752 | 526 | 461 |

*Εικόνα 30: Στην εικόνα αυτή παρουσιάζονται οι χρόνοι εκτέλεσης των Map και Reduce φάσεων, οι χρόνοι εκτέλεσης των αλγορίθμων περικοπής στοιχείων S καθώς και οι συνολικοί χρόνοι εκτέλεσης προκειμένου να συγκριθεί η αποδοτικότητα των αλγορίθμων περικοπής στοιχείων S που υπάρχουν. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα των χρόνων εκτέλεσης είναι λογαριθμική.*

Στην Εικόνα 31 παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις της MapReduce διαδικασίας. Πιο αναλυτικά το γράφημα παρουσιάζει το πλήθος των στοιχείων S που δέχονται ως είσοδο τα MapReduce Jobs, το πλήθος των στοιχείων S που μεταφέρονται από την Map προς την Reduce φάση και τον μέσο όρο του πλήθους των στοιχείων S ανά Reducer. Αξίζει να σημειωθεί ότι ο άξονας που περιγράφει το πλήθος των στοιχείων είναι σε λογαριθμική κλίμακα για λόγους παρουσίασης. Παρατηρείται ότι το πλήθος των στοιχείων S που δέχονται οι αλγόριθμοι ως είσοδο είναι το ίδιο, καθώς χρησιμοποιούνται τα ίδια σύνολα δεδομένων για την σύγκριση των αλγορίθμων. Όσο αναφορά το πλήθος των στοιχείων S που μεταφέρονται στο δίκτυο από την Map στην Reduce φάση, ο αλγόριθμος ExtremeScore & R-Lists παρουσιάζει την καλύτερη συμπεριφορά μεταφέροντας τα λιγότερα δεδομένα. Πιο αναλυτικά ο αλγόριθμος ExtremeScore μεταφέρει περίπου 1.000.000 περισσότερα στοιχεία από τον ExtremeScore & R-Lists, ενώ ο αλγόριθμος R-Lists μεταφέρει περίπου 1.500.566.932 από τον ExtremeScore & R-Lists. Αυτές οι διαφορές είναι πολύ σημαντικές καθώς κατά την Reduce φάση θα κληθούν να υπολογιστούν τα Top-k ερωτήματα περισσότερων στοιχείων με αποτέλεσμα να υπάρχουν καθυστερήσεις στην εκτέλεση του αλγορίθμου αλλά και υπερφόρτωση του δικτύου. Διατηρώντας σταθερό το πλήθος των Reducers και στα 3 πειράματα ο μέσος φόρτος των Reducers με στοιχεία S ακολουθεί ανάλογη συμπεριφορά με αυτή του πλήθους των στοιχείων S που μεταφέρονται προς τους Reducers. Τέλος μπορεί να συμπεράνει κανείς ότι ο αλγόριθμος ExtremeScore & R-Lists πραγματοποιεί την περικοπή των περισσότερων στοιχείων.




| | RLists | ExtremeScore | NotRealBounds & RLists |
|---|---|---|---|
| S | 109249998 | 109249998 | 109249998 |
| S'' | 1565409985 | 65903522 | 64843053 |
| S''/Reducer | 1805547 | 76013 | 74790 |

*Εικόνα 31: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

## 7.4 Σύγκριση Απλοϊκού και Σύνθετου αλγορίθμου

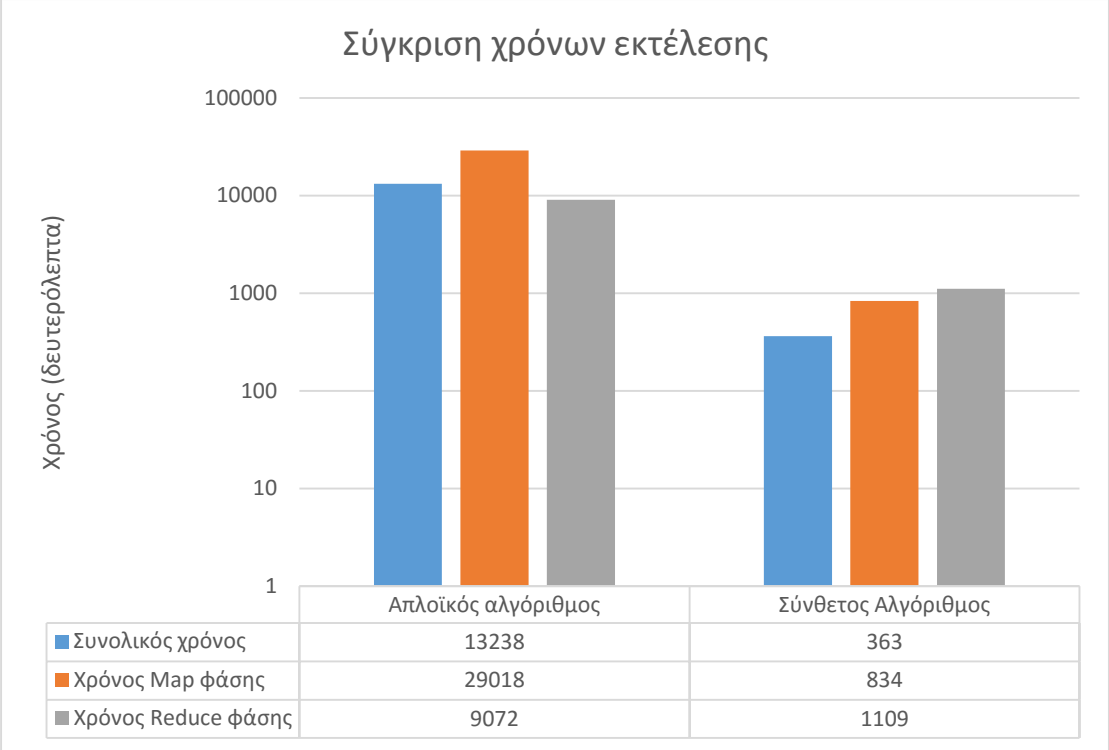
Στην εργασία αυτή προτάθηκαν δύο αλγόριθμοι ο Απλοϊκός αλγόριθμος και ο Σύνθετος αλγόριθμος. Όπως παρουσιάστηκε στα προηγούμενα κεφάλαια, ο Απλοϊκός αλγόριθμος αναπαριστά μια απλή λύση του προβλήματος με κάποιες εμφανείς ευπάθειες ενώ αντίθετα ο Σύνθετος αλγόριθμος υπόσχεται την επίλυση του προβλήματος. Σε αυτό το σημείο κρίνεται αναγκαία η σύγκριση μεταξύ των δύο αυτών αλγορίθμων καθώς πρέπει να εξεταστούν οι διαφορές τους.

Εφόσον κρίνεται αναγκαία η σύγκριση των αλγορίθμων αυτών, πρέπει να οριστούν τα μέτρα σύγκρισης τα οποία θα αναδείξουν τον καλύτερο αλγόριθμο. Ο χρόνος εκτέλεσης του αλγορίθμου θα αποτελέσει το βασικότερο κριτήριο σύγκρισης των αλγορίθμων. Επιπλέον το πλήθος των στοιχείων που μεταφέρονται στο δίκτυο τόσο των S όσο και των W, θα παρατηρηθούν καθώς ο φόρτος του δικτύου αποτελεί μια σημαντική παράμετρο για ένα MapReduce Job. Τέλος δεν μπορούν να παραληφθούν οι χρόνοι των Mappers και των Reducers καθώς κρίνεται αναγκαία η παρατήρηση της κατανομής του συνολικού χρόνου σε αυτές τις δύο φάσεις.

Σε αυτό το σημείο πρέπει να καθοριστούν τα πειράματα τα οποία θα εκτελεστούν για την ανάδειξη του αποδοτικότερου αλγορίθμου. Αξίζει να σημειωθεί ότι όλοι οι παράγοντες επηρεάζουν τα αποτελέσματα των αλγορίθμων. Παρόλα αυτά εκτελέστηκε μόνο ένα πείραμα καθώς λόγω της ευπάθειας του απλού αλγορίθμου αδυνατεί να διαχειριστεί μεγάλα σύνολα δεδομένων, πιο αναλυτικά παρουσιάζεται η ευπάθεια αυτή κατά την περιγραφή του αλγορίθμου. Λόγω αυτής της ευαισθησίας ο Απλοϊκός αλγόριθμος επιλύει περιορισμένα το πρόβλημα και όχι συνολικά, έτσι αναδεικνύεται ως καλύτερος αλγόριθμος ο Σύνθετος. Αξίζει όμως έστω για κάποια σύνολα δεδομένων να πραγματοποιηθεί μια σύγκριση των αλγορίθμων αυτών.

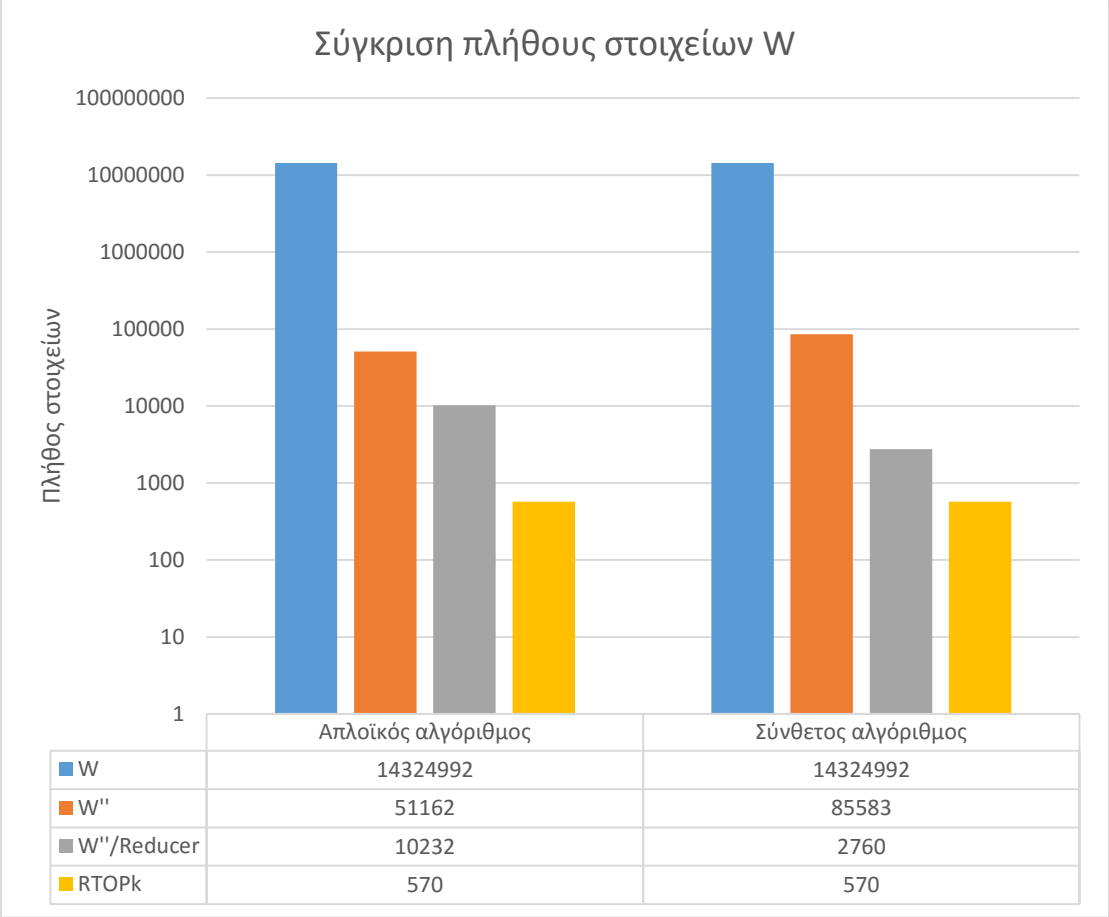
Για το πείραμα το οποίο εκτελέστηκε χρησιμοποιήθηκαν σύνολα δεδομένων 4-διαστάσεων. Όπου το σύνολο δεδομένων S ακολουθεί uniform (UN) κατανομή, το μέγεθος του είναι 500 MB και το πλήθος των στοιχείων του είναι 10,9 εκατομμύρια. Επιπλέον το πλέγμα $G_s$ το οποίο δημιουργήθηκε για το σύνολο δεδομένων S, ώστε να χρησιμοποιηθεί από τον Σύνθετο αλγόριθμο αποτελείται από 256 κελιά. Όσο αναφορά το σύνολο δεδομένων W ακολουθεί uniform (UN) κατανομή, το μέγεθός του είναι 500 MB και το πλήθος των στοιχείων του είναι 14,3 εκατομμύρια. Επιπλέον δημιουργήθηκαν 31 ομάδες από την τμηματοποίηση του χώρου W ώστε να χρησιμοποιηθούν από τον Σύνθετο αλγόριθμο. Σε αυτό το σημείο αξίζει να σημειωθεί μια διαφορά μεταξύ των δύο αλγορίθμων, ο Σύνθετος αλγόριθμος εκτελείται με 31 Reducers ενώ ο Απλοϊκός αλγόριθμος με 5 Reducers. Αυτό συμβαίνει γιατί το πλήθος των Reducers στον Σύνθετο αλγόριθμο καθορίζεται από το πλήθος των ομάδων των weighting vectors. Το πλήθος των Reducers όσο αναφορά τον Απλοϊκό αλγόριθμο επιλέχτηκε να είναι 5 ως βέλτιστο, καθώς περισσότεροι Reducers θα τον καθυστερούσαν δραματικά λόγω της ευπάθειάς του να στείλει όλο το S'' σε όλους τους Reducers.

Στην Εικόνα 32 παρουσιάζονται οι χρόνοι εκτέλεσης των δύο αλγορίθμων. Μετρήθηκαν οι χρόνοι εκτέλεσης των Map και Reduce φάσεων καθώς και οι συνολικοί χρόνοι εκτέλεσης. Οι χρόνοι αυτοί παρουσιάζονται σε δευτερόλεπτα ενώ αξίζει να σημειωθεί ότι ο άξονας του γραφήματος είναι σε λογαριθμική κλίμακα για την καλύτερη αναπαράσταση των δεδομένων. Όπως μπορεί να παρατηρηθεί ο Σύνθετος Αλγόριθμος έχει πολύ μικρότερους χρόνους σε όλα τα πεδία. Πιο αναλυτικά ο συνολικός χρόνος εκτέλεσης του Απλοϊκού αλγορίθμου είναι περίπου 3 ώρες και 40 λεπτά δηλαδή είναι 36 φορές αργότερος από τον Σύνθετο αλγόριθμο που εκτελέστηκε σε μόλις 6 λεπτά περίπου. Αυτό δείχνει ότι οι ιδιότητες που αξιοποιεί ο Σύνθετος αλγόριθμος αποφέρουν καλύτερα αποτελέσματα από τις ιδιότητες που αξιοποιεί ο Απλοϊκός αλγόριθμος. Πιο συγκεκριμένα ο χρόνος εκτέλεσης των Mappers είναι πολύ μικρότερος στον Σύνθετο αλγόριθμο, αυτό οφείλεται στο γεγονός ότι ο Απλοϊκός αλγόριθμος υπολογίζει τα τοπικά Reverse Top-k αποτελέσματα προκειμένου να κάνει περικοπή των στοιχείων $w \in W$. Η πράξη αυτή κοστίζει σε χρόνο αντίθετα με την περικοπή των στοιχείων αυτών βάση του πλέγματος $G_S$ όπως γίνεται στον Σύνθετο αλγόριθμο. Επιπλέον ο Απλοϊκός αλγόριθμος κατά την Map φάση πραγματοποιεί περικοπή των στοιχείων $p \in S$ βάση της κυριαρχίας του q, αντίθετα ο Σύνθετος αλγόριθμος χρησιμοποιεί την περικοπή βάση κυριαρχίας του q συνδυαστικά με τον αλγόριθμο Not Real Bound & R-Lists. Είναι προφανές ότι ο Σύνθετος αλγόριθμος αφιερώνει περισσότερο χρόνο για την περικοπή των στοιχείων αυτών αλλά το τελικό αποτέλεσμα είναι εμφανώς καλύτερο. Κατά την Reduce φάση πραγματοποιείται ο υπολογισμός των τελικών αποτελεσμάτων. Παρόλο που και οι δύο αλγόριθμοι εκτελούν τον RTA για τον υπολογισμό του Reverse Top-k ερωτήματος, ο Σύνθετος αλγόριθμος εκτελείται πάλι γρηγορότερα. Αυτό οφείλεται σε τρεις λόγους, πρώτον στο γεγονός ότι σταμάτησαν 15 από τους 31 Reducers λόγω του αλγορίθμου Not Real Bound & R-Lists γλιτώνοντας τον υπολογισμό αρκετών Reverse Top-k ερωτημάτων. Δεύτερον στο γεγονός ότι γίνεται καλύτερη περικοπή των στοιχείων $p \in S$, οπότε όσα Reverse Top-k υπολογίστηκαν στον Σύνθετο αλγόριθμο είχαν ως είσοδο λιγότερα στοιχεία και τρίτον γιατί πραγματοποιείται επίσης καλύτερη περικοπή των στοιχείων $w \in W$ οπότε υπολογίζονται λιγότερα Top-k ερωτήματα για την απάντηση ενός Reverse Top-k ερωτήματος. Τέλος αξίζει να σημειωθεί η παρατήρηση ότι η Map φάση του Σύνθετου αλγόριθμου απαιτεί λιγότερο χρόνο από την Reduce φάση του αντίθετα με τον Απλοϊκό αλγόριθμο. Αυτό συμβαίνει διότι η εκτέλεση του RTA για μεγαλύτερα σύνολα δεδομένων κοστίζει σε χρόνο και ο Απλοϊκός αλγόριθμος εκτελεί τον RTA στην Map φάση του.



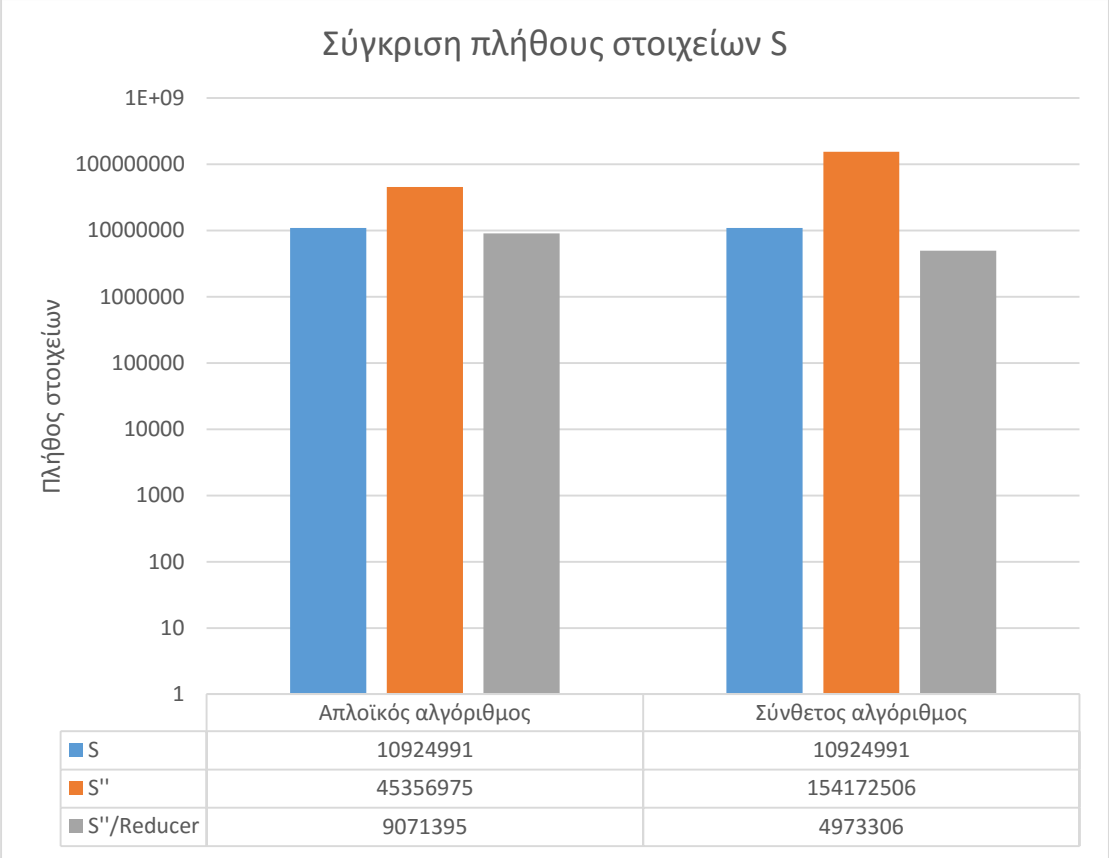

*Εικόνα 32: Στο γράφημα αυτό παρουσιάζονται οι χρόνοι εκτέλεσης των αλγορίθμων σε δευτερόλεπτα. Οι χρόνοι της εκτέλεσης της Map και της Reduce φάσης προκύπτουν από το άθροισμα των χρόνων εκτέλεσης των Mappers και των Reducers αντίστοιχα, για αυτό το άθροισμα τους παρατηρείται να είναι μεγαλύτερο από τον συνολικό χρόνο εκτέλεσης. Όσο αναφορά τον άξονα με τις χρονικές τιμές παρουσιάζεται σε λογαριθμική κλίμακα.*

Στην Εικόνα 33 παρουσιάζονται τα πλήθη των στοιχείων W σε διάφορες φάσεις της MapReduce διαδικασίας. Πιο αναλυτικά το γράφημα παρουσιάζει το πλήθος των στοιχείων W που δέχονται ως είσοδο οι αλγόριθμοι, το πλήθος των στοιχείων W που μεταφέρονται από την Map προς την Reduce φάση, τον μέσο όρο του πλήθους των στοιχείων W ανά Reducer και το πλήθος των τελικών αποτελεσμάτων καθώς και αυτά είναι W στοιχεία. Αξίζει να σημειωθεί ότι ο άξονας που περιγράφει το πλήθος των στοιχείων είναι σε λογαριθμική κλίμακα για λόγους παρουσίασης. Παρατηρείται ότι το πλήθος των στοιχείων W που δέχονται οι αλγόριθμοι ως είσοδο είναι το ίδιο, όπως και η έξοδος (RTOPk) των αλγορίθμων αυτών. Επιπλέον ο Απλοϊκός αλγόριθμος πραγματοποιεί καλύτερη περικοπή των στοιχείων W κατά την πρώτη φάση αλλά όπως αναφέρθηκε προηγουμένως συνοδεύεται με υψηλό χρονικό κόστος αυτός ο τρόπος περικοπής που χρησιμοποιείται. Ακόμα αυτή η διαφορά δεν είναι τόσο σημαντική καθώς το πλήθος των στοιχείων W που καλείται να επεξεργαστεί κάθε Reducer είναι μικρότερο στον Σύνθετο αλγόριθμο βελτιώνοντας έτσι σημαντικά τον χρόνο εκτέλεσης των Reducers παρόλο που είναι περισσότεροι στον Σύνθετο αλγόριθμο. Αξίζει να σημειωθεί σε αυτό το σημείο ότι μια τυχόν αύξηση των Reducers στον Απλοϊκό αλγόριθμο θα έχει ως αποτέλεσμα την χειροτέρευση όλων των άλλων παραμέτρων λόγω της ευπάθειάς του. Λόγω των παραπάνω προκύπτει ότι ο Σύνθετος αλγόριθμος καλείται να επεξεργαστεί λιγότερα στοιχεία W ανά Reducer.




*Εικόνα 33: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 34 παρουσιάζονται τα πλήθη των στοιχείων S σε διάφορες φάσεις της MapReduce διαδικασίας. Πιο αναλυτικά το γράφημα παρουσιάζει το πλήθος των στοιχείων S που δέχονται ως είσοδο οι αλγόριθμοι, το πλήθος των στοιχείων S που μεταφέρονται από την Map προς την Reduce φάση και τον μέσο όρο του πλήθους των στοιχείων S ανά Reducer. Αξίζει να σημειωθεί ότι ο άξονας που περιγράφει το πλήθος των στοιχείων είναι σε λογαριθμική κλίμακα για λόγους παρουσίασης. Παρατηρείται ότι το πλήθος των στοιχείων S που δέχονται οι αλγόριθμοι ως είσοδο είναι το ίδιο, το ίδιο συμβαίνει και για το πλήθος S που δέχεται ο κάθε Mapper για αυτό και δεν παρουσιάζεται. Ο Απλοϊκός αλγόριθμος μεταφέρει λιγότερα δεδομένα στο δίκτυο κάτι που είναι λογικό καθώς έχει μόλις 5 Reducers αντί για 31 που έχει ο Σύνθετος αλγόριθμος. Σε αυτό το σημείο αξίζει να σημειωθεί ότι αν ο Απλοϊκός αλγόριθμος κατάφερνε να έχει 31 Reducers θα μετέφερε περίπου 2 φορές περισσότερα στοιχεία από ότι μεταφέρει ο Σύνθετος αλγόριθμος. Από αυτό προκύπτει ότι η μέθοδος περικοπής δεδομένων που πραγματοποιεί ο Σύνθετος αλγόριθμος είναι καλύτερη από αυτή του Απλοϊκού αλγορίθμου. Επιπλέον παρατηρείται ότι επειδή ο Σύνθετος αλγόριθμος έχει στην διάθεσή του περισσότερους Reducers το πλήθος των στοιχείων κατανέμεται καλύτερα και έτσι ο Σύνθετος αλγόριθμος έχει λιγότερα στοιχεία S ανά Reducer από τον Απλοϊκό αλγόριθμο. Το γεγονός αυτό λειτουργεί υπέρ του Σύνθετου αλγορίθμου καθώς κάθε Reducer θα εκτελείται γρηγορότερα.




*Εικόνα 34: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Από τα παραπάνω μπορεί κανείς να καταλήξει στο συμπέρασμα ότι ο Σύνθετος αλγόριθμος είναι καλύτερος από τον Απλοϊκό αλγόριθμο. Η δυνατότητα του Σύνθετου αλγορίθμου να λειτουργεί με πιο κατανεμημένο τρόπο χρησιμοποιώντας ταχύτερους αλγορίθμους περικοπής δεδομένων είναι τα χαρακτηριστικά τα οποία του δίνουν το προβάδισμα αυτό. Επειδή ο Απλοϊκός αλγόριθμος έχει περιορισμένες δυνατότητες κατανεμημένης επεξεργασίας (δεν μπορεί να αυξηθεί σε μεγάλο βαθμό το πλήθος των Reducers) δεν θα μελετηθεί περαιτέρω. Στην συνέχει αντίθετα θα ακολουθήσει μια ανάλυση του Σύνθετου αλγορίθμου.

## 7.5 Ανάλυση ευαισθησίας του Σύνθετου αλγορίθμου

Προηγουμένως υπήρξε μία σύγκριση του Απλοϊκού αλγόριθμου με τον Σύνθετο αλγόριθμο όπου αναδείχθηκε ο Σύνθετος αλγόριθμος ως καλύτερος. Αυτό το οποίο δεν παρουσιάστηκε ως τώρα είναι η συμπεριφορά του Σύνθετου αλγορίθμου. Πρέπει να μελετηθεί αν μπορεί να χειριστεί μεγάλα σύνολα δεδομένων, ποικίλα ερωτήματα, αν έχει ευπάθειες και αν ναι να αναγνωριστούν, δηλαδή να μελετηθεί αν προσφέρει μια ικανοποιητική λύση για το πρόβλημα της κατανεμημένης επεξεργασίας ενός Reverse Top-k ερωτήματος. Έτσι κρίνεται αναγκαία η περαιτέρω πειραματική αξιολόγηση του Σύνθετου αλγόριθμου. Εφόσον κρίνεται απαραίτητη η πειραματική αξιολόγηση του Σύνθετου αλγορίθμου πρέπει να οριστούν τα μέτρα αξιολόγησης του αλγορίθμου. Ο χρόνος εκτέλεσης του Map Reduce Job κρίνεται ως το σημαντικότερο μέτρο αξιολόγησης, ενώ δεν μπορούν να παραληφθούν οι χρόνοι των Map και Reduce φάσεων. Ακόμα τα δεδομένα τα οποία μεταφέρονται στο δίκτυο αποτελούν κριτήριο αξιολόγησης, καθώς από αυτά μπορεί να παρατηρηθεί ο φόρτος του δικτύου καθώς και την επιρροή αυτού στον συνολικό χρόνο εκτέλεσης. Επιπλέον πρέπει να μετρηθούν τα στοιχεία τα οποία περικόπτονται από την κυριαρχία βάση του q, τον αλγόριθμο ExtremeScore, τον R-Lists και τον αλγόριθμο που στηρίζεται στον προσεγγιστικό υπολογισμό της κατάταξης του q βάση του $G_S$. Τέλος απαραίτητη κρίνεται η μέτρηση των Reducers των οποίων σταματάνε χάρη στον αλγόριθμο ExtremeScore.

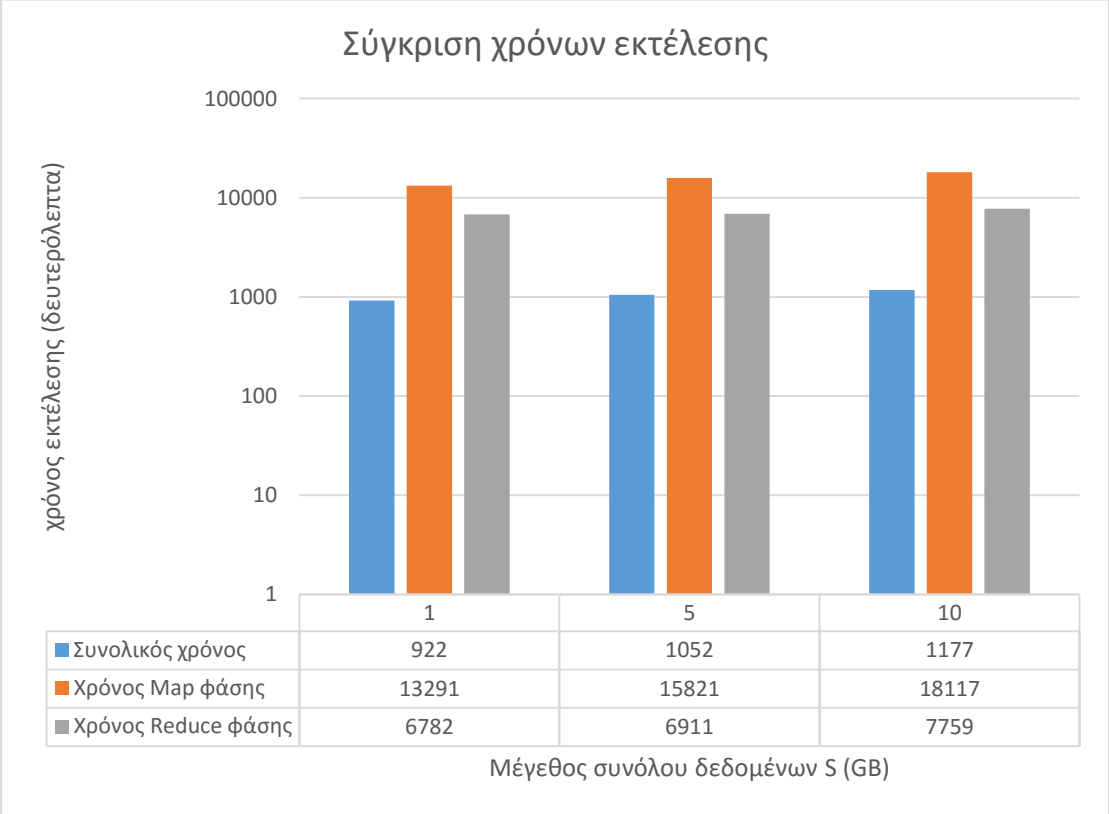
Σε αυτό το σημείο πρέπει να καθοριστούν τα πειράματα τα οποία θα εκτελεστούν για την αξιολόγηση του Σύνθετου αλγορίθμου. Αξίζει να σημειωθεί ότι όλοι οι παράγοντες επηρεάζουν τα αποτελέσματα και την συμπεριφορά του αλγορίθμου. Οπότε θα πραγματοποιηθούν πειράματα στα οποία θα μεταβάλλονται όλες οι παράμετροι προκειμένου να μελετηθεί η συμπεριφορά του αλγορίθμου στην εκάστοτε μεταβολή. Λόγω του μεγάλου πλήθος των παραμέτρων καθώς και της προηγούμενης ανάγκης επιλέχτηκε να τεθεί ένα πείραμα ως βασικό και έπειτα σε κάθε πείραμα να μεταβάλλεται και από μία παράμετρος. Πιο αναλυτικά οι παράμετροι καθώς και το πλήθος τους παρουσιάζονται στην Εικόνα 35, σημειώνεται ότι έχει σημειωθεί το βασικό πείραμα με έντονα γράμματα. Με αυτό τον τρόπο δίνεται η δυνατότητα να παρατηρηθούν οι συνέπειες τις μεταβολής κάθε παραμέτρου. Σημειώνεται ότι το πλέγμα $G_S$ δημιουργείται σε όλα τα πειράματα χωρίζοντας την κάθε διάσταση σε 10 τμήματα. Επιπλέον υπάρχουν 867 ομάδες weighting vector. Στην συνέχεια θα αναλυθούν τα πειράματα βάση της μεταβλητής που μεταβάλλεται κάθε φορά.

| S | Κατανομή S | W | k | Διαστάσεις |
|---|---|---|---|---|
| 1 GB | UNIFORM | 5 GB | 10 | 4 |
| **5 GB** | **UNIFORM** | **5 GB** | **10** | **4** |
| 10 GB | UNIFORM | 5 GB | 10 | 4 |
| 5 GB | CORELATED | 5 GB | 10 | 4 |
| 5 GB | ANTICORELATED | 5 GB | 10 | 4 |
| 5 GB | UNIFORM | 1 GB | 10 | 4 |
| 5 GB | UNIFORM | 10 GB | 10 | 4 |
| 5 GB | UNIFORM | 5 GB | 50 | 4 |
| 5 GB | UNIFORM | 5 GB | 100 | 4 |
| 5 GB | UNIFORM | 5 GB | 10 | 2 |
| 5 GB | UNIFORM | 5 GB | 10 | 6 |

*Εικόνα 35: Σε αυτόν τον πίνακα παρουσιάζονται οι παράμετροι των πειραμάτων που θα εκτελεστούνε για την αξιολόγηση του Σύνθετου αλγορίθμου. Σημειώνεται με έντονα γράμματα το βασικό πείραμα.*

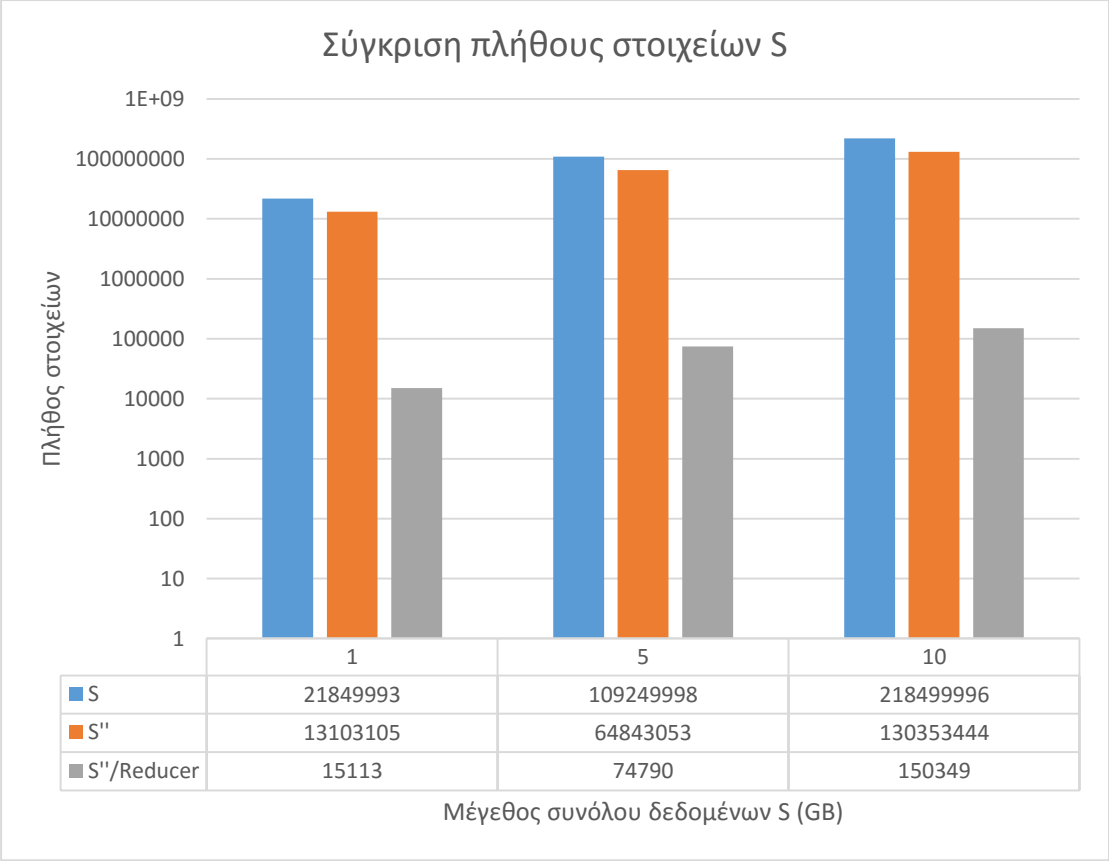
### 7.5.1 Πειράματα μεταβολής μεγέθους S

Σε αυτή την ομάδα πειραμάτων θα μελετηθούν οι συνέπειες που έχει η μεταβολή του συνόλου δεδομένων S για τον Σύνθετο αλγόριθμο. Όσο αναφορά τους συνολικούς χρόνους εκτέλεσης όπως φαίνεται και στην Εικόνα 36 αυξάνονται όσο αυξάνεται το μέγεθος του συνόλου δεδομένων S αλλά οι διαφορές είναι μικρές, καθώς για το σύνολο δεδομένων μεγέθους 1 GB απαιτούνται 15 λεπτά ενώ για το σύνολο δεδομένων μεγέθους 10 GB απαιτούνται 19 λεπτά. Δηλαδή για μία αύξηση μεγέθους του συνόλου δεδομένων 10 φορές απαιτείται μόλις 27% παραπάνω χρόνος εκτέλεσης. Σχετικά με τον χρόνο εκτέλεσης των Map φάσεων παρατηρείται επίσης ότι αυξάνεται όσο αυξάνεται το πλήθος των στοιχείων S, κάτι που είναι λογικό καθώς οι Mappers καλούνται να περικόψουν περισσότερα στοιχεία S, κάτι που μπορεί να παρατηρηθεί στην Εικόνα 38. Ο χρόνος εκτέλεσης των Reducers αυξάνεται κατά τον ίδιο τρόπο παρουσιάζοντας ακόμη μικρότερες διαφορές, αυτό συμβαίνει γιατί καθώς αυξάνεται το σύνολο δεδομένων S τόσο καλύτερη περικοπή δεδομένων γίνεται όπως φαίνεται και στην Εικόνα 39. Έτσι ο κάθε Reducer καλείται να επεξεργαστεί λιγότερα Top-k ερωτήματα.



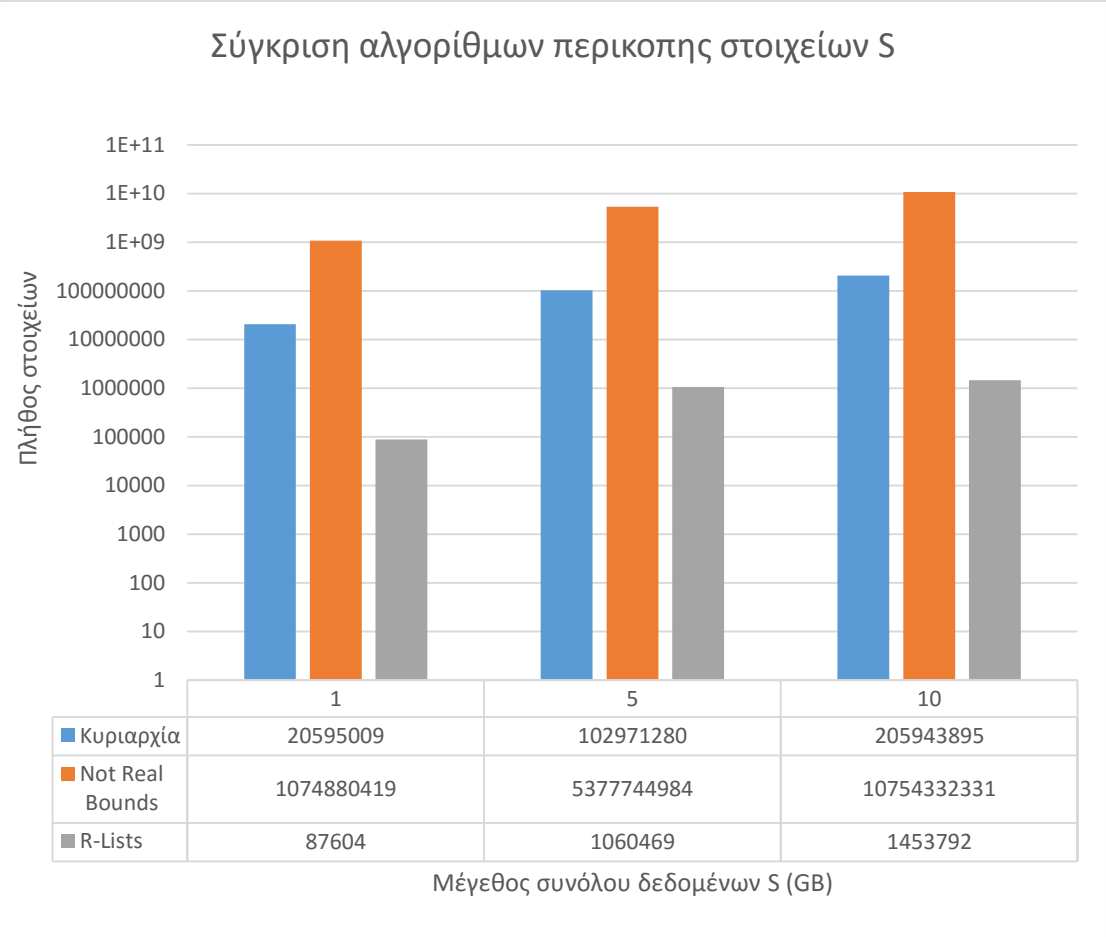

*Εικόνα 36: Σύγκριση χρόνων εκτέλεσης για τα πειράματα κατά τα οποία μεταβάλλεται μόνο το μέγεθος του συνόλου δεδομένων S. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του χρόνου εκτέλεσης είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 37 παρουσιάζει το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία S που δίνονται ως είσοδο στο MapReduce Job αυξάνονται καθώς αυξάνεται το μέγεθος του συνόλου δεδομένων. Έπειτα πραγματοποιείται μια περικοπή των δεδομένων που στέλνονται προς τους Reducers η οποία θα μελετηθεί στην συνέχεια αλλά όπως μπορεί κανείς να δει σε αυτό το σημείο όσο αυξάνεται το πλήθος των στοιχείων S, τόσο στέλνονται πιο πολλά στοιχεία στους Reducers. Διατηρώντας το ίδιο πλήθος Reducers και στα 3 πειράματα είναι λογικό να αυξάνεται ο φόρτος του κάθε Reducer όσο αναφορά τα στοιχεία S όπως φαίνεται στο διάγραμμα. Ο μέσος όρος των στοιχείων S που καλείται να επεξεργαστεί ένας Reducer για το σύνολο δεδομένων 1 GB είναι 10 φορές μικρότερος σχετικά με το πείραμα του συνόλου δεδομένων S στα 10 GB.



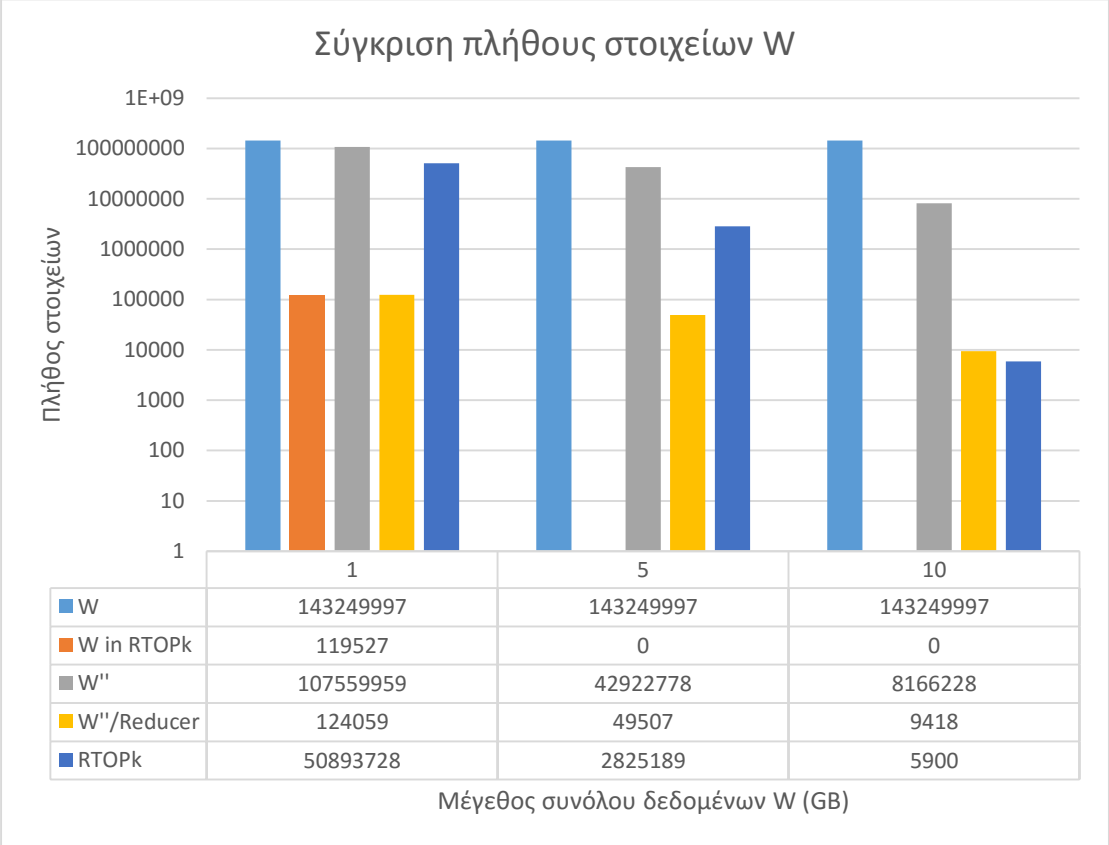

*Εικόνα 37: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 38 παρουσιάζονται οι μετρήσεις σχετικά με την περικοπή των στοιχείων S κατά την Map φάση. Όπως ειπώθηκε προηγουμένως όσο αυξάνεται το σύνολο δεδομένων S τόσο αυξάνονται και τα στοιχεία που στέλνονται προς τους Reducers. Σε αυτό το σημείο παρατηρείται επίσης ότι όσο αυξάνεται το σύνολο δεδομένων S αυξάνεται και το πλήθος των στοιχείων που περικόπτουν οι τρεις αλγόριθμοι. Αρχικά ο αλγόριθμος περικοπής βάση της κυριαρχίας του q περικόπτει περισσότερα δεδομένα όσο αυξάνεται το πλήθος των στοιχείων S, το ποσοστό των στοιχείων των οποίων περικόπτει παραμένει σταθερό και στα 3 πειράματα, κάτι που είναι λογικό καθώς επιλέγεται το q από συγκεκριμένο k-skyband όπως αναφέρθηκε προηγουμένως. Σχετικά με τον αλγόριθμο περικοπής ExtremeScore παρατηρείται επίσης ότι το ποσοστό των στοιχείων των οποίων περικόπτει παραμένει σταθερό και στα 3 πειράματα. Ενώ αντίθετα ο αλγόριθμος R-Lists ενώ αυξάνονται τα δεδομένα που περικόπτει το ποσοστό μειώνεται καθώς για το σύνολο δεδομένων 1 GB περικόπτει περίπου το 8%, για το σύνολο δεδομένων 5 GB περικόπτει περίπου το 2% και για το σύνολο δεδομένων 10 GB περικόπτει περίπου το 1,3%. Από τα παραπάνω παρατηρούνται ότι οι αλγόριθμοι περικοπής ανταποκρίνονται αντιστοίχως στο μέγεθος του συνόλου δεδομένων S, με μόνη διαφορά τον αλγόριθμο R-Lists ο οποίος περικόπτει περισσότερα στοιχεία όσο αυξάνεται το σύνολο δεδομένων S αλλά το ποσοστό των στοιχείων που περικόπτει μειώνεται, δηλαδή γίνεται λιγότερο αποδοτικός όσο μεγαλώνει το σύνολο δεδομένων S.




*Εικόνα 38: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S τα οποία περικόπτονται από κάθε αλγόριθμο. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

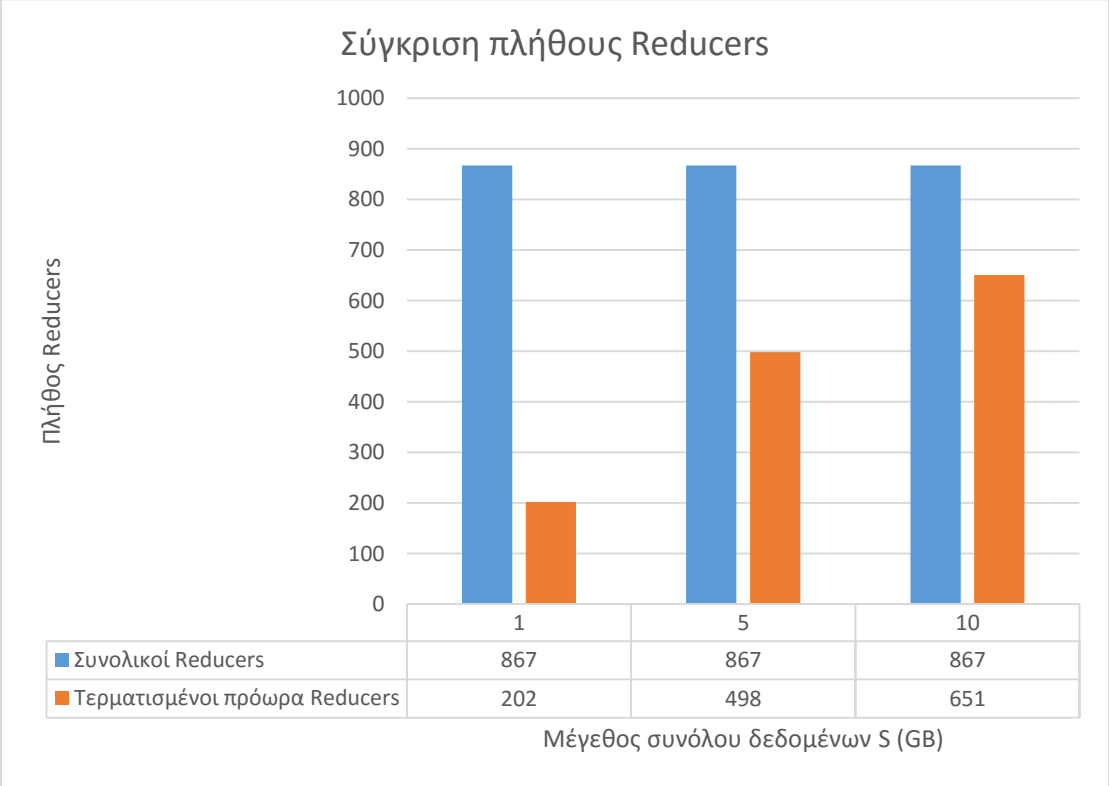
Η Εικόνα 39 παρουσιάζει το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αρχικά παρατηρείται ότι το πλήθος των στοιχείων W εισόδου είναι ίδιο και στις 3 περιπτώσεις καθώς δεν μεταβάλλεται το μέγεθος αυτό. Έπειτα το πλήθος των στοιχείων W για τα οποία είναι γνωστό βάση του πλέγματος $G_S$ από την Map φάση ότι ανήκουν στα τελικά αποτελέσματα είναι μη μηδενικό μόνο για το σύνολο δεδομένων του 1 GB. Αυτό συμβαίνει διότι το q έχει λιγότερους ανταγωνιστές (στοιχεία S) οπότε ανήκει σε περισσότερα Top-k αποτελέσματα κάτι που είναι γνωστό μερικές φορές από το $G_S$. Για τον ίδιο λόγω παρατηρείται και η μείωση των στοιχείων W που στέλνονται προς τους Reducers όσο αυξάνεται το μέγεθος του συνόλου δεδομένων S. Δηλαδή επειδή υπάρχουν περισσότεροι ανταγωνιστές του q το πλέγμα $G_S$ αναγνωρίζει μερικές φορές ότι το q δεν βρίσκεται στα Top-k αποτελέσματα. Τα στοιχεία τα οποία τελικά στέλνονται στους Reducers είναι αυτά τα οποία βάση του πλέγματος $G_S$ δεν μπορούν να αποσαφηνιστούν αν ανήκουν ή όχι στα τελικά Reverse Top-k αποτελέσματα. Βάση του πλέγματος $G_S$ ήταν γνωστό αν ένα στοιχείο W ανήκει ή όχι στα τελικά Reverse Top-k αποτελέσματα για το 25% των στοιχείων στο σύνολο δεδομένων S του 1 GB, για το 70% των στοιχείων στο σύνολο δεδομένων S των 5 GB και για το 94% των στοιχείων στο σύνολο δεδομένων S των 10 GB. Αντίστοιχα παρατηρείται ότι μειώνεται ο μέσος όρος των στοιχείων W που επεξεργάζεται ο κάθε Reducer καθώς και τα τελικά αποτελέσματα. Μπορεί κανείς να καταλήξει στο συμπέρασμα ότι όσο αυξάνεται το σύνολο δεδομένων τόσο αυξάνεται η αποδοτικότητα της περικοπής στοιχείων βάση του πλέγματος $G_S$. Αυτό επίσης οδηγεί στην μείωση των Top-k ερωτημάτων κατά την Reduce φάση.




| | 1 | 5 | 10 |
|---|---|---|---|
| W | 143249997 | 143249997 | 143249997 |
| W in RTOPk | 119527 | 0 | 0 |
| W'' | 107559959 | 42922778 | 8166228 |
| W''/Reducer | 124059 | 49507 | 9418 |
| RTOPk | 50893728 | 2825189 | 5900 |

*Εικόνα 39: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

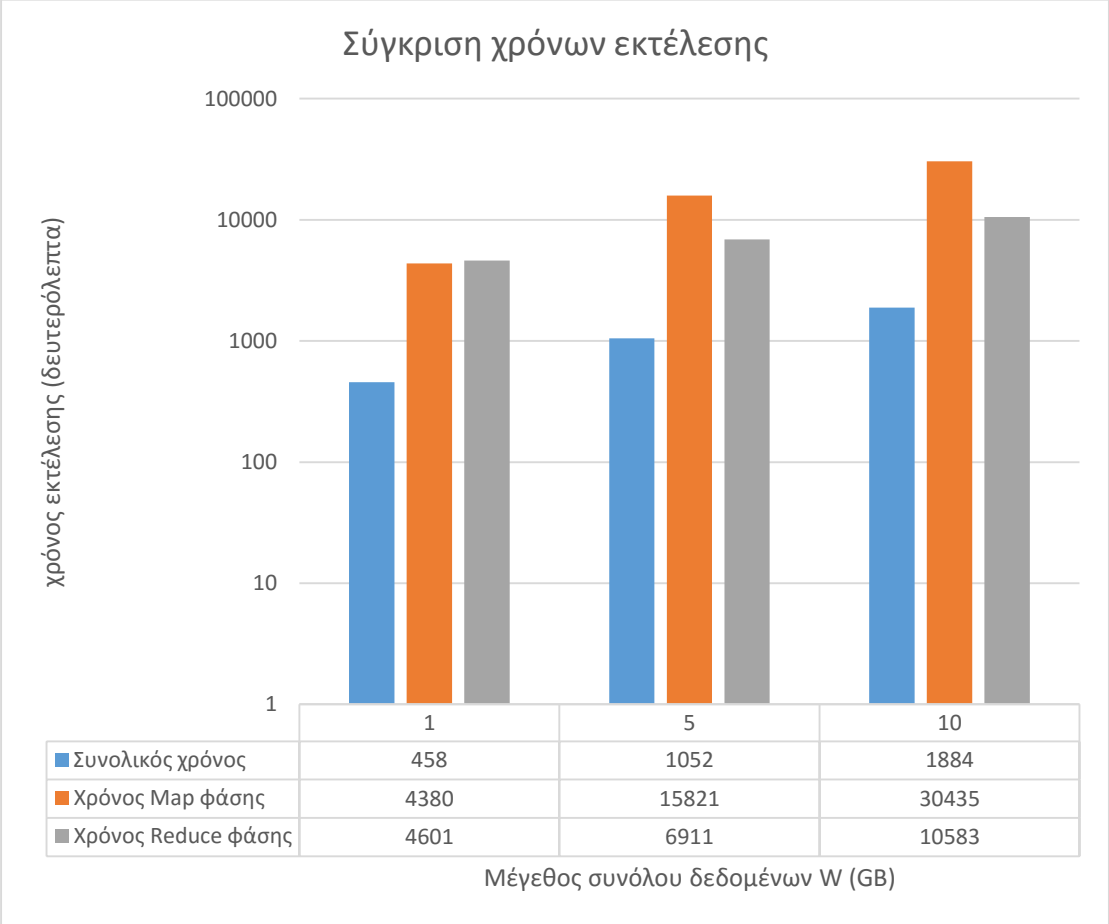
Στην Εικόνα 40 παρουσιάζεται το πλήθος των Reducers που εκτελούνται καθώς και το πλήθος αυτών που σταματούν πρόωρα χάρη στον αλγόριθμο ExtremeScore. Όπως είναι φυσικό το πλήθος των Reducers είναι ίδιο και στα 3 πειράματα. Όσο αναφορά το πλήθος των Reducers που σταματάνε πρόωρα παρατηρείται ότι αυξάνονται όσο αυξάνεται το μέγεθος του συνόλου δεδομένων S. Αυτό οφείλεται στον μεγαλύτερο ανταγωνισμό που έχει το στοιχείο q, οπότε βρίσκονται συχνά περισσότερα από k στοιχεία που έχουν καλύτερη βαθμολογία για όλα τα στοιχεία μιας ομάδας που επεξεργάζεται ένας Reducer, οπότε αυτός σταματά να λειτουργεί. Γλιτώνοντας έτσι τον υπολογισμό πολλών Top-k ερωτημάτων κατά την Reduce φάση.




*Εικόνα 40: Σε αυτό το διάγραμμα παρουσιάζονται τόσο οι συνολικοί Reducers, όσο και αυτοί που σταμάτησαν πρόωρα χάρη στον αλγόριθμο ExtremeScore.*

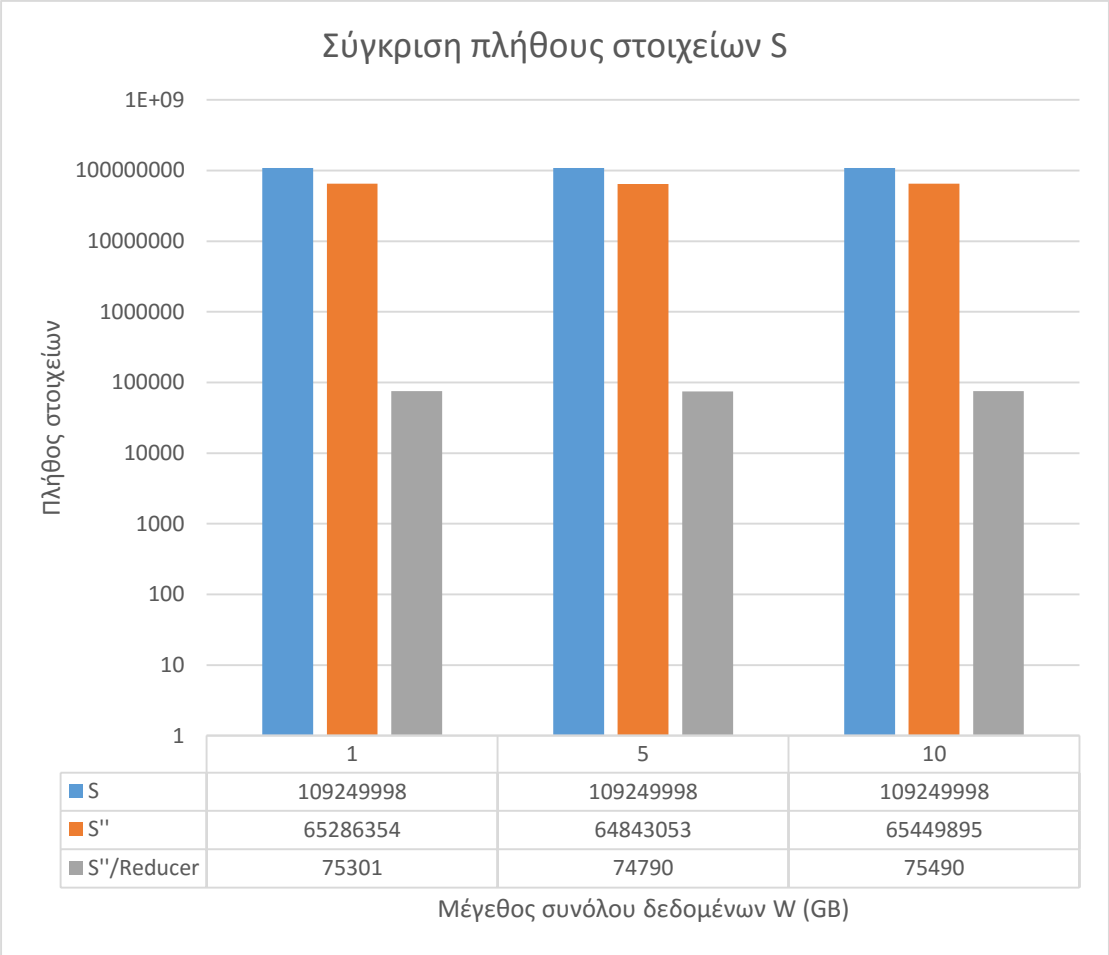
### 7.5.2 Πειράματα μεταβολής μεγέθους W

Σε αυτή την ομάδα πειραμάτων θα μελετηθούν οι συνέπειες που έχει η μεταβολή του συνόλου δεδομένων W για τον Σύνθετο αλγόριθμο. Όσο αναφορά τους συνολικούς χρόνους εκτέλεσης όπως φαίνεται και στην Εικόνα 41 αυξάνονται ανάλογα με το μέγεθος του συνόλου δεδομένων W. Όπως παρατηρείται στο διάγραμμα το πείραμα για το σύνολο δεδομένων του 1 GB απαιτεί περίπου 8 λεπτά ενώ για το πείραμα του συνόλου δεδομένων των 10 GB απαιτεί περίπου 31 λεπτά. Δηλαδή για τον δεκαπλασιασμό του συνόλου δεδομένων W απαιτήθηκε περίπου ο τετραπλάσιος χρόνος για την εκτέλεση του Map Reduce Job. Σχετικά με τον χρόνο εκτέλεσης των Map φάσεων παρατηρείται επίσης ότι αυξάνεται όσο αυξάνεται το πλήθος των στοιχείων W, κάτι που είναι λογικό καθώς οι Mappers καλούνται να περικόψουν περισσότερα στοιχεία W. Η διαδικασία αυτή είναι πιο χρονοβόρα από την περικοπή των στοιχείων S, για αυτό και οι χρόνοι σε αυτή την περίπτωση είναι μεγαλύτεροι. Ο χρόνος εκτέλεσης των Reducers αυξάνεται κατά τον ίδιο τρόπο παρουσιάζοντας όμως μικρότερες διαφορές σε σχέση με την Map φάση, αυτό συμβαίνει γιατί ο κάθε Reducer καλείται να επεξεργαστεί περισσότερα Top-k ερωτήματα.




*Εικόνα 41: Σύγκριση χρόνων εκτέλεσης για τα πειράματα κατά τα οποία μεταβάλλεται μόνο το μέγεθος του συνόλου δεδομένων W. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του χρόνου εκτέλεσης είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 42 παρουσιάζει το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία S που δίνονται ως είσοδο στο MapReduce Job είναι σταθερά καθώς δεν μεταβάλλεται το μέγεθος του συνόλου δεδομένων S. Έπειτα παρατηρούνται ότι τα στοιχεία S που αποστέλλονται στους Reducers έχουν ελάχιστες διαφορές όπως και ο μέσος όρος των στοιχείων S που επεξεργάζεται ένας Reducer. Αυτό οφείλεται όπως αναλύεται και παρακάτω στο ότι τα W στοιχεία δεν επηρεάζουν την περικοπή των στοιχείων S.



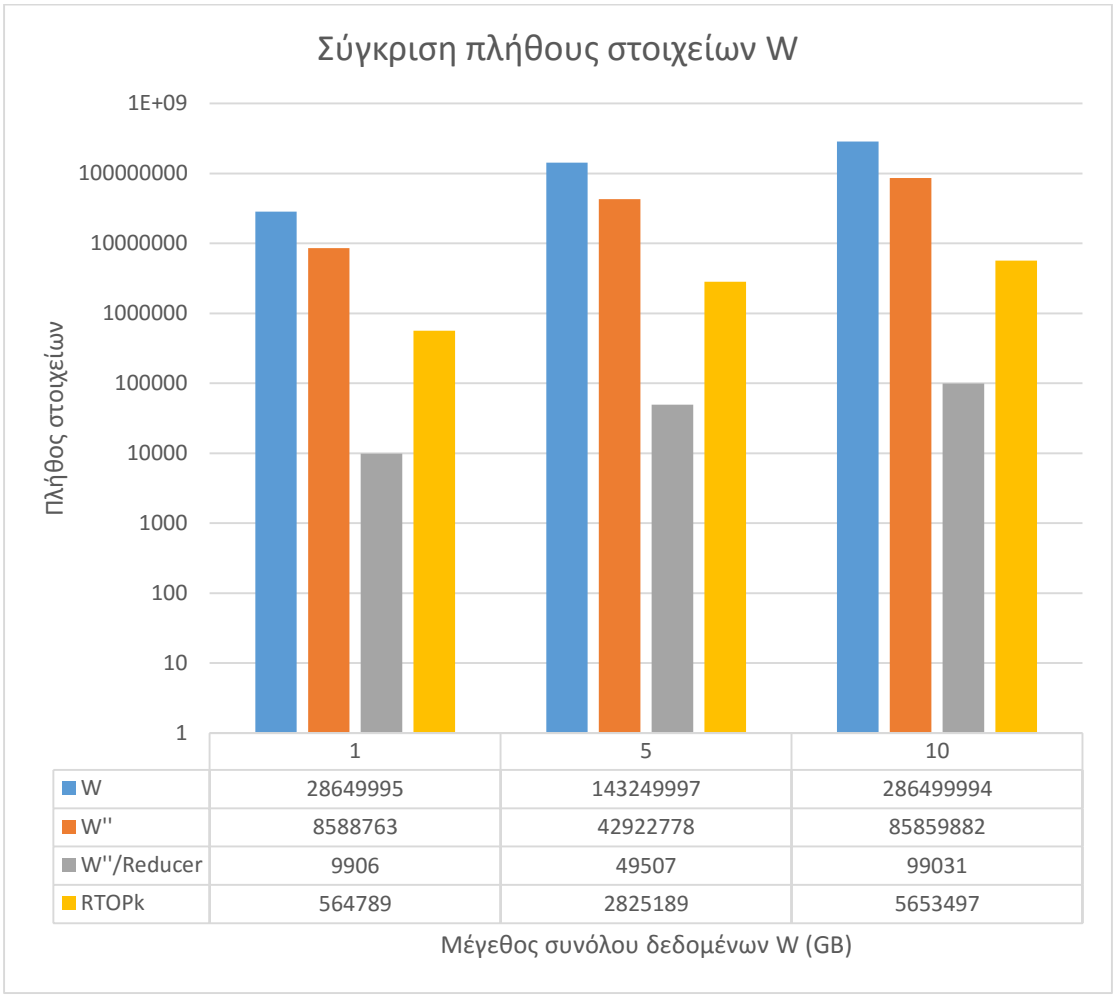

*Εικόνα 42: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 43 παρουσιάζονται οι μετρήσεις σχετικά με την περικοπή των στοιχείων S κατά την Map φάση. Όπως ειπώθηκε προηγουμένως η περικοπή των στοιχείων S δεν επηρεάζεται από τα W στοιχεία. Έτσι τα αποτελέσματα είναι ίδια για τον αλγόριθμο που βασίζεται στην κυριαρχία και τον αλγόριθμο ExtremeScore. Όσο αφορά τον αλγόριθμο R-Lists παρατηρούνται κάποιες μικρές διαφορές. Αυτό οφείλεται στο γεγονός ότι η περικοπή των στοιχείων εξαρτάται από την σειρά των στοιχείων που θα αναγνωστούν. Έτσι επειδή η σειρά αυτή δεν είναι πάντα ίδια υπάρχουν κάποιες μικρές διαφορές στα στοιχεία τα οποία περικόπτει χωρίς όμως να επηρεάζουν το πείραμα.


| | 1 | 5 | 10 |
|---|---|---|---|
| Κυριαρχία | 102971280 | 102971280 | 102971280 |
| Not Real Bounds | 5377744984 | 5377744984 | 5377744984 |
| R-Lists | 617168 | 1060469 | 453627 |

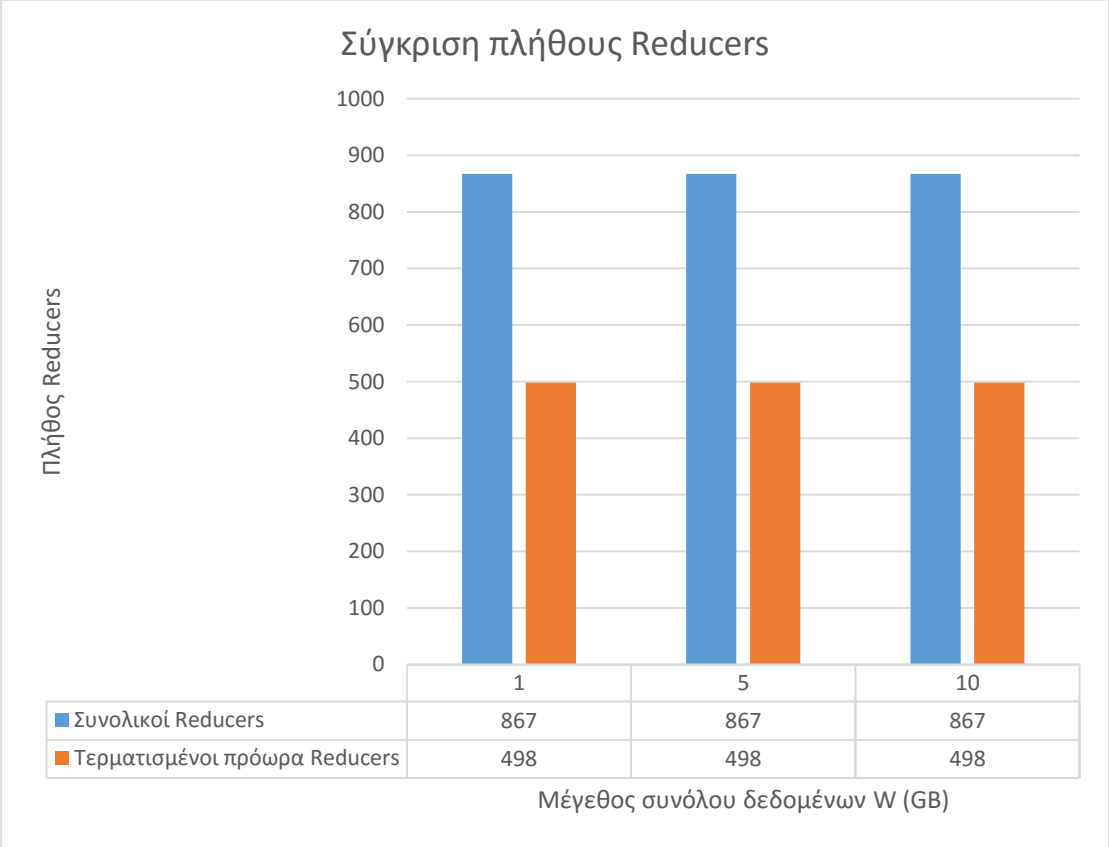
*Εικόνα 43: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S τα οποία περικόπτονται από κάθε αλγόριθμο. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 44 παρουσιάζει το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία W που δίνονται ως είσοδο στο MapReduce Job αυξάνονται καθώς αυξάνεται το μέγεθος του συνόλου δεδομένων. Επίσης τα στοιχεία W τα οποία στέλνονται προς τους Reducers είναι περισσότερα όσο αυξάνεται το μέγεθος του συνόλου δεδομένων W. Κάτι που είναι απολύτως φυσιολογικό καθώς με περισσότερα στοιχεία W αυξάνονται οι πιθανότητες του q να ανήκει σε περισσότερα Top-k αποτελέσματα, αυτό εξάλλου φαίνεται καθαρά από το πλήθος των τελικών Reverse Top-k αποτελεσμάτων που αυξάνεται ανάλογα με το μέγεθος του συνόλου δεδομένων W. Οπότε επειδή διατηρείται σταθερό το πλήθος των Reducers αυξάνεται ο φόρτος του κάθε Reducer βάση του μέσου όρου των στοιχείων W που επεξεργάζεται κάθε Reducer. Αξίζει να σημειωθεί ότι το ποσοστό της περικοπής των στοιχείων W βάση του πλέγματος $G_S$ παραμένει σταθερό στο 70%.




*Εικόνα 44: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

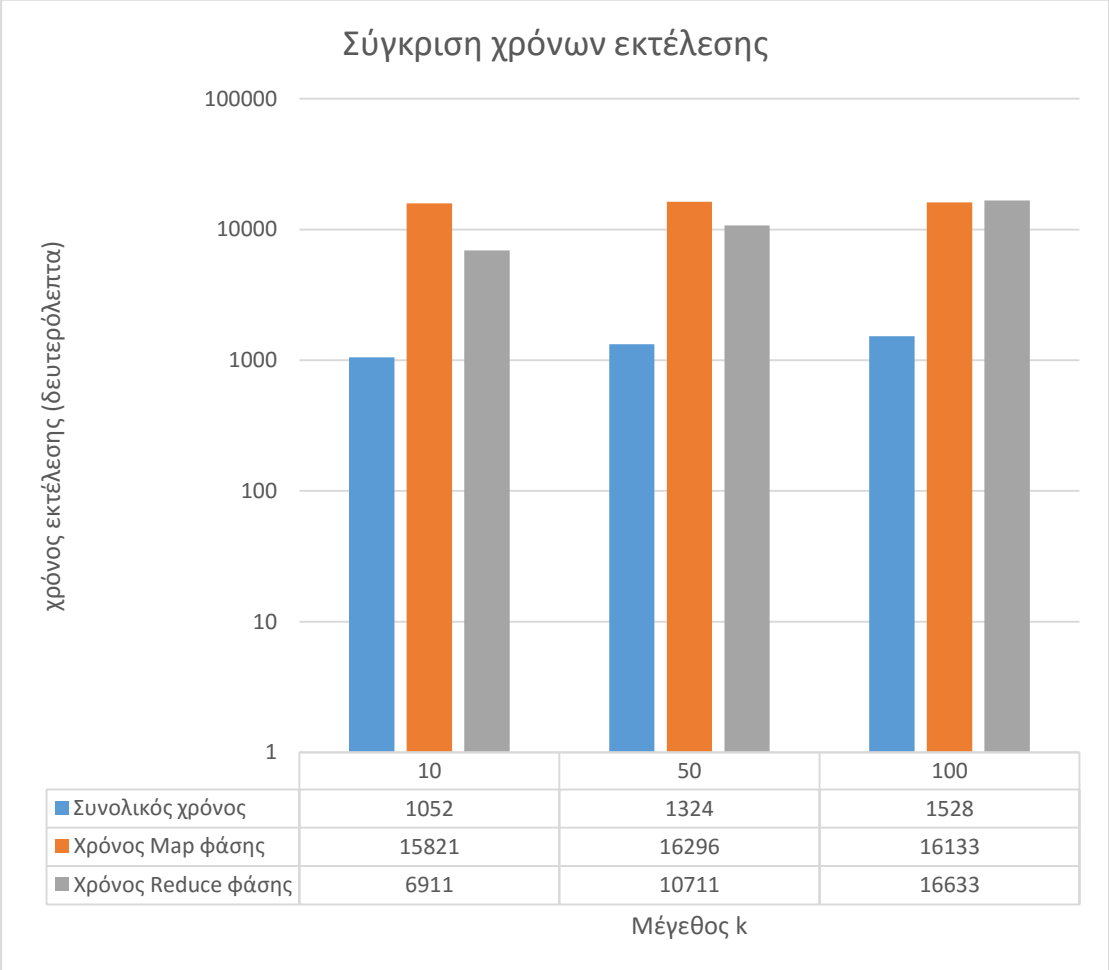
Στην Εικόνα 45 παρουσιάζεται το πλήθος των Reducers που εκτελούνται καθώς και το πλήθος αυτών που σταματούν πρόωρα χάρη στον αλγόριθμο ExtremeScore. Όπως είναι φυσικό το πλήθος των Reducers είναι ίδιο και στα 3 πειράματα. Επιπλέον το πλήθος των Reducers οι οποίοι σταμάτησαν πρόωρα παραμένει πάλι σταθερό καθώς εξαρτάται από τα στοιχεία S και το $G_W$ τα οποία ήταν τα ίδια και στα 3 πειράματα.




*Εικόνα 45: Σε αυτό το διάγραμμα παρουσιάζονται τόσο οι συνολικοί Reducers, όσο και αυτοί που σταμάτησαν πρόωρα χάρη στον αλγόριθμο ExtremeScore.*

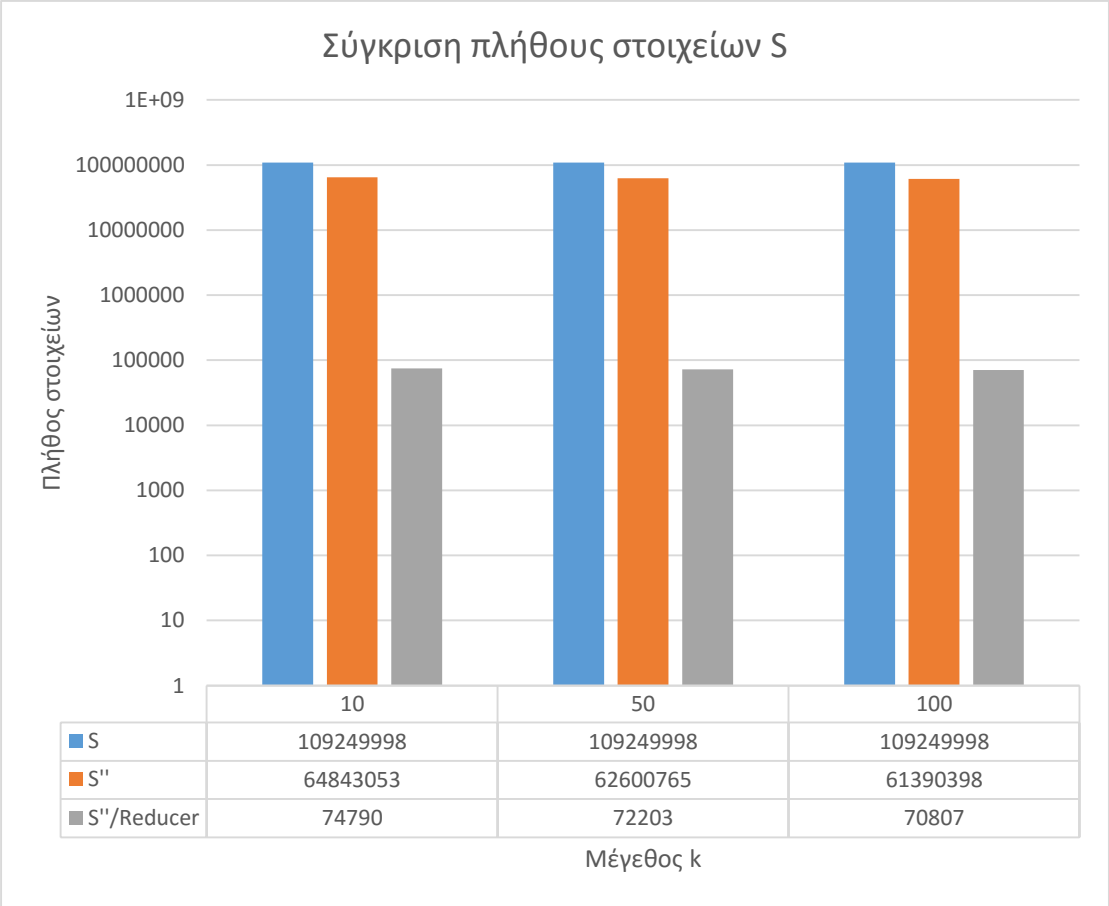
### 7.5.3 Πειράματα μεταβολής μεγέθους k

Σε αυτή την ομάδα πειραμάτων θα μελετηθούν οι συνέπειες που έχει η μεταβολή του συνόλου δεδομένων W για τον Σύνθετο αλγόριθμο. Όσο αναφορά τους συνολικούς χρόνους εκτέλεσης όπως φαίνεται και στην Εικόνα 46 αυξάνονται όσο αυξάνεται και το k. Για k ίσο με 10 χρειάστηκαν περίπου 17 λεπτά για την εκτέλεση του MapReduce Job ενώ για k ίσο με 100 χρειάστηκαν 25 λεπτά περίπου. Πιο συγκεκριμένα όσο αναφορά τους χρόνους εκτέλεσης της Map φάσης αυξάνονται ανάλογα με το k αλλά έχουν μικρές διαφορές μεταξύ τους, αυτό οφείλεται στο γεγονός ότι μόνο ο χρόνος εκτέλεσης του αλγορίθμου R-Lists επηρεάζεται από το k οπότε και καθυστερεί. Σχετικά με τον χρόνο εκτέλεσης των Reducers αυξάνεται ανάλογα με το k και επίσης παρατηρείται μεγαλύτερη διαφορά μεταξύ τους σε σχέση με αυτούς της Map φάσης. Αυτό συμβαίνει γιατί ο RTA αλγόριθμος που εκτελείται κατά την Reduce φάση καθυστερεί όσο αυξάνεται το k.




*Εικόνα 46: Σύγκριση χρόνων εκτέλεσης για τα πειράματα κατά τα οποία μεταβάλλεται μόνο το μέγεθος του k. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του χρόνου εκτέλεσης είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 47 παρουσιάζει το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία S που δίνονται ως είσοδο στο MapReduce Job είναι σταθερά καθώς δεν μεταβάλλεται το μέγεθος του συνόλου δεδομένων S. Όσο αναφορά το πλήθος των στοιχείων S το οποίο στέλνεται στους Reducers για επεξεργασία μειώνεται όσο αυξάνεται το k αλλά οι διαφορές είναι μικρές. Αυτό συμβαίνει χάρη στον αλγόριθμο R-Lists όπου επειδή κρατά μεγαλύτερο buffer (k σε πλήθος) περικόπτει αποδοτικότερα τα δεδομένα. Επίσης επειδή το πλήθος των Reducers παραμένει σταθερό και στα 3 πειράματα, μειώνεται και ο μέσος όρος στοιχείων S που επεξεργάζεται κάθε Reducer όσο αυξάνεται το k. Αυτό οδηγεί σε εξοικονόμηση χρόνου κατά την Reduce φάση επειδή υπάρχουν λιγότερα στοιχεία S αλλά εξοικονομείται χρόνος και από την μεταφορά δεδομένων πάνω από το δίκτυο.



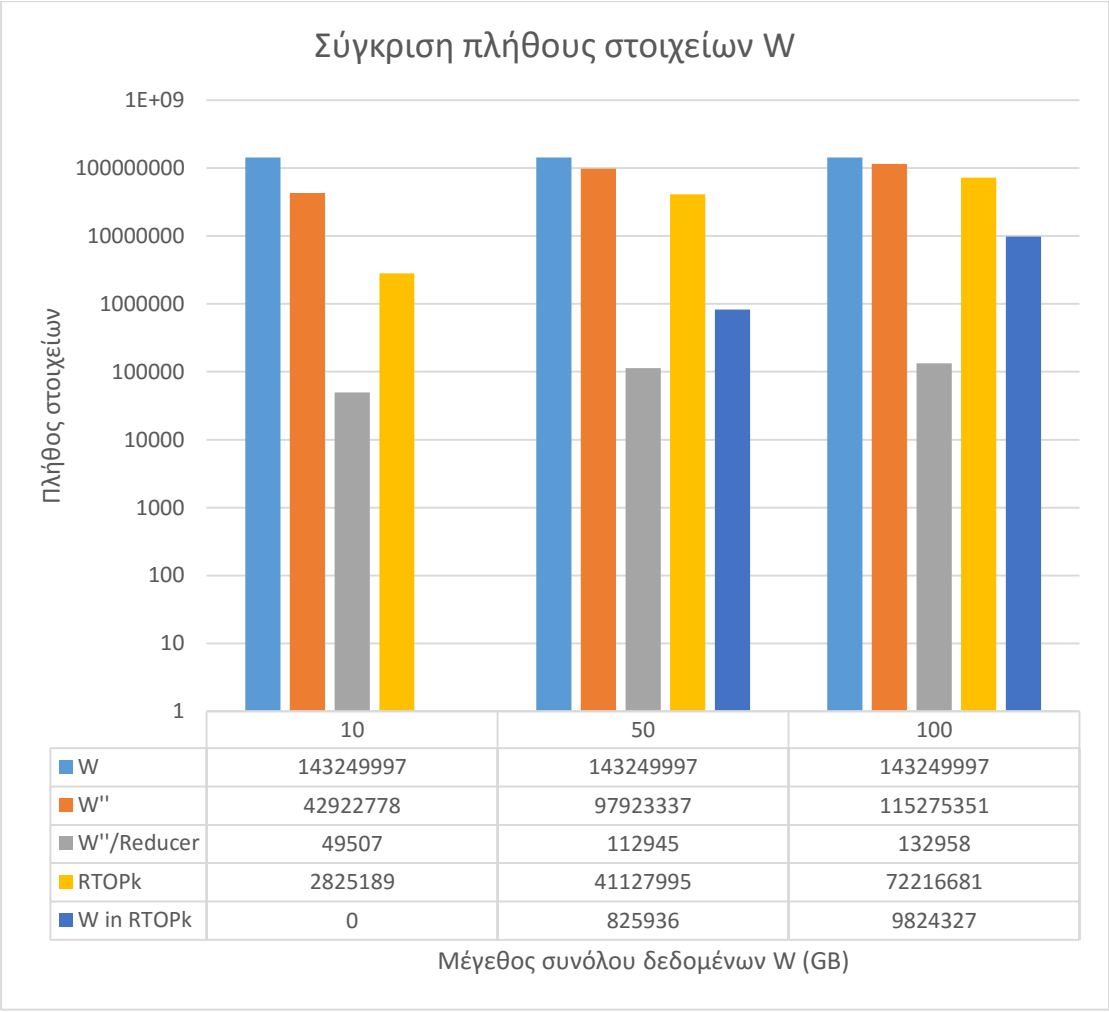

| | 10 | 50 | 100 |
|---|---|---|---|
| S | 109249998 | 109249998 | 109249998 |
| S'' | 64843053 | 62600765 | 61390398 |
| S''/Reducer | 74790 | 72203 | 70807 |

*Εικόνα 47: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 48 παρουσιάζονται οι μετρήσεις σχετικά με την περικοπή των στοιχείων S κατά την Map φάση. Παρατηρείται ότι ο αλγόριθμος που στηρίζεται στην κυριαρχία του q και ο αλγόριθμος ExtremeScore έχουν τα ίδια αποτελέσματα καθώς δεν επηρεάζονται από την μεταβολή του k στην Map φάση. Αντίθετα ο αλγόριθμος R-Lists επηρεάζεται από το k και πιο συγκεκριμένα όσο μεγαλώνει το k τόσο αποδοτικότερη περικοπή στοιχείων πραγματοποιεί. Αυτό οφείλεται στο γεγονός ότι κρατά ένα buffer k-στοιχείων, όσο μεγαλύτερος είναι αυτός ο buffer μπορεί να κρίνει καλύτερα ένα στοιχείο S. Έτσι όσο αυξάνεται το k αυξάνεται και το πλήθος που περικόπτει ο R-Lists αλγόριθμος, οι διαφορές που παρατηρούνται φυσικά είναι μικρές.

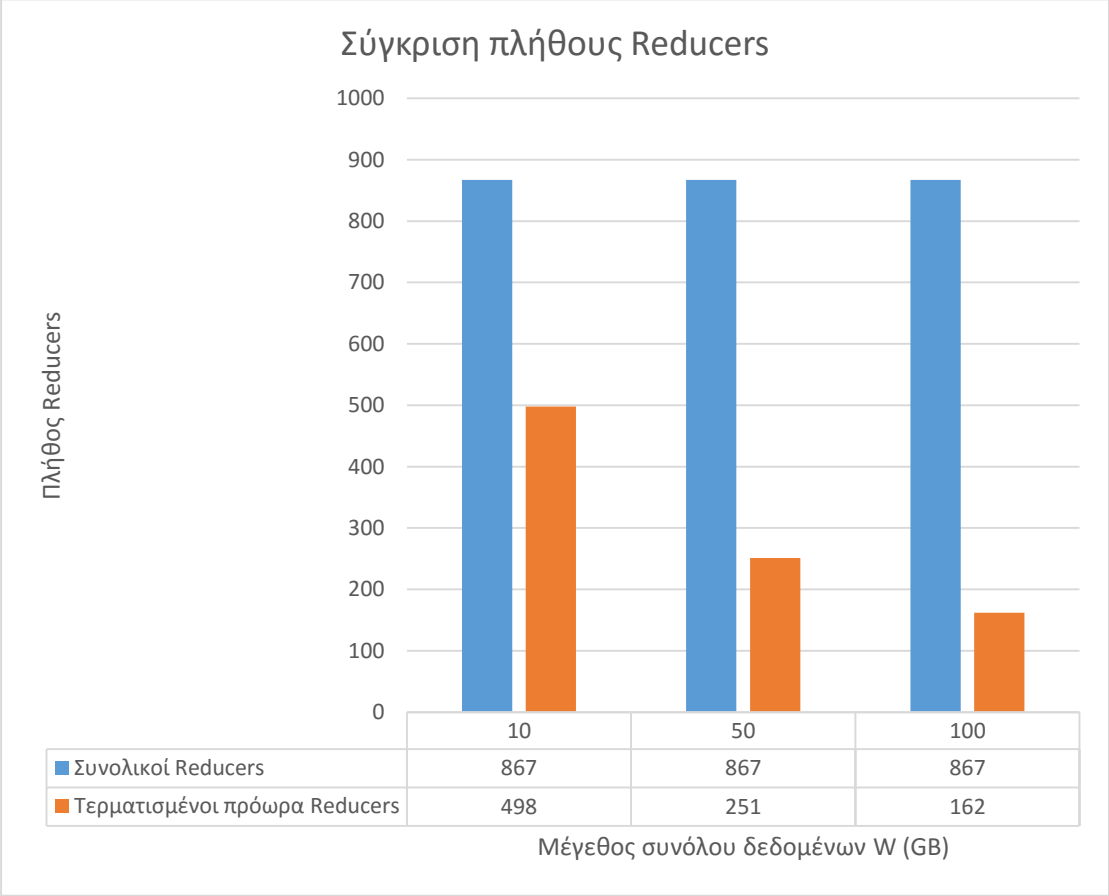

*Εικόνα 48: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S τα οποία περικόπτονται από κάθε αλγόριθμο. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 49 παρουσιάζει το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία W που δίνονται ως είσοδο στο MapReduce Job είναι σταθερά καθώς δεν μεταβάλλεται το μέγεθος του συνόλου δεδομένων. Από την άλλη τα στοιχεία W τα οποία στέλνονται προς τους Reducers για επεξεργασία αυξάνονται. Αυτό συμβαίνει γιατί όσο αυξάνεται το k τόσο αυξάνεται η πιθανότητα του q να ανήκει σε περισσότερα Top-k αποτελέσματα. Αυτό φαίνεται και από το τελικό Reverse Top-k αποτέλεσμα το οποίο αυξάνεται ανάλογα με το k. Έτσι ο μέσος όρος στοιχείων W που επεξεργάζεται κάθε Reducer αυξάνεται αναλόγως. Κάτι τέτοιο κανονικά θα αύξανε κατακόρυφα τον υπολογισμό των Top-k ερωτημάτων κατά την Reduce φάση αλλά κάτι τέτοιο μετριάζεται χάρη στο πλέγμα $G_S$ το οποίο ανακαλύπτει από την Map φάση στοιχεία W τα οποία ανήκουν στα τελικά Reverse Top-k αποτελέσματα και έτσι δεν υπολογίζεται το Top-k τους στην Reduce φάση αλλά γράφονται απευθείας στα αποτελέσματα. Πιο συγκεκριμένα για k ίσο με 100 ανακαλύπτεται από την Map φάση περίπου το 14% των τελικών αποτελεσμάτων, ενώ για k ίσο με 50 αυτός ο αριθμός μειώνεται στο 2% και για k ίσο με 0 δεν ανακαλύπτονται τέτοια στοιχεία.




*Εικόνα 49: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 50 παρουσιάζεται το πλήθος των Reducers που εκτελούνται καθώς και το πλήθος αυτών που σταματούν πρόωρα χάρη στον αλγόριθμο ExtremeScore. Όπως είναι φυσικό το πλήθος των Reducers είναι ίδιο και στα 3 πειράματα. Ο αλγόριθμος ExtremeScore επηρεάζεται κατά την Reduce φάση από το k, καθώς πρέπει να βρει k στοιχεία τα οποία πρέπει να έχουν καλύτερη βαθμολογία από το q για όλα τα στοιχεία W μιας ομάδας προκειμένου να σταματήσει ο Reducer ο οποίος επεξεργάζεται την ομάδα αυτή. Έτσι όσο αυξάνεται το k τόσο μειώνονται οι πιθανότητες να βρεθούν k τέτοια στοιχεία. Για τον λόγο αυτό παρατηρείται και η μείωση των Reducers που σταματούν πρόωρα κατά την αύξηση του μεγέθους k.

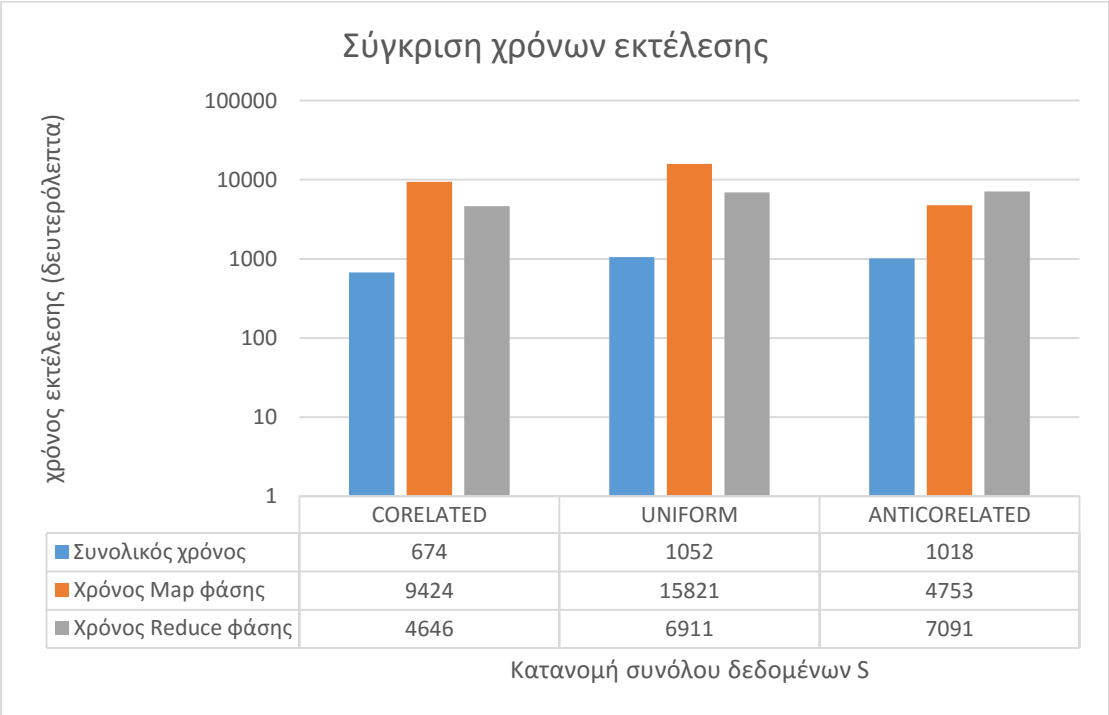



*Εικόνα 50: Σε αυτό το διάγραμμα παρουσιάζονται τόσο οι συνολικοί Reducers, όσο και αυτοί που σταμάτησαν πρόωρα χάρη στον αλγόριθμο ExtremeScore.*

### 7.5.4 Πειράματα μεταβολής κατανομής S

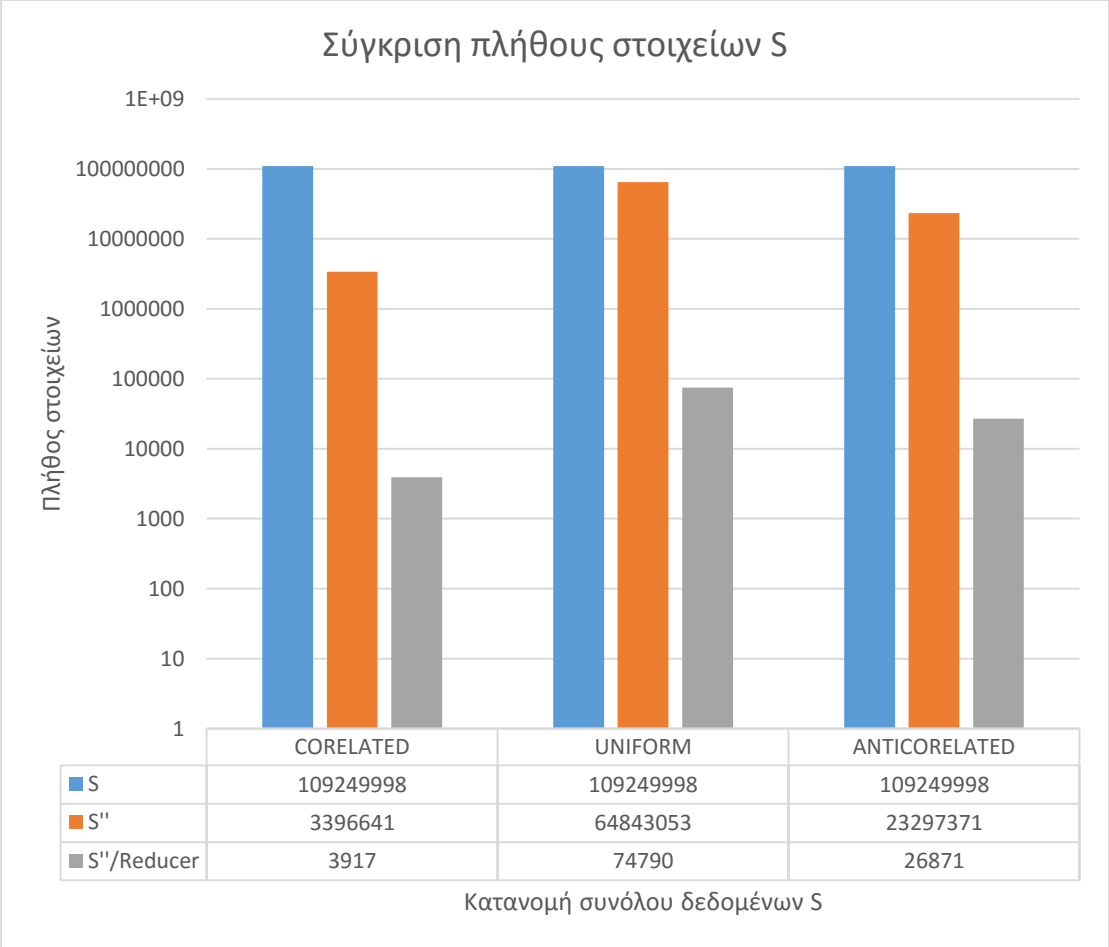
Σε αυτή την ομάδα πειραμάτων θα μελετηθούν οι συνέπειες που έχει η κατανομή του συνόλου δεδομένων S για τον Σύνθετο αλγόριθμο. Διεξήχθησαν πειράματα με τις κατανομές correlated και anticorrelated, όσο αναφορά το πείραμα σχετικά με την correlated κατανομή αναμένονται καλύτερα αποτελέσματα καθώς το q θα κυριαρχεί έναντι πολλών στοιχείων S. Αντίθετα στο πείραμα που αφορά την anticorrelated κατανομή αναμένονται να γίνει περικοπή λιγότερων στοιχείων και ο αλγόριθμος να παρουσιάσει καθυστερήσεις. Όσο αναφορά τους συνολικούς χρόνους εκτέλεσης όπως φαίνεται και στην Εικόνα 51 το πείραμα σχετικά με την correlated κατανομή παρουσίασε τους μικρότερους χρόνους εκτέλεσης. Πιο αναλυτικά εκτελέστηκε 7 λεπτά γρηγορότερα από το πείραμα σχετικά με την uniform κατανομή, ενώ 6 λεπτά γρηγορότερα από το πείραμα με την anticorrelated κατανομή. Αντίστοιχα οι χρόνοι των Map και των Reduce φάσεων μειώθηκαν αναλόγως. Η μείωση αυτή μπορεί να εξηγηθεί από την καλύτερη περικοπή που γίνεται τόσο στα S όσο και στα W στοιχεία, καθώς και από το πλήθος των Reducers που σταμάτησαν πρόωρα. Περισσότερες λεπτομέρειες για την περικοπή των δεδομένων θα δοθούν στην συνέχεια.

Όσο αναφορά τους συνολικούς χρόνους εκτέλεσης όπως φαίνεται και στην Εικόνα 51 το πείραμα σχετικά με την anticorrelated κατανομή δεν παρουσίασε τους μεγαλύτερους χρόνους όπως αναμενόταν. Αντίθετα εκτελέστηκε περίπου 1’ γρηγορότερα από το πείραμα σχετικά με την uniform κατανομή. Η μείωση αυτή οφείλεται στο γεγονός ότι πραγματοποιείται καλύτερη πρόβλεψη των στοιχείων W για το αν ανήκουν στα τελικά αποτελέσματα κατά την Map φάση βάση του $G_S$. Επίσης ενδιαφέρον παρουσιάζει το γεγονός ότι ο χρόνος εκτέλεσης της Map φάσης είναι μικρότερος από τον χρόνο εκτέλεσης της Reduce φάσης. Αυτό οφείλεται στο γεγονός ότι δεν υπάρχουν πρόωρα τερματισμένοι Reducers, καθώς και στο γεγονός ότι αυξάνεται η πιθανότητα για ένα W στοιχείο να ανήκει στα τελικά αποτελέσματα οπότε εκτελούνται περισσότερα Top-k ερωτήματα κατά την Reduce φάση.



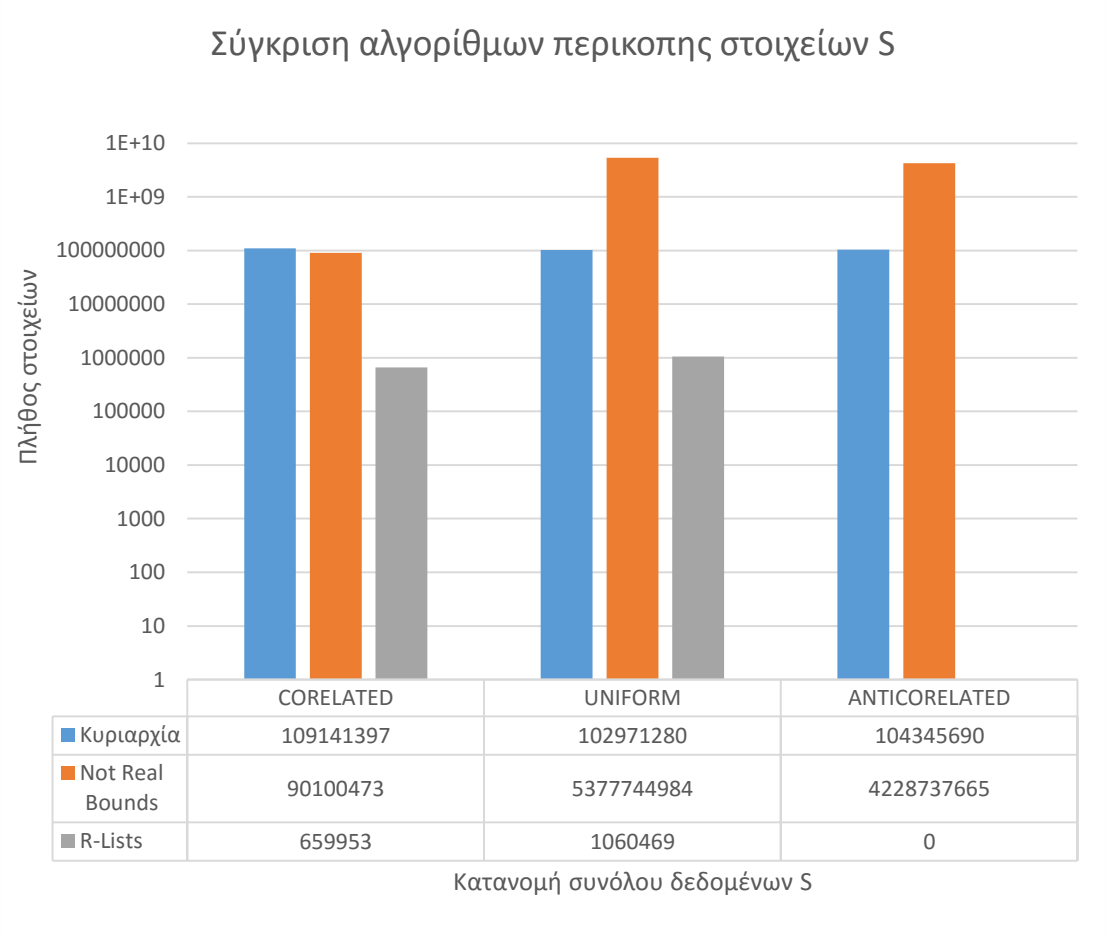

*Εικόνα 51: Σύγκριση χρόνων εκτέλεσης για τα πειράματα κατά τα οποία μεταβάλλεται μόνο η κατανομή του συνόλου δεδομένων S. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του χρόνου εκτέλεσης είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 52 παρουσιάζει το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία S που δίνονται ως είσοδο στο MapReduce Job είναι σταθερά καθώς δεν μεταβάλλεται το μέγεθος του συνόλου δεδομένων S. Όσο αναφορά τα στοιχεία S που στέλνονται προς τους Reducers, το πείραμα με την correlated κατανομή στέλνει τα λιγότερα στοιχεία με μεγάλη διαφορά σχετικά με τα άλλα δύο πειράματα κάτι που οφείλεται στην περικοπή των στοιχείων S βάση της κυριαρχίας του q. Αντίθετα το πείραμα σχετικά με την anticorrelated κατανομή δεν στέλνει τα περισσότερα στοιχεία αλλά αυτό με την uniform κατανομή, αξίζει να σημειωθεί ότι η διαφορά είναι επίσης μεγάλη. Διατηρώντας το πλήθος των Reducers σταθερό για όλα τα πειράματα ο μέσος όρος των στοιχείων S που καλείται να επεξεργαστεί κάθε Reducer ακολουθεί αντίστοιχη συμπεριφορά.



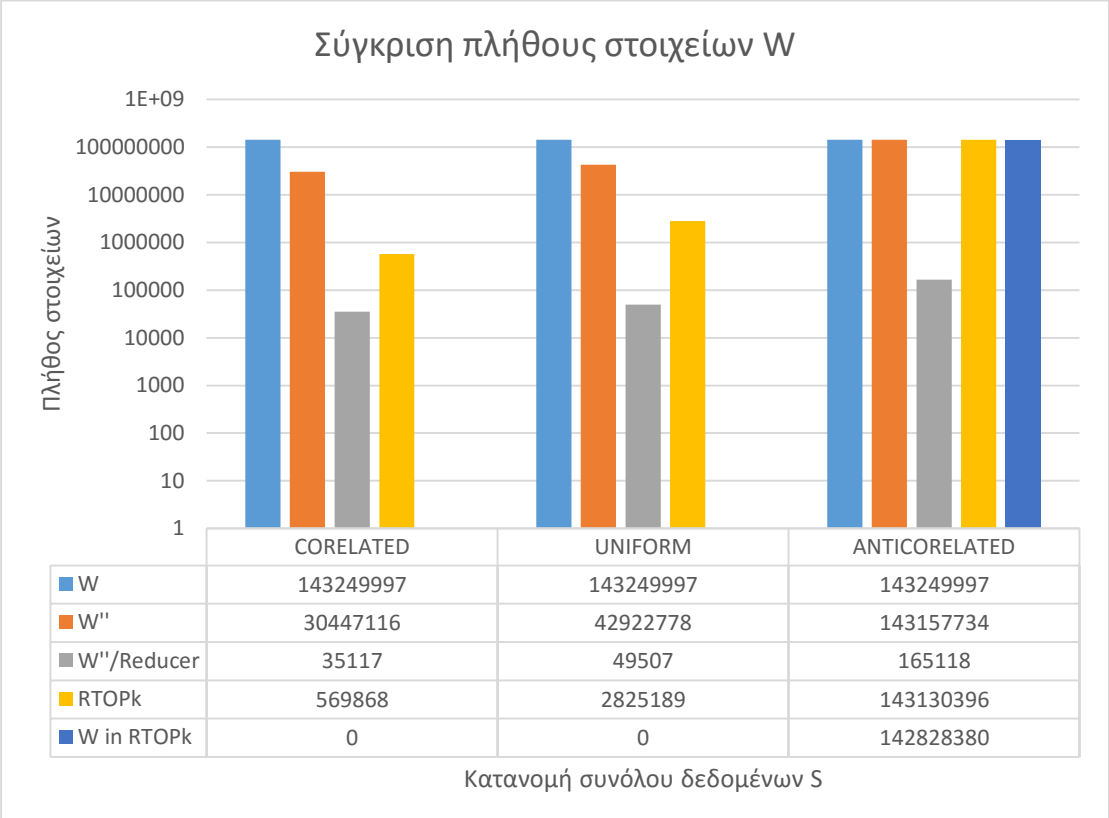

*Εικόνα 52: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 53 παρουσιάζονται οι μετρήσεις σχετικά με την περικοπή των στοιχείων S κατά την Map φάση. Όσο αναφορά την περικοπή των στοιχείων βάση της κυριαρχίας του q όπως αναμενόταν η αποδοτικότερη περικοπή δεδομένων παρουσιάζεται στο πείραμα με την correlated κατανομή. Κάτι που είναι πολύ σημαντικό καθώς στην χειρότερη περίπτωση τα στοιχεία που επιβιώνουν από την κυριαρχία του q θα σταλούν προς όλους τους Reducers, δηλαδή πολλαπλασιάζεται το μέγεθος αυτό και μπορεί να δημιουργήσει μεγάλο φόρτο στο δίκτυο. Σχετικά με την περικοπή των στοιχείων βάση του αλγορίθμου ExtremeScore, στο πείραμα με την correlated κατανομή περικόπτονται το 95% των στοιχείων που πρόκειται να σταλούν στους Reducers ενώ το ποσοστό αυτό είναι 98% για την uniform κατανομή και 99% για την anticorrelated κατανομή. Από την άλλη ο αλγόριθμος περικοπής στοιχείων S R-Lists δουλεύει καλύτερα για την uniform κατανομή περικόπτοντας περισσότερα στοιχεία, έπειτα για την correlated κατανομή και όσο αφορά την anticorrelated κατανομή η συνεισφορά του στην περικοπή των στοιχείων S είναι μηδενική.




*Εικόνα 53: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S τα οποία περικόπτονται από κάθε αλγόριθμο. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

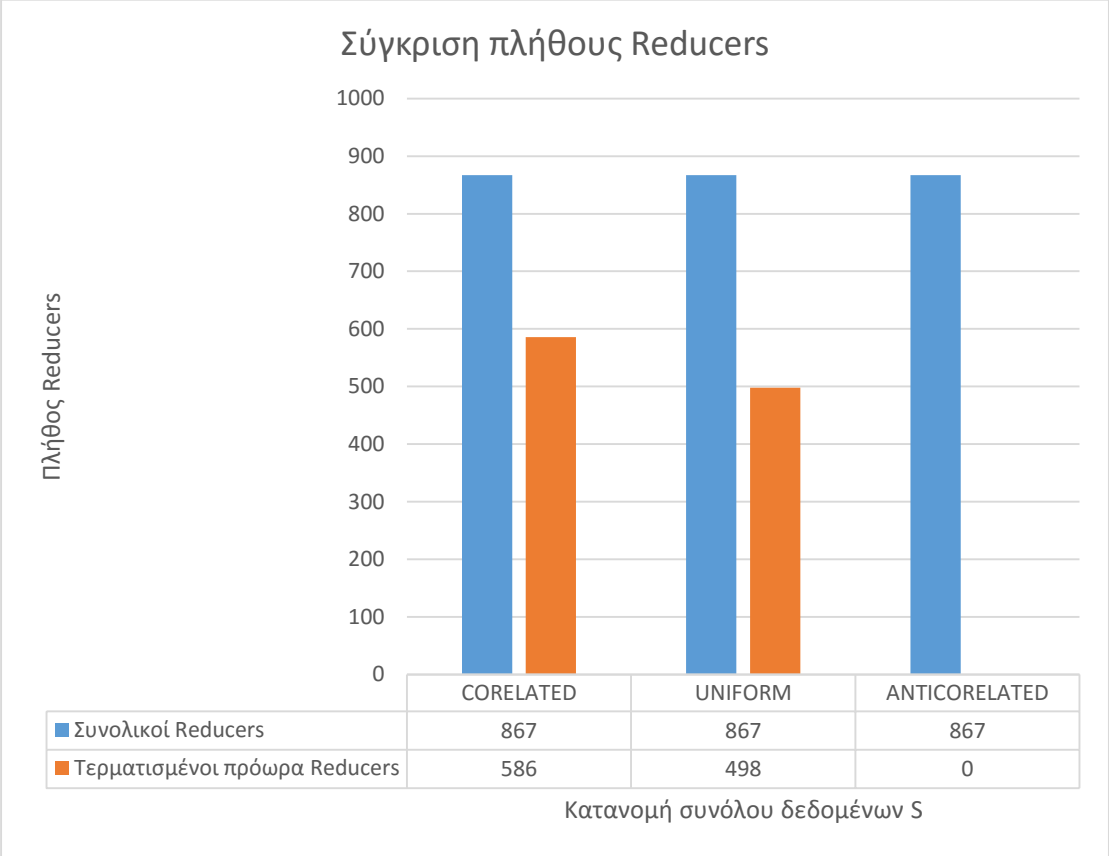
Η Εικόνα 54 παρουσιάζει το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία W που δίνονται ως είσοδο στο MapReduce Job είναι σταθερά καθώς δεν μεταβάλλεται το μέγεθος του συνόλου δεδομένων. Όσο αναφορά το πλήθος των στοιχείων W που μεταφέρονται στο δίκτυο όπως ήταν αναμενόμενο είναι λιγότερα για το πείραμα με την correlated κατανομή, περισσότερα για το πείραμα με την uniform κατανομή και ακόμη περισσότερα για το πείραμα με την anticorrelated κατανομή. Αξίζει να σημειωθεί ότι τα πειράματα σχετικά με την correlated και την uniform κατανομή δεν παρουσιάζουν σημαντικές διαφορές μεταξύ τους αντίθετα με το πείραμα σχετικά με την anticorrelated κατανομή το οποίο διαφέρει σημαντικά από τα άλλα δύο. Επειδή το πλήθος των Reducers διατηρείται σταθερό αντίστοιχη συμπεριφορά παρατηρείται και για τον μέσο όρο στοιχείων W που καλείται να επεξεργαστεί κάθε Reducer. Αντίστοιχη συμπεριφορά παρατηρείται και για τον μέσο όρο στοιχείων W που επεξεργάζεται κάθε Reducer καθώς το πλήθος των Reducers παραμένει σταθερό σε όλα τα πειράματα. Το σημαντικό στοιχείο σε αυτό το σημείο είναι τα στοιχεία W τα οποία είναι γνωστό ήδη από την Map φάση ότι ανήκουν στα τελικά Reverse Top-k αποτελέσματα βάση του $G_S$ όπου δεν χρειάζεται να υπολογιστεί το Top-k αυτών κατά την Reduce φάση και απλά γράφονται στα αποτελέσματα. Το πλήθος των στοιχείων αυτών είναι μηδενικό για την uniform και την correlated κατανομή. Αντίθετα το πλήθος των στοιχείων αυτών είναι ιδιαίτερα μεγάλο για την anticorrelated κατανομή καλύπτοντας το 99,8% των τελικών αποτελεσμάτων και το 99,7% των στοιχείων W που μεταφέρθηκαν στο δίκτυο. Αυτό συμβαίνει διότι αυξάνεται η πιθανότητα του q να ανήκει στα τελικά Reverse Top-k αποτελέσματα όπως και όντως συμβαίνει, έτσι κάποια από αυτά αναγνωρίζονται ήδη από την Map φάση γλιτώνοντας έτσι τον υπολογισμό πάρα πολλών Top-k ερωτημάτων κατά την Reduce φάση.




| | CORELATED | UNIFORM | ANTICORELATED |
|---|---|---|---|
| W | 143249997 | 143249997 | 143249997 |
| W'' | 30447116 | 42922778 | 143157734 |
| W''/Reducer | 35117 | 49507 | 165118 |
| RTOPk | 569868 | 2825189 | 143130396 |
| W in RTOPk | 0 | 0 | 142828380 |

*Εικόνα 54: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

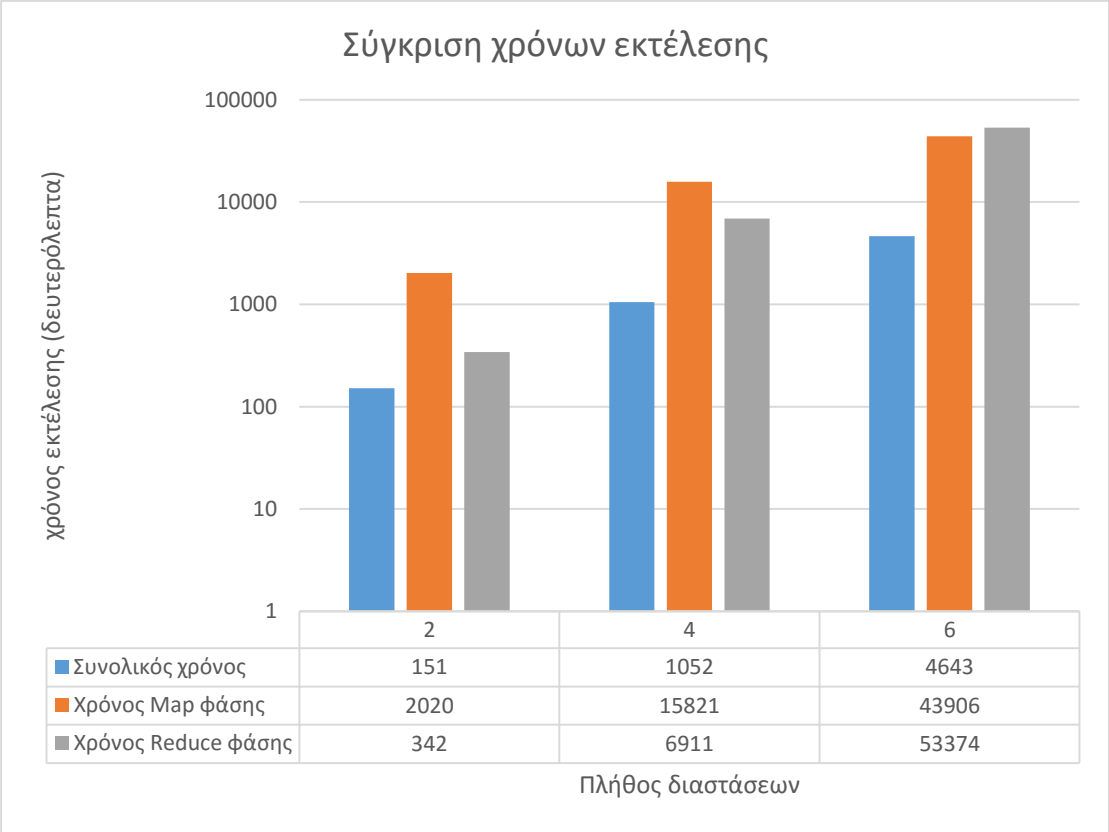
Στην Εικόνα 55 παρουσιάζεται το πλήθος των Reducers που εκτελούνται καθώς και το πλήθος αυτών που σταματούν πρόωρα χάρη στον αλγόριθμο ExtremeScore. Όπως είναι φυσικό το πλήθος των Reducers είναι ίδιο και στα 3 πειράματα. Αντίθετα όπως ήταν και αναμενόμενο άλλωστε το πλήθος των Reducers που σταματάνε πρόωρα διαφέρει ανάλογα με την κατανομή των δεδομένων S. Παρατηρείται ότι στο πείραμα σχετικά με την correlated κατανομή παρουσιάζεται το μεγαλύτερο πλήθος των Reducers που σταματάνε ενώ στο πείραμα με την uniform κατανομή το αμέσως μεγαλύτερο, αντίθετα με το πείραμα σχετικά με την anticorrelated κατανομή όπου το πλήθος αυτό είναι μηδενικό. Αυτό οφείλεται στο γεγονός ότι στην anticorrelated κατανομή βάση του q που δόθηκε αναμένεται να βρίσκονται στοιχεία από όλους τους Reducers στα τελικά Reverse Top-k αποτελέσματα τα οποία λόγω της φύσης της κατανομής επηρεάζονται από διαφορετικά στοιχεία S, έτσι κάθε Reducer πρέπει να εξετάσει τα δεδομένα του. Αντίθετα στην correlated κατανομή τα στοιχεία W επηρεάζονται από λιγότερες ομάδες στοιχείων W έτσι ομάδες οι οποίες δεν επηρεάζουν το αποτέλεσμα δεν εξετάζονται σταματώντας τους Reducers που τις επεξεργάζονται.




*Εικόνα 55: Σε αυτό το διάγραμμα παρουσιάζονται τόσο οι συνολικοί Reducers, όσο και αυτοί που σταμάτησαν πρόωρα χάρη στον αλγόριθμο ExtremeScore.*

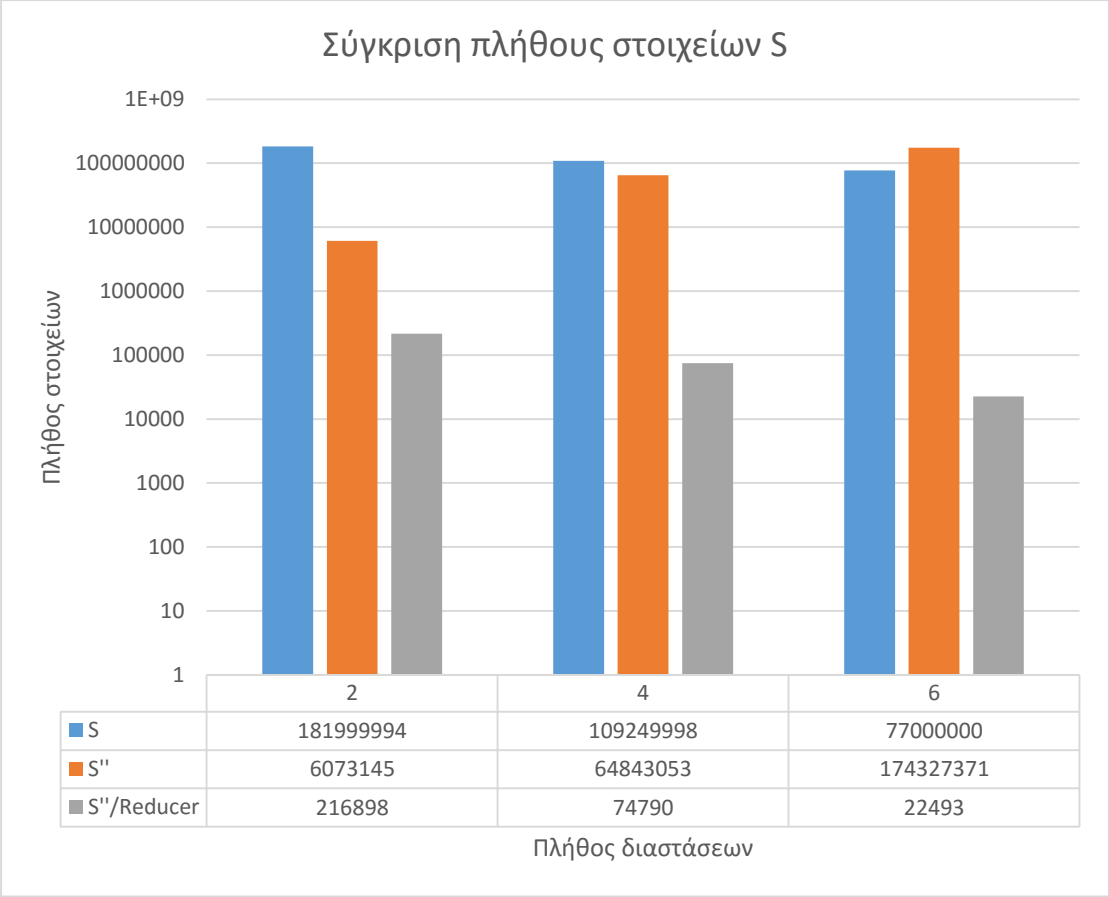
### 7.5.5 Πειράματα μεταβολής των διαστάσεων

Σε αυτή την ομάδα πειραμάτων θα μελετηθούν οι συνέπειες της μεταβολής του πλήθους των διαστάσεων για τον Σύνθετο αλγόριθμο. Διεξήχθησαν πειράματα σε 2, 4 και 6 διαστάσεις. Αξίζει να σημειωθεί ότι για να παραμείνει σταθερό το μέγεθος των συνόλων δεδομένων S, W μειώνεται το πλήθος των στοιχείων τους όσο αυξάνεται το πλήθος των διαστάσεων, καθώς όσο περισσότερες διαστάσεις έχει ένα στοιχείο τόσο περισσότερο χώρο καταλαμβάνει. Όσο αναφορά τους συνολικούς χρόνους εκτέλεσης όπως φαίνεται και στην Εικόνα 56 αυξάνονται όσο αυξάνεται και το πλήθος των διαστάσεων. Παρατηρείται επίσης ότι οι διαφορές μεταξύ τους είναι μεγάλες, για παράδειγμα ο χρόνος εκτέλεσης στο πείραμα σχετικά με τις 2 διαστάσεις απαιτεί μόλις 2,5 λεπτά ενώ στο πείραμα σχετικά με τις 6 διαστάσεις απαιτούνται 1 ώρα και 17,5 λεπτά για την εκτέλεσή του. Πιο συγκεκριμένα παρατηρείται αύξηση των χρόνων εκτέλεσης των Map φάσεων ανάλογα με το πλήθος των διαστάσεων καθώς όλοι οι αλγόριθμοι για την περικοπή των δεδομένων επηρεάζονται άμεσα από αυτό. Επίσης ενδιαφέρον παρουσιάζει η αύξηση του χρόνου εκτέλεσης των Reducers και πιο συγκεκριμένα ο ρυθμός αύξησης αυτού, όπου στο πείραμα των 6 διαστάσεων χρειάζεται περισσότερο χρόνο από την αντίστοιχη Map φάση. Αυτό συμβαίνει διότι όπως θα αναλυθεί και στην συνέχεια, όσο αυξάνεται το πλήθος των διαστάσεων τόσο μειώνεται η αποδοτικότητα των αλγορίθμων περικοπής δεδομένων. Επομένως κάθε Reducer καλείται να επεξεργαστεί περισσότερα δεδομένα ενώ παράλληλα ο RTA αλγόριθμος επιβαρύνεται και προσθέτει και αυτός καθυστερήσεις λόγω της αύξησης του πλήθους των διαστάσεων.



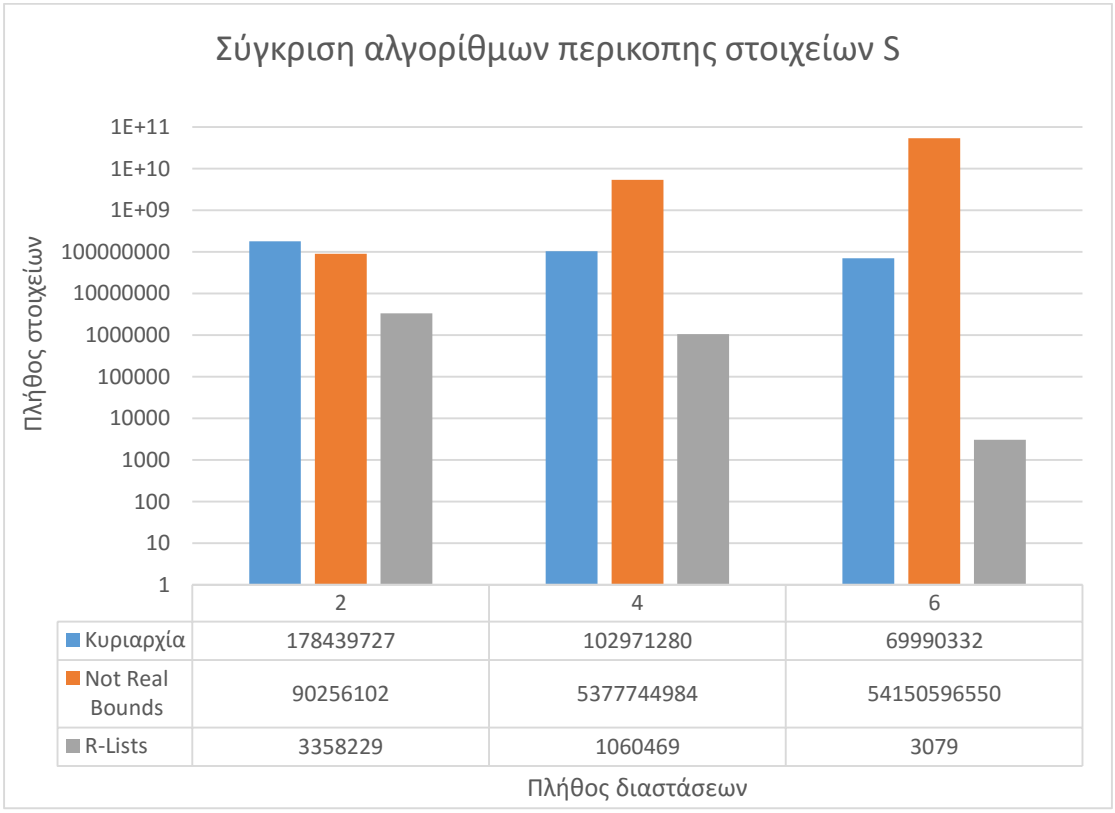

*Εικόνα 56: Σύγκριση χρόνων εκτέλεσης για τα πειράματα κατά τα οποία μεταβάλλεται μόνο το πλήθος των διαστάσεων. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του χρόνου εκτέλεσης είναι σε λογαριθμική κλίμακα.*

Η Εικόνα 57 παρουσιάζει το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία S που δίνονται ως είσοδο στο MapReduce Job είναι διαφορετικά καθώς προκειμένου να κρατηθεί σταθερό το μέγεθος του συνόλου δεδομένων S, χρειάζεται να μειώνεται το πλήθος των στοιχείων του όσο αυξάνονται οι διαστάσεις επειδή όσο περισσότερες διαστάσεις έχει ένα στοιχείο τόσο περισσότερο χώρο καταλαμβάνει. Σχετικά με το πλήθος των στοιχείων S που μεταφέρονται στο δίκτυο παρατηρείται μία αύξηση ανάλογα με την αύξηση του πλήθους των διαστάσεων. Η αύξηση αυτή οφείλεται σε δύο παράγοντες, πρώτον στο γεγονός ότι δεν γίνεται τόσο αποδοτική περικοπή των δεδομένων και δεύτερον στο γεγονός ότι όσο αυξάνεται το πλήθος των διαστάσεων τόσο αυξάνεται το πλήθος των Reducers. Η αύξηση του συνόλου των Reducers οφείλεται στο γεγονός ότι πρέπει να διατηρηθεί σταθερό το πλήθος των τμημάτων που χωρίζεται η κάθε διάσταση για να δημιουργηθούν οι ομάδες των weighting vectors. Αυτό σημαίνει ότι πρέπει να στείλει το ίδιο στοιχείο σε ακόμα περισσότερους Reducers. Αντίθετα επειδή το πλήθος των Reducers αυξάνεται όσο αυξάνεται το πλήθος των διαστάσεων, ο μέσος όρος στοιχείων S που επεξεργάζεται κάθε Reducer μειώνεται όσο αυξάνεται το πλήθος των διαστάσεων.




*Εικόνα 57: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

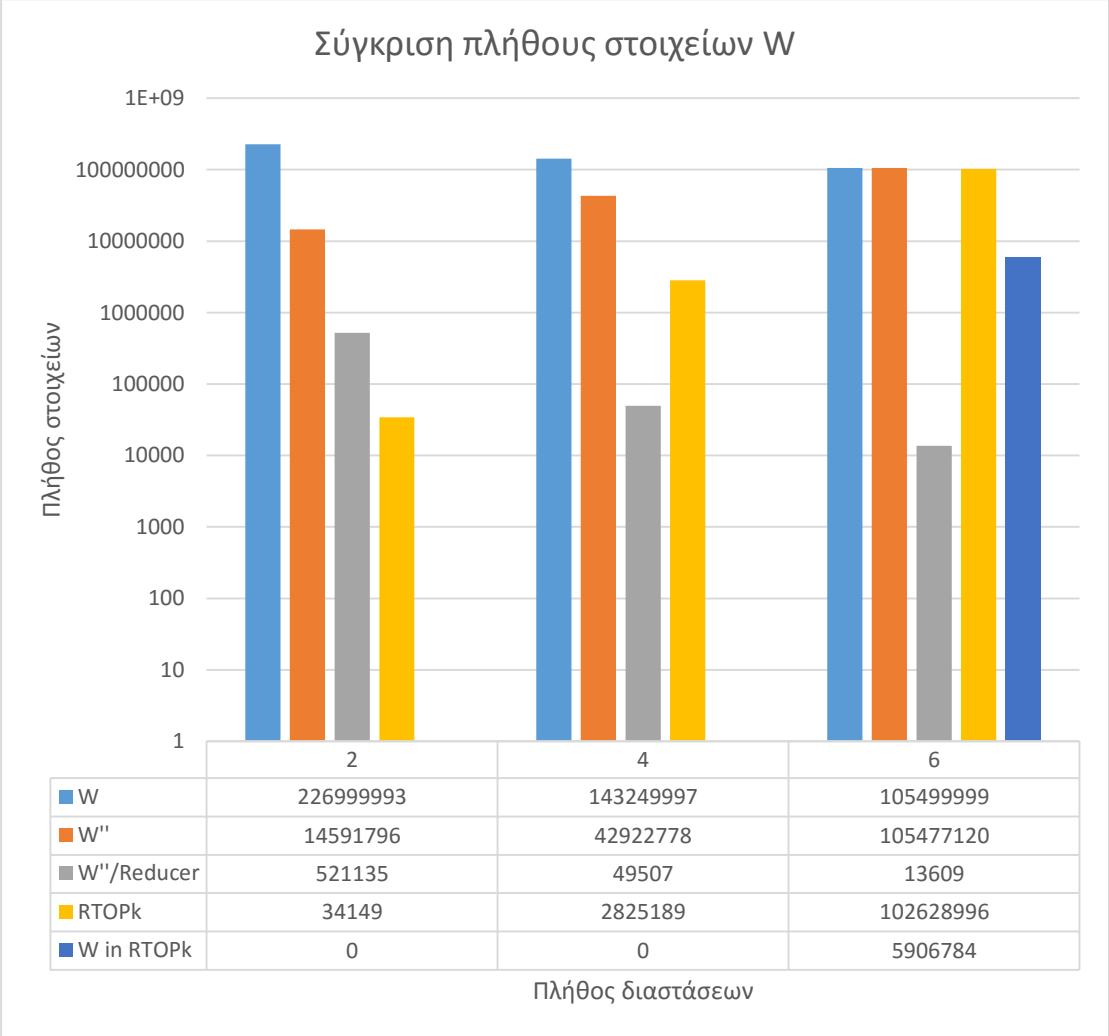
Στην Εικόνα 58 παρουσιάζονται οι μετρήσεις σχετικά με την περικοπή των στοιχείων S κατά την Map φάση. Όσο αναφορά την περικοπή των στοιχείων βάση της κυριαρχίας του q όπως αναμενόταν η αποδοτικότερη περικοπή δεδομένων παρουσιάζεται στο πείραμα με τις 2 διαστάσεις ενώ η λιγότερο αποδοτική περικοπή δεδομένων παρουσιάζεται στο πείραμα με τις 6 διαστάσεις. Αξίζει να σημειωθεί ότι το όσο αυξάνονται οι διαστάσεις τόσο μειώνεται το πλήθος των στοιχείων του συνόλου δεδομένων S. Πιο συγκεκριμένα για τις 2 διαστάσεις περικόπτεται από την κυριαρχία του q το 98% των στοιχείων, ενώ για τις 4 διαστάσεις το 94% και για τις 6 διαστάσεις το 90%. Οι διαφορές αυτές μπορεί να φαίνονται μικρές αλλά είναι πολύ κρίσιμες καθώς το πλήθος των στοιχείων που επιβιώνουν από την περικοπή αυτή στην χειρότερη περίπτωση θα σταλεί προς όλους τους Reducers όπου το πλήθος αυτών αυξάνεται με την αύξηση των διαστάσεων (λόγω των τιμών που πρέπει να κρατηθούν σταθερές για την πειραματική αξιολόγηση, στην πραγματικότητα μπορεί να κρατάτε ίσο ή ακόμα και να μειώνεται, είναι επιλογή του χρήστη). Σχετικά με την αποδοτικότητα του ExtremeScore αλγόριθμου κατά την αύξηση του πλήθους των διαστάσεων παρατηρείται ότι βελτιώνεται η αποδοτικότητά του καθώς για τις 2 διαστάσεις περικόπτεται το 90% των στοιχείων που θα στελνόντουσαν στους Reducers, αντίστοιχα για τις 4 διαστάσεις το ποσοστό αυτό αγγίζει το 98% ενώ για τις 6 διαστάσεις το ποσοστό αυτό είναι 99,6%. Παρόλα αυτά στις 6 διαστάσεις παρατηρείται ότι στέλνονται περισσότερα στοιχεία καθώς το πλήθος των Reducers είναι 7750, όπου επιχειρείται να σταλεί κάθε στοιχείο S σε καθέναν από αυτούς. Η αποδοτικότητα του αλγορίθμου R-Lists μειώνεται σημαντικά όσο αυξάνεται το πλήθος των διαστάσεων. Πιο συγκεκριμένα στις 2 διαστάσεις περικόπτει το 3%, ενώ στις 4 και στις 6 διαστάσεις το ποσοστό αυτό μειώνεται δραματικά όπως παρατηρείται και στο διάγραμμα.




| | 2 | 4 | 6 |
|---|---|---|---|
| Κυριαρχία | 178439727 | 102971280 | 69990332 |
| Not Real Bounds | 90256102 | 5377744984 | 54150596550 |
| R-Lists | 3358229 | 1060469 | 3079 |

*Εικόνα 58: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων S τα οποία περικόπτονται από κάθε αλγόριθμο. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

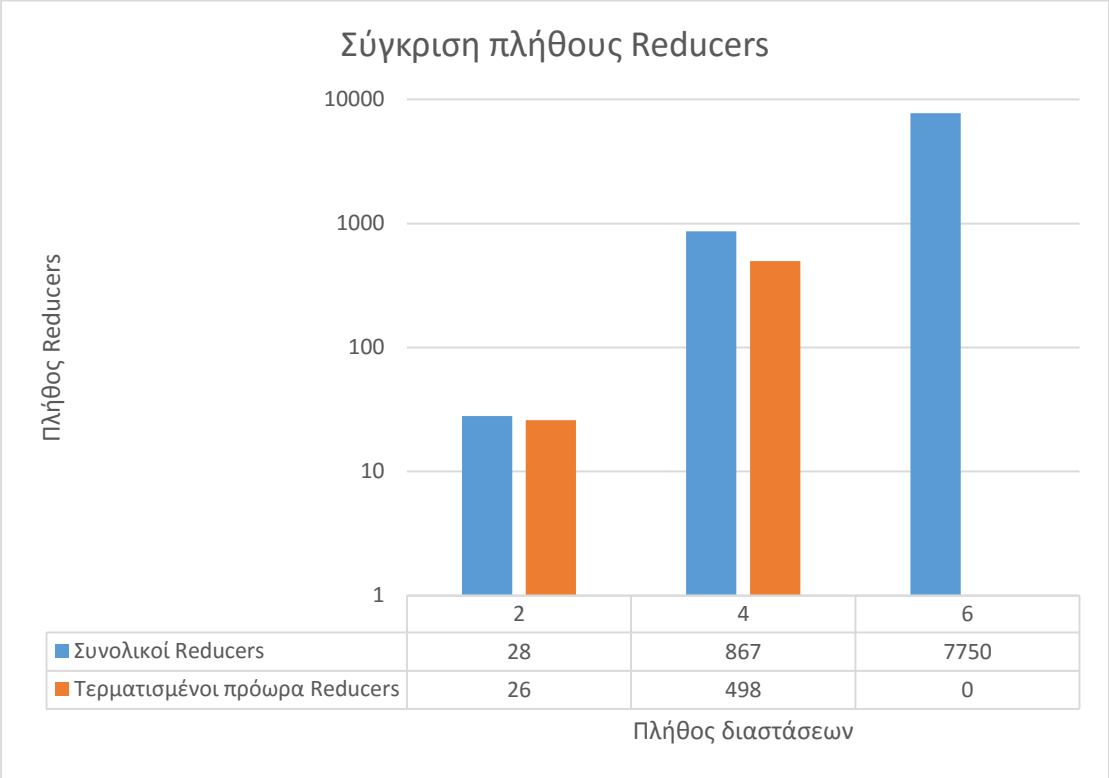
Η Εικόνα 59 παρουσιάζει το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Όπως παρατηρείται τα στοιχεία W που δίνονται ως είσοδο στο MapReduce Job είναι διαφορετικά καθώς προκειμένου να κρατηθεί σταθερό το μέγεθος του συνόλου δεδομένων W χρειάζεται να μειώνεται το πλήθος των στοιχείων του όσο αυξάνονται οι διαστάσεις επειδή όσο περισσότερες διαστάσεις έχει ένα στοιχείο τόσο περισσότερο χώρο καταλαμβάνει. Σχετικά με το πλήθος των στοιχείων W που μεταφέρονται στο δίκτυο παρατηρείται μία αύξηση ανάλογα με την αύξηση του πλήθους των διαστάσεων. Αυτό οφείλεται στο γεγονός ότι η επιλογή του q μπορεί να πληροί κάποια κριτήρια αλλά ποικίλει, και αυτό παρατηρείται στο πείραμα με τις 6 διαστάσεις το οποίο έχει πολλά Reverse Top-k αποτελέσματα. Το μόνο ασφαλές συμπέρασμα που μπορεί να σημειωθεί σε αυτό το σημείο είναι ότι όσο αυξάνεται το πλήθος των διαστάσεων (δηλαδή και το πλήθος των Reducers) τόσο μειώνεται ο μέσος όρος στοιχείων W που καλείται να επεξεργαστεί ένας Reducer.




| | 2 | 4 | 6 |
|---|---|---|---|
| W | 226999993 | 143249997 | 105499999 |
| W'' | 14591796 | 42922778 | 105477120 |
| W''/Reducer | 521135 | 49507 | 13609 |
| RTOPk | 34149 | 2825189 | 102628996 |
| W in RTOPk | 0 | 0 | 5906784 |

*Εικόνα 59: Στο γράφημα αυτό παρουσιάζεται το πλήθος των στοιχείων W σε διάφορες φάσεις του MapReduce Job. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των στοιχείων είναι σε λογαριθμική κλίμακα.*

Στην Εικόνα 60 παρουσιάζεται το πλήθος των Reducers που εκτελούνται καθώς και το πλήθος αυτών που σταματούν πρόωρα χάρη στον αλγόριθμο ExtremeScore. Όπως αναφέρθηκε και προηγουμένως το πλήθος των Reducers αυξάνεται ανάλογα με το πλήθος των διαστάσεων. Αυτό συμβαίνει λόγω των τιμών που πρέπει να κρατηθούν σταθερές για την πειραματική αξιολόγηση, στην πραγματικότητα μπορεί να κρατείται ίσο ή ακόμα και να μειώνεται, είναι επιλογή του χρήστη. Από την άλλη πλευρά το ποσοστό των Reducers που σταματούν πρόωρα μειώνεται όσο αυξάνεται το πλήθος των διαστάσεων. Αυτό οφείλεται κυρίως στην δυσκολία της ύπαρξης k στοιχείων που να έχουν καλύτερη βαθμολογία για όλα τα στοιχεία W μιας ομάδας, όσο αυξάνεται το πλήθος των διαστάσεων. Ακόμα το αποτέλεσμα αυτό επηρεάζεται και από την επιλογή του q που παρόλο που τηρούνται κάποιοι κανόνες για την επιλογή του τα αποτελέσματα διαφέρουν.




*Εικόνα 60: Σε αυτό το διάγραμμα παρουσιάζονται τόσο οι συνολικοί Reducers, όσο και αυτοί που σταμάτησαν πρόωρα χάρη στον αλγόριθμο ExtremeScore. Αξίζει να σημειωθεί ότι η κλίμακα του άξονα του πλήθους των Reducers είναι σε λογαριθμική κλίμακα.*

# 8. Σχετικές έρευνες

Πρόσφατα, η αποδοτικότητα των Top-k ερωτημάτων έχει τραβήξει το ενδιαφέρον της κοινότητας των ερευνητών που ασχολούνται με τις βάσεις δεδομένων. Επειδή η επεξεργασία των Reverse Top-k ερωτημάτων σχετίζεται σημαντικά με την επεξεργασία των Top-k ερωτημάτων, παρουσιάζονται συνοπτικά κάποιες έρευνες που έχουν γίνει στον τομέα αυτό. Η Onion τεχνική ευρετηριοποίησης όπως παρουσιάζεται στο (7) υπολογίζει σε φάση προ-επεξεργασίας το convex hull των δεδομένων σε επίπεδα. Έπειτα η εκτίμηση ενός Top-k ερωτήματος πραγματοποιείται διατρέχοντας τα επίπεδα προς τα μέσα, ξεκινώντας από το εξωτερικό hull. Το PREFER όπως παρουσιάζεται στο (8) χρησιμοποιεί materialized views των Top-k αποτελεσμάτων σύμφωνα με αυθαίρετες συναρτήσεις βαθμολόγησης. Κατά την επεξεργασία ενός ερωτήματος, επιλέγεται το materialized view που αντιστοιχεί στην συνάρτηση βαθμολόγησης που μοιάζει περισσότερο με αυτή του ερωτήματος και εξετάζει ένα υποσύνολο δεδομένων σε αυτό.

Όσο αναφορά τα Reverse Top-k ερωτήματα προτάθηκαν στις δημοσίευσες (1) και (9) προκειμένου να εκτιμηθεί το πιθανό αντίκτυπο ενός προϊόντος στην αγορά, με βάση το πλήθος των χρηστών που θα είχαν αυτό το προϊόν στα Top-k αποτελέσματά τους σύμφωνα με τις προτιμήσεις τους. Δόθηκε ο ορισμός των monochromatic και των bichromatic Reverse Top-k ερωτημάτων όπου ο πρώτος δεν γνωρίζει τις προτιμήσεις των χρηστών, ενώ ο δεύτερος τις γνωρίζει. Επιπλέον προτάθηκαν αλγόριθμοι για την απάντηση των ερωτημάτων αυτών όπως ο RTA αλγόριθμος για την απάντηση των bichromatic Reverse Top-k ερωτημάτων. Αργότερα παρουσιάστηκε ο Branch and bound αλγόριθμος όπως παρουσιάζεται στο (10) για την βελτίωση του υπολογισμού σε σχέση με τις μεθόδους που είχαν προταθεί μέχρι τότε. Ακόμα έχουν παρουσιαστεί αρκετές εφαρμογές των Reverse Top-k ερωτημάτων, όπως αυτή της αναγνώρισης των προϊόντων με την μεγαλύτερη επιρροή (11). Μία ακόμα εφαρμογή αφορά την παρακολούθηση της δημοτικότητας των τοποθεσιών βάση της κινητικότητας των χρηστών (12). Στην δημοσίευση αυτή υπάρχει μια δυναμική ιδιότητα που είναι η απόσταση των χρηστών από το σημείο ενδιαφέροντος, καθορίστηκε επίσης το Distance-based Reverse Top-k ερώτημα.

Ο υπολογισμός πολλαπλών Top-k ερωτημάτων έχει μελετηθεί στην δημοσίευση (13), όπου δοσμένου ενός συνόλου δεδομένων και ενός συνόλου συναρτήσεων βαθμολόγησης προτείνεται μια μέθοδος υπολογισμού των Top-k ερωτημάτων για όλες τις συναρτήσεις βαθμολόγησης (All Top-k ερώτημα). Στην πιο απλή περίπτωση θα έπρεπε να υπολογιστούν τα Top-k ερωτήματα για κάθε συνάρτηση αλλά η μέθοδος αυτή εκμεταλλεύεται το γεγονός ότι όμοιες συναρτήσεις παρουσιάζουν κοινά αποτελέσματα και έτσι αποφεύγεται ο υπολογισμός όλων των Top-k ερωτημάτων. Επίσης προτείνεται και μια μέθοδος η οποία μπορεί να χρησιμοποιηθεί για τον υπολογισμό των Reverse Top-k ερωτημάτων, διότι αν είναι δυνατός ο υπολογισμός όλων των Top-k ερωτημάτων είναι απλό πρόβλημα να υπολογιστεί το Reverse Top-k ερώτημα. Ωστόσο, όπως παρουσιάστηκε στην πειραματική αξιολόγηση ο υπολογισμός ενός All Top-k ερωτήματος κοστίζει περισσότερο από τον υπολογισμό ενός Reverse Top-k ερωτήματος και αξίζει να πραγματοποιηθεί σε περιπτώσεις που χρειάζεται να απαντηθούν πολλαπλά Reverse Top-k ερωτήματα.

Η δημοσίευση (14), αποτελεί έναν οδηγό με προβλήματα και τεχνικές επίλυσης για την επεξεργασία δεδομένων στο μοντέλο MapReduce. Πιο αναλυτικά επικεντρώνεται στην αποδοτική και παράλληλη επεξεργασία ερωτημάτων στο μοντέλο MapReduce. Αποτυπώνει τις αδυναμίες του μοντέλου αυτού σε βάθος, παρουσιάζει διάφορες προτάσεις για την επεξεργασία διάφορων ερωτημάτων στον μοντέλο αυτό. Επιπλέον συγκρίνει τις προτάσεις για την επεξεργασία κοινών ερωτημάτων, παρουσιάζοντας τα θετικά και τα αρνητικά στοιχεία αυτών (πχ. αν ένας αλγόριθμος χρειάζεται προ-επεξεργασία ή όχι).

# 9. Συμπεράσματα

Στην εργασία αυτή παρουσιάστηκε ένας αλγόριθμος που προτάθηκε για το πρόβλημα της κατανεμημένης επεξεργασίας ενός Reverse Top-k ερωτήματος σε κατανεμημένο περιβάλλον. Στόχος ήταν να προταθεί ένας αλγόριθμος που μπορεί να διαχειριστεί μεγάλα δεδομένα, πραγματοποιώντας επεξεργασία με κατανεμημένο και παράλληλο τρόπο για την απάντηση Reverse Top-k ερωτημάτων σε λογικό χρόνο.

Όπως φάνηκε και από την πειραματική αξιολόγηση ο αλγόριθμος αυτός ανέβασε τον πήχη καθώς απαντούσε σε λογικό χρόνο Reverse Top-k ερωτήματα πάνω από μεγάλα σύνολα δεδομένων, τα οποία ήταν πολύ μεγαλύτερα από αυτά που χρησιμοποιούντο μέχρι τώρα στις έρευνες για αυτό τον τύπο ερωτήματος. Επίσης ο αλγόριθμος προσαρμόζεται σε διάφορες μεταβολές των παραμέτρων του καθώς οι αλγόριθμοι που χρησιμοποιήθηκαν «συμπληρώνουν» πολλές φορές ο ένας τον άλλο καθώς με την μεταβολή μιας παραμέτρου μπορεί ένας αλγόριθμος να μην διαχειρίζεται καλά την κατάσταση προσθέτοντας βάρη στην απάντηση του ερωτήματος αλλά από την άλλη η μεταβολή της τιμής αυτής αυξάνει την αποδοτικότητα ενός άλλου αλγορίθμου διατηρώντας έτσι μία ισορροπία. Χαρακτηριστικό παράδειγμα είναι αυτό της μεταβολής του μεγέθους του συνόλου δεδομένων S όπου από την μία στέλνονται περισσότερα δεδομένα προς τους Reducers αλλά από την άλλη οι Reducers γλιτώνουν σημαντικό χρόνο εκτέλεσης καθώς γνωρίζουν από τους Mappers αρκετά στοιχεία W που ανήκουν ήδη στα τελικά αποτελέσματα.

Παρόλα αυτά υπάρχει περιθώριο να γίνουν πολλές βελτιώσεις. Για παράδειγμα ο αλγόριθμος αυτός αντιστοιχεί το πλήθος των Reducers με το πλήθος των ομάδων των weighting vectors, κάτι που ίσως να μην είναι επιθυμητό σε όλες τις περιπτώσεις. Ακόμα από την πειραματική αξιολόγηση προκύπτει ότι θα παρουσίαζε καθυστερήσεις για μεγάλο πλήθος διαστάσεων πχ. 50. Θα μπορούσε επίσης να μελετηθεί ο τρόπος δημιουργίας των πλεγμάτων και των ομάδων των weighting vectors ώστε να αυξάνουν την αποδοτικότητα του αλγορίθμου ή η δυναμική επιλογή αυτών ανάλογα με το q.

# Παράρτημα

## Περιβάλλον εκτέλεσης αλγορίθμων

Σε αυτό το σημείο θα περιγραφεί το περιβάλλον στο οποίο μπορούν να εκτελεστούν οι αλγόριθμοι. Αρχικά θα πρέπει να είναι εγκατεστημένη η έκδοση 7 της Java σε κάθε κόμβο που θα εκτελεστούν οι αλγόριθμοι. Επιπροσθέτως πρέπει να είναι εγκατεστημένη η έκδοση του Hadoop 2.6.0 καθώς αυτή χρησιμοποιούν και οι δύο αλγόριθμοι. Οποιαδήποτε νέα η παλιότερη έκδοση των παραπάνω μπορεί να προκαλέσει προβλήματα συμβατότητας.

## Εγχειρίδιο χρήσης Απλού αλγορίθμου

Σε αυτό το σημείο περιγράφεται ο τρόπος με τον οποίο μπορεί κανείς να εκτελέσει το Απλό αλγόριθμο. Ανάλογα με το περιβάλλον εκτέλεσης θέτεται ως εκτελέσιμο αρχείο το .jar αρχείο που βρίσκεται στον φάκελο του απλού αλγορίθμου. Έπειτα δίνονται ως είσοδο δύο αρχεία S και W, τα οποία πρέπει να έχουν την παρακάτω μορφή. Τα στοιχεία της κάθε γραμμής χωρίζονται μεταξύ τους με κενά.

| id | διάσταση-1 | … | διάσταση-d |
|---|---|---|---|
| **1** | 23.325 | | 45.345 |
| **2** | 34.352 | | 23.453 |

Οι παράμετροι που δίνονται στον αλγόριθμο περιγράφονται παρακάτω. Αξίζει να σημειωθεί ότι δίνονται σε μορφή Command-Line Arguments (Java args). Δηλαδή κάθε κενό σημειώνει και ότι ακολουθεί μια επόμενη παράμετρος. Οι παράμετροι καθώς και οι θέσεις τους περιγράφονται παρακάτω.

| Θέση παραμέτρου | Περιγραφή |
|---|---|
| **args[0]** | Το μέγεθος k (πχ. 10) |
| **args[1]** | Το ερώτημα q, προσοχή δεν πρέπει να περιέχει κενά. Τα κόμματα διαχωρίζουν τις διαστάσεις ενώ η τελειά το δεκαδικό μέρος (πχ. [14.4,23.4,23.42] για 3-διαστάσεις). |
| **args[2]** | Το path του αρχείου που αντιστοιχεί στο σύνολο δεδομένων S. Προσοχή δεν πρέπει να περιέχει κενά (πχ. /George/S.txt) |
| **args[3]** | Το path του αρχείου που αντιστοιχεί στο σύνολο δεδομένων W. Προσοχή δεν πρέπει να περιέχει κενά (πχ. /George/W.txt) |
| **args[4]** | Το path που θα εξαχθούν τα δεδομένα. Προσοχή δεν πρέπει να περιέχει κενά (πχ. /George/Results) |
| **args[5]** | Η τιμή βάση της οποίας θα ταξινομηθούν τα weighing vectors. Συνήθως είναι 0.5 και πιθανώς δεν θα χρειαστεί να αλλάξει. |
| **args[6]** | Ο τύπος του αλγορίθμου που επιθυμείται να εκτελεστεί. Οι δυνατές τιμές είναι οι εξής FULL_MULTIPLE_REDUCERS, FULL, NoSortingW, NoSortingWNoDominate. Στα πλαίσια της εργασίας αυτής χρησιμοποιήθηκε ο τύπος FULL_MULTIPLE_REDUCERS. Οι υπόλοιποι τύποι απλώς υπάρχουν και δεν χρειάστηκε να γίνει περαιτέρω αναφορά στα πλαίσια της εργασίας αυτής. |
| **args[7]** | Το πλήθος των Reducers (πχ. 5). Χρησιμοποιείται μόνο για τον τύπο FULL_MULTIPLE_REDUCERS. |

## Εγχειρίδιο χρήσης Σύνθετου αλγορίθμου

Σε αυτό το σημείο περιγράφεται ο τρόπος με τον οποίο μπορεί κανείς να εκτελέσει το Σύνθετο αλγόριθμο. Ανάλογα με το περιβάλλον εκτέλεσης θέτεται ως εκτελέσιμο αρχείο το .jar αρχείο που βρίσκεται στον φάκελο του Σύνθετου αλγορίθμου. Έπειτα δίνονται ως είσοδο δύο αρχεία S και W, τα οποία πρέπει να έχουν την παρακάτω μορφή. Τα στοιχεία της κάθε γραμμής χωρίζονται μεταξύ τους με κενά.

| id | διάσταση-1 | ... | διάσταση-d |
|---|---|---|---|
| **1** | 23.325 | | 45.345 |
| **2** | 34.352 | | 23.453 |

Επιπλέον στον αλγόριθμο αυτό δίνεται ως είσοδος το πλέγμα για το σύνολο δεδομένων S. Το οποίο πρέπει να έχει την παρακάτω μορφή. Τα στοιχεία της κάθε γραμμής χωρίζονται μεταξύ τους με κενά.

| id | Πλήθος στοιχείων | Κάτω όριο διάσταση-1 | ... | Κάτω όριο διάσταση-d | Άνω όριο διάσταση-1 | ... | Άνω όριο διάσταση-d |
|---|---|---|---|---|---|---|---|
| **1** | 100 | 23.325 | | 25.345 | 78.75 | | 68.67 |
| **2** | 50 | 34.352 | | 23.453 | 65.76 | | 67.67 |

Επιπλέον στον αλγόριθμο αυτό δίνεται ως είσοδος  οι ομάδες των weighting vectors. Το οποίο πρέπει να έχει την παρακάτω μορφή. Τα στοιχεία της κάθε γραμμής χωρίζονται μεταξύ τους με κενά.

| id | Κάτω όριο διάσταση-1 | ... | Κάτω όριο διάσταση-d | Άνω όριο διάσταση-1 | ... | Άνω όριο διάσταση-d |
|---|---|---|---|---|---|---|
| **1** | 0.75 | | 0.0 | 1 | | 0.25 |
| **2** | 0.5 | | 0.25 | 0.75 | | 0.5 |

Οι παράμετροι που δίνονται στον αλγόριθμο περιγράφονται παρακάτω. Αξίζει να σημειωθεί ότι δίνεται το path ενός φακέλου σε μορφή Command-Line Arguments (Java args), ο οποίος φάκελος περιέχει τα αρχεία των πειραμάτων που θα εκτελεστούνε. Κάθε αρχείο περιέχει τις παραμέτρους του εκάστοτε πειράματος οι οποίες περιγράφονται παρακάτω.

| Όνομα παραμέτρου | Περιγραφή |
|---|---|
| K | Το μέγεθος k |
| Query | Το ερώτημα q, προσοχή δεν πρέπει να περιέχει κενά. Τα κόμματα διαχωρίζουν τις διαστάσεις ενώ η τελειά το δεκαδικό μέρος. |
| PathS | Το path του αρχείου που αντιστοιχεί στο σύνολο δεδομένων S. |
| PathGridS | Το path του αρχείου που αντιστοιχεί στο πλέγμα του συνόλου δεδομένων S. |
| PathW | Το path του αρχείου που αντιστοιχεί στο σύνολο δεδομένων W. |
| PathGridW | Το path του αρχείου που αντιστοιχεί στις ομάδες weighting vectors. |
| PathOutput | Το path που θα εξαχθούν τα δεδομένα. |
| ReducersNumber | Το πλήθος των Reducers. Πρέπει να είναι ίσο με το πλήθος των ομάδων των weighting vectors. |
| AlgorithmForS | Ο τύπος αλγορίθμου για περικοπή των στοιχείων S. Οι δυνατές τιμές είναι RealBounds, Rlists, NotRealBounds, CompineNotRealBoundsAndRLists. Αξίζει να σημειωθεί ότι ο τύπος αλγορίθμου RealBounds δεν χρησιμοποιείται στην εργασία αυτή. Όπου NotRealBounds εννοείται ο ExtremeScore. |
| GridForS | Ο τύπος πλέγματος που θα χρησιμοποιηθεί. Οι δυνατές τιμές είναι Simple, DominateAndAntidominateArea, Tree, TreeDominateAndAntidominateArea. |
| JobName | Το όνομα του πειράματος αυτού. |
| AlgorithmForRtopk | Ο τύπος του Reverse Top-k αλγορίθμου που εκτελείται στην Reduce φάση. Οι δυνατές τιμές είναι RTA, BRS. Ο τύπος BRS δεν χρησιμοποιείται στα πλαίσια της εργασίας αυτής. |

Παρακάτω παρουσιάζεται ένα παράδειγμα με κάποιες τιμές που μπορεί να έχει ένα τέτοιο αρχείο.

```
K=10
Query=[6456.57153,339201.0,136210.0,102653.8]
PathS=/user/input/5Suni4
PathGridS=/user/input/5Suni4.grid10
PathW=/user/input/5Wuni4
PathGridW=/user/input/W4.grid10
PathOutput=/user/Results
ReducersNumber=867
AlgorithmForS=CompineNotRealBoundsAndRLists
GridForS=TreeDominateAndAntidominateArea
JobName=My_Job_Name
AlgorithmForRtopk=RTA
```

# Βιβλιογραφία

1. Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, Kjetil Nørvåg. Reverse Top-k Queries. *ICDE.* 2010: 365 - 376.

2. hadoop.apache.org. *Apache Hadoop.* [Online] 01 16, 2016 https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

3. hadooptutorial.wikispaces.com. [Online] 01 13, 2016. http://hadooptutorial.wikispaces.com/MapReduce.

4. White, Tom. *Hadoop: The Definitive Guide, Third Edition.* s.l. : O'Reilly Media / Yahoo Press, 2012.

5. Hadoop 2.6.0 Documentation. [Online] 06 2015. https://hadoop.apache.org/docs/stable/api/.

6. Yahoo! developer.yahoo.com. [Online] 01 13, 2016. https://developer.yahoo.com/hadoop/tutorial/.

7. Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, John R. Smith. The onion technique: indexing for linear optimization queries. *SIGMOD Conference 2000:* 391 - 402.

8. Vagelis Hristidis, Nick Koudas, Yannis Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. *SIGMOD Conference 2001:* 259-270.

9. Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, Kjetil Nørvåg. Monochromatic and Bichromatic Reverse Top-k Queries. *IEEE Trans. Knowl. Data Eng. 23(8):* 1215 - 1229 .

10. Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, Yannis Kotidis. Branch-and-bound algorithm for reverse top-k queries. *SIGMOD Conference 2013:* 481 - 492 .

11. Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, Yannis Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB 3(1).* 2010: 364-372.

12. Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg. Monitoring reverse top-k queries over mobile devices. *MobiDE 2011:* 17-24.

13. Leong Hou U, Mamoulis N., Cheung D.W. Efficient All Top-k Computation - A Unified Solution for All Top-k, Reverse Top-k and Top-m Influential Queries. *IEEE Trans. Knowl. Data Eng. 25(5).* 2013: 1015 - 1027.

14. Christos Doulkeridis, Kjetil NØrvåg. A survey of large-scale analytical query processing in MapReduce. *VLDB J. 23(3).* 2014: 355-380.