Ατομική Διπλωματική Εργασία

ΑΝΑΠΤΥΞΗ ΚΑΙ ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΒΕΛΤΙΩΣΗ ΤΗΣ ΕΠΙΔΟΣΗΣ ΔΙΚΤΥΩΝ ΠΑΡΑΔΟΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ ΜΕ ΠΟΛΛΑΠΛΟΥΣ ΕΞΥΠΗΡΕΤΗΤΕΣ ΠΕΡΙΕΧΟΜΕΝΟΥ

Έλλη Ζαβού

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη και Ανάλυση Αλγορίθμων για βελτίωση της επίδοσης Δικτύων Παράδοσης Περιεχομένου με Πολλαπλούς Εξυπηρετητές Περιεχομένου

Έλλη Ζαβού

Επιβλέποντες Καθηγητές
Γεωργίου Χρύσης και Πάλλης Γιώργος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους επιβλέποντες καθηγητές μου, Επίκουρο Καθηγητή Χρύση Γεωργίου και Λέκτορα Γιώργο Πάλλη, για την ευκαιρία που μου έδωσαν να εργαστώ σε αυτή την εργασία αλλά και για την εμπιστοσύνη που μου έδειξαν ώστε να την φέρω εις πέρας.

Θα ήθελα επίσης να τους ευχαριστήσω για την άψογη συνεργασία μας, για την πολύτιμη βοήθεια που μου παρείχαν όλο αυτό το διάστημα καθώς επίσης για την καθοδήγηση και τις συμβουλές που μου έδιναν. Τους ευχαριστώ για τον χρόνο που αφιέρωσαν και το ενδιαφέρον τους να με στηρίξουν.

Θα ήθελα επιπλέον να ευχαριστήσω την οικογένειά μου για τη στήριξη που μου έδωσαν όλα αυτά τα χρόνια αλλά και τον φίλο και συμφοιτητή μου Νίκο Δημητρίου, με τον οποίο συνεργαστήκαμε άψογα όλο αυτό το διάστημα, αφού η διπλωματική του ενός συμπληρώνει τη διπλωματική του άλλου.

Περίληψη

Τα Δίκτυα Παράδοσης Περιεχομένου (ΔΠΠ - CDNs) χρησιμοποιούνται ευρέως για τη μετάδοση δεδομένων στο Διαδίκτυο. Παρουσιάζουν μεγάλο ερευνητικό ενδιαφέρον λόγω της υψηλής ποιότητας υπηρεσιών που προσφέρουν, μεταφέροντας το περιεχόμενο στους τελικούς χρήστες.

Η κατανόηση της πλοήγησης των χρηστών στον Παγκόσμιο Ιστό (Π.Ι.) είναι σημαντική για τη βελτίωση της ποιότητας της πληροφορίας και της ταχύτητας της προσπέλασης ευρείας κλίμακας πόρων του Π.Ι. Η ομαδοποίηση των χρηστών του Π.Ι. με βάση την πλοήγησή τους σε συνόδους αποτελεί μία σύνηθες πρακτική με σκοπό να προσδιορισθούν τα πρότυπα και οι ομοιότητες τα οποία διαχειρίζονται από διαδικτυακές αλληλεπιδραστικές εφαρμογές (αναζήτηση, ηλεκτρονικό εμπόριο κλπ). Μελετώντας την πλοήγηση των χρηστών στο Διαδίκτυο έχει παρατηρηθεί ότι στα ενδιαφέροντα των χρηστών υπάρχει αυξημένη ζήτηση ιστοσελίδων παρόμοιου περιεχομένου από διαφορετικές ομάδες χρηστών.

Ένα άλλο σημαντικό ζήτημα είναι να καθοριστεί το περιεχόμενο που θα διατεθεί από τους διάφορους εξυπηρετητές του Παγκόσμιου Ιστού για να το διαχειριστεί κάποιο διαθέσιμο ΔΠΠ. Σε αυτή την εργασία προτείνεται μία νέα προσέγγιση, σύμφωνα με τη ζήτηση των χρηστών σε ιστοσελίδες παρόμοιου περιεχομένου, για να καθοριστεί το περιεχόμενο που θα αντιγραφεί στους εξυπηρετητές αναπαραγωγής.

Οι ιστοσελίδες ενός Διαδικτυακού τόπου ομαδοποιούνται με βάση τη συνδεσμολογία τους και οι ομάδες αυτές ονομάζονται κοινότητες του Π.Ι. Είναι αυτές οι οποίες θα αντιγραφούν στους εξυπηρετητές αναπαραγωγής. Στα πλαίσια της διπλωματικής εργασίας αναπτύσσονται αποτελεσματικοί αλγόριθμοι, οι οποίοι εντοπίζουν κοινότητες του Π.Ι. από διάφορους πηγαίους εξυπηρετητές κοινού ενδιαφέροντος και τις αντιγράφουν στους κατάλληλους εξυπηρετητές αναπαραγωγής. Μέσω ενός αναλυτικού περιβάλλοντος προσομοίωσης, και χρησιμοποιώντας τόσο συνθετικά όσο και πραγματικά δεδομένα, οι προτεινόμενοι αλγόριθμοι αξιολογούνται.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο και Στόχος εργασίας

Στην εργασία αυτή, μελετούνται τα Δίκτυα Παράδοσης Περιεχομένου - ΔΠΠ (CDN: Content Delivery Networks) [2,3,4,6] και κυρίως αλγόριθμοι που θα βοηθήσουν στην αύξηση της επίδοσης τους. Γιατί όμως μας ενδιαφέρουν τα Δίκτυα Παράδοσης Περιεχομένου;

Αρχικά, θα πρέπει να πούμε τι είναι Δίκτυα Παράδοσης Περιεχομένου (ΔΠΠ) και ποιο σκοπό εξυπηρετούν στην Επιστήμη της Πληροφορικής. Τα ΔΠΠ είναι δίκτυα με πολλαπλούς εξυπηρετητές που βρίσκονται κατανεμημένοι ανά το παγκόσμιο και στους οποίους βρίσκονται αποθηκευμένα αντίγραφα περιεχομένων ιστοσελίδων. Χρησιμοποιώντας ΔΠΠ οι πληροφορίες βρίσκονται πιο κοντά στους χρήστες, μειώνοντας δραστικά την κυκλοφορία στο δίκτυο καθώς επίσης και τους χρόνους προσπέλασης της προσφερόμενης πληροφορίας. Με αυτό τον τρόπο οι χρήστες του συστήματος εξυπηρετούνται γρηγορότερα και υπάρχει λιγότερη συμφόρηση στο δίκτυο.

Για να λειτουργεί σωστά και αποδοτικά ένα ΔΠΠ, λαμβάνονται υπόψη τρεις σημαντικοί παράγοντες, η τοπολογία του δικτύου, η επιλογή του περιεχομένου που

αντιγράφεται στους εξυπηρετητές αναπαραγωγής και η πολιτική που ακολουθείται για την αντιγραφή των περιεχομένων [2,4].

Έχοντας δει λοιπόν τι είναι τα Δίκτυα Παράδοσης Περιεχομένου και τι προσφέρουν στην Επιστήμη της Πληροφορικής, ας δούμε ένα πρόβλημα που υπάρχει, το οποίο λειτούργησε σαν κίνητρο στην ανάπτυξη αυτής της μελέτης. Σύμφωνα με τα ενδιαφέροντα των χρηστών του διαδικτύου, παρατηρείται αυξημένη ζήτηση ιστοσελίδων παρόμοιου περιεχομένου από διαφορετικές ομάδες χρηστών. Για παράδειγμα, άτομα που ενδιαφέρονται για αγορές στο Διαδίκτυο, θα αναζητήσουν ως συνήθως και στο Amazon.com αλλά και στο BarnesandNoble.com. Το ίδιο ισχύει για πολλά είδη ενδιαφέροντος, που αφορούν αναζητήσεις σε ιστοσελίδες όπως: CNN.com και BBC.co.uk, IEEE.org και ACM.org, NASA.gov και ScienceDaily.com κ.ά. Επομένως, η αρχική μας ιδέα, ήταν ότι η επίδοση του δικτύου θα αυξηθεί, αν υπάρχουν σε συγκεκριμένους εξυπηρετητές αναπαραγωγής, ιστοσελίδες με κοινού ενδιαφέροντος περιεχόμενο, σύμφωνα με τις προτιμήσεις των χρηστών. Στη διπλωματική αυτή εργασία λοιπόν, αυτό που μας απασχολεί έχει να κάνει με την *επιλογή του περιεχομένου που αντιγράφεται* από τους πηγαίους εξυπηρετητές.

Σκοπός και συνεισφορά αυτής της διπλωματικής εργασίας ήταν:

- Η ανάπτυξη των αλγορίθμων όπου θα εντοπίζουν ιστοσελίδες και έπειτα κοινότητες που γίνονται δημοφιλείς σε μεγάλο ποσοστό χρηστών, καθώς και θα αντιγράφουν το περιεχόμενο στους κατάλληλους εξυπηρετητές αναπαραγωγής.
- Αξιολόγηση των προτεινόμενων αλγορίθμων και της απόδοσης του δικτύου. Θα πρέπει να δούμε πότε αξίζει να γίνεται η συγχώνευση των κοινού ενδιαφέροντος κοινοτήτων στους ίδιους εξυπηρετητές και σε τι βαθμό επηρεάζει την απόδοση του δικτύου.

1.2 Μεθοδολογία που ακολουθήθηκε

Αρχικά, χρειάστηκε να μελετηθεί η βιβλιογραφία που αφορά τα ΔΠΠ, ούτως ώστε να κατανοηθεί σε βάθος η σημασία τους στο χώρο της Πληροφορικής και η χρήση τους κυρίως στο Διαδίκτυο. Έπειτα, κύριο σημείο της μελέτης, ήταν ο ξεκάθαρος ορισμός του τι θεωρούμε ιστοσελίδες κοινού ενδιαφέροντος για τους χρήστες και τι είναι

βαθμός ομοιότητας δυο ιστοσελίδων. Αφού σκοπός της διπλωματικής ήταν από την αρχή ο έλεγχος του αν η συγχώνευση των δημοφιλών ιστοσελίδων σε κοινούς εξυπηρετητές αυξάνει την επίδοση του δικτύου, πρώτο βήμα ήταν φυσικά να καθορίσουμε τι θεωρούμε δημοφιλείς ιστοσελίδες – όχι η κάθε ιστοσελίδα από μόνη της, αλλά σύνολο ιστοσελίδων να έχουν την ίδια ζήτηση από τους χρήστες τους δικτύου.

Το επόμενο βήμα ήταν η σχεδίαση αλγορίθμου για την κατασκευή δομής συσχέτισης χρηστών με τις ιστοσελίδες που ζητούν από το δίκτυο. Η δυσκολία εδώ, ήταν κυρίως η μορφή που είχαν τα δεδομένα που μπορούσαμε να εξάγουμε από ήδη υπάρχοντα εργαλεία και το πώς θα γίνει η σωστή σύνδεση μεταξύ χρηστών και αιτημάτων σε ιστοσελίδες από διαφορετικούς εξυπηρετητές. Εδώ ήταν και η σημασία της μελέτης αυτής, αφού δεν έχει γίνει άλλη προσπάθεια συνένωσης περιεχομένων από διαφορετικούς πηγαίους εξυπηρετητές για αντιγραφή τους στους εξυπηρετητές αναπαραγωγής του ΔΠΠ. Ουσιαστικά, είχαμε στη διάθεσή μας ένα εργαλείο για τη δημιουργία αιτήσεων, το οποίο δίνει ρεαλιστικά δεδομένα αιτημάτων διαδικτύου από έναν εξυπηρετητή. Όμως η δομή που θα χρησιμοποιούταν από τους επόμενους αλγόριθμους θέλαμε να είναι η αποδοτικότερη, να μη δεσμεύεται αχρείαστος χώρος στη μνήμη του υπολογιστή αλλά και να μπορεί μετά να γίνεται εύκολη αναζήτηση και χρήση της. Έτσι, καταλήξαμε να κάνουμε δύο είδη αλγορίθμων – ένα με χρήση πίνακα και ένα με χρήση συνδεδεμένων λιστών – και έπειτα να τους συγκρίνουμε, όχι μόνο θεωρητικά αλλά και πρακτικά.

Το επόμενο βήμα ήταν η σχεδίαση και ανάλυση αλγορίθμου για την εύρεση βαθμού ομοιότητας δύο ιστοσελίδων από δύο διαφορετικούς πηγαίους εξυπηρετητές, σύμφωνα με τους χρήστες που τις ζητούν. Αυτός, θα χρησιμοποιηθεί από τον επόμενο αλγόριθμο, για την εύρεση ζευγών ιστοσελίδων που έχουν βαθμό ομοιότητας μεγαλύτερο από ένα όριο Χ. Στη συνέχεια, επόμενο βήμα ήταν η μετατροπή των δυο τελευταίων αλγορίθμων σε μορφή που να επιτρέπει εύρεση πλειάδων ιστοσελίδων με βαθμό ομοιότητας μεγαλύτερο από όριο Χ για k εξυπηρετητές (άρα κ-άδες) καθώς επίσης και η ανάλυση της χρονικής και χωρικής πολυπλοκότητας τους.

Τέλος, σχεδιάστηκε αλγόριθμος για την εύρεση και συνένωση κοινοτήτων κοινού ενδιαφέροντος, από διαφορετικούς πηγαίους εξυπηρετητές, με βαθμό ομοιότητας μεγαλύτερο από το όριο Υ, για την αντιγραφή τους στους κατάλληλους εξυπηρετητές αναπαραγωγής του συστήματος. Έγινε ανάλυση της πολυπλοκότητας και για αυτό τον αλγόριθμο και στο τέλος η πειραματική αξιολόγηση της μεθόδου μετά την υλοποίηση που περιγράφεται στην εργασία [1].

1.3 Οργάνωση/Δομή διπλωματικής

Στο Κεφάλαιο 2 παρουσιάζεται κάποιο γνωστικό υπόβαθρο, απαραίτητο για την πλήρη κατανόηση των εννοιών που χρησιμοποιούνται στη διπλωματική εργασία αλλά και την εκτίμηση της αξίας που έχει αυτή η μελέτη. Θα περιγραφεί αναλυτικά τι είναι τα Δίκτυα Παράδοσης Περιεχομένου, τι εξυπηρετούν στην Επιστήμη της Πληροφορικής και πως χρησιμοποιούνται. Ακόμα θα εξηγηθεί τι είναι οι Κοινότητες Ιστοσελίδων και ο Αλγόριθμος CiBC [3], έννοιες απαραίτητες για τα επόμενα κεφάλαια. Επεξηγούνται επίσης κάποιες ορολογίες που χρησιμοποιούνται σε όλη την έκταση της διπλωματικής, έννοιες που θα μας απασχολήσουν σημαντικά, αφού αποτελούν τη βάση της μελέτης αυτής.

Στο Κεφάλαιο 3 δίνεται αρχικά ο ορισμός για τις Κοινού Ενδιαφέροντος Ιστοσελίδες/Αντικείμενα (ΚΕΑ) και περιγράφονται αναλυτικά όλοι οι αλγόριθμοι που σχεδιάστηκαν για την εύρεση τους στα Δίκτυα Παράδοσης Περιεχομένου. Παρουσιάζεται επίσης η χρονική και χωρική πολυπλοκότητα του καθενός από αυτούς. Αντίστοιχα, στο Κεφάλαιο 4 δίνεται ο ορισμός για τις Κοινού Ενδιαφέροντος Κοινότητες (ΚΕΚ) και περιγράφονται οι αλγόριθμοι που σχεδιάστηκαν για την εύρεσή τους στα ΔΠΠ. Περιλαμβάνεται επίσης και η ανάλυση της πολυπλοκότητάς τους.

Μετά από την υλοποίηση και τα πειράματα που έγιναν στη διπλωματική [1], στο Κεφάλαιο 5 παρουσιάζονται κάποια συμπεράσματα από όλη την εργασία, καθώς και κάποιες εισηγήσεις για μελλοντικές εργασίες που μπορούν να γίνουν, είτε σαν προέκταση της μελέτης αυτής είτε σαν επιπλέον μελέτες στον τομέα των Δικτύων Παράδοσης Περιεχομένου.

Κεφάλαιο 2

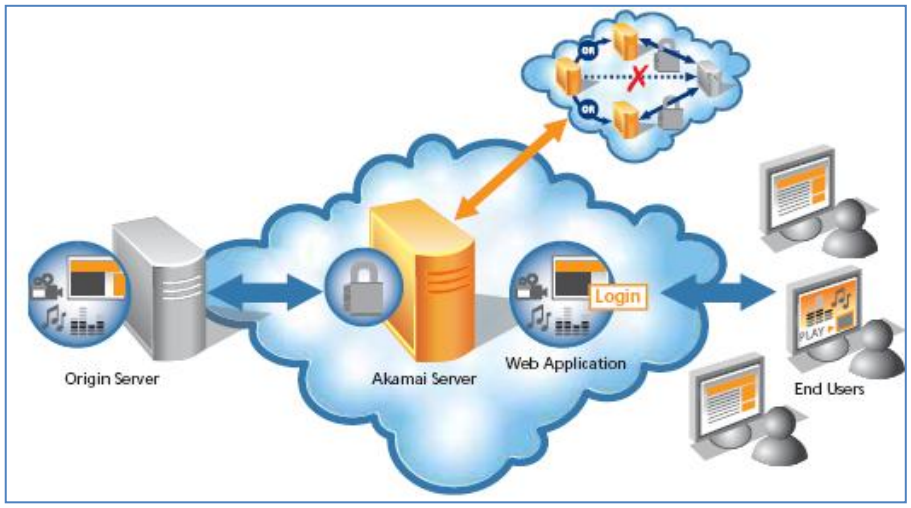
Γνωστικό Υπόβαθρο

2.1 Δίκτυα Παράδοσης Περιεχομένου

Τα Δίκτυα Παράδοσης Περιεχομένου (ΔΠΠ) [2,3,4,6] είναι δίκτυα με πολλαπλούς πηγαίους εξυπηρετητές (origin servers) στους οποίους βρίσκονται αποθηκευμένες ιστοσελίδες, εξυπηρετητές αναπαραγωγής (surrogate servers) – εξυπηρετητές που βρίσκονται σε διάφορα σημεία ανά το παγκόσμιο – στους οποίους αποθηκεύονται αντίγραφα από το περιεχόμενο των ιστοσελίδων των πηγαίων εξυπηρετητών, δρομολογητές (routers) και χρήστες (users). Σκοπός του συστήματος είναι να μεταδίδονται γρήγορα οι πληροφορίες στους χρήστες του διαδικτύου, για γρηγορότερη εξυπηρέτηση τους και να γίνεται καλύτερη διαχείριση των πληροφοριών που διακινούνται στο δίκτυο.

Σε ένα ΔΠΠ, υπάρχουν ουσιαστικά δύο ροές διακίνησης πληροφοριών, όπως φαίνεται και στο πιο κάτω Σχήμα.2.1., μια μεταξύ εξυπηρετητών αναπαραγωγής και χρηστών, και μια μεταξύ πηγαίων εξυπηρετητών και εξυπηρετητών αναπαραγωγής. Με τον τρόπο αυτό *μειώνεται η συμφόρηση* του δικτύου – οι αιτήσεις από τους χρήστες δεν γίνονται στους πηγαίους εξυπηρετητές και έτσι αποφεύγεται η συμφόρηση και οι καθυστερήσεις στη μετάδοση των μηνυμάτων και των πληροφοριών στο δίκτυο - και *αυξάνεται η διαθεσιμότητα και η κατανομή των περιεχομένων* – αφού υπάρχουν

αντίγραφα των ιστοσελίδων σε πολλούς εξυπηρετητές αναπαραγωγής και οι χρήστες μπορούν να εξυπηρετούνται γρηγορότερα, από διαφορετικούς εξυπηρετητές [4].



Σχήμα 2.1. Δίκτυο Παράδοσης Περιεχομένου [8]

Για να λειτουργεί σωστά και αποδοτικά ένα ΔΠΠ, λαμβάνονται υπόψη τρεις σημαντικοί παράγοντες, όπως αναφέρεται και προηγουμένως. *Η τοπολογία του δικτύου*, που είναι ουσιαστικά η γεωγραφική τοποθεσία των εξυπηρετητών αναπαραγωγής, *η επιλογή του περιεχομένου που αντιγράφεται* στους εξυπηρετητές αναπαραγωγής και *η πολιτική που ακολουθείται για την αντιγραφή* των περιεχομένων από τους πηγαίους εξυπηρετητές στους εξυπηρετητές αναπαραγωγής του δικτύου [2,4].

Όσο αφορά την τοπολογία του δικτύου, είναι σημαντικό να επιλεγούν οι θέσεις των εξυπηρετητών αναπαραγωγής με τέτοιο τρόπο ούτως ώστε να μεγιστοποιηθεί η απόδοση του δικτύου και να ελαχιστοποιηθεί το κόστος της υποδομής του. Όσο πιο πολλοί εξυπηρετητές αναπαραγωγής τόσο το καλύτερο θα έλεγε κανείς. Όμως δεν θα θέλαμε να έχουμε πολύ πυκνά τους εξυπηρετητές γιατί θα ήταν μεγαλύτερο το κόστος τους αλλά και το κόστος σε χρόνο και σε υπολογιστική δύναμη για την αντιγραφή των περιεχομένων και την συχνή ενημέρωσή τους σε πάρα πολλούς εξυπηρετητές.

Σημαντική είναι επίσης και η επιλογή του περιεχομένου που θα αντιγραφεί στους διάφορους εξυπηρετητές αναπαραγωγής, αφού η αντιγραφή ολόκληρων των ιστοσελίδων, αν και ιδανική δεν είναι εφικτή, λόγω περιορισμένης μνήμης στους εξυπηρετητές αλλά και λόγω δυσκολίας στην ανανέωση των περιεχομένων –

δημιουργούνται προβλήματα συνέπειας του περιεχομένου που βρίσκεται στους διάφορους εξυπηρετητές στο δίκτυο. Έτσι χρησιμοποιούνται διάφοροι τρόποι επιλογής, είτε με βάση δηλαδή τις προτιμήσεις των χρηστών, είτε με βάση τη συνδεσμολογία των ιστοσελίδων μεταξύ τους μέσω υπερσυνδέσμων.

Έχοντας λοιπόν την τοπολογία των εξυπηρετητών και το περιεχόμενο που πρέπει να αντιγραφεί, είναι σημαντικό να αποφασιστεί η πολιτική που θα ακολουθήσουμε για την αντιγραφή των περιεχομένων. Οι τρεις δημοφιλέστερες πολιτικές είναι:

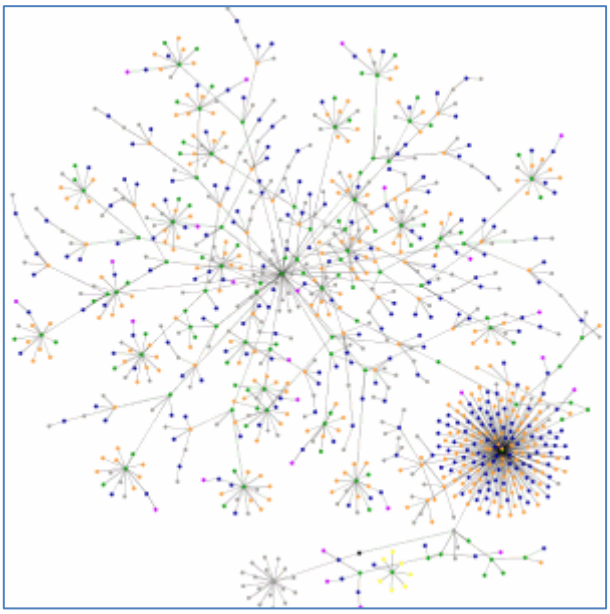
(1) η **προανάκτηση και συνεργασία (cooperative push-based),** όπου τα περιεχόμενα ανακτούνται από πριν στους κατάλληλους εξυπηρετητές αναπαραγωγής και μετά οι εξυπηρετητές συνεργάζονται για να κρατήσουν τα περιεχόμενά τους ενημερωμένα. Ανάλογα με την αίτηση του χρήστη επιλέγεται ο πιο κοντινός εξυπηρετητής που έχει το περιεχόμενο που ζητά ο χρήστης για να εξυπηρετήσει την αίτηση. Αν δεν υπάρχει σε εξυπηρετητής αναπαραγωγής τότε η αίτηση πηγαίνει στον πηγαίο εξυπηρετητή.

(2) η **ανάκτηση όταν χρειάζεται και χωρίς συνεργασία (uncooperative pull-based)**, όπου οι αιτήσεις πηγαίνουν στον κοντινότερο εξυπηρετητή αναπαραγωγής και αν το περιεχόμενο δεν υπάρχει στον εξυπηρετητή αυτό, η αίτηση μεταφέρεται είτε σε γειτονικό εξυπηρετητή αναπαραγωγής είτε στον πηγαίο εξυπηρετητή. Συγκεκριμένα αντιγράφεται το περιεχόμενο από τον πηγαίο εξυπηρετητή στον εξυπηρετητή αναπαραγωγής. Στην πολιτική αυτή οι εξυπηρετητές αναπαραγωγής δεν συνεργάζονται μεταξύ τους και έτσι δεν γνωρίζει ο ένας τι περιέχει ο γείτονάς του.

(3) η **ανάκτηση όταν χρειάζεται και συνεργασία (cooperative pull-based)**, όπου οι αιτήσεις πηγαίνουν στον κοντινότερο εξυπηρετητή αναπαραγωγής και αν το περιεχόμενο δεν υπάρχει στον εξυπηρετητή αυτό τότε αντιγράφεται από τον κοντινότερο εξυπηρετητή αναπαραγωγής που το περιέχει. Εδώ υπάρχει συνεργασία των εξυπηρετητών αφού γνωρίζουν από ποιον να κάνουν την αίτηση αν δεν έχουν οι ίδιοι το περιεχόμενο που ψάχνουν.

2.2 Κοινότητες Ιστοσελίδων και Αλγόριθμος CiBC

Κοινότητα Ιστοσελίδων (Web Community) [3], είναι μια ομάδα από ιστοσελίδες (web pages) που συνδέονται μεταξύ τους και ανήκουν στο σύνολο των ιστοσελίδων ενός ιστότοπου (website). Αν θεωρήσουμε τον ιστότοπο ένα γράφο, όπου οι κόμβοι είναι ιστοσελίδες και οι ακμές είναι οι σύνδεσμοι μεταξύ τους (hyperlinks), τότε οι κοινότητες, είναι ουσιαστικά υπο-γράφοι, οι οποίοι έχουν περισσότερες ακμές στο εσωτερικό τους (που συνδέουν δηλαδή τις ιστοσελίδες μεταξύ τους μέσα στην κοινότητα) και λιγότερες προς τα έξω της κοινότητας (που την συνδέουν με άλλες κοινότητες). Στο Σχήμα 2.2. πιο κάτω φαίνεται ένα παράδειγμα γράφου ιστότοπου όπου ξεχωρίζουν κατά κάποιο τρόπο μερικές κοινότητες ιστοσελίδων του. Βλέπουμε την πυκνή και αραιή συνδεσιμότητα σε μερικά σημεία όπου δημιουργούνται οι κοινότητες του γράφου.



Σχήμα 2.2. Γράφος Ιστότοπου με Κοινότητες Ιστοσελίδων [9]

Μια εύρωστη τεχνική που προτάθηκε για την επιλογή των περιεχομένων που αντιγράφονται στους εξυπηρετητές αναπαραγωγής ενός ΔΠΠ, είναι με την εύρεση κοινοτήτων στις ιστοσελίδες του πηγαίων εξυπηρετητών [3]. Ο αλγόριθμος CiBC προτάθηκε για να βρίσκει τις κοινότητες μεταξύ ιστοσελίδων σύμφωνα με τη Μεταξύ Κεντρικότητα (Betweenness Centrality) κάθε κόμβου, στο γράφο που αποτελούν τα περιεχόμενα ενός πηγαίου εξυπηρετητή (κάθε ιστοσελίδα θεωρείται κόμβος στο γράφο

και οι σύνδεσμοι μεταξύ τους αποτελούν τις ακμές). Η Μεταξύ Κεντρικότητα κάθε κόμβου είναι το πόσο κεντρικός είναι ο κόμβος στο γράφο. Αν έχει μεγάλη Μεταξύ Κεντρικότητα σημαίνει πως πολλά από τα μικρότερα μονοπάτια μεταξύ δυο κόμβων περνούν από εκεί και επομένως ο χρήστης είναι πιο πιθανό να ζητήσει αυτή την ιστοσελίδα αν είναι σε κάποια γειτονική. Ο αλγόριθμος CiBC υπολογίζει αρχικά την Μεταξύ Κεντρικότητα κάθε κόμβου και έπειτα, δημιουργεί τις κοινότητες γύρω από τους κόμβους που έχουν την ελάχιστη τιμή. Σύμφωνα με το βαθμό συνδεσιμότητας μεταξύ των κοινοτήτων μπορεί να ενώσει και κάποιες από τις κοινότητες κάνοντας τις μεγαλύτερες. Μετά, ανάλογα με τη μνήμη του εξυπηρετητή, μπορεί να κάνει αλλαγές στους κόμβους των κοινοτήτων, ούτως ώστε να χωρούν στη μνήμη των εξυπηρετητών αναπαραγωγής που θα αντιγραφούν.

Για τον αλγόριθμο αυτό, ο χρόνος που χρειάζεται για να υπολογιστούν οι κοινότητες είναι ανάλογος με τον αριθμό των κόμβων (δηλαδή με τον αριθμό των ιστοσελίδων του εξυπηρετητή) ενώ η επίδοσή του επηρεάζεται γραμμικά από τον αριθμό των ακμών (δηλαδή των συνδέσεων μεταξύ των ιστοσελίδων του εξυπηρετητή).

2.3 Γενικές Ορολογίες

Σε ολόκληρη τη διπλωματική εργασία χρησιμοποιούνται κάποιες συγκεκριμένες ορολογίες και για την πλήρη κατανόηση της μελέτης εξηγούνται στο μέρος αυτό, ούτως ώστε να γίνεται ξεκάθαρο σε όλους τους αναγνώστες το περιεχόμενο.

Αρχικά για τα Δίκτυα Παράδοσης Περιεχομένου αναφέρονται συχνά με τη συντομογραφία ΔΠΠ. Ακόμη, χρησιμοποιούνται οι όροι αντικείμενο ιστού (web-object) και υποκείμενο (subject) οι οποίοι αντιστοιχούν στους όρους ιστοσελίδα (web-page) και χρήστης (user). Σύνολα αντικειμένων ιστού θεωρούμε τους πηγαίους εξυπηρετητές ενώ καταχωρητές αντικειμένων ιστού θεωρούμε τους εξυπηρετητές αναπαραγωγής.

Ακόμη, είναι σημαντικό να αναφέρουμε πως για την ορολογία χρήστης, χρησιμοποιείται η έννοια της συνόδου (session) [5]. Αναγνωριστικό του χρήστη δεν

είναι κάποια IP διεύθυνση. Για να μπορούμε να ξεχωρίζουμε από ποιόν έγινε η κάθε αίτηση, χρησιμοποιείται η έννοια της συνόδου, στην οποία ανήκουν κάποιες αιτήσεις. Αν αλλάξει η σύνοδος τότε θεωρούμε πως είναι διαφορετικός χρήστης.

Πιο κάτω παρουσιάζεται ο Πίνακας2.1. με τις βασικές μεταβλητές που χρησιμοποιούνται από τους αλγορίθμους σε όλη την έκταση της διπλωματικής.

Μεταβλητή	Σημασία
O_i	Πηγαίος Εξυπηρετητής i
S	Σύνολο χρηστών που κάνει αιτήσεις στο δίκτυο
S_i	Σύνολο χρηστών που ζητούν την ιστοσελίδα i
X	Όριο κοινού ενδιαφέροντος Ιστοσελίδων
R	Δομή που περιέχει όλες τις αιτήσεις του δικτύου
w	Πλήθος αντικειμένων ιστού στα οποία γίνονται αιτήσεις
s	Πλήθος χρηστών που κάνει αιτήσεις στο δίκτυο
Structure	Δομή που αντικατοπτρίζει τη σχέση μεταξύ ιστοσελίδων και χρηστών που τις ζητούν
TotSubjects	Δομή που δείχνει πόσοι χρήστες ζήτησαν κάθε ιστοσελίδα του δικτύου
Origin	Δομή που δείχνει σε ποιο πηγαίο εξυπηρετητή ανήκει κάθε ιστοσελίδα του δικτύο
WC_i	Σύνολο κοινοτήτων από πηγαίο εξυπηρετητή i
wc_i	Κοινότητα από πηγαίο εξυπηρετητή i
u(wc)	Αριθμός μοναδικών χρηστών που κάνουν αίτηση σε αντικείμενα της κοινότητας wc
topK(wc)	Σύνολο από τις K πιο δημοφιλείς ιστοσελίδες της κοινότητας wc
z	Ποσοστό ζευγών κοινού ενδιαφέροντος ιστοσελίδων από το σύνολο των ζευγών ιστοσελίδων δυο κοινοτήτων
Y	Όριο κοινού ενδιαφέροντος Κοινοτήτων Ιστοσελίδων

Πίνακας 2.1. Σημασία Σημαντικότερων Μεταβλητών

Κεφάλαιο 3

Κοινού Ενδιαφέροντος Αντικείμενα

3.1 Ορισμός ΚΕΑ

Αρχικά, ορίσαμε τι θεωρούμε Ιστοσελίδες Κοινού Ενδιαφέροντος (ΙΚΕ) για τους χρήστες ή αλλιώς Κοινού Ενδιαφέροντος Αντικείμενα-Ιστού (ΚΕΑ) και τι είναι βαθμός ομοιότητας δυο ιστοσελίδων/αντικειμένων ιστού.

Δυο αντικείμενα ιστού θεωρούμε πως είναι κοινού ενδιαφέροντος, αν το ποσοστό των υποκειμένων, από το σύνολο όλων των υποκειμένων που κάνουν αιτήσεις στα σύνολα αντικειμένων ιστού, που ζητούν και τα δυο αντικείμενα, είναι μεγαλύτερο ή ίσο με **X,** όπου το **X** ορίζεται ως το όριο ποσοστού κοινού ενδιαφέροντος ή αλλιώς βαθμός ομοιότητας.

Συγκεκριμένα, έχουμε δυο αντικείμενα ιστού, Α και Β. Το σύνολο των υποκειμένων που κάνουν αιτήσεις στα σύνολα των αντικειμένων είναι S. Αν το ποσοστό των υποκειμένων από το σύνολο S, που ζητούν και τα δυο αντικείμενα, είναι περισσότερο από το ποσοστό **X** που θέτουμε ως όριο, τότε τα αντικείμενα ιστού Α και Β είναι κοινού

ενδιαφέροντος για τα υποκείμενα που τα ζητούν. Μας ενδιαφέρει όμως, τα αντικείμενα αυτά να είναι από διαφορετικά σύνολα αντικειμένων ιστού, ούτως ώστε η αντιγραφή τους σε καταχωρητές αντικειμένων να επιφέρει αποδοτικότερα αποτελέσματα (να είναι από διαφορετικούς πηγαίους εξυπηρετητές).

$O_1, O_2 : \{\sigma\acute{\upsilon}\nu o\lambda\alpha\ \alpha\nu\tau\iota\kappa\epsilon\iota\mu\acute{\epsilon}\nu\omega\nu\ \iota\sigma\tau o\acute{\upsilon}\}$

$S : \{\upsilon\pi o\kappa\epsilon\acute{\iota}\mu\epsilon\nu\alpha\ \pi o\upsilon\ \kappa\acute{\alpha}\nu o\upsilon\nu\ \alpha\iota\tau\acute{\eta}\sigma\epsilon\iota\varsigma\ \gamma\iota\alpha\ \alpha\nu\tau\iota\kappa\epsilon\acute{\iota}\mu\epsilon\nu\alpha\ \iota\sigma\tau o\acute{\upsilon}\}$

$A \in O_1$ *(A αντικείμενο ιστού από το σύνολο* O_1*)*

$B \in O_2$ *(B αντικείμενο ιστού από το σύνολο* O_2*)*

$S_A : \{\upsilon\pi o\kappa\epsilon\acute{\iota}\mu\epsilon\nu\alpha\ \pi o\upsilon\ \zeta\eta\tau o\acute{\upsilon}\nu\ \tau o\ \alpha\nu\tau\iota\kappa\epsilon\acute{\iota}\mu\epsilon\nu o\ A\}$

$S_B : \{\upsilon\pi o\kappa\epsilon\acute{\iota}\mu\epsilon\nu\alpha\ \pi o\upsilon\ \zeta\eta\tau o\acute{\upsilon}\nu\ \tau o\ \alpha\nu\tau\iota\kappa\epsilon\acute{\iota}\mu\epsilon\nu o\ B\}$

X: $\acute{o}\rho\iota o\ \pi o\sigma o\sigma\tau o\acute{\upsilon}\ \kappa o\iota\nu o\acute{\upsilon}\ \epsilon\nu\delta\iota\alpha\phi\acute{\epsilon}\rho o\nu\tau o\varsigma\ \alpha\nu\tau\iota\kappa\epsilon\iota\mu\acute{\epsilon}\nu\omega\nu$

$$If\ \frac{|S_A\ \cap\ S_B\ |}{|\ S\ |} \geq X\ \ then$$

$A\ \kappa\alpha\iota\ B\ \alpha\nu\tau\iota\kappa\epsilon\acute{\iota}\mu\epsilon\nu\alpha\ \kappa o\iota\nu o\acute{\upsilon}\ \epsilon\nu\delta\iota\alpha\phi\acute{\epsilon}\rho o\nu\tau o\varsigma$

Με τον ίδιο τρόπο, για **n** αντικείμενα ιστού, θεωρούμε πως είναι κοινού ενδιαφέροντος για τα υποκείμενα, αν το πλήθος των υποκειμένων που τα ζητούν δια το σύνολο όλων των υποκειμένων που κάνουν αιτήσεις στο δίκτυο, είναι ίσο ή περισσότερο από το όριο **X**. Είναι απαραίτητο τα αντικείμενα να προέρχονται από n διαφορετικά σύνολα αντικειμένων ιστού.

$$If\ \frac{|S_1\ \cap\ S_2\ \cap\ S_3 \cap \ldots \cap S_n|}{|\ S\ |} \geq X\ \ then$$

$1, 2, 3, \ldots n\ \alpha\nu\tau\iota\kappa\epsilon\acute{\iota}\mu\epsilon\nu\alpha\ \kappa o\iota\nu o\acute{\upsilon}\ \epsilon\nu\delta\iota\alpha\phi\acute{\epsilon}\rho o\nu\tau o\varsigma$

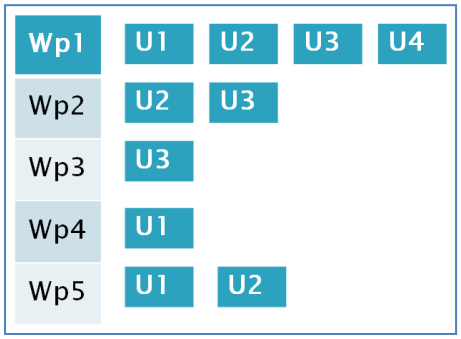
3.2 Αλγόριθμος κατασκευής δομής

Ο πρώτος αλγόριθμος που έπρεπε να σχεδιαστεί ήταν για τη δημιουργία μιας δομής που θα έκανε τη σύνδεση μεταξύ των χρηστών του δικτύου και των ιστοσελίδων που παρέχονται από τους πηγαίους εξυπηρετητές. Η λύση ήταν φυσικά μέσα από τις αιτήσεις των χρηστών. Χρησιμοποιώντας αρχικά μια γεννήτρια αιτήσεων (request generator) θα έχουμε σε αρχείο τις αιτήσεις των χρηστών σε τέτοια μορφή που θα καθιστά τη δημιουργία της δομής όσο πιο εύκολη γίνεται.

Στο σημείο αυτό, αξίζει να σημειωθεί πως ο αλγόριθμος είναι τέτοιος ώστε να κατασκευάζει τη δομή που εμείς επιθυμούμε, είτε είναι πίνακας, είτε συνδεδεμένες λίστες. Αυτό, γιατί θέλαμε να συγκρίνουμε την πολυπλοκότητα των δυο περιπτώσεων και να δούμε πότε είναι αποδοτικότερη η χρήση της κάθε μιας.

	User1	User2	User3
Wp1	1	1	1
Wp2	0	1	1
Wp3	0	0	1
Wp4	1	0	0
Wp5	1	1	0

Σχήμα 4.1. Πίνακας



Σχήμα 4.2. Συνδεδεμένες Λίστες

Ένα παράδειγμα κατασκευής δομής είναι το πιο πάνω, όπου το Σχήμα 4.1. είναι δυσδιάστατος πίνακας αντικειμένων – χρηστών, ενώ το Σχήμα 4.2. είναι μονοδιάστατος πίνακας των αντικειμένων ιστού με δείχτες σε συνδεδεμένες λίστες από τους χρήστες που κάνουν αιτήσεις σε αυτές.

3.2.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός έχει σαν εισόδους: τις αιτήσεις των υποκειμένων για τα σύνολα από αντικείμενα ιστού - που δημιουργήθηκαν από τη γεννήτρια αιτήσεων - το πλήθος των αντικειμένων, το πλήθος των υποκειμένων και τον τύπο της δομής που θέλουμε να δημιουργηθεί. Σκοπός της υπό-ρουτίνας αυτής είναι η κατασκευή της δομής συσχέτισης

αντικείμενα-υποκείμενα είτε με πίνακα είτε με συνδεδεμένη λίστα. Θέλουμε επίσης να δημιουργηθούν οι πίνακες **TotSubjects** και **Origin** οι οποίοι αντιπροσωπεύουν το σύνολο των υποκειμένων που ζήτησαν κάποιο αντικείμενο ιστού και τον αντιπροσωπευτικό αριθμό του συνόλου αντικειμένων (πηγαίο εξυπηρετητή) στον οποίο ανήκει το καθένα, αντίστοιχα. Οι πίνακες αυτοί θα βοηθήσουν στους επόμενους αλγόριθμους και είναι καλύτερο να δημιουργηθούν καθώς δημιουργείται και η δομή συσχέτισης χρηστών με ιστοσελίδες **Structure.**

Επομένως ακολουθούνται τα εξής βήματα:

1) Οι αιτήσεις των υποκειμένων (χρήστες), το πλήθος των υποκειμένων, το πλήθος των αντικειμένων ιστού (ιστοσελίδες) και ο τύπος της δομής που θα δημιουργηθεί από τον αλγόριθμο δίνονται σαν είσοδος στην υπο-ρουτίνα.
2) Με βάση τις αιτήσεις και τον τύπο που δόθηκαν, δημιουργείται δομή (**Structure**) που αντιστοιχεί κάθε υποκείμενο με τα αντικείμενα ιστού που ζήτησε. Καθώς δημιουργείται η δομή Structure, υπολογίζεται και ο συνολικός αριθμός των υποκειμένων που ζητούν κάθε αντικείμενο ξεχωριστά (**TotSubjects**), όπως επίσης και το σύνολο αντικειμένων ιστού στο οποίο ανήκει το καθένα (**Origin**).
3) Αποτέλεσμα του αλγορίθμου είναι η κατασκευή των τριών δομών που θα δοθούν σε επόμενους αλγόριθμους σαν δεδομένα εισόδου.

Input: R: set of requests,
w : amount of web-objects
s : amount of subjects requesting web-objects in the network
type: Boolean variable, 'false' for array and 'true' for linked-list

Output: Structure, TotSubjects, Origin

Structures: we consider the subjects to have names from 0 to **s** and web-objects from 0 to **w**. They are created this way from the script creating the requests [1].
TotSubjects : array where TotSubjects[i] is the amount of subjects requesting the i^{th} web-object
Origin: array where Origin[i] is the identity of the set in which the i^{th} web-object belongs to (origin server identifier)

Algorithm:

```
1:  Create TotSubjects[ w ] and initialize all with 0
2:  Create Origin[ w ] and initialize all with 0
3:  Create Structure, TotSubjects and Origin:
4:  If ( type == false ) then  //array
5:      Create U[ w ][ s ] and Initialize all with 0
6:      For all r ∈ R Do
7:          If ( Structure[r.web-object][r.subject] == 1 ) then //check if the same request
8:              Continue;                    //has already been marked in Structure
9:          Structure[r.web-object][r.subject] = 1  //web-object is requested by subject
```

```
10:             TotSubjects[r.web-object] ++;
11:             Origin[r.web-object] = r.set
12:     End for
13: Else  //linked-list
14:     Create Structure[ w ] and initialize all with null
15:     For all r ∈ R Do
16:             Node *head = Structure[r.web-object].head
17:             Node *tail = Structure[r.web-object].tail
18:             Node n = new Node()
19:             n.subject = r.subject
20:             n.next = null
21:             If ( head == NULL ) then  //if the new subject is the first to be in the list add it
22:                     Structure[r.web-object].head = n
23:                     Structure[r.web-object].tail = n
24:                     TotSubjects[r.web-object]++
25:                     Origin[r.web-object] = r.set
26:                     continue
27:             End if
28:             If ( r.subject < head→subject ) then //if the new subject has smaller ID
29:                     n.next = head           // than the first on the list, put it first
30:                     Structure[r.web-object]. head = n
31:                     TotSubjects[r.web-object]++
32:                     Origin[r.web-object] = r.set
33:             Else if ( r.subject > tail→subject ) then //if the new subject has larger ID
34:                                             // than the last on the list, put it last
35:                     Tail→next = n
36:                     Structure[r.web-object].tail = n
37:                     TotSubjects[r.web-object]++
38:                     Origin[r.web-object] = r.set
39:             Else            //search the list to find the right place for the new subject
40:                     Node *cur = head
41:                     Node *pre = null
42:                     While (cur != null ) Do
43:                             If ( cur→subject > r.subject ) then //the new node must be
44:                                     n.next = cur      // placed before the current node
45:                                     pre→next = n
46:                                     TotSubjects[r.web-object] ++
47:                                     Origin[r.web-object] = r.set
48:                             Else if ( cur→subject == r.subject ) then //the same
49:                                     Free(n)         //subject has already been added
50:                                     Break;
51:                             Else
52:                                     pre = cur
53:                                     cur = cur→next
54:                             End if
55:                     End while
56:             End if
57:     End for
58: End if
59: Return <Structure, TotSubject, Origin>
```

Στις πρώτες 2 γραμμές του αλγορίθμου γίνεται η αρχικοποίηση των πινάκων TotSubjects και Origin σύμφωνα με το πλήθος των αντικειμένων του δικτύου. Στις γραμμές 4 – 12, είναι η περίπτωση δημιουργίας πίνακα για τη συσχέτιση χρηστών και ιστοσελίδων. Αρχικοποιείται ο πίνακας Structure και έπειτα για κάθε αίτηση, ελέγχεται αν έχει ήδη καταχωρηθεί και αν όχι, τοποθετείται στην κατάλληλη θέση του πίνακα ο αριθμός 1 (δείκτης - flag), αυξάνεται ο αριθμός αιτήσεων του αντικειμένου που αντιστοιχεί στην αίτηση – κατάλληλη θέση στον πίνακα TotSubjects – και σημειώνεται ο πηγαίος εξυπηρετητής του αντικειμένου στην αντίστοιχη θέση του πίνακα Origin.

Στις γραμμές 13 – 58, είναι η περίπτωση δημιουργίας συνδεδεμένων λιστών για κάθε αντικείμενο του δικτύου. Αρχικά, δημιουργείται και αρχικοποιείται ο πίνακας Structure, ο οποίος έχει θέσεις για όλα τα αντικείμενα ιστού. Έπειτα, για κάθε αίτηση που γίνεται στο δίκτυο, βρίσκεται η λίστα που αντιστοιχεί στην ιστοσελίδα που αφορά η αίτηση και δημιουργείται ένας καινούριος κόμβος για να βρεθεί η κατάλληλη θέση εισαγωγής του στη λίστα. Στις γραμμές 21 – 27 γίνεται έλεγχος αν η λίστα είναι άδεια, ούτως ώστε να μπει ο νέος κόμβος αμέσως στην κατάλληλη θέση και να μην γίνουν επιπρόσθετοι έλεγχοι. Έπειτα, ελέγχονται 2 περιπτώσεις ακόμα πριν αρχίσει να γίνεται σειριακή αναζήτηση στην υπάρχουσα λίστα. Η πρώτη (γραμμές 28 – 32) είναι αν ο χρήστης που κάνει την αίτηση έχει μικρότερη ταυτότητα (id) από αυτόν που είναι στην αρχή της λίστας, για να μπει ο νέος στην αρχή και να είναι ταξινομημένοι. Η δεύτερη (γραμμές 33- 38) είναι αν ο χρήστης που κάνει την αίτηση έχει μεγαλύτερη ταυτότητα από αυτόν που είναι στο τέλος της λίστας, ούτως ώστε να μπει ο νέος στο τέλος απ’ ευθείας. Αν δεν ισχύει καμία από τις περιπτώσεις, τότε θα εκτελεστεί το επόμενο κομμάτι του αλγορίθμου (γραμμές 39 – 56) όπου γίνεται σειριακή αναζήτηση στη λίστα για την κατάλληλη θέση του νέου κόμβου ούτως ώστε να παραμένει ταξινομημένη η λίστα. Επέλεξα να γίνεται ταξινομημένη εισαγωγή των δεδομένων, ούτως ώστε στους επόμενους αλγόριθμους που θα χρειάζεται αναζήτηση να γίνεται πιο γρήγορα. Αν οι λίστες που δημιουργούνται είναι αρκετά μεγάλες, ο χρόνος που ξοδεύεται στην αρχή για τη δημιουργία τους θα αξίζει γιατί θα κερδίζει ο αλγόριθμος στο χρόνο αναζήτησης αργότερα.

3.2.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Αφού σχεδιάστηκε ο αλγόριθμος, πρέπει να γίνει και η ανάλυση της πολυπλοκότητάς του, χρονικής και χωρικής ούτως ώστε να δούμε το μέγεθος χρόνου και χώρου της λύσης που προτείνουμε. Η ανάλυση θα δείξει κατά πόσο αξίζει να τους υλοποιήσουμε και θα μας δώσει τις πρώτες ενδείξεις για το τι θα περιμένουμε από την πειραματική αξιολόγηση των αλγορίθμων.

Συγκεκριμένα, ο αλγόριθμος κατασκευής δομής είναι αρκετά σημαντικός όσο αφορά τη δομή που θα χρησιμοποιήσουμε τελικά στα πειράματα. Και όπως φαίνεται πιο κάτω, ανάλογα με τη δομή που επιλέγουμε έχουμε να κάνουμε κάποιους συμβιβασμούς. Η χρησιμοποίηση λίστας γλυτώνει πόρους από το πρόγραμμα αφού φυλάγονται μόνο τα χρήσιμα δεδομένα, ενώ ο πίνακας δεσμεύει χώρο για όλες τις περιπτώσεις αιτήσεων είτε έγιναν είτε όχι. Αντίθετα, όσο αφορά τον χρόνο, τα δεδομένα του πίνακα μπορούν να ανακτηθούν αμέσως σε χρόνο Θ(1), ενώ οι λίστες χρειάζονται περισσότερο χρόνο να δημιουργηθούν ταξινομημένες ούτως ώστε να κάνουν αργότερα την αναζήτηση γρηγορότερη, και ο χρόνος προσπέλασης κάποιου κόμβου εξαρτάται από τη θέση του στη λίστα. Έτσι, αναλύοντας σε βάθος τον αλγόριθμο προκύπτουν οι πιο κάτω πολυπλοκότητες.

<u>Χρονική πολυπλοκότητα:</u>

Οι δομές Origin και TotSubjects είναι οι ίδιες ανεξάρτητα από τη δομή του Structure που θα ζητείται. Αρχικοποιούνται στην αρχή του αλγορίθμου και χρειάζεται χρόνος Θ(w) για τον κάθε πίνακα, αφού έχουν size ίσο με το πλήθος των web-objects του δικτύου.

Στη συνέχεια του αλγορίθμου υπάρχουν δυο περιπτώσεις που πρέπει να αναλυθούν ξεχωριστά, για πίνακα και για συνδεδεμένη λίστα, για να γίνει αργότερα η σύγκριση μεταξύ τους και να βρούμε σε ποιες περιπτώσεις είναι καλύτερα να χρησιμοποιείται η κάθε δομή. Δεν λαμβάνεται υπ' όψη η αρχικοποίηση πιο πάνω, αφού είναι κοινή για τις δυο περιπτώσεις.

Χρησιμοποιώ συμβολισμό **T** για **time**.

Για πίνακα (αναφορά στις γραμμές του αλγορίθμου):

5: χρειάζεται χρόνος Θ(w*s) για να αρχικοποιηθεί η δομή Structure με 0 σε όλες τις θέσεις.

6 - 12: για κάθε αίτηση r, χρειάζονται:

- χρόνος Θ(1) για να ελεγχτεί η θέση στον πίνακα Structure που αντιστοιχεί με την αίτηση του r.web-object από το r.subject,
- χρόνος Θ(1) για να θέσει 1 την τιμή του πίνακα στη θέση που αντιστοιχεί με την αίτηση του r.web-object από το r.subject,
- χρόνος Θ(1) για να αυξήσει τον μετρητή για τα υποκείμενα που ζητούν το αντικείμενο ιστού,
- χρόνος Θ(1) για να θέσει το σύνολο των αντικειμένων ιστού στο οποίο ανήκει το αντικείμενο της αίτησης.

Έτσι έχουμε συνολικά χρόνο $T_{array} = \Theta(w*s) + \Theta(|R|)*\Theta(1) \rightarrow$ $\boxed{T_{array} = \Theta\,(w*s + |R|)}$

Για συνδεδεμένη λίστα:

14: χρειάζεται χρόνος Θ(w) για την αρχικοποίηση όλων των θέσεων του πίνακα Structure.

15 – 57: για κάθε αίτηση r, χρειάζονται:

- χρόνος Θ(1) για κάθε γραμμή μεταξύ 16 και 20 και
- Για τις 4 περιπτώσεις:
    1. αν το υποκείμενο που κάνει την αίτηση είναι το πρώτο που ελέγχεται για το συγκεκριμένο αντικείμενο ιστού, τότε ο νέος κόμβος θα μπει χωρίς ψάξιμο στην πρώτη θέση της αντίστοιχης λίστας. Χρειάζεται χρόνος Θ(1) για κάθε γραμμή μεταξύ 21 και 27
    2. αν το υποκείμενο που κάνει την αίτηση είναι μικρότερο από το πρώτο υποκείμενο (head) που έχει στη λίστα του το αντικείμενο, χρειάζεται χρόνος Θ(1) για κάθε γραμμή μεταξύ 28 και 32
    3. αν το υποκείμενο που κάνει την αίτηση είναι μεγαλύτερο από το τελευταίο υποκείμενο (tail) που έχει στη λίστα του το αντικείμενο, χρειάζεται χρόνος Θ(1) για κάθε γραμμή μεταξύ των 33 και 38
    4. αν το υποκείμενο που κάνει την αίτηση είναι μεταξύ του πρώτου (head) και του τελευταίου (tail), χρειάζεται χρόνος Θ(1) για κάθε γραμμή μεταξύ 39 και 41 για τους δείχτες που θα χρησιμοποιηθούν στην αναζήτηση, χρόνος T_{search} για να

προσπελάσει τη λίστα με τα υποκείμενα του αντίστοιχου αντικειμένου, μέχρι να βρεθεί η κατάλληλη θέση για το νέο υποκείμενο, και Θ(1) για κάθε γραμμή μεταξύ 43 και 47 για την πρόσθεση του νέου κόμβου, είτε για τις γραμμές 48 με 50 σε περίπτωση που το υποκείμενο αυτό υπάρχει ήδη στη λίστα.

Στο T_{search} περιλαμβάνονται οι γραμμές 51 με 53 για κάθε κόμβο που περνιέται οι οποίες χρειάζονται χρόνο Θ(1). Έστω ότι έχουμε ένα αντικείμενο ιστού, i, στο οποίο γίνεται η αίτηση από ένα υποκείμενο. Το αντικείμενο αυτό, ζητούν συνολικά s_i υποκείμενα. Ο χρόνος που χρειάζεται για να βρεθεί η κατάλληλη θέση ενός υποκειμένου το οποίο κάνει αίτηση σε αυτό το αντικείμενο θα είναι το πολύ s_i, αν χρειαστεί να προσπελάσει ολόκληρη τη λίστα με τα υποκείμενα. Έτσι, $T_{search} = O(s_i)$ για το αντικείμενο ιστού i στη χειρότερη περίπτωση.

Κάθε φορά που εισάγεται ένα νέο υποκείμενο στη λίστα s_i η λίστα μεγαλώνει κατά 1, έτσι για το επόμενο μπορεί να γίνουν περισσότερες συγκρίσεις μέχρι να βρεθεί η κατάλληλη θέση που του αντιστοιχεί.

Έτσι λοιπόν ο συνολικός χρόνος $T_{linked\text{-}list} = \Theta(w) + \Theta(|R|)*O(s_i)$ (όπου i αντικείμενο και s_i το συνολικό πλήθος των υποκειμένων που το ζητούν)

→ $T_{linked\text{-}list} = \Theta(w) + O(\sum_{i=0}^{w} s_i)$ $T_{linked\text{-}list} = \Theta(w) + O(\sum_{\kappa=0}^{w}(\sum_{m=0}^{\kappa} m))$

Η *χειρότερη περίπτωση* είναι όταν όλα τα υποκείμενα ζητούν όλα τα αντικείμενα ιστού και οι αιτήσεις για κάθε αντικείμενο γίνονται με τέτοιο τρόπο που πρέπει να γίνεται η μεγαλύτερη αναζήτηση στη λίστα των υποκειμένων του αντικειμένου, με αποτέλεσμα ο χρόνος να γίνεται $T_{linked\text{-}list} = \Theta(w) + \Theta(\sum_{i=0}^{w} s_i)$ και το s_i γίνεται ίσο με το πλήθος των υποκειμένων, s. Επομένως $\mathbf{T_{linked\text{-}list}} = \Theta(w) + \Theta(w*s) = \mathbf{\Theta(w + w*s)}$

Η *καλύτερη περίπτωση* είναι όταν οι αιτήσεις για κάθε αντικείμενο ιστού γίνονται με φθίνουσα ή αύξουσα σειρά με αποτέλεσμα οι κόμβοι της λίστας να προστίθενται χωρίς να γίνεται αναζήτηση στη λίστα των υποκειμένων του και έτσι ο συνολικός χρόνος να γίνεται $\mathbf{T_{linked\text{-}list}} = \Theta(w) + \Theta(|R|) = \mathbf{\Theta(w + |R|)}$

Ακόμα και η καλύτερη περίπτωση στη σειρά των αιτήσεων να ισχύει, αν οι αιτήσεις περιλαμβάνουν όλα τα υποκείμενα να ζητούν όλα τα αντικείμενα ο χρόνος θα γίνει ίσος με τον χειρότερο γιατί θα ισχύει |R| = w*s.

Γενικά, το σύνολο των αιτήσεων έχει μέγεθος $|R| = \sum_{k=0}^{w} s_k$ αφού κάθε αίτηση αντιστοιχεί σε ένα υποκείμενο που ζητά ένα από τα αντικείμενα. Στην περίπτωση της συνδεδεμένης λίστας, κάθε αίτηση αντιστοιχεί σε ένα κόμβο λίστας, ενώ στην περίπτωση του πίνακα, είναι απλά μια επανάληψη που κάνει την αντίστοιχη θέση του πίνακα από 0 σε 1.

Όπως φαίνεται από τους χρόνους των δυο περιπτώσεων, για πίνακα, ο χρόνος είναι ξεκάθαρος για τη δημιουργία της δομής $\mathbf{T_{array} = \Theta(w*s + |R|) = \Theta(w*s + \sum_{i=0}^{o} s_i)}$. Αντιθέτως, για συνδεδεμένη λίστα, ο χρόνος διαμορφώνεται ανάλογα με το πόσο πυκνή είναι η σύνδεση μεταξύ χρηστών και ιστοσελίδων αλλά και με τη σειρά που εισάγονται οι αιτήσεις στο πρόγραμμα. Αυτό, γιατί, αν η σύνδεση μεταξύ χρηστών και ιστοσελίδων είναι μεγάλη, ο χρόνος κατασκευής των λιστών θα είναι ίσος ή μεγαλύτερος από αυτόν του πίνακα. Και αν η σειρά των αιτήσεων είναι τέτοια που αναγκάζει το πρόγραμμα να κάνει πολλές επαναλήψεις στην αναζήτηση της κατάλληλης θέσης, ο χρόνος θα είναι σαφώς μεγαλύτερος από αυτόν του πίνακα. Γενικά, $\mathbf{T_{linked\text{-}list} = O(w + \sum_{i=0}^{o} s_i)}$. Ο χρόνος κατασκευής της δομής αυτής θα είναι ουσιαστικά καλύτερος από τον χρόνο του πίνακα στην περίπτωση που η σύνδεση μεταξύ χρηστών και ιστοσελίδων δεν είναι πολύ μεγάλη. Για να δούμε όμως και τι γίνεται με τον χώρο που χρειάζονται οι δυο δομές.

<u>Χωρική πολυπλοκότητα:</u>

Στον αλγόριθμο αυτό, η μόνη δομή που διαφέρει στις δυο περιπτώσεις, χρήσης πίνακα ή συνδεδεμένης λίστας, είναι η Structure, η οποία και θα έχει διαφορετική απαίτηση και σε χώρο. Οι υπόλοιπες δομές, TotSubjects και Origin θα έχουν τον ίδιο χώρο και για τις δυο περιπτώσεις.

Αρχικά οποιοσδήποτε τύπος απλής μεταβλητής, όπως integer ή Boolean είναι χώρου Θ(1).

Χρησιμοποιώ το συμβολισμό **S** για το **space**:

Χώρος για τον πίνακα TotSubjects: $S_{TotSubjects} = \Theta(w)*\Theta(1) = \Theta(w)$

Χώρος για τον πίνακα Origin: $S_{Origin} = \Theta(w)*\Theta(1) = \Theta(w)$

Για πίνακα:

$S_{StructureTable} = \Theta(w*s)*\Theta(1) = \Theta(w*s)$

Για συνδεδεμένη λίστα:

$S_{StructureLinked\text{-}list} = \Theta(w)*\Theta(2)$ *(χώρος για δείχτες σε head και tail nodes)* $+ \Theta(|R|)*\Theta(2)$ *(χώρος για για τον αριθμό του subject + δείχτης στο επόμενο node, για όλα τα nodes των λιστών)*

Εδώ πρέπει να σημειώσω πως το σύνολο R (requests) έχει μέγεθος το λιγότερο **w** και το μέγιστο **w*s**, αφού θεωρούμε πως για να υπάρχει ένα αντικείμενο ιστού στα δεδομένα εισόδου πρέπει να έχει γίνει κάποια αίτηση για αυτό, έστω από ένα υποκείμενο (χρήστη του δικτύου) και η μεγαλύτερη σχέση που μπορεί να υπάρχει μεταξύ τους είναι όλα τα υποκείμενα να κάνουν αίτηση για όλα τα αντικείμενα, οπότε και το |R| θα έχει τη μέγιστη τιμή που προανέφερα.

Επομένως, όσο αφορά το χώρο που χρειάζεται η κάθε περίπτωση, λαμβάνοντας υπ' όψη και το χώρο των δυο κοινών πινάκων, για πίνακα ο συνολικός χώρος είναι $S_{table} = \Theta(w*s + 2*w)$, ενώ για συνδεδεμένες λίστες $S_{linked\text{-}list} = \Theta(2*w + 2*|R| + 2*w) = \Theta(2*|R| + 4*w)$. Για να μπορούμε να τα συγκρίνουμε πρέπει να αφήσουμε μόνο τις ίδιες μεταβλητές. Και αφού γνωρίζουμε πως $w \leq |R| \leq w*s$, αυτό σημαίνει πως $\Theta(2*w + 4*w) = \Theta(6*w) = \Theta(w) \leq S_{linked\text{-}list} \leq \Theta(2*w*s + 4*w) = \Theta(w*s + w)$

➔ $\Theta(w) \leq S_{linked\text{-}list} \leq \Theta(w*s + w)$ → **$S_{linked\text{-}list} = O(w*s)$ και $S_{linked\text{-}list} = \Omega(w)$**

➔ $S_{table} = \Theta(w*s + 2*w)$ → **$S_{table} = \Theta(w*s)$**

3.3 Αλγόριθμος Similarity2wo: Βαθμός Ομοιότητας 2 Αντικειμένων

Αφού σχεδιάστηκε ο αλγόριθμος κατασκευής δομής η οποία συνδέει τους χρήστες με τις αιτήσεις που κάνουν στις ιστοσελίδες του δικτύου, έπρεπε τώρα να γίνει ο αλγόριθμος ο οποίος θα βρίσκει το ποσοστό ομοιότητας 2 αντικειμένων, τα οποία είναι από διαφορετικούς πηγαίους εξυπηρετητές. Η υπό-ρουτίνα αυτή, μαζί με τον πρώτο αλγόριθμο θα χρησιμοποιηθούν αργότερα από τα επόμενα στάδια της μεθοδολογίας, μέχρι να εκπληρωθεί πλήρως ο τελικός σκοπός. Ουσιαστικά εδώ χρησιμοποιείται ο ορισμός ΚΕΑ, χωρίς να γίνεται ο έλεγχος στο τέλος αν το ποσοστό ομοιότητας ξεπερνά κάποιο όριο.

3.3.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός παίρνει σαν είσοδο τη δομή **Structure** που δημιουργείται από τον πρώτο αλγόριθμο και τις τιμές **w** και **s.** Επιπρόσθετη είσοδο θεωρούμε δυο αντικείμενα ιστού από διαφορετικούς πηγαίους εξυπηρετητές, για τα οποία θα υπολογίσει το ποσοστό κοινού ενδιαφέροντος. Ουσιαστικά θα βρει το ποσοστό των υποκειμένων, από το σύνολο των υποκειμένων που κάνουν αιτήσεις στο σύστημα, ζητούν και τα δυο αντικείμενα ιστού.

1) Τα δυο αντικείμενα (A και B) που θα ελεγχτούν από τον αλγόριθμο δίνονται σαν είσοδος, μαζί με τη δομή Structure που θα χρησιμοποιηθεί και τις τιμές w και s.
2) Από τη δομή **Structure** βρίσκουμε τις αντίστοιχες θέσεις που αφορούν τα δυο αντικείμενα και μετρούμε πόσα υποκείμενα ζήτησαν και τα δυο αντικείμενα αυτά (A και B).
3) Τέλος, η υπό-ρουτίνα επιστρέφει το αποτέλεσμα του ποσοστού ομοιότητας.

Input: A, B: web objects from sets O_1 and O_2
Structure
w, s
Output: similarity percentage of the 2 web objects
Structures: we consider the subjects to have names from 0 to **s** and web-objects from 0 to **w**.
Structure : shows the relation between subjects and web-objects
Algorithm:

```
1:  Initialize count with 0//count is a counter for the number of subjects requesting both
                           //A and B
2:  If ( Structure is an array ) then //array
3:      For  i=0 to s Do            //for every subject increase the count variable
4:          If ( Structure[A][i] == 1 and Structure[B][i] == 1 ) then //if the subject requested
5:              count += 1                                          //both A and B
6:          End if
7:      End for
8:  Else  //linked-list
9:      Node *cur1 = Structure[A].head,  *cur2 = Structure[B].head
10:     While( cur1 != NULL and cur2 != NULL ) Do
11:         If( cur1→subject == cur2→subject ) then
12:             count++
13:             cur1 = cur1→next
14:             cur2 = cur2→next
15:         Else if( cur1→subject < cur2→subject ) then
16:             cur1 = cur1→next
17:         Else
18:             cur2 = cur2→next
19:         End if
20:     End while
21: End if
22: Return (count/s)      //subjects requesting both A and B/all subjects requesting web-objects
```

Στον αλγόριθμο πιο πάνω βλέπουμε και πάλι τις δυο περιπτώσεις εκτελέσεων ανάλογα με το τι ζητήθηκε από τον πρώτο αλγόριθμο να υλοποιηθεί, πίνακας ή συνδεδεμένες λίστες. Στην πρώτη περίπτωση εκτελούνται οι γραμμές 2 – 7 όπου είναι μια επαναληπτική διαδικασία κατά την οποία για κάθε υποκείμενο ελέγχεται αν έχει ζητήσει και τα δυο αντικείμενα ιστού που δίνονται στην είσοδο. Αν ναι, τότε αυξάνεται ο μετρητής. Στη δεύτερη περίπτωση εκτελούνται οι γραμμές 8 – 21 όπου χρησιμοποιούνται δυο προσωρινοί δείχτες, υπεύθυνοι για τις λίστες με τα υποκείμενα των δυο αντικειμένων που μας ενδιαφέρουν. Οι δυο λίστες προσπελαύνονται ταυτόχρονα και αν υπάρχει το ίδιο υποκείμενο και στις δυο λίστες τότε αυξάνεται ο μετρητής. Η αναζήτηση αυτή γίνεται πιο εύκολη αφού προνοήσαμε για την ταξινομημένη τοποθέτηση των υποκειμένων στις λίστες από τον πρώτο αλγόριθμο και έτσι θα είναι και πιο γρήγορη. Και στις δυο περιπτώσεις στο τέλος γίνεται διαίρεση με τον συνολικό αριθμό υποκειμένων του δικτύου, για να βρεθεί το ποσοστό.

3.3.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Η ανάλυση της χρονικής και χωρικής πολυπλοκότητας είναι απαραίτητη σε κάθε αλγόριθμο που σχεδιάζουμε, αφού μόνο έτσι μπορούμε να δούμε τι να περιμένουμε από την υλοποίηση σε θέματα χρόνου αλλά και πόρων που θα δεσμεύονται.

Ο αλγόριθμος αυτός όπως φαίνεται και πιο κάτω στην ανάλυση έχει χρειάζεται χρόνο ανάλογο με τον αριθμό των χρηστών που κάνουν αιτήσεις στο δίκτυο και χώρο ανάλογο με το σύνολο των αιτήσεων που έγιναν. Αυτό, γιατί πρέπει να ελεγχτεί πόσα από όλα τα υποκείμενα ζήτησαν τα δυο συγκεκριμένα αντικείμενα που μας ενδιαφέρουν και γιατί χρησιμοποιείται η δομή Structure που δημιουργήθηκε από τον προηγούμενο αλγόριθμο.

<u>Χρονική πολυπλοκότητα:</u>

Ο χρόνος που χρειάζεται ο αλγόριθμος αυτός χωρίζεται σε δυο περιπτώσεις, ανάλογα με τη δομή Structure που χρησιμοποιείται. Εξαιρείται η πρώτη γραμμή που είναι ίδιας χρονικής πολυπλοκότητας και για τις δυο περιπτώσεις και έχει χρόνο Θ(1).

<u>Για πίνακα:</u>

οι γραμμές 2 – 7 όπου γίνεται μια επανάληψη για κάθε υποκείμενο (διασχίζονται ταυτόχρονα δυο γραμμές του πίνακα) με αποτέλεσμα ο χρόνος να είναι **$T = \Theta(s)$**

<u>Για λίστες:</u>

οι γραμμές 8 – 21, όπου ουσιαστικά διατρέχονται οι συνδεδεμένες λίστες των αντικειμένων Α και Β, με αποτέλεσμα ο χρόνος να είναι **<u>$T = O(s)$</u>**

Αυτό γιατί στην *καλύτερη περίπτωση* οι δυο λίστες των αντικειμένων θα είναι άδειες με αποτέλεσμα ο χρόνος να είναι **$T = \Theta(1)$** , ενώ στη *χειρότερη περίπτωση* ο χρόνος θα είναι το άθροισμα της προσπέλασης της λίστας του Α και της λίστας του Β αν προχωρούσαν ένα ένα από κάθε λίστα, δηλαδή **$T = O(s_A + s_B)$** που είναι της τάξης O(s). Έχει όμως κάτω φράγμα **$\Omega(\min(s_A + s_B))$**, αφού δεν μπορεί να είναι μικρότερο από το μέγεθος της μικρότερης λίστας από τις δυο.

Χωρική πολυπλοκότητα:

Η μόνη δομή που χρησιμοποιείται είναι η Structure, έτσι ο χώρος που χρειάζεται είναι:

Για πίνακα:

$S_{StructureTable} = \Theta(w*s)*\Theta(1) = \Theta(w*s) \rightarrow S = \Theta(w*s) + \Theta(1)$ *(για το count)*

→ S = Θ(w*s)

Για συνδεδεμένη λίστα:

$S_{StructureLinked\text{-}list} = \Theta(w)*\Theta(2)$ *(χώρος για δείχτες σε head και tail nodes)*

$+\Theta(\sum_{i=0}^{w} s_i)*\Theta(2)$ *(χώρος για για τον αριθμό του subject + δείχτης στο επόμενο node, για όλα τα nodes των λιστών)* **→ S = $\Theta(\sum_{i=0}^{w} s_i)$ = O(w*s)**

3.4 Αλγόριθμος SimilarityKwo: Βαθμός Ομοιότητας k Αντικειμένων

Ο αλγόριθμος αυτός είναι η προέκταση του προηγούμενου αλγόριθμου, αφού εδώ μας απασχολεί το ποσοστό ομοιότητας όχι μεταξύ μόνο δυο αντικειμένων ιστού αλλά μεταξύ k, που ανήκουν και πάλι σε διαφορετικούς πηγαίους εξυπηρετητές το καθένα.

3.4.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός παίρνει σαν είσοδο τη δομή **Structure** και τις τιμές **w, s** από την προηγούμενη υπό-ρουτίνα. Επιπρόσθετη είσοδος είναι το **k** το οποίο καθορίζει για πόσα αντικείμενα θα υπολογιστεί ο βαθμός ομοιότητας, καθώς επίσης και το σύνολο των **k** αντικειμένων, για τα οποία θα υπολογίσει το ποσοστό κοινού ενδιαφέροντος. Ουσιαστικά θα βρει πόσα υποκείμενα, από το σύνολο των υποκειμένων που κάνουν αιτήσεις στο σύστημα, ζητούν και τα k αντικείμενα.

1) Τα k αντικείμενα που θα ελεγχτούν από τον αλγόριθμο δίνονται σαν είσοδος, μαζί με τη δομή Structure που θα χρησιμοποιηθεί και τις τιμές w και s.
2) Από τη δομή **Structure** βρίσκουμε τα k αντικείμενα ιστού και μετρούμε πόσα υποκείμενα ζήτησαν και τα k αντικείμενα (A[1], A[2], A[3],..., A[k]).
3) Τέλος, το αποτέλεσμα του ποσοστού ομοιότητας επιστρέφεται από την υπό-ρουτίνα στο πρόγραμμα που την καλεί.

Input: A[k]: a table with the web objects from sets O_1, O_2,$O_{3,\ldots,}$ O_k
Structure
w, s and k
Output: similarity percentage of the k web objects
Structure: we consider the subjects to have names from 0 to **s** and web-objects from 0 to **w**.
Structure : shows the relation between subjects and web-objects
Algorithm:

```
1:  Initialize count with 0//count is a counter for the number of subjects requesting all objects
2:  If ( Structure is an array ) then //array
3:      For  i=0 to s Do            //for every subject increase the count variable
4:          For j = 0 to k Do
5:              If (Structure[A[j]][i] != 1) //if the subject didn't request all the web-
6:                  Break;              //objects
7:              End if
8:          End for
9:          If ( j==k+1 ) then
10:             count ++  //if the subject requested all the web-objects
11:         End if
12:     End for
13: Else  //linked-list
14:     Create Node *cur[k]
15:     For j=0 to k Do
16:         cur[j] = Structure[A[j]].head
17:     End for
18:     For i=0 to s Do //for every subject check if it requests all K-objects
19:         For j=0 to k Do
20:             While ( cur[j] AND cur[j]→subject < i ) Do
21:                 cur[j] = cur[j]→next
22:             End while
23:             If( cur[j] == NULL ) then
24:                 Break;
25:             Else If( cur[j]→subject > i ) then
26:                 i = cur[j]->subject
27:                 Break;
28:             End if
29:         End for
30:         If( j != k+1 ) then
31:             If( cur[j] == NULL ) then
32:                 Break;
33:             Else
34:                 Continue without increasing i;
35:             End if
36:         Else
37:             count++
38:         End if
39:     End for
40: End if
41: Return (count/s)      //subjects requesting all k-objects/all subjects requesting web-objects
```

Στον πιο πάνω αλγόριθμο, όπως και στον Similarity2wo βλέπουμε για τις δυο περιπτώσεις δομής Structure, την εύρεση ποσοστού κοινού ενδιαφέροντος κάποιων συγκεκριμένων αντικειμένων ιστού, σε αυτή την περίπτωση k. Στις γραμμές 2 – 12 φαίνεται ο κώδικας που εκτελείται στην περίπτωση που η δομή Structure είναι πίνακας.

Ακολουθείται μια επαναληπτική διαδικασία για κάθε υποκείμενο, όπου ελέγχεται αν γίνεται αίτηση στο δίκτυο για όλα τα k αντικείμενα. Αν ναι, αυξάνεται ο μετρητής. Αντίστοιχα, στις γραμμές 13 – 40 βρίσκεται ο κώδικας που εκτελείται όταν η δομή Structure είναι συνδεδεμένες λίστες, η οποία είναι παρόμοια με αυτήν του αλγόριθμου Similarity2wo. Ουσιαστικά χρησιμοποιούνται k προσωρινοί δείχτες στις λίστες των k αντικειμένων και γίνεται πάλι μια ταυτόχρονη προσπέλασή τους για έλεγχο αν όλα τα αντικείμενα ζητήθηκαν από το συγκεκριμένο υποκείμενο κάθε φορά. Η ιδέα εκμεταλλεύεται το γεγονός ότι οι λίστες είναι ταξινομημένες και έτσι και πάλι η αναζήτηση γίνεται πολύ γρήγορα. Στο τέλος, γίνεται η διαίρεση με το σύνολο όλων των υποκειμένων που κάνουν αιτήσεις στο δίκτυο για να επιστραφεί το ποσοστό στη σωστή μορφή.

3.4.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Ο αλγόριθμος αυτό είναι η γενικευμένη μορφή του προηγούμενου, αφού βρίσκει το ποσοστό ομοιότητας μεταξύ k αντικείμενα, αντί για 2 και αυτό φαίνεται και από την ανάλυση της πολυπλοκότητας του, τόσο στο χρόνο όσο και στο χώρο που χρειάζεται. Αυτό, αφού παρατηρούμε πως ο χρόνος αλλά και ο χώρος έχει πολλαπλασιαστεί με τη μεταβλητή k.

Χρονική πολυπλοκότητα:

Ο χρόνος που χρειάζεται ο αλγόριθμος αυτός χωρίζεται και πάλι σε δυο περιπτώσεις, ανάλογα με τη δομή Structure που χρησιμοποιείται. Εξαιρείται η πρώτη γραμμή που είναι ίδιας χρονικής πολυπλοκότητας και για τις δυο περιπτώσεις και έχει χρόνο Θ(1).

Για πίνακα:

οι γραμμές 3 – 12 όπου γίνεται επανάληψη για κάθε αντικείμενο, ελέγχεται με μια δεύτερη επανάληψη (γραμμές 4 – 8) αν το υποκείμενο αυτό κάνει αίτηση για όλα τα k αντικείμενα του πίνακα A που δίνονται στην είσοδο. Αν κάποιο δεν ζητήθηκε από το υποκείμενο, η επανάληψη σταματά και συνεχίζεται ο έλεγχος με το επόμενο υποκείμενο. Σαν αποτέλεσμα ο χρόνος να είναι **T = O(s*k)**

Για λίστες:

Στις γραμμές 14 – 17, όπου ουσιαστικά δημιουργείτε πίνακας με δείχτες σε κάθε μια από τις λίστες των k αντικειμένων, χρειάζεται χρόνος Θ(k). Στις γραμμές 18 – 39, γίνεται μια επανάληψη με την ίδια λογική αυτής του πίνακα. Για κάθε υποκείμενο δηλαδή γίνεται έλεγχος κατά πόσο ζητά όλα τα k αντικείμενα που ζητούνται. Η βασική ιδέα είναι ότι για κάθε υποκείμενο γίνεται μια επανάληψη (γραμμές 19 - 29) όπου σε κάθε λίστα των k αντικειμένων ψάχνουμε το υποκείμενο i. Αν υπάρχει σε όλες τις λίστες, τότε σημαίνει πως το υποκείμενο αυτό ζητά όλα τα k αντικείμενα, έτσι στη γραμμή 37 αυξάνεται ο μετρητής. Από τη στιγμή που τα υποκείμενα βρίσκονται ταξινομημένα στις λίστες, γνωρίζουμε αμέσως αν ένα υποκείμενο δεν υπάρχει μόλις βρούμε κάποιο πιο μεγάλο από αυτό που ψάχνουμε. Αφού λοιπόν δεν υπάρχει, κάνουμε το i ίσο με το νέο που βρήκαμε, προσπερνώντας έτσι πιθανόν, αρκετά υποκείμενα. Ο χρόνος λοιπόν που χρειάζεται είναι **T = O(s*k)**

Χωρική πολυπλοκότητα:

Οι δομές που χρησιμοποιούνται είναι οι ίδιες με αυτές στον προηγούμενο αλγόριθμο, με τον επιπρόσθετο πίνακα A από k αντικείμενα. Έτσι, ο χώρος που χρειάζεται είναι:

Για πίνακα:

$S_{Atable} = \Theta(k)*\Theta(1) = \Theta(k),\ k \leq w \rightarrow O(w)$

$S_{StructureTable} = \Theta(w*s)*\Theta(1) = \Theta(w*s)$

➔ $S = \Theta(w*s) + \Theta(k) + \Theta(1)$ *(για το count)* → **$S = \Theta(w*s + k) = O(w*s + w)$**

Για συνδεδεμένη λίστα:

$S_{Atable} = \Theta(k)*\Theta(1) = \Theta(k),\ k \leq w \rightarrow O(w)$

$S_{cur} = \Theta(k)*\Theta(1) = \Theta(k)$

$S_{StructureLinked\text{-}list} = \Theta(w)*\Theta(2)$ *(χώρος για δείχτες σε head και tail nodes)* $+\Theta($ $\sum_{i=0}^{w} s_i)*\Theta(2)$ *(χώρος για για τον αριθμό του subject + δείχτης στο επόμενο node, για όλα τα nodes των λιστών)*

➔ $S = \Theta(\sum_{i=0}^{w} s_i)*\Theta(2) + \Theta(1)$ *(για το count)* $+ 2*\Theta(k)$

➔ **$S = \Theta(\sum_{i=0}^{w} s_i + k) = O(w*s + w)$**

3.5 Αλγόριθμος CommInt2wo: Κοινού Ενδιαφέροντος ζεύγη Αντικειμένων

Αφού σχεδιάστηκε αλγόριθμος που βρίσκει το ποσοστό ομοιότητας δυο αντικειμένων ιστού, επόμενο βήμα ήταν η σχεδίαση αλγορίθμου που βρίσκει τα ζεύγη αντικειμένων από δυο διαφορετικούς πηγαίους εξυπηρετητές τα οποία θεωρούνται κοινού ενδιαφέροντος για τους χρήστες του δικτύου.

3.5.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός παίρνει τις δομές Structure, TotSubjects, Origin και τις τιμές w και s από τον πρώτο αλγόριθμο, και χρησιμοποιώντας τον αλγόριθμο Similarity2wo που βρίσκει το ποσοστό κοινού ενδιαφέροντος δυο αντικειμένων, βρίσκει όλα τα ζεύγη από αντικείμενα (τα αντικείμενα είναι από διαφορετικά σύνολα) που είναι κοινού ενδιαφέροντος μεγαλύτερου του ορίου το οποίο θέτουμε ως **X** (καθορίζει το κατώτατο όριο για το ποσοστό κοινού ενδιαφέροντος). Το X καθορίζεται κατά την εκκίνηση του προγράμματος.

Επομένως ακολουθούνται τα πιο κάτω βήματα:

1) Το όριο **X**, οι δομές Structure, TotSubjects, Origin και οι τιμές w και s που θα χρησιμοποιηθούν, δίνονται σαν είσοδος στον αλγόριθμο.
2) Από τη δομή **TotSubjects**, επιλέγονται τα αντικείμενα που έχουν ποσοστό ζήτησης μεγαλύτερο ή ίσο του **X** και εισάγονται στις νέες δομές $\mathbf{H_1}$ και $\mathbf{H_2}$ ανάλογα με το σύνολο αντικειμένων στο οποίο ανήκουν (για αυτό χρησιμοποιείται η δομή Origin από την αρχική υπό-ρουτίνα).
3) Από τα σύνολα $\mathbf{H_1}$ και $\mathbf{H_2}$ δημιουργούνται ζεύγη αντικειμένων ιστού, και χρησιμοποιώντας τον αλγόριθμο Similarity2wo βρίσκουμε το ποσοστό κοινού ενδιαφέροντος μεταξύ τους. Αν είναι κοινού ενδιαφέροντος – έχουν δηλαδή ποσοστό ομοιότητας μεγαλύτερο η ίσο του X – τα τοποθετούμε στο σύνολο **K** που θα είναι το αποτέλεσμα.

Input: X: threshold for common interest
Structure, TotSubjects, Origin
w and s

Output: K = { $<A_1,B_1>$, $<A_2,B_2>$, … }: A set of web-object pairs that are of common interest from the subjects

Δομή: we consider the subjects to have names from 0 to **s** and web-objects from 0 to **w**.
Structure : shows the relation between subjects and web-objects
TotSubjects: array where T[i] is the amount of subjects requesting the i^{th} web-object
Origin : array where Q[i] is the set in which the i^{th} web-object belongs to
H_1: set of web-objects from set 1
H_2: set of web-objects from set 2

Algorithm:

1: **H_1 and H_2 empty**
2: **For i = 0 to w Do**
3: *//if the percentage of subjects requesting the i^{th} web-object from all the subjects requesting web-objects is more than or equal to X put it in one of the sets H_1 and H_2*
4: **If** (TotSubjects[i]/s $\geq$ X) **then**
5: **If (** Origin[i] == 1 **) then**
6: $H_1 = \{i\} \cup H_1$
7: **Else**
8: $H_2 = \{i\} \cup H_2$
9: **End if**
10: **End for**
11: **For** a $\in H_1$ **Do**
12: **For** b $\in H_2$ **Do**
13: **If (Similarity2wo(a, b, Structure, s, w) ≥ X) then**
14: $K = \{<a,b>\} \cup K$
15: **End if**
16: **End for**
17: **End for**
18: **Return** K

Στις γραμμές 2 – 10 γίνεται ουσιαστικά ο διαχωρισμός των αντικειμένων, με ποσοστό ζήτησης από τους χρήστες του δικτύου μεγαλύτερο ή ίσο του X, σε δυο ξεχωριστά σύνολα ανάλογα με τον πηγαίων εξυπηρετητών από τον οποίο προέρχονται. Έπειτα, στις επόμενες γραμμές, 11 – 17 υπάρχουν δυο επαναληπτικές δομές που δημιουργούν όλους τους συνδυασμούς ζευγών για τα αντικείμενα από τους δυο πηγαίους εξυπηρετητές. Για κάθε ζεύγος, στη γραμμή 13 καλείται ο αλγόριθμος Similarity2wo όπου υπολογίζεται ο βαθμός ομοιότητας των δυο αντικειμένων και ελέγχεται αν είναι μεγαλύτερος ή ίσος με το όριο X. Αν ναι, τότε φυλάγεται στη δομή K και επιστρέφεται στο τέλος του αλγορίθμου.

3.5.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Στον αλγόριθμο εύρεσης ζευγών κοινού ενδιαφέροντος ιστοσελίδων έχει προστεθεί η διαδικασία δημιουργίας συνδυασμών των ιστοσελίδων που ελέγχονται για το ποσοστό ομοιότητάς τους και έτσι πρέπει να δούμε πώς έχει διαμορφωθεί ο χρόνος αλλά και οι απαιτήσεις χώρου όλης της διαδικασίας.

Χρονική πολυπλοκότητα:

Γραμμές 2 – 10: Ο χρόνος που χρειάζεται για να χωριστούν τα αντικείμενα που έχουν ποσοστό ζήτησης μεγαλύτερο ή ίσο από το ποσοστό X που δίνεται είναι $T_{separate} = \Theta(w)$

Γραμμές 11 – 17: Στο κομμάτι αυτό δημιουργούνται ουσιαστικά τα ζεύγη μεταξύ των περισσότερο ζητούμενων αντικειμένων από τα δυο σύνολα και συγκρίνονται με τη χρήση του αλγόριθμου Similarity2wo που περιγράφηκε πιο πάνω. Ο χρόνος λοιπόν που χρειάζεται είναι:

Για πίνακα:

$T_{array} = \Theta(|H_1|*|H_2|)* T_{Similarity2woArray} = \Theta(|H_1|*|H_2|) * \Theta(s)$

Τα H_1 και H_2 περιέχουν αντικείμενα, που έχουν συνολικό πλήθος ο. Έτσι, τα $|H_1| \leq w$ και $|H_2| \leq w$. Επομένως, $\mathbf{T_{array} = O(w*w*s)}$

Για συνδεδεμένη λίστα:

$T_{linked\text{-}list} = \Theta(|H_1|*|H_2|)* T_{\#1.1linked\text{-}list} = \Theta(|H_1|*|H_2|) * O(s) = O(w*w) * O(s)$

➔ $\mathbf{T_{linked\text{-}list} = O(w*w*s)}$

Εδώ βλέπουμε πως η χρονική πολυπλοκότητα των δυο περιπτώσεων είναι η ίδια ασυμπτοτικά. Ουσιαστικά όμως το αποτέλεσμα και των δυο περιπτώσεων εξαρτάται από το πλήθος των ζευγών που θα δημιουργηθούν. Αλλά και από τη δομή – αραιή ή πυκνή σχέση ιστοσελίδων και χρηστών – στις συνδεδεμένες λίστες. Μπορούμε να πούμε πως ο χρόνος θα είναι της ίδιας τάξης τελικά, όμως τα πειράματα δείχνουν τις διαφορές.

<u>Χωρική πολυπλοκότητα:</u>

Αφού οι δομές που χρησιμοποιούνται είναι οι ίδιες με αυτές στον αλγόριθμο δημιουργίας δομής μαζί με τις επιπρόσθετες που χρειάζεται ο αλγόριθμος εύρεσης ποσοστού ομοιότητας, ο χώρος που χρειάζεται θα είναι:

<u>Για πίνακα:</u>

$S = S_{Structure} + S_{TotSubjects} + S_{Origin} + S_{H1} + S_{H2} + S_K$

➔ $S = \Theta(w*s) + \Theta(w) + \Theta(w) + \Theta(|H_1|)*\Theta(1) + \Theta(|H_2|)*\Theta(1) + \Theta(|K|)*\Theta(1)*2$

➔ $S = \Theta(w*s + 2*w) + \Theta(|H1| + |H2|) + O(|H1|*|H2|)$ *(η περίπτωση όπου όλα τα ζεύγη θα είναι κοινού ενδιαφέροντος)*

Αφού $|H_1| + |H_2|$ αποτελούν μαζί υποσύνολο όλων των web-objects, $|H_1| + |H_2| \leq w$

→ $S = \Theta(w + w*s) + O(w) + O(|H_1|*|H_2|)$

➔ **Γενικά: $S = O(w*s + w + w*w)$**

<u>Για συνδεδεμένη λίστα:</u>

$S = S_{Structure} + S_{TotSubjects} + S_{Origin} + S_{H1} + S_{H2} + S_K$

➔ $S = O(w*s) + \Theta(w) + \Theta(w) + \Theta(|H_1|)*\Theta(1) + \Theta(|H_2|)*\Theta(1) + \Theta(|K|)*2*\Theta(1)$

➔ $S = O(w*s) + 2*\Theta(w) + O(w) + O(|H1|*|H2|)$

➔ **Γενικά: $S = O(w*s + w + w*w)$**

Η διαφορά των δύο ασυμπτοτικών αναλύσεων χώρου πιο πάνω, είναι ουσιαστικά στο χώρο της λίστας, η οποία εξαρτάται και πάλι από τη σύνδεση των ιστοσελίδων με τους χρήστες. Για να είμαστε πιο ακριβείς θα ήταν καλύτερα αντί να πούμε O(w*s), να λέγαμε $\Theta(\sum_{i=0}^{w} s_i)$ για το $S_{Structure}$, που είναι βασικά το πλήθος των κόμβων στη δομή. Έχει όμως μέγιστη τιμή το w*s.

3.6 Αλγόριθμος CommIntKwo: Κοινού Ενδιαφέροντος k-άδες Αντικειμένων

Μετά από τη σχεδίαση του πιο πάνω αλγορίθμου CommInt2wo, που βρίσκει τα ζεύγη αντικειμένων ιστού από δυο διαφορετικούς πηγαίους εξυπηρετητές, τα οποία θεωρούνται κοινού ενδιαφέροντος για τους χρήστες του δικτύου, σειρά είχε η γενίκευση του για πλειάδες από k αντικείμενα ιστού κοινού ενδιαφέροντος.

3.6.1 Περιγραφή Αλγορίθμου

Παρόμοια με τον αλγόριθμο CommInt2wo, ο αλγόριθμος αυτός παίρνει τις δομές Structures, TotSubjects, Origin και τις τιμές w και s από τον πρώτο αλγόριθμο, και χρησιμοποιώντας τον αλγόριθμο SimilarityKwo που βρίσκει το ποσοστό κοινού ενδιαφέροντος k αντικειμένων, βρίσκει όλες τις πλειάδες από k αντικείμενα (τα αντικείμενα είναι από διαφορετικά σύνολα) που είναι κοινού ενδιαφέροντος μεγαλύτερου του ορίου το οποίο θέτουμε ως **X** (καθορίζει το κατώτατο όριο για το ποσοστό κοινού ενδιαφέροντος). Το X καθορίζεται και πάλι κατά την εκκίνηση του προγράμματος. Ο αλγόριθμος όμως αυτός, σαν είσοδο έχει επίσης και το k, που αντιστοιχεί στον αριθμό των αντικειμένων που θέλουμε να υπολογίσουμε ποσοστό ομοιότητας μεταξύ τους, δηλαδή το μέγεθος των πλειάδων που θέλουμε να δημιουργήσουμε.

Επομένως ακολουθούνται τα πιο κάτω βήματα:

1) Το threshold **X**, οι δομές Structure, TotSubjects, Origin και οι τιμές w, s, k που θα χρησιμοποιηθούν, δίνονται σαν είσοδος στον αλγόριθμο.
2) Από τη δομή **TotSubjects**, επιλέγονται τα αντικείμενα που έχουν ποσοστό ζήτησης μεγαλύτερο ή ίσο του **X** και εισάγονται στις νέες δομές **H[k]** ανάλογα με το σύνολο αντικειμένων στο οποίο ανήκουν (για αυτό χρησιμοποιείται η δομή **Origin** από την αρχική υπό-ρουτίνα).
3) Από τα σύνολα αυτά δημιουργούνται k-άδες από αντικείμενα ιστού, και χρησιμοποιώντας τον αλγόριθμο SimilarityKwo βρίσκουμε το ποσοστό κοινού ενδιαφέροντος μεταξύ τους. Αν είναι κοινού ενδιαφέροντος, τα τοποθετούμε στο σύνολο **K** που θα είναι το αποτέλεσμα.

Input: X: threshold for common interest
Structure, TotSubjects, Origin
w, s and k

Output: K = { $<A_1,A_2,\ldots,A_N>$, $<A_1,A_2,\ldots,A_N>$, … }: A set of web-object k-pairs that are of common interest from the subjects

Structure: we consider the subjects to have names from 0 to **s** and web-objects from 0 to **w**.
Structure : shows the relation between subjects and web-objects
TotSubjects: array where T[i] is the amount of subjects requesting the i^{th} web-object
Origin : array where Q[i] is the set in which the i^{th} web-object belongs to
H[i]: set of web-objects from set i

Algorithm:

```
1:  Create H[k] of empty sets
2:  For i = 0 to w Do
3:  //if the percentage of subjects requesting the i^th web-object from all the subjects requesting web-objects is more than or equal to X put it in one of the sets H[K]
4:      If ( TotSubjects[i]/s ≥ X ) then
5:          For j=0 to k Do //put the i^th object in the corresponding origin server set
6:              If ( Origin[i] == j ) then
7:                  H[j] = {i} ∪ H[j]
8:                  Break;
9:              End if
10:         End for
11:     End if
12: End for
13: Create Node* cur[k] //pointers to objects of the K sets
14: For i=0 to k Do //initialize
15:     cur[k] = H[i].0
16: End for
17: While cur[0] != NULL Do
18:     For i=0 to k Do
19:         A[i] = cur[i]
20:     End for
21:     If ( SimilarityKwo(A[],Structure, s, w, k) ≥ X ) then
22:         K = {<A[]>} ∪ K
23:     End if
24:     cur[K] = cur[k].next
25:     For j=k to 1 Do
26:         If( cur[j] == NULL ) then
27:             cur[j] = H[j].0
28:             cur[j-1] = cur[j-1].next
29:         Else
30:             Break;
31:         End if
32:     End for
33: End while
34: Return K
```

Στις γραμμές 2 – 12 γίνεται ουσιαστικά ο διαχωρισμός των αντικειμένων ιστού, με ποσοστό ζήτησης από τους χρήστες του δικτύου μεγαλύτερο ή ίσο του X, σε k ξεχωριστά σύνολα ανάλογα με τον πηγαίο εξυπηρετητή από τον οποίο προέρχονται. Έπειτα, στις επόμενες γραμμές, 13 – 33 δημιουργούνται k δείχτες, αρχικοποιούνται με

το πρώτο στοιχείο κάθε συνόλου που δημιουργήθηκε πιο πριν, και χρησιμοποιούνται για να κατασκευάσουν όλους τους συνδυασμούς πλειάδων μεγέθους k για τα αντικείμενα από τους k πηγαίους εξυπηρετητές. Για κάθε πλειάδα, στη γραμμή 21 καλείται ο αλγόριθμος SimilarityKwo. όπου υπολογίζεται ο βαθμός ομοιότητας των k αντικειμένων και ελέγχεται αν είναι μεγαλύτερος ή ίσος με το όριο X. Αν ναι, τότε φυλάγεται στη δομή K και επιστρέφεται στο τέλος του αλγορίθμου.

3.6.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Ο αλγόριθμος αυτό είναι η γενικευμένη μορφή του προηγούμενου, αφού βρίσκει τις πλειάδες αντικειμένων μεγέθους k κοινού ενδιαφέροντος, αντί για ζεύγη. Εδώ δεν φαίνεται ξεκάθαρα η γενικευμένη αυτή μορφή σε σχέση με τον προηγούμενο αλγόριθμο, γιατί περιπλέκεται λίγο ο τρόπος που δημιουργούνται οι πλειάδες ιστοσελίδων για τις οποίες βρίσκουμε το ποσοστό ομοιότητας. Έτσι λοιπόν η πολυπλοκότητα διαμορφώνεται όπως πιο κάτω.

Χρονική πολυπλοκότητα:

Γραμμές 2 – 12: Ο χρόνος που χρειάζεται για να χωριστούν σε k διαφορετικά σύνολα, τα αντικείμενα ιστού που έχουν ποσοστό ζήτησης μεγαλύτερο ή ίσο από το ποσοστό X που δίνεται είναι $T_{separate}$ = O(w*k)

Γραμμές 13 – 16: Στο κομμάτι αυτό δημιουργείται και αρχικοποιείται ο πίνακας με τους δείχτες για κάθε ένα από τα k σύνολα των k αντικειμένων. Χρειάζεται χρόνο Θ(k).

Γραμμές 17 – 33: δημιουργούνται ουσιαστικά οι k-άδες από τα περισσότερο ζητούμενα αντικείμενα από k διαφορετικούς πηγαίους εξυπηρετητές. Συγκρίνονται με τη χρήση του αλγόριθμου SimilarityKwo που περιγράφηκε πιο πάνω και μπαίνουν στο σύνολο με τις k-άδες κοινού περιεχομένου . Ο χρόνος που χρειάζεται κάθε επανάληψη είναι:

γραμμές 18-20: Θ(k) +

γραμμές 21-23: O(s*k) +

γραμμές 25-32: O(k)

και όλα αυτά O(**max(**|cur[i]|**)*(** $\sum_{i=0}^{k}|cur[i]|$ **)**) φορές, για να ελεγχτούν όλες οι k-άδες που δημιουργούνται. Αυτό, γιατί έχουμε k λίστες, η κάθε μια με διαφορετικό αριθμό κόμβων, |cur[i]|, και οι οποίες πρέπει να συνδυαστούν μεταξύ τους για να γίνουν όλες οι πιθανές πλειάδες.

Επομένως, T = O(w*k + k + **max(**|cur[i]|**)*(** $\sum_{i=0}^{k}|\boldsymbol{cur}[\boldsymbol{i}]|$ **) * (k + s*k + k)**)

➔ T = O(w*k + **max(**|cur[i]|**)*(** $\sum_{i=0}^{k}|\boldsymbol{cur}[\boldsymbol{i}]|$ **) * (s*k)**)

Εδώ δεν πρέπει να γίνουν δυο διαφορετικές αναλύσεις, για πίνακα και συνδεδεμένες λίστες, γιατί γνωρίζουμε ήδη πως ο χρόνος που χρειάζονται για τον αλγόριθμο SimilarityKwo είναι ο ίδιος και στις δυο περιπτώσεις. Αφού δεν υπάρχει θέμα χρήσης επιπρόσθετου πίνακα ή λίστας στο κομμάτι αυτό, η ασυμπτοτική ανάλυση είναι η ίδια.

<u>Χωρική πολυπλοκότητα:</u>

Αφού οι δομές που χρησιμοποιούνται είναι οι ίδιες με αυτές στον αλγόριθμο CommInt2wo, ο χώρος που χρειάζεται θα είναι περίπου ο ίδιος, με τη μόνη διαφορά τα k σύνολα αντί των 2 και τους k δείχτες που χρησιμοποιούνται. Έτσι λοιπόν:

<u>Για πίνακα:</u>

$S = S_{Structure} + S_{TotSubjects} + S_{Origin} + \sum_{i=0}^{k} S_i + S_K$

➔ $S = \Theta(w*s) + \Theta(w) + \Theta(w) + \Theta(\sum_{i=0}^{k}|H_i|)*\Theta(1) + \Theta(|K|)*\Theta(1)*k$

➔ $S = \Theta(w*s + 2*o) + \Theta(\sum_{i=0}^{k}|H_i|) + O($ **max(**|cur[i]|**)*(** $\sum_{i=0}^{k}|\boldsymbol{cur}[\boldsymbol{i}]|$ **) *k**$)$ *(η περίπτωση όπου όλα τα ζεύγη θα είναι κοινού ενδιαφέροντος είναι και το άνω όριο)*

Αφού $\sum_{i=0}^{k}|H_i|$ αποτελούν υποσύνολο όλων των web-objects, $\sum_{i=0}^{k}|H_i| \leq w$

➔ S = Θ(w + w*s) + O(w) + O(**max(**|cur[i]|**)*(** $\sum_{i=0}^{k}|\boldsymbol{cur}[\boldsymbol{i}]|$ **) *k**)

➔ **S = O(w*s + w + max(|cur[i]|)*(** $\sum_{i=0}^{k}|\boldsymbol{cur}[\boldsymbol{i}]|$ **) *k)**

<u>Για συνδεδεμένη λίστα:</u>

$S = S_{Structure} + S_{TotSubjects} + S_{Origin} + \sum_{i=0}^{k} S_i + S_K$

➔ $S = O(w*s) + \Theta(w) + \Theta(w) + \Theta(\sum_{i=0}^{k}|H_i|)*\Theta(1) + \Theta(|K|)*\Theta(1)*k$

➔ **<u>S = O(w*s + w + max(|cur[i]|)*(</u>** $\sum_{i=0}^{k}|\boldsymbol{cur}[\boldsymbol{i}]|$ **<u>) *k)</u>**

Και σε αυτό τον αλγόριθμο η διαφορά των δύο ασυμπτοτικών αναλύσεων χώρου πιο πάνω, είναι ουσιαστικά στο χώρο της λίστας, η οποία εξαρτάται και πάλι από τη σύνδεση των ιστοσελίδων με τους χρήστες. Για να είμαστε πιο ακριβείς θα ήταν καλύτερα αντί να πούμε O(w*s), να λέγαμε $\Theta(\sum_{i=0}^{w} s_i)$ για το $S_{Structure}$, που είναι βασικά το πλήθος των κόμβων στη δομή. Έχει όμως μέγιστη τιμή το w*s.

3.7 Αλγόριθμος NchooseK Δημιουργία k-άδων Origin Servers από N

Μετά από τη σχεδίαση των προηγούμενων αλγορίθμων και την ανάλυση της πολυπλοκότητάς τους, θα πρέπει να ολοκληρώσουμε το ενδεχόμενο των n πηγαίων εξυπηρετητών στο δίκτυο. Οι αλγόριθμοι μας και πιο συγκεκριμένα ο αλγόριθμος CommIntKwo βρίσκει πλειάδες από k αντικείμενα ιστού που είναι κοινού ενδιαφέροντος και το κάθε αντικείμενο ανήκει σε ξεχωριστό πηγαίο εξυπηρετητή. Σε ένα ΔΠΠ όμως, με n πηγαίους εξυπηρετητές, μας ενδιαφέρει να βρούμε πλειάδες κοινού ενδιαφέροντος από διαφορετικό αριθμό πηγαίων εξυπηρετητών και όχι απαραίτητα από n διαφορετικούς. Έτσι, χρειαζόμαστε ένα αλγόριθμο που θα κάνει k-άδες από τους πηγαίους εξυπηρετητές για τις οποίες αργότερα θα εκτελούνται οι προηγούμενοι αλγόριθμοι για να βρεθούν διάφορα είδη πλειάδων ιστοσελίδων κοινού ενδιαφέροντος. Καταλήξαμε λοιπόν στον πιο κάτω αλγόριθμο.

3.7.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός παίρνει εισόδους: τον συνολικό αριθμό πηγαίων εξυπηρετητών στο δίκτυο, **n**, και τον αριθμό **k** που καθορίζει το μέγεθος των πλειάδων που θα δημιουργηθούν. Σκοπός του είναι να δημιουργήσει όλες τις πλειάδες μεγέθους k που μπορούν να υπάρχουν από n πηγαίων εξυπηρετητών, ούτως ώστε μετά, για κάθε πλειάδα να χρησιμοποιηθεί ο αλγόριθμος CommIntKwo για να βρεθούν τα αντικείμενα κοινού ενδιαφέροντος από κάθε πλειάδα πηγαίων εξυπηρετητών. Έχοντας λοιπόν τα αποτελέσματα θα μπορούμε να βρούμε ένα βέλτιστο ποσοστό κοινού ενδιαφέροντος και θα μπορεί να χρησιμοποιηθεί και σε επίπεδο κοινοτήτων για να γίνει αργότερα πιο καλή τοποθέτηση των κοινοτήτων στους εξυπηρετητές αναπαραγωγής.

Ακολουθούνται τα πιο κάτω βήματα:

1) Τα **n** και **k** που θα χρησιμοποιηθούν, δίνονται σαν είσοδος στον αλγόριθμο.
2) Υπολογίζεται το **x**, που είναι ο αριθμός των τελικών πλειάδων $\binom{n}{k}$ για να μπορεί να δημιουργηθεί η δομή **C** (combinations) με τα αποτελέσματα.
3) Δημιουργείται δομή **P** (pointers) μεγέθους k, η οποία έχει το ρόλο δειχτών στους αριθμούς 0 – (n-1) ούτως ώστε να μην επαναλαμβάνονται οι πλειάδες.

4) Μετά από υπολογισμούς, δημιουργούνται όλοι οι συνδυασμοί των πηγαίων εξυπηρετητών στη δομή **C**.

Input: n: amount of origin servers
k: amount of origins in each pair
Output: C[X][k] = { <A_1,A_2,…,A_k>, <A_1,A_2,…,A_k>, … }: set of X total k-pairs of origin servers
Structures: x: amount of total k-pairs to be created
c: counter for the X pairs to be created
P[k] = { <$head_1$,val_1,$tail_1$>, <$head_2$,val_2,$tail_2$>, … }: set of k "pointers" (with head, value and tail in order to control the pairs to be created)
Algorithm:

```
1:  Calculate X using an iterative function factorial() to find ! (lines 44-50)
2:  x = n! / ( k! * (n-k)! )
3:  C[x][k] of type Int
4:  P[k] of type Node  //Node { <head, val, tail> }
5:  Initialize P
6:  For i=0 to k Do
7:      P[i].head = i
8:      P[i].val = i
9:      P[i].tail = n-k+i
10: End for
11: c = 0
12: While ( P[0].head <= P[0].tail ) Do
13:     For i=0 to k Do
14:         C[c][i] = P[i].val
15:     End for
16:     c++
17:     P[k-1].val++
18:     For j=k-1 to 0 Do
19:         If( P[j].val == (P[j].tail + 1) )
20:             P[j].head++
21:             P[j].val = P[j].head
22:             P[j-1].val++
23:         Else
24:             Break;
25:         End if
26:     End for
27: End while
28: Return C
29:
30: Function factorial(n){
31:     f = 1
32:     For i=1 to n Do
33:         f = f * i
34:     End for
35:     Return f;
36: }
```

Στον πιο πάνω αλγόριθμο, οι γραμμές 1 – 11 είναι βασικά οι αρχικοποιήσεις όλων των μεταβλητών και δομών που χρησιμοποιούνται για την κατασκευή των k-άδων. Έπειτα, στις γραμμές 12 – 27 βλέπουμε μια επαναληπτική δομή που περιλαμβάνει την κατασκευή όλων των πιθανών συνδυασμών. Στο τέλος, γραμμές 30 – 36 βλέπουμε και μια συνάρτηση που υπολογίζει με επανάληψη το factorial κάποιου αριθμού. Χρησιμοποιείται στον υπολογισμό εύρεσης του πλήθους των πλειάδων που πρέπει να δημιουργηθούν, στη γραμμή 2.

3.7.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Ο αλγόριθμος αυτός υλοποιεί ουσιαστικά το μαθηματικό $\binom{n}{k}$ και έτσι αναμένεται ο χρόνος του να είναι ανάλογος του n καθώς επίσης και του k, με απαιτήσεις χώρου ανάλογες με το $\binom{n}{k}$ αφού τόσες θα είναι και οι πλειάδες που θα δημιουργηθούν.

<u>Χρονική πολυπλοκότητα:</u>

Γραμμές 30 – 36: Ο χρόνος που χρειάζεται για να υπολογιστεί το παραγοντικό ενός αριθμού, χρησιμοποιείται ουσιαστικά στη γραμμή 2, είναι Θ(n), όπου n ο αριθμός για τον οποίο υπολογίζεται το παραγοντικό.

Επομένως ο χρόνος που χρειάζεται για τη γραμμή 2 είναι: Θ(n + k + (n-k))

Γραμμές 3 – 11: Χρειάζεται χρόνος Θ(3k+1) για την αρχικοποίηση του πίνακα P

Γραμμές 13 – 15: Χρόνος Θ(k)

Γραμμές 16 – 17: Χρόνος Θ(2)

Γραμμές 18 – 26: Χρόνος O(k*4)

Γραμμές 12 – 27: Ολόκληρη η επαναληπτική δομή while χρειάζεται να εκτελεστεί ουσιαστικά x φορές! (όσες είναι και οι κ-άδες που θα δημιουργηθούν) Επομένως, για το while loop χρειάζεται χρόνος O(x*(k+2+ k*4)) = O(5k*x), όπου x = n! / (k! * (n-k)!)

Τελικά, T = O(n+k+(n-k) + 3k+1 + 5k*(n!/(k!*(n-k)!))) =

➔ **T = O(2n + 3k + 5k(n!/(k!(n-k)!)))**

<u>Χωρική πολυπλοκότητα:</u>

Οι δομές που χρησιμοποιούνται στον πιο πάνω αλγόριθμο είναι ουσιαστικά, οι μεταβλητές n,k,x,c που είναι integers, και οι πίνακες C[x][k] και P[k]:

N: integer: Θ(1)

K: integer: Θ(1)

X: integer: Θ(1)

c: integer: Θ(1)

C[x][k]: Θ(k*x) = Θ(k*(n!/(k!*(n-k)!)))

P[k]: Θ(3k)

➔ **S = Θ(3k + k*(n!/(k!*(n-k)!)))**

Κεφάλαιο 4

Κοινού Ενδιαφέροντος Κοινότητες Ιστοσελίδων

4.1 Ορισμός ΚΕΚ και Αλγόριθμος CommInt2wc: Εύρεση Κοινού Ενδιαφέροντος ζεύγη κοινοτήτων

Τελικός σκοπός, ήταν να εισάγουμε τις κοινότητες στον αλγόριθμό μας. Να βρίσκουμε τις κοινού ενδιαφέροντος κοινότητες (ΚΕΚ) από διαφορετικούς πηγαίους εξυπηρετητές, ούτως ώστε να αντιγράφονται μαζί στους εξυπηρετητές αναπαραγωγής. Θα πρέπει να θέσουμε ένα δεύτερο όριο, **Υ**, το οποίο θα καθορίζει το ποσοστό κοινού ενδιαφέροντος δυο κοινοτήτων που μας ενδιαφέρει. Αν δυο κοινότητες έχουν βαθμό ομοιότητας μεγαλύτερο ή ίσο από αυτό το ποσοστό τότε θα θεωρούνται κοινού ενδιαφέροντος.

Είναι σημαντικό να αναφέρουμε εδώ πως για τις κοινού ενδιαφέροντος κοινότητες ο ορισμός είναι μόνο για 2 πηγαίους εξυπηρετητές, όπως και η πειραματική αξιολόγηση που υλοποιήθηκε στο [1] με τη χρήση ενός προσομοιωτή, του CDNsim [7]. Αυτό, γιατί η διαδικασία γινόταν πιο χρονοβόρα και πιστεύουμε πως η πειραματική αξιολόγηση σε επίπεδο 2 πηγαίων εξυπηρετητών ήταν αρκετή για να δείξει αν αξίζει και σε ποιες περιπτώσεις, να γίνεται αυτή η διαδικασία για βελτιστοποίηση στα ΔΠΠ. Ουσιαστικά η προέκταση του αλγορίθμου για εφαρμογή της τεχνικής σε μεγαλύτερα δίκτυα, με περισσότερους πηγαίους εξυπηρετητές, μπορεί να γίνει δημιουργώντας ζεύγη από τους

πηγαίους εξυπηρετητές, χρησιμοποιώντας τον αλγόριθμο NchooseK που παρουσιάστηκε στο προηγούμενο κεφάλαιο, και επαναλαμβάνοντας τη διαδικασία που χρειάζεται για το κάθε ζευγάρι, για να βρίσκονται οι κοινού ενδιαφέροντος κοινότητες που αντιστοιχούν.

4.1.1 Ορισμός και Περιγραφή Αλγορίθμου

Για να βρούμε το ποσοστό κοινού ενδιαφέροντος κοινοτήτων από 2 πηγαίους εξυπηρετητές, ακολουθείται ο πιο κάτω αλγόριθμος:

1) Αρχικά, χρησιμοποιώντας τον αλγόριθμο **CiBC** βρίσκουμε τις κοινότητες αντικειμένων ιστού (κοινότητες ιστοσελίδων) κάθε συνόλου αντικειμένων (πηγαίου εξυπηρετητή) στο δίκτυο – $\mathbf{WC_1}$**,** $\mathbf{WC_2}$**.**
2) Έπειτα, για κάθε κοινότητα, των δυο συνόλων, μετρούμε τον αριθμό μοναδικών υποκειμένων (χρηστών) που κάνουν αιτήσεις για αντικείμενα της συγκεκριμένης κοινότητας – **u(wc).**
3) Μετά, πρέπει να πάρουμε από τα σύνολα αντικειμένων, ζεύγη από κοινότητες – έστω $\mathbf{wc_1} \in \boldsymbol{WC_1}$,$\mathbf{wc_2} \in \boldsymbol{WC_2}$ – και να ελέγξουμε κατά πόσο είναι συγκρίσιμες. Ουσιαστικά, για να είναι συγκρίσιμες πρέπει τα $u(wc_1)$ και $u(wc_2)$ να είναι της ίδιας τάξης (να διαφέρουν μόνο σε μια επιτρεπτή απόκλιση **c**).
4) Για τα ζεύγη κοινοτήτων αντικειμένων ιστού που είναι συγκρίσιμα, θεωρούμε πως είναι υποψήφια για να θεωρηθούν κοινού ενδιαφέροντος. Έτσι, από κάθε κοινότητα παίρνουμε ένα αριθμό αντικείμενα, τα K πιο περιζήτητα web objects – $\mathbf{topK(wc_1)}$**,** $\mathbf{topK(wc_2)}$**.** Είναι τα K αντικείμενα κάθε κοινότητας που έχουν το μεγαλύτερο αριθμό μοναδικών υποκειμένων που τα ζητούν. Έτσι αφού γίνει ταξινόμηση των αντικειμένων κάθε υποψήφιας κοινότητας με βάση τον αριθμό των υποκειμένων που τα ζήτησαν, επιλέγονται τα πρώτα K. Αυτό θα μπορούσε να είναι και ποσοστό στον αριθμό των αντικειμένων κάθε κοινότητας, όμως επιλέξαμε να πάρουμε K αντικείμενα από κάθε υποψήφια κοινότητα ούτως ώστε να γίνεται σύγκριση ίσου αριθμού αντικειμένων από κάθε κοινότητα.
5) Από τα topK αντικείμενα κάθε κοινότητας, κάνουμε όλα τα πιθανά ζεύγη αντικειμένων μεταξύ των κοινοτήτων για τα οποία βρίσκουμε, από τον **αλγόριθμο Similarity2wo** που έχουμε ήδη υλοποιήσει, τον βαθμό ομοιότητας μεταξύ τους. Αν

το ποσοστό είναι μεγαλύτερο από **X** (το όριο που θέσαμε για κριτήριο ομοιότητας δυο αντικειμένων) τότε αυξάνουμε ένα μετρητή. Στο τέλος, υπολογίζουμε το ποσοστό ζευγών αντικειμένων, που είναι κοινού ενδιαφέροντος, από όλα τα ζεύγη που δημιουργούνται από τα topK αντικείμενα κάθε κοινότητας – **z.**

6) Αν το ποσοστό **z** είναι μεγαλύτερο ή ίσο από **Y**, τότε και οι δύο υποψήφιες κοινότητες είναι τελικά κοινού ενδιαφέροντος.
7) Τα βήματα 4,5,6 επαναλαμβάνονται για όλα τα ζεύγη από συγκρίσιμες κοινότητες.

X: $όριο\ ποσοστού\ κοινού\ ενδιαφέροντος\ ιστοσελίδες$

Y: $όριο\ ποσοστού\ κοινού\ ενδιαφέροντος\ κοινότητες$

O_1, O_2 : $\{πηγαίοι\ εξυπηρετητές\}$

WC_1, WC_2 : $\{σύνολα\ κοινοτήτων\}$

WC_1: $\{wc: wc \in O_1\}$

WC_2: $\{wc: wc \in O_2\}$

$u(wc)$: $αριθμός\ μοναδικών\ χρηστών\ που\ ζητούν\ αντικείμενα\ από\ την\ κοινότητα\ wc$

$If\ \ u(wc_1) = c * u(wc_2)\ \ then\ wc_1\ και\ wc_2\ είναι\ συγκρίσημες\ κοινότητες$

$Για\ κάθε\ ζεύγος\ αντικειμένων\ από\ \langle topK(wc_1), topK(wc_2)\rangle$

$$If\ \ Similarity2wo\big(topK(wc_1), topK(wc_2)\big) \geq X\ \ then\ count++$$

$$z = \frac{count}{αριθμός\ συνολικών\ ζευγών\ αντικειμένων\ από\ τις\ δυο\ κοινότητες}$$

$If\ \ z\ \geq Y\ then\ wc_1\ και\ wc_2\ είναι\ κοινού\ ενδιαφέροντος$

Είναι σημαντικό να αναφέρουμε πως το όριο **X** θα πρέπει να είναι μεταξύ κάποιων ορίων, $\mathbf{X_{min} \leq X \leq X_{max}}$

Από κάτω ($\mathbf{X_{min}}$) φράσσεται από το ποσοστό ομοιότητας αντικειμένων ιστού του ίδιου πηγαίου εξυπηρετητή. Θα πρέπει να γίνουν πειραματικές εκτελέσεις εύρεσης ποσοστού ομοιότητας μεταξύ αντικειμένων στον ίδιο πηγαίο εξυπηρετητή (από διάφορους

πηγαίους εξυπηρετητές φυσικά) για να βρεθεί ένα κατάλληλο ποσοστό. Αυτό, γιατί στον ίδιο πηγαίο εξυπηρετητή περιμένουμε να υπάρχει μεγαλύτερη ομοιότητα για μικρά Χ. Το $\mathbf{X_{min}}$ θα ήταν ίσως καλύτερα να βρεθεί από αντικείμενα ιστού της ίδιας κοινότητας ιστοσελίδων, όμως στην υλοποίηση υπάρχει δυσκολία στην απομόνωση των αντικειμένων μιας μόνο κοινότητας. Το πιο γενικό λοιπόν, είναι να πάρουμε αντικείμενα από τον ίδιο πηγαίο εξυπηρετητή, χωρίς να λάβουμε υπ' όψη μας τις κοινότητες στις οποίες ανήκουν.

Από πάνω ($\mathbf{X_{max}}$) φράσσεται από το ποσοστό Χ που θα βρεθεί πειραματικά από την πρώτη φάση της διπλωματικής, η οποία είναι βασικά η εύρεση ποσοστού ομοιότητας μεταξύ αντικειμένων από διαφορετικούς πηγαίους εξυπηρετητές. Αυτό, περιμένουμε να είναι αρκετά μεγαλύτερο από το X_{min} .

Input: sets of web communities from 2 origin servers – WC_1, WC_2
Structure, TotSubjects, Origin
X, Y, w, s
Output: set with pairs of common interest web communities
Structures: Structure: shows the relation between subjects and web-objects – represents the requests in the network,
C: set of comparable web communities pairs,
Result: set with pairs of common interest web communities
Algorithm:

Find the amount of unique users requesting each web community for the two origin servers

```
1:  Create structure of WC1 and WC2
2:  For wc in WC1 or WC2 Do
3:      u(wc) = 0
4:      For i=0 to s Do
5:          For w in wc Do
6:              If ( Structure[w][i]==1 ) then
7:                  u(wc) ++
8:                  break
9:              End if
10:         End for
11:     End for
12: End for
```

Sort the objects in each web community according to the number of users requesting each one

```
13: For wc in WC1 or WC2 Do
14:     Sort objects in wc  //using merge sort
15: End for
```

Find comparable pairs of web communities

```
16: C is the empty set of comparable pairs
17: For wc1 in WC1 Do
18:     For wc2 in WC2 Do
```

```
19:         If ( u(wc1) = c*u(wc2) ) then
20:             C = C ∪ {<wc1,wc2>}
21:         End if
22:     End for
23: End for
    Take the top K (first K in sorted list) web objects of each web candidate community and
    create pairs for which to find the common interest percentage
24: Result is the empty set of web communities with common interest
25: For each pair <wc1,wc2> in C Do
26:     G1 = top K web objects in wc1
27:     G2 = top K web objects in wc2
28:     count = 0
29:     For w1 in G1 Do
30:         For w2 in G2 Do
31:             If ( Similarity2wo(w1, w2, Structure, w, s) ≥ X ) then
32:                 count ++
33:             End if
34:         End for
35:     End for
36:     z = count/(|G1|*|G2|)
37:     If ( z >= Y ) then
38:         Result = Result ∪ {<wc1,wc2>}
39:     End if
40: End
```

Αν και ο πιο πάνω ψευδοκώδικας είναι από μόνος του αρκετά επεξηγηματικός, στις γραμμές 1 – 12 βρίσκουμε ουσιαστικά τον αριθμό των μοναδικών χρηστών που αιτούνται αντικείμενα σε κάθε κοινότητα ιστοσελίδων. Χρησιμοποιούνται τρεις φωλιασμένες επαναληπτικές δομές, όπου για κάθε κοινότητα των 2 πηγαίων εξυπηρετητών, για κάθε υποκείμενο ελέγχεται πόσα αντικείμενα ζητά, μέσω της δομής Structure που θεωρούμε πως έχουμε από τον πρώτο αλγόριθμο του κεφαλαίου 3, που κατασκευάζει ουσιαστικά τη δομή συσχέτισης χρηστών με τις ιστοσελίδες του δικτύου.

Έπειτα, στις γραμμές 13 – 15 θεωρούμε πως γίνεται χρήση κάποιας συνάρτησης merge sort για την ταξινόμηση των αντικειμένων κάθε κοινότητας με βάση τον αριθμό μοναδικών χρηστών που τα ζητούν. Εδώ χρησιμοποιείται ο πίνακας TotSubjects, επίσης από τον πρώτο αλγόριθμο κατασκευής δομής για τις αιτήσεις του δικτύου.

Στις γραμμές 16 – 23 γίνεται η εύρεση των συγκρίσιμων ζευγών κοινοτήτων από τους δυο διαφορετικούς πηγαίους εξυπηρετητές, τα οποία φυλάγονται στην καινούρια δομή C που δημιουργείται. Η γραμμή 19 ελέγχει: **If (** $u(wc_1) = c*u(wc_2)$ **) .** Αυτό ουσιαστικά

σημαίνει να είναι τα δυο web communities της ίδιας τάξης και μπορεί να εννοηθεί και αλλιώς: $u(wc_1) - c*u(wc_1) \leq u(wc_2) \leq u(wc_1) + c*u(wc_1)$.

Στις γραμμές 24 – 40 γίνεται το τελικό βήμα. Από το σύνολο των συγκρίσιμων κοινοτήτων, γίνεται επανάληψη της διαδικασίας που ακολουθεί για κάθε ζεύγος. Επιλέγονται τα topK αντικείμενα ιστού από κάθε κοινότητα και με μια επαναληπτική δομή δημιουργούνται ζεύγη από αντικείμενα, για τα οποία καλείται ο αλγόριθμος Similarity2wo για να βρεθεί το ποσοστό ομοιότητας. Αν αυτό ξεπερνά το X που θέσαμε, τότε το ζεύγος θεωρείται κοινού ενδιαφέροντος και έτσι αυξάνεται ένας μετρητής. Μετά το τέλος των επαναλήψεων για όλα τα πιθανά ζεύγη αντικειμένων των συγκεκριμένων κοινοτήτων, γίνεται μια διαίρεση για να βρεθεί το z, το ποσοστό των ζευγών από αντικείμενα από το σύνολο των ζευγών για τις δυο κοινότητες. Στις γραμμές 37 – 39 ελέγχεται αν το z είναι μεγαλύτερο ή ίσο από το ποσοστό Y που θέσαμε για να εισαχθεί το ζεύγος κοινοτήτων στο τελικό αποτέλεσμα κοινού ενδιαφέροντος ζευγών κοινοτήτων.

4.1.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Όπως και στους αλγόριθμους που αφορούσαν κοινού ενδιαφέροντος ιστοσελίδες, έτσι και για τις κοινού ενδιαφέροντος κοινότητες πρέπει να γίνει ανάλυση της πολυπλοκότητάς τους, τόσο για χρονική όσο και για χωρική αξιολόγηση. Παρόλο που η υλοποίηση έχει γίνει μόνο για δομή Structure τύπου δυσδιάστατου πίνακα, η ανάλυση πολυπλοκότητας έγινε και για συνδεδεμένες λίστες για να δούμε αν υπάρχει κάποια σημαντική διαφορά.

Χρονική πολυπλοκότητα:

Χρήση πίνακα για δομή Structure:

Ο χρόνος που χρειάζεται η διαδικασία αυτή, υπολογίζοντας και τη διαδικασία δημιουργίας δομής Structure είναι ο εξής:

Δομή Structure: $\Theta(w*s + |R|)$

Γραμμές 1-12: $O(s*w*|WC_1 \mathbf{U} WC_2|)$

Γραμμές 13-15: $O(|WC_1 \mathbf{U} WC_2|*nlogn)$ where n is the number of objects in each web community which is $O(w) = O(|WC_1 \mathbf{U} WC_2|*|wc|*log(|wc|)) = O(|WC_1 \mathbf{U} WC_2|*w*logw) = O(\sum_{i=0}^{|WC_1 \boldsymbol{U} WC_2|} i \log i)$

Γραμμές 16-23: $\Theta(|WC_1|*|WC_2|)$

Γραμμές 24-40: $O(\mathbf{(}|WC_1|*|WC_2|\mathbf{)*((}K+K) + K*K*T_{Similarity2wo}\mathbf{)}) = O(\mathbf{((}|WC_1|*|WC_2|\mathbf{)*(} s*K + K*K*s\mathbf{))}$

<u>Επομένως:</u> $T = \Theta(o*s + |R|) + O(s*o*|WC_1 \mathbf{U} WC_2|) + O(\sum_{i=0}^{|WC_1 \boldsymbol{U} WC_2|} i \log i)$
$+ \Theta(|WC_1|*|WC_2|) + O(\mathbf{((}|WC_1|*|WC_2|\mathbf{)*(} s*K + K*K*s\mathbf{))}$

<u>Χρήση λιστών για δομή Structure:</u>

Για την περίπτωση που η δομή Structure, η οποία δίνει τη σχέση των υποκειμένων με τα αντικείμενα ιστού, είναι υλοποιημένη με λίστες, ο πιο πάνω αλγόριθμος θα έχει αλλαγή στη γραμμή 6 βασικά, όπου ελέγχουμε αν ο χρήστης i ζήτησε το αντικείμενο w για να μετρήσουμε για κάθε κοινότητα πόσοι μοναδικοί χρήστες ζητούν αντικείμενα από αυτή.

```
6:  If ( Structure[w][i]==1 ) then
7:      u(wc) ++
8:      break
9:  End if
```

Ο κώδικας λοιπόν σε αυτό το σημείο γίνεται:

```
6:  Node *cur = Structure[w].head
7:  While ( cur ) Do
8:      If (cur.subject < i) then
9:          cur = cur->next
10:     Else if(cur.subject == i) then
11:         u(wc) ++
12:         Break
13:     Else
14:         Break
15:     End if
16: End do
17: If (cur) then
18:     Break
19: End if
```

Ο χρόνος που χρειάζεται επομένως ο αλγόριθμος αυτός για τις λίστες αυξάνεται μόνο στο πιο πάνω κομμάτι, όπου χρειάζεται επιπλέον λίγος χρόνος για να βρίσκει το υποκείμενο στη λίστα κάθε αντικειμένου.

Έτσι, για τις γραμμές 1-21 ο χρόνος θα είναι: O(|WC_1 **U** WC_2|*s*o*|s_w|) όπου |s_w| είναι το μέγεθος της λίστας με τα υποκείμενα που ζητούν το αντικείμενο w. Επομένως ο χρόνος αυτός είναι και: O($\sum_{i=0}^{w} s_i$ *s*|WC_1 **U** WC_2|)

Άρα: T = Θ(w*s + |R|) + O($\sum_{i=0}^{w} s_i$ *s*|WC_1 **U** WC_2|) + O($\sum_{i=0}^{|WC_1 \ U \ WC_2|} i \log i$)

+ Θ(|WC_1|*|WC_2|) + O(**(**|WC_1|*|WC_2|**)*(** s*K + K*K*s**)**)

<u>Χωρική πολυπλοκότητα:</u>

<u>Ανάλυση για πίνακες:</u>

Ο χώρος που χρειάζεται η διαδικασία αυτή είναι ο ίδιος με το χώρο που χρειάζεται ο αλγόριθμος Similarity2wo με τις επιπρόσθετες δομές που χρησιμοποιεί ο αλγόριθμος αυτός.

Χώρος αλγόριθμου Similarity2wo: Θ(o*s)

WC_1 and WC_2: Θ(|WC_1 + WC_2|*n) where n is the number of objects in each web community of the two sets which is O(w)

wc: integer → Θ(1)

u(): integer*|WC_1 + WC_2| → Θ(|WC_1 + WC_2|)

C: O(|WC_1 + WC_2|*2)

G_1 and G_2: Θ(2*K)

count: integer → Θ(1)

Result: O(|WC_1 + WC_2|*2) (έχει το πολύ όσα ζεύγη υπάρχουν στη δομή C)

Επομένως: **O(w*s + |WC_1 + WC_2|*w + 2*K)**

<u>Ανάλυση για λίστες:</u>

Ο χώρος που χρειάζεται η διαδικασία είναι ο ίδιος με το χώρο που χρειάζεται για πίνακες με τη διαφορά στο χώρο που χρειάζεται ο αλγόριθμος Similarity2wo για λίστες που είναι O(w*s).

Επομένως ο απαιτούμενος χώρος παραμένει ακριβώς ο ίδιος: **O(w*s + |WC_1 + WC_2|*w + 2*K)**

4.2 Αλγόριθμος AssignWCtoSurr: Ανάθεση Κοινοτήτων στους Εξυπηρετητές Αναπαραγωγής

Τελικό βήμα ήταν η ανάθεση των κοινοτήτων ιστού στους εξυπηρετητές αναπαραγωγής που θα αντιγραφούν. Επομένως, στο τέλος του αλγορίθμου πρέπει να δημιουργείται το αρχείο τοποθέτησης (placement file) [1], το οποίο δείχνει την αντιστοιχία αντικειμένων με εξυπηρετητές αναπαραγωγής.

Ουσιαστικά, έξοδος τους αλγόριθμου αυτού δεν θα είναι μόνο ένα αρχείο, αλλά πολλά, ανάλογα με το πόσοι εξυπηρετητές αναπαραγωγής χρησιμοποιούνται τελικά. Δημιουργείται ένα αρχείο για κάθε εξυπηρετητή αναπαραγωγής, το οποίο περιέχει τις ταυτότητες κάθε αντικειμένου που θα αντιγραφεί εκεί και ένα γενικό αρχείο τοποθέτησης το οποίο αναθέτει στον κάθε εξυπηρετητή αναπαραγωγής το αρχείο που του αντιστοιχεί.

4.2.1 Περιγραφή Αλγορίθμου

Για τη δημιουργία του αρχείου τοποθέτησης, χρειάζεται να ακολουθηθούν τα ακόλουθα βήματα:

1) Δεδομένα εισόδου του αλγορίθμου είναι ουσιαστικά τα αποτελέσματα από τον προηγούμενο αλγόριθμο, με τα ζεύγη κοινού ενδιαφέροντος κοινότητες.
2) Το πρόγραμμα ελέγχει κατά πόσο κάθε κοινότητα που καθορίζεται να αντιγραφεί σε ένα συγκεκριμένο εξυπηρετητή αναπαραγωγής, έχει κάποιο ζευγάρι που πρέπει να αντιγραφεί μαζί της. Έτσι, αναθέτει την κάθε κοινότητα σε κάποιον εξυπηρετητή και στο τέλος δημιουργεί τα αρχεία εξόδου.

Input: Result: pairs of common interest web communities
Output: placement file
Structures: Result[]: pairs of common interest web communities,
sur: counter used from the assignment of surrogate servers,
sur1[], sur2[]: tables of size $|WC_1|$,$|WC_2|$ used for the assignment of surrogate server in the position corresponding to the web-community of origin servers 1 and 2 respectively
NoSurrogates: variable keeping the amount of surrogates used in the network
Algorithm:

```
1:  Assign communities to surrogates
2:  Initialize sur1[|WC1|] and sur2[|WC2|]
3:  sur = 0
4:  For each pair <wc1,wc2> in Result[] Do
5:      If( sur1[wc1] != -1 ) then //a surrogate has already be assigned to community wc1
6:              If( sur2[wc2] != -1 ) then
7:                      Continue
8:              Else
9:                      sur2[wc2] = sur1[wc1]
10:             End if
11:     Else if( sur2[wc2] != -1 ) then
12:             sur1[wc1] = sur2[wc2]
13:     Else
14:             sur1[wc1] = sur
15:             sur2[wc2] = sur
16:             sur++
17:     End if
18: End for
19: Place unassigned communities to surrogates (randomly)
20: For i=0 to |WC1|      Do
21:     If( sur1[i] == -1 ) then
22:             sur1[i] = rand*NoSurrogates
23:     End if
24: End for
25: For i=0 to |WC2|      Do
26:     If( sur2[i] == -1 ) then
27:             sur2[i] = rand*NoSurrogates
28:     End if
29: End for
30: Create placement file(a loop for sur1 and one for sur2)
```

Στις γραμμές 1 – 18 τοποθετούνται σε εξυπηρετητές πρώτα τα ζεύγη των κοινοτήτων που θεωρούμε κοινού ενδιαφέροντος από τον προηγούμενο αλγόριθμο. Έπειτα, στις γραμμές 19 – 29 τοποθετούνται οι υπόλοιπες κοινότητες σε τυχαίους εξυπηρετητές στο δίκτυο και τέλος στη γραμμή 30 αναφέρεται η δημιουργία του τελικού αρχείου. Κώδικας για το σημείο αυτό θεώρησα πως δεν είναι απαραίτητος και το αφήνω στην κρίση του αυτού που θα υλοποιήσει τον τρόπο που θα το κάνει. Ουσιαστικά όμως, είναι απλά μια επαναληπτική δομή η οποία περνά από τους πίνακες sur1[] και sur2[] και γράφει τα δεδομένα τους σε αρχείο.

4.2.2 Ανάλυση Χρονικής και Χωρικής Πολυπλοκότητας

Χρονική πολυπλοκότητα:

Ο χρόνος που χρειάζεται το πιο πάνω κομμάτι αλγορίθμου είναι:

-- $O(|WC_1 + WC_2|*2)$ *(όσα είναι τα ζεύγη κοινοτήτων κοινού ενδιαφέροντος)*

-- $O(|WC_1|+|WC_2|)$ *(χρόνος προσπέλασης των δυο πινάκων για ανάθεση surrogate server στις κοινότητες που δεν ανατέθηκε κανένας)*

-- $O(|WC_1|+|WC_2|)$ *(προσπέλαση των συμπληρωμένων πινάκων για δημιουργία αρχείου)*

→ **$T = O(|WC_1 + WC_2| + |WC_1| + |WC_2|) = O(|WC_1 + WC_2|)$**

Χωρική πολυπλοκότητα:

Ο χώρος που χρειάζεται είναι ουσιαστικά το μέγεθος των δομών που χρησιμοποιούνται:

Result: $O(|WC_1 + WC_2|*2)$

sur: $\Theta(1)$

sur1: $O(|WC_1|)$

sur2: $O(|WC_2|)$

NoSurrogates: $\Theta(1)$

→ **$O(|WC_1 + WC_2| + |WC_1| + |WC_2|) = O(|WC_1 + WC_2|)$**

Κεφάλαιο 6

Συμπεράσματα

5.1 Γενικά Συμπεράσματα

Αρχικά, από την ανάλυση των πρώτων αλγορίθμων, για τα κοινού ενδιαφέροντος αντικείμενα, αλλά και με μια πειραματική αξιολόγηση τους, φαίνεται πως η χρήση πινάκων είναι πιο γρήγορη παρά λιστών όταν η σύνδεση ιστοσελίδων και χρηστών που τις ζητούν είναι μεγάλη. Αυτό, γιατί στις λίστες χρειάζεται δαπάνη πολύτιμου χρόνου στην αναζήτηση, ενώ στον πίνακα η πληροφορία που χρειάζεται ο αλγόριθμος είναι αμέσως διαθέσιμη και προσπελάσιμη. Παρόλο που στην περίπτωση που η σύνδεση ιστοσελίδων και χρηστών είναι μικρή, από θέμα χώρου είναι καλύτερες οι λίστες γιατί δεν θα δεσμεύουν αχρείαστο χώρο, όμως ο χρόνος που είναι επίσης σημαντικός έχει περίπου τα ίδια αποτελέσματα. Παρατηρήσαμε πως σε πολλά από τα πειράματα οι λίστες ήταν πιο γρήγορες, όμως σε άλλα ήταν αρκετά πιο αργές. Αυτό, εξαρτιόταν και από τη σειρά των αιτήσεων, γιατί όπως εξήγησα και στο Κεφάλαιο 4 έχει μεγάλη σημασία στην κατασκευή της δομής με τις συνδεδεμένες λίστες. Γι' αυτό αποφασίσαμε στα πειράματα που έγιναν αργότερα για τις κοινού ενδιαφέροντος κοινότητες [1] να χρησιμοποιηθούν πίνακες, οι οποίοι είναι και πιο εύκολοι στη χρήση και πιο σταθεροί στο χρόνο που χρειάζονται. Γνωρίζαμε πως ο χρόνος που θα χρειάζονταν τα πειράματα ήταν πολύ καλός και θα ήταν και τετριμμένος αφού είναι τάξεως Θ.

Ακόμα, μετά από πολλά πειραματικά αποτελέσματα, καταλήξαμε σε καλύτερη μέση τιμή του ποσοστού Χ 20%, όμως επειδή υπήρξαν διάφορες περιπτώσεις, ανάλογα και

με τις πολιτικές που χρησιμοποιήθηκαν στα πειράματα, ένα καλό διάστημα για το X είναι μεταξύ 15% και 25% περίπου.

Όσο αφορά το ποσοστό Y, τα πειράματα δεν ήταν εντελώς ξεκάθαρα. Τις περισσότερες φορές, όντος ο χρόνος ανταπόκρισης του δικτύου ήταν καλύτερος με την αντιγραφή των κοινού ενδιαφέροντος κοινοτήτων στους ίδιους εξυπηρετητές, όμως οι φορές που έκανε τη μεγαλύτερη διαφορά ήταν διαφορετικές σε κάθε περίπτωση. Αυτό που παρατηρήθηκε γενικά είναι ότι όσο πιο μικρό είναι το X, τόσο πιο μεγάλο μπορεί να γίνει το Y δίνοντας καλύτερα αποτελέσματα. Αντίθετα, αν βάλουμε πολύ μεγάλο το X, για να βρίσκει κοινού ενδιαφέροντος κοινότητες ο αλγόριθμος και να δίνουν καλά αποτελέσματα, πρέπει το Y να είναι αρκετά μικρό. Οι περιπτώσεις λοιπόν που έδωσαν καλύτερο χρόνο ανταπόκρισης στο δίκτυο ήταν είτε όταν το X ήταν γύρο στο 1-3% και το Y 25-50%, είτε όταν το X ήταν 10-20% και το Y 3-5%. Γενικότερα όμως το Y δεν φαίνεται να είχε καλά αποτελέσματα πάνω από 55%. Είναι όμως ένα ικανοποιητικό ποσοστό που έχει όντως βελτιστοποιήσεις στο δίκτυο.

Όσο αφορά το πλήθος των εξυπηρετητών αναπαραγωγής του δικτύου, παρατηρήθηκε γενικά πως δεν έχει μεγάλη διαφορά, είτε χρησιμοποιούνται 50 είτε 100 εξυπηρετητές. Ο χρόνος διαφοροποιείται μόλις στο πέμπτο δεκαδικό του χρόνου. Στην πραγματικότητα όμως, όταν το δίκτυο θα πρέπει να ανταποκριθεί σε εκατομμύρια αιτήσεις, πολύ περισσότερες από τις αιτήσεις που μπορέσαμε να κάνουμε πειράματα, η διαφορά στο χρόνο θα είναι σημαντική. Από τις γραφικές παραστάσεις που παρουσιάζονται στη διπλωματική [1] φαίνεται πως για τα ζεύγη ιστότοπων που έχουν κοινό σημασιολογικό περιεχόμενο, όπως για παράδειγμα BBC.com CNN.com, οι 100 εξυπηρετητές αναπαραγωγής δίνουν καλύτερα αποτελέσματα από τους 50. Αντίθετα, για τα ζεύγη ιστότοπων που δεν έχουν κοινό σημασιολογικό περιεχόμενο, δεν μπορούμε να έχουμε ξεκάθαρα συμπεράσματα, αφού στις περιπτώσεις των 1000 χρηστών, οι 50 εξυπηρετητές απέδιδαν καλύτερα, ενώ στους 5000 χρήστες καλύτερα αποτελέσματα είχαν οι 100 εξυπηρετητές.

5.2 Μελλοντικές Εργασίες

Στα πλαίσια μελλοντικών μελετών, θα μπορούσαν να γίνουν αρκετά ως προέκταση αυτής της εργασίας αλλά και γενικά στα Δίκτυα Παράδοσης Περιεχομένου. Αρχικά πιστεύω πως θα ήταν καλό να γίνει πειραματική αξιολόγηση της μεθόδου αυτής όχι μόνο σε μια από τις πολιτικές αντιγραφής των περιεχομένων στους εξυπηρετητές, αλλά σε όλες, ούτως ώστε να γίνει και σύγκριση για να φανεί πότε είναι καλύτερα να εφαρμόζεται, σύμφωνα με το δίκτυο που υπάρχει και την πολιτική που ακολουθείται.

Επιπλέον, θα μπορούσαν να γίνουν παρόμοιοι αλγόριθμοι, αντίστοιχοι με αυτούς που σχεδιάστηκαν στη διπλωματική αυτή εργασία, σε επίπεδο όμως παράλληλης επεξεργασίας, σε όσα κομμάτια του προβλήματος γίνεται φυσικά να εφαρμοστεί παραλληλισμός. Για παράδειγμα, ο πρώτος αλγόριθμος, για τη δημιουργία δομής συσχέτισης ιστοσελίδων και χρηστών, είναι κάτι που δεν γίνεται παράλληλα, μπορεί να γίνει μόνο σειριακά. Το ίδιο πιστεύω και για τους αλγορίθμους εύρεσης ζευγών ή πλειάδων κοινού ενδιαφέροντος ιστοσελίδων, όσο αφορά το κομμάτι δημιουργίας των ζευγών ή πλειάδων που θα εξεταστούν. Μπορεί όμως η εύρεση του ποσοστού ομοιότητας κάθε ζεύγους ή πλειάδας να γίνει παράλληλα για πολλά ζεύγη ή πλειάδες. Έτσι, θα κερδίζεται χρόνος για τον έλεγχο του αν τα ζεύγη ή οι πλειάδες είναι τελικά κοινού ενδιαφέροντος. Πιστεύω πως θα ήταν ενδιαφέρον να δούμε την επίδοσή τους σε ένα κατανεμημένο σύστημα όπως αυτό των Δικτύων Παράδοσης Περιεχομένου, ιδικά σε σύγκριση με τους σειριακούς αλγόριθμους.

Θεωρώ ακόμη δελεαστική την ιδέα οι κατανεμημένοι αλγόριθμοι να εκτελούνται από τους εξυπηρετητές αναπαραγωγής του δικτύου και να βρίσκουν οι ίδιοι τα περιεχόμενα που είναι καλύτερα να φυλάγονται εκεί, εκμεταλλευόμενοι των αιτήσεων που γίνονται από τους χρήστες προς αυτούς – που είναι λογικά και οι κοντινότεροί τους. Θα εκμεταλλευόμαστε έτσι και την ακριβής τοπολογία του δικτύου αλλά και των τοπικών αιτήσεων των χρηστών της περιοχής. Νομίζω όμως πως στον CDNsim [1,5] δεν μπορεί να γίνει κάτι τέτοιο, έτσι ίσως μια τροποποίησή του να ήταν αρχικά απαραίτητη. Επίσης, θα πρέπει οι εξυπηρετητές αναπαραγωγής να συνεργάζονται για να ξέρουν που βρίσκεται η κάθε πληροφορία, πράγμα που ίσως δεν είναι τόσο εύκολο να ελεγχτεί.

Βιβλιογραφία

[1] Ν. Δημητρίου, ΑΔΕ "Ανάπτυξη Αλγορίθμων για βελτίωση της επίδοσης Δικτύων Παράδοσης Περιεχομένου με Πολλαπλούς Εξυπηρετητές Περιεχομένου – Προσομοίωση", Πανεπιστήμιο Κύπρου, 2011.

[2] A.Vakali and G.Pallis, "Content Delivery Networks: Status and Trends", IEEE Internet Computing, Noveber-December 2003

[3] D. Katsaros, G. Pallis, K. Stamos, A. Vakali, A. Sidiropoulos and Y. Manolopoulos, "CDNs Content Outsourcing via Generalized Communities", IEEE Computer Society, 2009.

[4] G. Pallis and A. Vakali, "Insight and Perspectives for Content Delivery Networks", Communications of the ACM, Vol.49, No.1, January 2006.

[5] G.Pallis, L.Angelis, A.Vakali, "Validation and Interpretation of Web Users' Sessions Clusters", Information Processing and Management, 2007

[6] K. Stamos, G. Pallis, A. Vakali, M.D. Dikaiakos, "Evaluating the Utility of Content Delivery Networks", ACM, 2009.

[7] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos and Y. Manolopoulos, "CDNsim: A Simulation Tool for Content Distribution Networks", ACM, Vol.20, No. 2, April 2010.

[8] Wiki Tern Paper – croft1 – CS 525 Spring 2011 – The Computer Science Agora
https://agora.cs.illinois.edu/display/cs525sp11/Wiki+Term+Paper+-+croft1

[9] Mathematics of Ranking
http://nalag.cs.kuleuven.be/research/workshops/ranking/programme.shtml