ΔΙΑΤΗΡΗΣΗ ΕΝΗΜΕΡΟΤΗΤΑΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ ΣΕ ΑΔΟΜΗΤΑ ΣΥΣΤΗΜΑΤΑ ΟΜΟΤΙΜΩΝ ΚΟΜΒΩΝ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Κωνσταντίνο Αλεξάκη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Νοέμβριος 2007

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω θερμά την καθηγήτριά μου, κυρία Ευαγγελία Πιτουρά για την ανεκτίμητη βοήθειά της και το χρόνο που μου αφιέρωσε όλο αυτό το διάστημα.
Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου, Σωτήρη, Καίτη και Ηλία για την αμέριστη συμπαράσταση που παρείχαν στο πρόσωπό μου, κάτι που μου είναι αρκετά δύσκολο να εκφραστεί με απλές λέξεις.
Ακόμη, θα ήθελα να ευχαριστήσω την εξεταστική επιτροπή, τους κυρίους Βασιλειάδη και Δημακόπουλο για το χρόνο τους και τις συμβουλές τους ως προς τις διορθώσεις της διατριβής μου.
Τέλος, θέλω να ευχαριστήσω τους συναδέλφους και τους φίλους που έκανα στα Ιωάννινα που με τη συμπαράστασή τους μου δώσανε κουράγιο και όρεξη να συνεχίσω μέχρι εδώ.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Κωνσταντίνος Αλεξάκης του Σωτηρίου και της Σαραντίας-Αικατερίνης. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Νοέμβριος, 2007. Διατήρηση Ενημερότητας Βάσεων Δεδομένων σε Αδόμητα Δίκτυα Ομότιμων Κόμβων.
Επιβλέπουσα: Ευαγγελία Πιτουρά.

Στην παρούσα εργασία εξετάζουμε την ενημερότητα των δεδομένων σε κατανεμημένες βάσεις δεδομένων πάνω σε αδόμητα συστήματα ομότιμων κόμβων. Η ενημερότητα ορίζεται ως η φρεσκάδα των δεδομένων, που την προσδιορίζει η ημερομηνία λήξης τους, δηλαδή τη χρονική στιγμή έπειτα από την οποία τα δεδομένα παύουν να ισχύουν. Η ημερομηνία λήξης, ανατίθεται από το σύστημα με τη μορφή χρονοσφραγίδας σε κάθε πλειάδα κατά την εισαγωγή της στην κατανεμημένη βάση δεδομένων. Στο σύστημα, κάθε κόμβος διαθέτει μια βάση δεδομένων, με τοπικά αποθηκευμένους πίνακες, πάνω στην οποία επιθυμούμε να εκτελούμε ερωτήσεις.

Στην εργασία έγινε μια επέκταση της παραδοσιακής SQL γλώσσας για την υποστήριξη των λειτουργιών συνάθροισης (aggregation) και συνένωσης (join) από τέτοια δίκτυα, έτσι ώστε να υποστηρίζουν κριτήρια ενημερότητας. Προτείνονται αλγόριθμοι για την υλοποίηση των λειτουργιών αυτών και δίνεται ο τρόπος υπολογισμού της ημερομηνίας λήξης του αποτελέσματος. Επιπλέον παρουσιάζονται διάφορα πρωτόκολλα ενημέρωσης (update) και αντιγραφής (replication) των πλειάδων στο σύστημα για λόγους φρεσκάδας της κατανεμημένης βάσης δεδομένων, ανοχής σφαλμάτων, βελτίωσης της αποδοτικότητας και επεκτασιμότητας του δικτύου. Τα πρωτόκολλα αυτά μελετούνται ως προς την ποιότητα υπηρεσιών (QoS) και δεδομένων (QoD) που προσφέρουν. Τέλος περιγράφονται αποτελέσματα ενός συνόλου πειραμάτων που δείχνουν την απόδοση των πρωτοκόλλων ενημέρωσης και αντιγραφής σε τέτοια δίκτυα.

EXTENDED ABSTRACT IN ENGLISH

Konstantinos Alexakis. MSc, Computer Science Department, University of Ioannina, Greece. November 2007. Freshness-Aware Processing in Unstructured Peer-to-Peer DBMSs. Thesis Supervisor: Evaggelia Pitoura.

Nowadays, internet has become a part of our everyday life with more than 1 billion users all over the world. A large number of them makes use of applications over Peer-to-Peer systems.

Peer-to-Peer systems make use of internet infrastructures and resources for network communication. They are overlay networks which run on top of Internet and provide the ability of sharing resources between users. Such systems are mainly used for exchanging resources such as storage space, bandwidth, computing power and content such as files.

In particular, we focus on a decentralized, unstructured, dynamic peer-to-peer system. In such a system, there is no distinction between clients and servers, but instead all peers have equal roles and function simultaneously as both clients and servers to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. In addition, peers may dynamically connect to or disconnect from the system without further notice.

We assume that each peer contains a database on which we need to process queries. Every tuple is extended with an attribute stating its expiration time. We associate with each tuple an expiration time in the form of a timestamp. Expiration times indicate when tuples cease to be current in a database.

Making use of data extended with expiration times is motivated by applications that involve sensor networks (e.g. temperature sensors), data streaming and location tracking for moving objects. Specifically, there is a need to introduce methods for processing SQL queries over the distributed database, as well as to indicate the freshness of the results expressed through the expiration time of the data. For this reason, we have extended the standard SQL aggregation and join operations and have introduced methods and algorithms for computing the expiration time of the final result.

Moreover, there is the need for updates and replications of tuples in order to assure fault tolerance, improve efficiency and scalability of the system. To this end, we introduce update and replication protocols that deal with the update process of obsolete tuples and the freshness of the distributed database. When we combine these protocols with query processing, we are interested in two sets of metrics. These are the Quality of Service (QoS) metrics to measure system performance and the Quality of Data (QoD) metrics to measure the freshness of the served data.

In order to improve system performance, we look into two Quality of Service guarantees. Namely, the *response time*, that is the number of hops and the *message overhead*, that is the overall number of messages to obtain the results. Moreover, we consider two metrics for the Quality of Data. These are the *freshness* metric which is proportional to the average expiration time of the result returned by a query and *recall* metric which is the percentage of the results returned with respect to all results in the network. Each combination of update protocols and query protocol achieves different QoS and QoD guarantees.

Finally, we have evaluated both the quality of data and the quality of service achieved by each protocol through simulation.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

Το κεφάλαιο αυτό αποτελεί μια εισαγωγή στο πρόβλημα το οποίο μελετάμε στην παρούσα εργασία. Στην ενότητα 1.1 ορίζεται σε γενικές γραμμές το πρόβλημα και παρουσιάζονται οι ανάγκες του σημερινού κόσμου που οδήγησαν σε αυτό. Στην ενότητα 1.2 δίνεται συνοπτικά ο σκοπός της εργασίας και τέλος στην 1.3 δίνεται η δομή της εργασίας αυτής.

1.1. Ορισμός του Προβλήματος

Στη σημερινή κοινωνία το διαδίκτυο έχει εισχωρήσει στην καθημερινή ζωή των ανθρώπων. Με την εξάπλωση της ευρυζωνικότητας στην Ευρώπη και σε όλο τον κόσμο, καθώς και με την ολοένα και αυξανόμενη ανάγκη των ανθρώπων για χρήση των υπολογιστών και του διαδικτύου για ανάγκες ταχύτερης πληροφόρησης, επικοινωνίας, μεταφοράς δεδομένων κ.α. σήμερα πάνω από ένα δισεκατομμύριο άνθρωποι χρησιμοποιούν το διαδίκτυο σε ολόκληρο τον πλανήτη [15]. Από αυτούς ένα πολύ μεγάλο μέρος χρησιμοποιούν εφαρμογές για συστήματα ομότιμων κόμβων (peer-to-peer).

Τα συστήματα ομότιμων κόμβων (peer-to-peer) είναι κατανεμημένα συστήματα ομότιμων κόμβων τα οποία χρησιμοποιούν την υποδομή και τους πόρους του διαδικτύου για την επικοινωνία μεταξύ των κόμβων. Τα συστήματα αυτά χρησιμοποιούνται κυρίως για διαμοιρασμό αρχείων μεταξύ των χρηστών, για

διαμοιρασμό της επεξεργαστικής ισχύς των κόμβων ή του εύρους δικτύου τους. Τα δίκτυα αυτά μετράνε περίπου οχτώ χρόνια ζωής και σήμερα αποτελούν μια από τις σημαντικότερες εφαρμογές διαδικτύου [8] με πάνω από το 50% της συνολικής κίνησης του διαδικτύου [16].

Τα τελευταία χρόνια ένα μεγάλο μέρος της ερευνητική κοινότητας έχει στραφεί στη μελέτη τέτοιων δικτύων με σκοπό κυρίως την αποδοτικότητα, την επεκτασιμότητα και την ανοχή σφαλμάτων που μπορούν να ανεχθούν. Στην παρούσα μελέτη εξετάζουμε δίκτυα ομότιμων κόμβων που είναι μη κεντρικοποιημένα, δεν υπάρχει δηλαδή η έννοια του εξυπηρετητή, αδόμητα (χωρίς καμία ιδιαίτερη δομή), δυναμικά (οι κόμβοι μπορούν να εισέρχονται και να αποχωρούν χωρίς προειδοποίηση) και που όλοι οι κόμβοι είναι ομότιμοι-ισάξιοι μεταξύ τους, με μια σημαντική διαφορά με τα υπάρχοντα δίκτυα: κάθε κόμβος διαθέτει μια βάση δεδομένων πάνω στην οποία επιθυμούμε να εκτελούμε ερωτήσεις και κάθε πλειάδα της βάσης δεδομένων διαθέτει μια ημερομηνία λήξης. Η ημερομηνία λήξης, είναι η χρονική στιγμή έπειτα από την οποία η πλειάδα λήγει και έτσι παύει να ισχύει. Αποτελεί ένα είδος φρεσκάδας της κάθε πλειάδας που της την αναθέτει το σύστημα με τη μορφή χρονοσφραγίδας σαν ιδιαίτερο γνώρισμα.

Η έννοια της ημερομηνίας λήξης σε δεδομένα δεν είναι καινούρια. Χρησιμοποιείται σε διάφορες εφαρμογές που έχουν να κάνουν με ροή δεδομένων (data streaming), με τοποθεσίες κινούμενων αντικειμένων, αισθητήρες θερμοκρασίας ή σε κλειδιά σε πρωτόκολλα κρυπτογράφησης ([1],[2]). Στην εργασία μας ενσωματώνουμε την έννοια της φρεσκάδας σαν ημερομηνία λήξης των πλειάδων, σε μια κατανεμημένη βάση δεδομένων σε δίκτυα ομότιμων κόμβων.

Σε ένα τέτοιο σύστημα, αρκετά είναι τα προβλήματα που δημιουργούνται. Ένα πρόβλημα που μελετάμε είναι το πώς θα υποστηρίζει τις κυριότερες λειτουργίες της παραδοσιακής SQL γλώσσας στην κατανεμημένη βάση δεδομένων και ποια θα είναι τελικά η φρεσκάδα του αποτελέσματος. Για το λόγο αυτό έγινε επέκταση της παραδοσιακής SQL για τις ανάγκες του συστήματός μας ως προς τις κύριες λειτουργίες συνάθροισης (aggregation) και συνένωσης (join), έτσι ώστε να υποστηρίζουν κριτήρια ενημερότητας. Προτείνονται διάφοροι αλγόριθμοι και

αναλύεται ο τρόπος με τον οποίο θα υπολογίζεται η φρεσκάδα του αποτελέσματος των ερωτημάτων αυτών.

Ένα επιπλέον πρόβλημα αποτελεί ο τρόπος ενημέρωσης (update) και αντιγραφής (replication) των δεδομένων-πλειάδων στο σύστημα που μελετάμε. Επειδή το σύστημα είναι δυναμικό, δηλαδή οι κόμβοι μπορούν να εισέρχονται και να αποχωρούν χωρίς προειδοποίηση, ή να δημιουργούν και να ενημερώνουν τις πλειάδες τους ανεξάρτητα από τους υπόλοιπους ομότιμους κόμβους, γίνεται επιτακτική η ανάγκη γρήγορης διάδοσης και αντιγραφής των ενημερωμένων πλειάδων στο σύστημα, για λόγους ανοχής σφαλμάτων. Επιπλέον καθώς οι πλειάδες των κόμβων λήγουν, ο κάθε κόμβος πρέπει να τις διαγράφει αλλά παράλληλα να προσπαθεί να τις ενημερώνει με αντίστοιχες πλειάδες των ομότιμών του. Για το λόγο αυτό προτείνονται διάφορα πρωτόκολλα ενημέρωσης και αντιγραφής δεδομένων στο σύστημα που μελετάμε, με σκοπό την ενημέρωση και τη φρεσκάδα της κατανεμημένης βάσης δεδομένων.

Στα πρωτόκολλα ενημέρωσης και αντιγραφής ενημερώσεων που προτείνονται, γίνεται μελέτη της απόδοσής τους, του φόρτου δικτύου που επιφέρουν καθώς και της ποιότητας υπηρεσιών (QoS) και δεδομένων (QoD) που προσφέρουν. Η ποιότητα υπηρεσιών και δεδομένων αποτελεί ένα είδος μετρικών των πρωτοκόλλων που μελετάμε. Η ποιότητα υπηρεσιών αναφέρεται στον αριθμό βημάτων για το επιθυμητό αποτέλεσμα και στο συνολικό αριθμό μηνυμάτων μέχρι την περάτωση της ερώτησης. Η ποιότητα δεδομένων αναφέρεται στο μέσο όρο της ημερομηνίας λήξης του επιστρεφόμενου αποτελέσματος και στο ποσοστό επί τοις εκατό των πλειάδων που επιστράφηκαν σε σχέση με όλες τις πλειάδες που υπάρχουν στο δίκτυο και θα έπρεπε να επιστραφούν.

Τέλος σημαντικό μέρος της παρούσας εργασίας αποτελεί η υλοποίηση και τα πειραματικά αποτελέσματα που υπέδειξε αυτή. Για το σκοπό της μελέτης δημιουργήθηκε υλοποίηση, που σκοπό είχε να προσομοιώσει το σύστημα ομότιμων κόμβων που μελετάμε. Στην υλοποίηση κάθε κόμβος μπορεί να ενεργοποιεί κάποιο ή κάποια από τα πρωτόκολλα που μελετήθηκαν θεωρητικά με σκοπό τη μελέτη της αποδοτικότητάς τους στη φρεσκάδα της κατανεμημένης βάσης δεδομένων.

1.2. Σκοπός της Εργασίας

Όπως προαναφέρθηκε, στην παρούσα μελέτη θα εξετασθεί η επεξεργασία ερωτήσεων, η ενημέρωση (update) και η αντιγραφή (replication) δεδομένων (πλειάδων) με ημερομηνία λήξης, σε συστήματα δυναμικών, αδόμητων και μη κεντρικοποιημένων ομότιμων κόμβων (peer-to-peer).

Λόγω αυτής της διαφοροποίησης της βάσης δεδομένων και της ύπαρξης της ημερομηνίας λήξης των πλειάδων, *σκοπός της παρούσας μελέτης* είναι:
α) να γίνει επέκταση της γλώσσας SQL ως προς την ημερομηνία λήξης των πλειάδων
β) να παρουσιαστούν αλγόριθμοι που να υποστηρίζουν πλήρως ερωτήσεις SQL σε κατανεμημένες βάσεις δεδομένων και να υπολογίζουν την ημερομηνία λήξης του επιστρεφόμενου αποτελέσματος
γ) να παρουσιαστούν πρωτόκολλα ενημέρωσης (update) των δεδομένων που σκοπό έχουν να εξετάζουν τη φρεσκάδα των δεδομένων και να πράττουν ανάλογα έτσι ώστε να παραμείνει η τοπική βάση δεδομένων ενημερωμένη
δ) να παρουσιαστούν πρωτόκολλα αντιγραφής (replication) των “φρέσκων” δεδομένων με στόχο πιο φρέσκα δεδομένα σε όλη την κατανεμημένη βάση δεδομένων και την ταχύτερη διάδοση των ενημερώσεων
ε) να γίνει μια θεωρητική μελέτη των πρωτοκόλλων ως προς την απόδοσή τους, το φόρτο δικτύου που επιφέρουν και την ποιότητα δεδομένων και υπηρεσιών που προσφέρουν, και τέλος,
στ) να γίνει πειραματική μελέτη της συμπεριφοράς ενός τέτοιου δικτύου όσο αφορά την απόδοση των πρωτοκόλλων που μελετήθηκαν θεωρητικά. Τα πειραματικά αποτελέσματα που θα παρουσιαστούν, θα πρέπει να δείξουν την επίπτωση του συνδυασμού των διαφορετικών πρωτοκόλλων στη φρεσκάδα της τοπικής βάσης δεδομένων των κόμβων, στην γρήγορη διάδοση των πλειάδων και τέλος στην απόδοση της αναζήτησης τυχαίων πλειάδων που ενδιαφέρουν τον κόμβο κάθε χρονική στιγμή με μικρό βάθος ερώτησης (TTL) κάθε φορά.

1.3. Δομή της Διατριβής

Στο Κεφάλαιο 2 γίνεται η περιγραφή του προβλήματος που μελετάμε. Εδώ μοντελοποιούμε το δίκτυο, δίνουμε μια πλήρη θεωρητική περιγραφή του δικτύου και των λειτουργιών του, δηλαδή τους τύπους των ερωτημάτων που έχουμε, τη βάση δεδομένων έχει ο κάθε κόμβος και τις μεθόδους αναζήτησης που μπορεί να γίνουν. Στο κεφάλαιο 3 παρουσιάζονται και αναλύονται οι διάφοροι τύποι των ερωτήσεων που υποστηρίζει το δίκτυο και δίνονται οι αλγόριθμοι που προτείνουμε για τις διάφορες λειτουργίες. Στο κεφάλαιο 4, παρουσιάζονται τα πρωτόκολλα ενημέρωσης και αντιγραφής που προτείνονται για τη διατήρηση των δεδομένων στην κατανεμημένη βάση δεδομένων που έχουμε. Επίσης στο κεφάλαιο γίνεται και αναφορά στην ποιότητα δεδομένων και υπηρεσιών που προσφέρει το κάθε πρωτόκολλο. Στο κεφάλαιο 5 γίνεται μια αναλυτική μελέτη ως προς την απόδοση των μεθόδων αναζήτησης με πλημμύρα και αναζήτησης με τυχαίες διαδρομές, καθώς και την απόδοση των πρωτοκόλλων ενημέρωσης και αντιγραφής δεδομένων στο σύστημα ως προς την ποιότητα δεδομένων και υπηρεσιών που προσφέρουν. Στο κεφάλαιο 6 παρουσιάζεται η υλοποίηση που δημιουργήθηκε για την προσομοίωση του δικτύου που μελετάμε καθώς και τα πειραματικά αποτελέσματα όπου εξετάζουμε την απόδοση των πρωτοκόλλων ενημέρωσης και αντιγραφής στο δίκτυο καθώς και την ποιότητα υπηρεσιών και δεδομένων που προσφέρουν αν συνδυαστούν με απλά ερωτήματα. Τέλος στο κεφάλαιο 7 δίνεται η σχετική εργασία, όπου παρουσιάζονται διάφορες σχετικές μελέτες που έχουν γίνει και στο κεφάλαιο 8 εκφράζονται διάφορα συμπεράσματα που αφορούν την παρούσα εργασία.

ΚΕΦΑΛΑΙΟ 2. ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Σκοπός του κεφαλαίου αυτού είναι να δοθεί η περιγραφή του προβλήματος και μια πλήρη εικόνα του συστήματος που μελετάται. Επίσης, δίνεται και μια εφαρμογή για το που αυτό μπορεί να χρησιμεύσει. Στην ενότητα 2.1 περιγράφεται το δίκτυο που μελετάμε. Καθώς το δίκτυο αυτό εντάσσεται στα δίκτυα ομότιμων κόμβων (peer-to-peer), στην ενότητα 2.1.1 γίνεται εισαγωγή στα peer-to-peer συστήματα και ανάλυση κυρίως των αρχιτεκτονικών τους. Στην επόμενη ενότητα (2.1.2) γίνεται μοντελοποίηση και πλήρη περιγραφή του προβλήματος. Στην ενότητα 2.2 περιγράφεται το πως γίνεται η διαχείριση της κατανεμημένης βάσης δεδομένων που έχει κάθε κόμβος. Επιπλέον, δίνεται έμφαση στην ποιότητα Δεδομένων και Υπηρεσιών και στους τύπους ερωτήσεων που μπορούν να υποστηριχθούν στο σύστημα. Τέλος, στην ενότητα 2.3, δίνεται μια εφαρμογή του πραγματικού κόσμου που αντικατοπτρίζει πλήρως το πρόβλημα που μελετάμε.

2.1. Περιγραφή του Δικτύου

Στην παρούσα ενότητα θα γίνει περιγραφή του συστήματος. Το σύστημα αυτό εντάσσεται στα δίκτυα ομότιμων κόμβων (peer-to-peer). Έτσι, αρχικά, στην υποενότητα 2.1.1 δίνεται μια εισαγωγή για τα peer-to-peer συστήματα όπου

αναλύονται τα διαφορετικά είδη τους και η αρχιτεκτονική τους, καθώς και γίνεται λόγος για τη μεγάλη δημοφιλία και σπουδαιότητά τους στον σημερινό κόσμο. Στην συνέχεια γίνεται μοντελοποίηση του συστήματος που μελετάμε (2.1.2) και τέλος περιγράφεται ο τρόπος αναζήτησης των δεδομένων στο σύστημα με παραδείγματα (2.1.3).

2.1.1. Δίκτυα Ομότιμων Κόμβων

Τα δίκτυα ομότιμων κόμβων (peer-to-peer), αντίθετα με τις παραδοσιακές εφαρμογές πελάτη-εξυπηρετητή (client-server), είναι κατανεμημένα συστήματα ομότιμων κόμβων τα οποία χρησιμοποιούν την υποδομή και τους πόρους του διαδικτύου για την επικοινωνία μεταξύ των κόμβων. Η επικοινωνία μεταξύ τους γίνεται πάνω από το φυσικό TCP/IP δίκτυο, χωρίς να απαιτείται κάποιος κεντρικός έλεγχος. Τα συστήματα αυτά χρησιμοποιούνται κυρίως για το διαμοιρασμό διαφόρων αρχείων μεταξύ των χρηστών. Κύριο χαρακτηριστικό των κόμβων είναι το γεγονός ότι είναι ομότιμοι, δηλαδή κάθε κόμβος συμμετέχει ισάξια στο σύστημα, έχει τα ίδια δικαιώματα και τις ίδιες υποχρεώσεις με τους υπόλοιπους.

Η αρχή έγινε το 1999 με το Napster [6] και λόγω της μεγάλης απήχησής του, ακολούθησαν και άλλες ευρέως διαδεδομένες εφαρμογές όπως είναι το Gnutella, KaZaA, BitTorrent κ.α. Τέτοια δίκτυα σήμερα αποτελούν το μεγαλύτερο ποσοστό της κίνησης του διαδικτύου [16] (με πάνω από το 50% της συνολικής κίνησης) και μέσω αυτών διακινείται συνολικός όγκος δεδομένων της τάξης του terra byte. Για το λόγο αυτό, τέτοια συστήματα, κέντρισαν το ενδιαφέρον της ερευνητικής κοινότητας εδώ και μερικά χρόνια. Έχει γίνει μεγάλη έρευνα για συστήματα πρώτης γενιάς όπως είναι τα παραπάνω, αλλά και δεύτερης γενιάς που βασίζονται σε DHT (distributed hash tables) όπως είναι το CAN και το CHORD.

Γενικά, υπάρχουν διάφορες αρχιτεκτονικές για δίκτυα ομότιμων κόμβων, όπως είναι τα κεντρικοποιημένα και τα αποκεντρικοποιημένα ή μη-κεντρικοποιημένα τα οποία διαχωρίζονται σε δομημένα και αδόμητα.

Κεντρικοποιημένα συστήματα [6], είναι συστήματα τα οποία χρησιμοποιούν έναν ή περισσότερους ισχυρούς κόμβους-εξυπηρετητές (servers), όπως γινόταν αρχικά με το Napster και σήμερα με το δίκτυο BitTorrent. Όλοι οι απλοί κόμβοι (peers) συνδέονται στο δίκτυο μέσω αυτών των εξυπηρετητών και οι τελευταίοι αναλαμβάνουν τις διαδικασίες για εισαγωγή/διαγραφή κόμβου από το σύστημα, καθώς και τις προωθήσεις των ερωτημάτων. Οι εξυπηρετητές επίσης διαθέτουν πληροφορίες που αφορούν το ποια αρχεία διανέμονται στο δίκτυο και ποιοι κόμβοι τα προσφέρουν. Κύριο μειονέκτημα αυτής της αρχιτεκτονικής είναι η μη-ανεκτικότητα σε σφάλματα. Όταν κάποιος server αποτύχει, λόγω του σημαντικού φόρτου εργασίας που είναι υποχρεωμένος να υποστεί, υπάρχει ο κίνδυνος της κατάρρευσης του δικτύου ή σε περίπτωση ύπαρξης εναλλακτικών εξυπηρετητών, της δυσλειτουργίας του. Ένα άλλο μειονέκτημα αποτελεί και περιορισμένη δυνατότητα κλιμάκωσης αυτών των δικτύων. Δηλαδή, τέτοια δίκτυα δεν μπορούν να δεχτούν απεριόριστο αριθμό από κόμβους διότι αυξάνεται κατά πολύ ο φόρτος στο εξυπηρετητή με αποτέλεσμα τη δυσλειτουργία όλου του συστήματος.

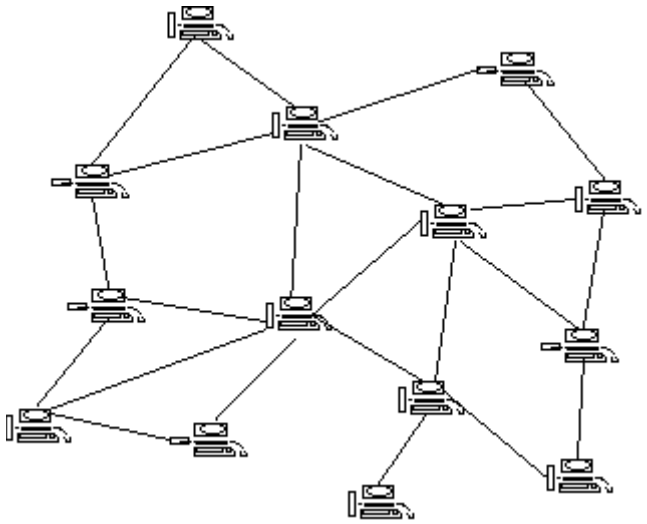
Στα αποκεντρικοποιημένα συστήματα [6] δεν υπάρχει η έννοια του εξυπηρετητή. Οι αιτήσεις του κάθε χρήστη δρομολογούνται από τους ίδιους τους χρήστες δυναμικά χωρίς τη μεσολάβηση κάποιου εξυπηρετητή (server). Εδώ ανάλογα με την κατασκευή του συστήματος μπορεί να αποφευχθεί αποτυχία δικτύου λόγω κατάρρευσης κόμβων ή υπερφόρτωσης κάποιων μεμονωμένων κόμβων στο δίκτυο και η κλιμάκωσή τους είναι κατά πολύ μεγαλύτερη.

Τα αποκεντρικοποιημένα συστήματα ανάλογα με τη κατασκευή τους χωρίζονται σε δομημένα και αδόμητα [6]. Δομημένα είναι τα συστήματα όπου η τοπολογία του δικτύου είναι αυστηρά ελεγχόμενη και τα αρχεία τοποθετούνται σε αυστηρά καθορισμένες θέσεις και όχι τυχαία, έτσι ώστε να είναι ταχύτερη και αξιόπιστη η αναζήτησή τους. Παραδείγματα δομημένων συστημάτων είναι το CAN [12] και το CHORD [13]. Τα συστήματα αυτά χρησιμοποιούν κατανεμημένους πίνακες κατακερματισμού (DHT) και ζεύγη τιμών-κλειδιών για να αποθηκεύσουν και να ανακτήσουν τα δεδομένα τους. Αδόμητα είναι τα συστήματα όπου δεν υπάρχει κάποιος έλεγχος στην τοπολογία του δικτύου ούτε στην τοποθέτηση των αρχείων. Σε αυτά τα συστήματα η αναζήτηση των δεδομένων είναι συνήθως τυφλή και η

δρομολόγηση των αιτήσεων γίνεται με πλημμύρα (flooding) ή με τυχαίες διαδρομές (random walks). Θα πρέπει να επισημανθεί ότι η τοπολογία του δικτύου αφορά τη συνδεσιμότητα των κόμβων μεταξύ τους και τον γενικό γράφο που σχηματίζεται από αυτούς. Όπως αναφέρθηκε και παραπάνω, η τοπολογία του δικτύου σχετίζεται με ένα overlay επίπεδο διότι χρησιμοποιεί την υπάρχουσα υποδομή του διαδικτύου.

2.1.2. Μοντελοποίηση του Δικτύου

Το σύστημα που θα μελετήσουμε θα έχει τη μορφή αδόμητου, μη κεντρικοποιημένου peer-to-peer συστήματος, με όλους τους κόμβους ισάξιους-ομότιμους μεταξύ τους. Επίσης το σύστημά μας, είναι δυναμικό, δηλαδή οι κόμβοι μπορεί να εισέρχονται και να αποχωρούν από το σύστημα αυτοβούλως, όσες φορές και όποτε το επιθυμούν. Σχηματικά το δίκτυο απεικονίζεται στο σχήμα 2.1.



Σχήμα 2.1 Αδόμητο, μη Κεντρικοποιημένο P2P Δίκτυο

Κάθε κόμβος μπορεί να είναι ένας σταθερός ή φορητός υπολογιστής, ή κάποιος υπολογιστής παλάμης (PDA) που χρησιμοποιεί την υποδομή του διαδικτύου για την επικοινωνία μεταξύ των άλλων κόμβων στο peer-to-peer δίκτυο.

Όπως έχει ήδη αναφερθεί στο σύστημά μας οι κόμβοι είναι ομότιμοι και ισάξιοι μεταξύ τους. Δηλαδή, δεν υπάρχει κάποια δομή με υπερκόμβους (super-peers) όπως εισήγαγε το KaZaA και χρησιμοποιεί ακόμα και τώρα η peer-to-peer εφαρμογή Skype [14]. Το δίκτυο είναι δυναμικό, δηλαδή οι κόμβοι μπορεί να εισέρχονται και να αποχωρούν όταν και όποτε το επιθυμούν και για το λόγο αυτό το σύστημα είναι και αρκετά αναξιόπιστο: δεν μπορεί να προβλεφθεί πότε και αν κάποιος κόμβος είναι διαθέσιμος και έτοιμος να απαντήσει μια ερώτηση. Άρα και το μονοπάτι της ερώτησης δεν είναι από πριν γνωστό και δημιουργείται δυναμικά.

Ο κάθε κόμβος στο δίκτυο μπορεί να λειτουργεί ανεξάρτητα από τους άλλους κόμβους. Δηλαδή, κάθε κόμβος μπορεί να εκτελεί διάφορα πρωτόκολλα ενημέρωσης ή αντιγραφής δεδομένων που θα αναλυθούν παρακάτω. Μπορεί να ενημερώνει τα δεδομένα του όποτε έχει τη δυνατότητα να το κάνει, ή να δημιουργεί μόνος του ενημερωμένα δεδομένα και έπειτα να τα διοχετεύει στους υπόλοιπους κόμβους. Τέλος κάθε κόμβος μπορεί να δημιουργεί οποιουδήποτε τύπου ερώτηση επιθυμεί τη στιγμή που το επιθυμεί. Για παράδειγμα ένας τύπος ερώτησης που μπορεί να υποβάλλει κάποιος κόμβος είναι η αναζήτηση κάποιου δεδομένου. Η πορεία της αναζήτησης μπορεί να γίνει με διάφορες μεθόδους που θα αναλύσουμε παρακάτω. Ανεξάρτητα από το τι δεδομένα υπάρχουν στο δίκτυο ή που βρίσκονται αυτά, ο κόμβος οποιαδήποτε στιγμή μπορεί να τα ζητήσει ασχέτως από την επιτυχία ή μη του αποτελέσματος. Επίσης, κάθε κόμβος παίρνει μέρος στην απάντηση της ερώτησης κάποιου γείτονα αν του ζητηθεί, με τη μοναδική προϋπόθεση να είναι διαθέσιμος.

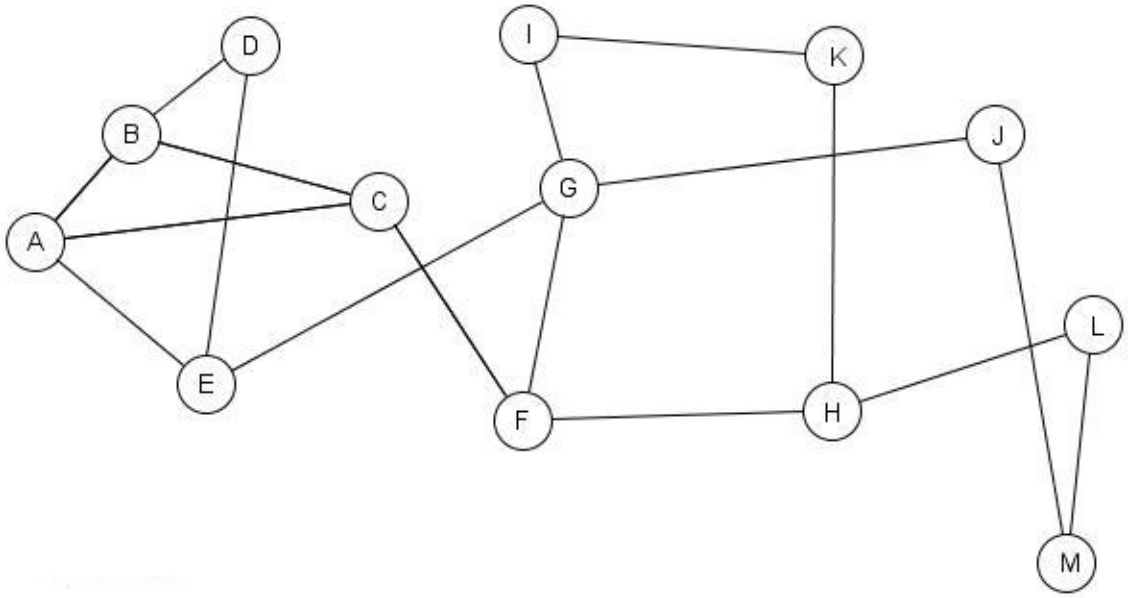
Πολύ σπουδαίο είναι ότι το σύστημα που μελετάται είναι αρκετά μεγάλο ως προς το πλήθος των κόμβων και επιπλέον είναι σημαντικά επεκτάσιμο. Μπορούν να συνδεθούν όσοι περισσότεροι κόμβοι γίνεται χωρίς να επηρεάζεται η καλή λειτουργία του (δεν αυξάνονται δηλαδή οι αποτυχίες κόμβων) ούτε και υπερφορτώνονται ορισμένοι κόμβοι καθώς δεν υπάρχουν υπερκόμβοι (super peers).

2.1.3. Μέθοδοι Αναζήτησης

Δύο μέθοδοι αναζήτησης που μπορούν να χρησιμοποιηθούν στο σύστημά μας είναι η μέθοδος της πλημμύρας (*flooding*) καθώς η μέθοδος random walks. Η μέθοδος της

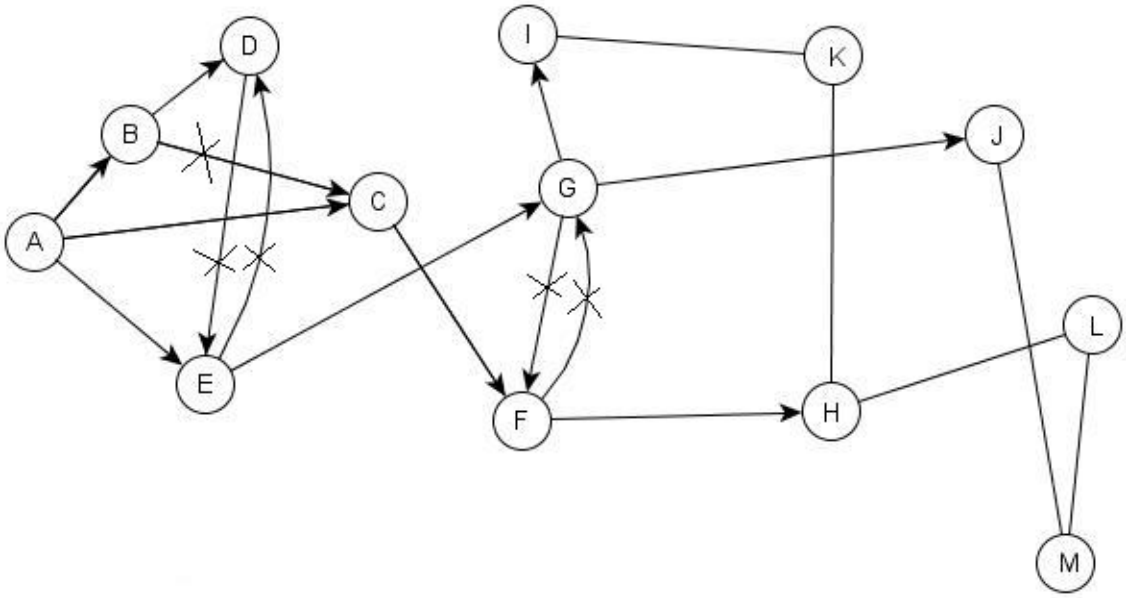
πλημμύρας [6] είναι ο πιο κλασικός αλγόριθμος για την επικοινωνία κόμβων μέσα σε ένα δίκτυο. Ο αλγόριθμος έχει ως εξής: κάποιος κόμβος επιθυμεί να υποβάλλει μια ερώτηση και επικοινωνεί με όλους τους γείτονές του. Στη συνέχεια οι γειτονικοί κόμβοι προωθούν το μήνυμα στους δικούς τους γείτονες, εκτός βέβαια από τον γείτονα από τον οποίο έλαβαν το μήνυμα. Η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί το δεδομένο που αναζητείται, ή η αναζήτηση να φτάσει σε κάποιο βάθος δικτύου (time to live-TTL). Το βάθος δικτύου το θέτει ο αρχικός κόμβος και κάθε φορά που κάποιος κόμβος προωθεί το μήνυμα το βάθος αυξάνεται κατά ένα. Όταν το βάθος δικτύου γίνει ίσο με το TTL το μήνυμα σταματάει να προωθείται και επιστρέφει προς τα πίσω από το ίδιο μονοπάτι μέχρι να φθάσει στον αρχικό κόμβο.

Με την πλημμύρα έχουμε ουσιαστικά μια κατά πλάτος (BFS) διάσχιση της τοπολογίας του δικτύου. Ας υποθέσουμε, λοιπόν, ότι έχουμε το δίκτυο του σχήματος 2.2



Σχήμα 2.2 Σχηματική Απεικόνιση του P2P Δικτύου

Ένα παράδειγμα αναζήτησης έχει ως εξής: Έστω ότι ο κόμβος A ξεκινάει μια αναζήτηση με TTL=3 και την προωθεί στους γείτονές του: B,C,E. Οι κόμβοι B,C,E εξετάζουν αν διαθέτουν το αρχείο που αναζητείται. Αν το διαθέτουν στέλνεται θετική απάντηση στον κόμβο A και η αναζήτηση σταματάει. Διαφορετικά προωθούν το μήνυμα της αναζήτησης και αυτοί με τη σειρά τους στους γείτονές τους D,F,G. Εκείνοι αν δεν διαθέτουν το αρχείο προωθούν την αναζήτηση στους I,H και J. Εκεί η προώθηση του μηνύματος σταματάει, καθώς το βάθος δικτύου είναι 3 και οι απαντήσεις πηγαίνουν προς τα πίσω το μονοπάτι μέχρι να φθάσουν τον A. Σημειώνουμε ότι αν το αρχείο βρεθεί σε ενδιάμεσο κόμβο τότε η προώθηση της αναζήτησης σταματάει και η απάντηση στέλνεται στον αρχικό κόμβο από το ίδιο μονοπάτι της αναζήτησης. Σχηματικά η προώθηση του μηνύματος παρουσιάζεται στο σχήμα 2.3.



Σχήμα 2.3 Προώθηση του Μηνύματος με τη Μέθοδο της Πλημμύρας με TTL=3

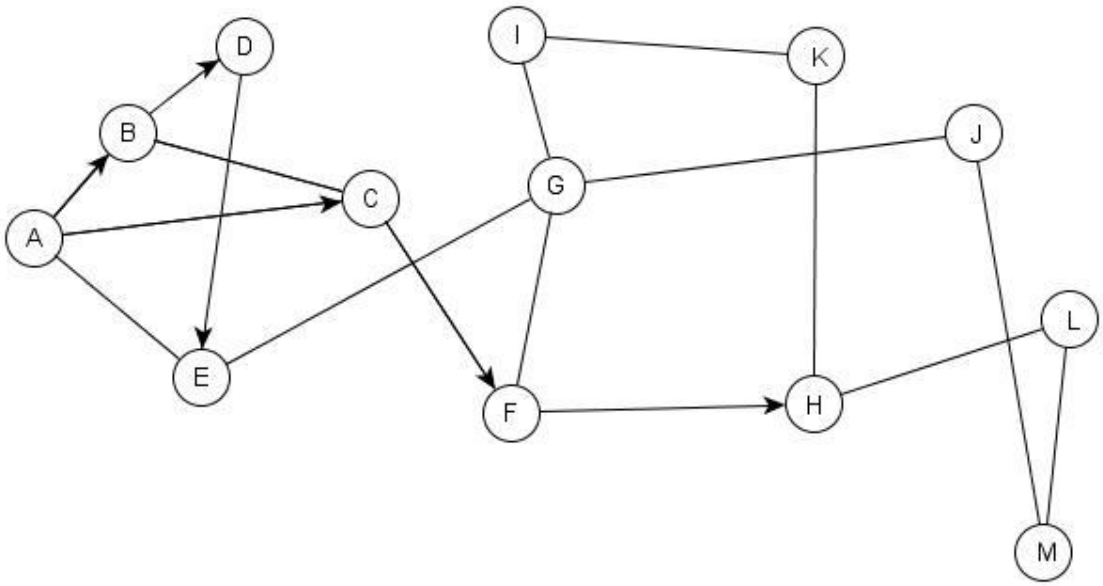
Από το σχήμα 2.3 παρατηρούμε ότι έχουμε αρκετούς κύκλους. Οι κόμβοι C,D,E,F,G θα λάβουν το μήνυμα δύο φορές. Σε τέτοια περίπτωση οι κόμβοι θα απορρίψουν το μήνυμα της αναζήτησης τη δεύτερη φορά που θα το δεχτούν χωρίς να το επεξεργαστούν. Γενικά, ο αλγόριθμος της πλημμύρας διασφαλίζει ότι όλοι οι κόμβοι

με βάθος μικρότερο του TTL θα λάβουν το μήνυμα, αλλά κάποιοι κόμβοι θα λάβουν το ίδιο μήνυμα πολλές φορές λόγω κύκλων στην τοπολογία του δικτύου. Όπως είναι εμφανές, ο μηχανισμός αυτός έχει κάποια σημαντικά μειονεκτήματα. Το δίκτυο πλημμυρίζεται από πάρα πολλά μηνύματα, που έχουν ως αποτέλεσμα τον αυξημένο φόρτο εργασίας στον εκάστοτε κόμβο και στο δίκτυο. Επιπλέον, η επιλογή της κατάλληλης τιμής για την παράμετρο TTL δεν είναι εύκολη. Αν η τιμή της είναι μεγάλη, το σύστημα επιβαρύνεται σημαντικά σε κάθε αναζήτηση, ενώ αν η τιμή της είναι μικρή, είναι πιθανό ο κόμβος που ξεκίνησε την αναζήτηση να μην εντοπίσει τα επιθυμητά αποτελέσματα, ακόμα και αν υπάρχουν κάπου στο δίκτυο. Τέλος ένα ακόμη πρόβλημα όπως αναφέρθηκε και προηγούμενα, είναι ότι το ίδιο μήνυμα μπορεί να περάσει από έναν κόμβο περισσότερες από μια φορές, εξαιτίας των πολλαπλών γειτόνων του. Αυτό έχει ως αποτέλεσμα η επιβάρυνση των κόμβων ως προς τα μηνύματα να παραμένει σημαντική. Αντιθέτως, η μέθοδος αναζήτησης αυτή, έχει το πλεονέκτημα να είναι αποδοτική σε δίκτυα με μεγάλη ικανότητα κλιμάκωσης. Επειδή το μονοπάτι της αναζήτησης δημιουργείται δυναμικά και δεν έχουμε την ύπαρξη εξυπηρετητών ή υπερκόμβων, άρα την υπερφόρτωση ορισμένων κόμβων στο δίκτυο, η αναζήτηση αποδίδει ανεξάρτητα από το πλήθος των κόμβων.

Εναλλακτικά αυτής της μεθόδου αναζήτησης, για εξοικονόμηση κυρίως μηνυμάτων χρησιμοποιήθηκε και η μέθοδος των *τυχαίων διαδρομών* (random walks) [6]. Σύμφωνα με αυτή τη μέθοδο ο αρχικός κόμβος υποβάλει μια ερώτηση, με τη μορφή μηνύματος, σε *k* τυχαία επιλεγμένους γείτονές του. Τα k αυτά μηνύματα ονομάζονται περιπατητές (walkers) και το καθένα ακολουθεί τυχαία τη δική του διαδρομή όπου κάθε ενδιάμεσος κόμβος τα προωθεί σε ένα τυχαίο γείτονά του σε κάθε βήμα. Οι *walkers* τερματίζονται όταν εντοπιστεί το αρχείο που αναζητείται ή σε διαφορετική περίπτωση με χρήση της παραμέτρου TTL, όπως και στη μέθοδο της πλημμύρας. Για να πετύχουμε τον τερματισμό όλων των walkers όταν εντοπιστεί το αρχείο υπάρχει η μέθοδος του “ελέγχου” (checking). Σύμφωνα με αυτήν κάθε walker ρωτάει τον αρχικό κόμβο για την επιτυχία ή όχι της αναζήτησης και έπειτα προωθεί το μήνυμα στον επόμενο κόμβο. Το σημαντικότερο πλεονέκτημα της μεθόδου αυτής, είναι ο μικρός αριθμός μηνυμάτων που παράγει. Στην χειρότερη περίπτωση, παράγει *k* TTL* μηνύματα. Τα μειονεκτήματά του είναι η αστάθεια στην απόδοση, αφού η επιτυχία

μιας αναζήτησης εξαρτάται από τις τυχαίες επιλογές των ενδιάμεσων κόμβων που έγιναν.

Το προηγούμενο παράδειγμα, η αναζήτηση στο δίκτυο με τη μέθοδο random walks από τον κόμβο Α θα είχε το αποτέλεσμα του σχήματος 2.4 με *k* =2 και TTL=3.



Σχήμα 2.4 Προώθηση του Μηνύματος με τη Μέθοδο Random Walks

Στο παράδειγμα ο κόμβος Α ξεκινάει μια ερώτηση με TTL=3 και το μήνυμα το προωθεί σε 2 τυχαία επιλεγμένους γείτονές του (k=2): B,C. Έπειτα προωθείται στο D,F και τέλος στο E,H, στη χειρότερη περίπτωση που δεν εντοπιστεί το αρχείο που αναζητάμε σε κάποιο ενδιάμεσο κόμβο. Στην περίπτωση που εντοπιστεί σε ενδιάμεσο κόμβο, η αναζήτηση σταματάει και το αποτέλεσμα επιστρέφει στο κόμβο Α από το ίδιο μονοπάτι. Παρατηρούμε ότι με τη μέθοδο αυτή έχουμε σαφώς λιγότερα μηνύματα από προηγουμένως, χρησιμοποιώντας flooding και περιορίζονται οι κύκλοι. Όμως περιορίζονται και οι κόμβοι που συναντάμε στην αναζήτηση καθώς εδώ συναντάμε 6 ενώ με την προηγούμενη μέθοδο 9 κόμβους.

2.2. Διαχείριση Δεδομένων

Στην παρούσα ενότητα γίνεται λόγος για τη διαχείριση της βάση δεδομένων που μπορεί να έχει ο κόμβος. Αρχικά, γίνεται λόγος για τη βάση δεδομένων που μπορεί να έχει κάθε κόμβος και τη διαφοροποίησή της από τις κοινές βάσεις δεδομένων. Στη συνέχεια ορίζουμε την ποιότητα δεδομένων και υπηρεσιών, δύο στοιχεία που μας βοηθάνε να ελέγξουμε την ευρωστία του δικτύου και την αποτελεσματικότητα των διαφόρων πρωτοκόλλων. Τέλος αναλύονται οι τύποι ερωτήσεων που μπορεί να θέτει κάποιος κόμβος στους ομότιμούς του στο δίκτυο.

2.2.1. Βάση Δεδομένων του Κόμβου

Στο δίκτυο κάθε κόμβος έχει μια βάση δεδομένων πάνω στην οποία εκτελούνται ερωτήσεις.

Η **βάση δεδομένων** που έχει κάθε κόμβος διαφέρει από τη βάση όπως τη γνωρίζουμε από την παραδοσιακή DBMS, καθώς και από αυτή που κατέχουν οι κόμβοι σε ένα παραδοσιακό peer-to-peer δίκτυο, δηλαδή μόνο από απλά αρχεία. Η βάση δεδομένων που διατηρεί κάθε κόμβος, αποτελείται από ένα σύνολο από πίνακες. Οι πίνακες αυτοί είναι τοπικά αποθηκευμένοι πίνακες, με τα περιεχόμενά τους τοπικά αποθηκευμένα στον κόμβο. Αποτελούνται από εγγραφές-πλειάδες που χρησιμοποιούνται για την απάντηση διαφόρων ερωτήσεων προερχομένων από τον ίδιο τον κόμβο ή από άλλους κόμβους στο δίκτυο. Επιπλέον κάθε κόμβος μπορεί να έχει πίνακες νοητούς ή υβριδικούς που προέρχονται από τους υπόλοιπους κόμβους στο δίκτυο. Οι *νοητοί* πίνακες περιέχουν δεδομένα που δεν είναι τοπικά αποθηκευμένα στον κόμβο, αλλά προέρχονται από τους κόμβους που βρίσκονται στο δίκτυο τη στιγμή της υποβολής της ερώτησης. Για να μπορεί ο κόμβος να επεξεργαστεί τέτοιους πίνακες, πρέπει να συλλέξει τις εγγραφές από τους ομότιμούς του και να τις αποθηκεύσει τοπικά.. Οι *υβριδικοί πίνακες*, περιέχουν δεδομένα που ένα μέρος είναι τοπικά τοποθετημένα ενώ το άλλο συλλέγεται από το δίκτυο δυναμικά. Οι υβριδικοί πίνακες είναι συνδυασμός των τοπικά αποθηκευμένων και των νοητών, αλλά και πάλι ισχύει ότι ο κόμβος χρειάζεται να συλλέξει και να αποθηκεύσει τοπικά τις εγγραφές από τους ομότιμούς του που τον ενδιαφέρουν και

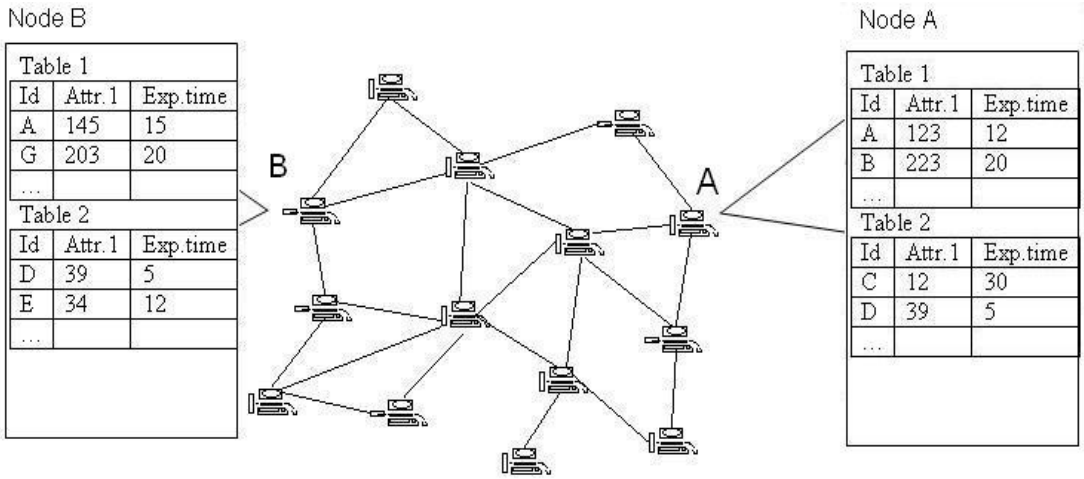
έπειτα να τις επεξεργαστεί. Το ποιοι πίνακες είναι νοητοί ή υβριδικοί δεν θα μας απασχολήσει στο υπόλοιπο της εργασίας μας, αναφέρθηκε απλά για να γίνει σαφές το γεγονός ότι το περιεχόμενο της βάσης δεδομένων στο κόμβο δεν είναι στατικό αλλά μεταβαλλόμενο, και εκτός από τις εγγραφές που αποθηκεύει τοπικά, περιλαμβάνει και ένα σύνολο εγγραφών που συλλέγονται κατά τη διάρκεια μιας ερώτησης από υπόλοιπους κόμβους. Οι εγγραφές που συλλέγονται, μπορεί να ανήκουν σε πίνακα που διαθέτει ο κόμβος ή και όχι, καθώς και εγγραφές που ήδη είναι τοπικά αποθηκευμένες αλλά είναι πιο πρόσφατες από τις ήδη υπάρχουσες.

Σημαντική διαφοροποίηση της βάσης δεδομένων που διαθέτει ο κόμβος, αποτελεί η έννοια της "*φρεσκάδας*" (freshness) των δεδομένων του.

Φρεσκάδα καλείται η ιδιότητα που χαρακτηρίζει το πόσο πρόσφατη είναι μια πλειάδα δεδομένων. Η φρεσκάδα εφαρμόζεται ως επέκταση σε κάθε πλειάδα του κόμβου και μπορεί να είναι είτε: i) *time-based:* δηλαδή ο χρόνος από την τελευταία ενημέρωση της πλειάδας, ii) *lag-based:* πλήθος των χαμένων ενημερώσεων είτε πιο απλά iii) *based on expiration time:* η ημερομηνία λήξης της πλειάδας.

Στην εργασία μας θα μας απασχολήσει μόνο η ημερομηνία λήξης της πλειάδας [1]: κάθε πλειάδα θα έχει ένα επιπλέον γνώρισμα που θα σχετίζεται με την ημερομηνία λήξης της. *Η ημερομηνία λήξης* της πλειάδας αποτελεί γνώρισμα που δηλώνει το χρόνο ζωής της πλειάδας δηλαδή τη χρονική στιγμή έπειτα από την οποία η πλειάδα λήγει και παύει να ισχύει. Για το λόγο αυτό πρέπει, όταν γίνεται μια ερώτηση τύπου SQL, να υπολογιστεί η ημερομηνία λήξης του επιστρεφόμενου αποτελέσματος. Επίσης, θα πρέπει να αποκλείονται όσες πλειάδες έχουν λήξει και να ενημερώνονται καταλλήλως.

Για την πλήρη κατανόηση του δικτύου που μελετάμε, δίνεται το σχήμα 2.5, όπου πέρα από τη σχηματική απεικόνιση του δικτύου όπως παρουσιάστηκε στο σχήμα 2.1, απεικονίζονται οι τοπικά αποθηκευμένοι πίνακες δύο κόμβων έστω των κόμβων A, B.



Σχήμα 2.5 Παραδείγματα Βάσης Δεδομένων των Κόμβων του Συστήματος

Όπως παρατηρούμε από το σχήμα 2.5, η βάση δεδομένων κάθε κόμβου μπορεί να περιέχει έναν ή περισσότερους τοπικά αποθηκευμένους πίνακες. Οι εγγραφές των πλειάδων έχουν ένα επιπλέον γνώρισμα την ημερομηνία λήξης τους. Η ημερομηνία λήξης δηλώνει τη χρονική στιγμή που η πλειάδα θα λήξει, για παράδειγμα κάποια πλειάδα με ημερομηνία λήξης 5, δηλώνει ότι τη χρονική στιγμή 5 θα λήξει. Η χρονική στιγμή μπορεί να είναι η ώρα του πραγματικού κόσμου (πχ 5h) έτσι ώστε όλοι οι κόμβοι να τη γνωρίζουν και να είναι ανεξάρτητη από τη χρονική στιγμή που η πλειάδα εισήλθε στο σύστημα.

Επίσης όπως παρατηρούμε στον τοπικά αποθηκευμένο πίνακα του κόμβου, κάθε εγγραφή έχει μοναδικό κλειδί (id) που την προσδιορίζει, αλλά σε πίνακα διαφορετικού κόμβου μπορεί να υπάρχει εγγραφή με ίδιο id όπου και αποτελεί αντίγραφο της πλειάδας. Για παράδειγμα ο κόμβος Α έχει στο πίνακα 2 μια εγγραφή με id = C. Ο κόμβος Β έχει ζητήσει από τον Α την εγγραφή με id = C και την έχει αποθηκεύσει τοπικά. Έτσι δημιουργείται ένα αντίγραφο της πλειάδας. Έπειτα, αυτό το αντίγραφο μπορεί να το χρησιμοποιήσει για την επεξεργασία ερωτήσεων αλλά μπορεί και να το ενημερώσει, δηλαδή να έχει μια καινούρια πληροφορία έτσι ώστε να μεταβάλλει ένα ή περισσότερα γνωρίσματα και την ημερομηνία λήξης του. Για παράδειγμα ο κόμβος Α και ο κόμβος Β έχουν ένα αντίγραφο μιας εγγραφής με id=A του πίνακα 1. Όμως ο κόμβος Α έχει ενημερώσει το αντίγραφο και πλέον το

αντίγραφο διαθέτει μια πιο μεταγενέστερη ημερομηνία λήξης. Αυτό το αντίγραφο καλείται ενημερωμένο αντίγραφο αυτής της εγγραφής.

2.2.2. Ποιότητα Δεδομένων και Υπηρεσιών

Κάθε κόμβος στο δίκτυο μπορεί να δημιουργεί νέες πλειάδες, να ενημερώνει τις ήδη υπάρχουσες και να μεταβάλλει τη τοπική βάση δεδομένων του ανεξάρτητα από τους υπόλοιπους κόμβους. Το ζήτημα που τίθεται πλέον είναι ότι από τη στιγμή που οι ενημερώσεις των δεδομένων γίνονται από τον κάθε κόμβο ξεχωριστά, πρέπει να βρεθεί ένας αποδοτικός τρόπος για ταχύτατη διάδοση των ενημερωμένων αντιγράφων κατά μήκος όλου του δικτύου, έτσι ώστε κάθε κόμβος να μπορεί να εντοπίζει τα δεδομένα που επιθυμεί εύκολα, με όσο το δυνατόν μικρότερο βάθος ερώτησης και τέλος η κατανεμημένη βάση δεδομένων να είναι όσο γίνεται πιο ενημερωμένη. Για το λόγο αυτό στο κεφάλαιο 4 παρουσιάζονται διάφορα πρωτόκολλα ενημέρωσης και δημιουργίας αντιγράφων έτσι ώστε να ενημερώνονται τα αντίγραφα που λήγουν και να διαδίδονται οι ενημερώσεις όσο γίνεται περισσότερο.

Στην ανάλυση των πρωτοκόλλων ενημέρωσης και δημιουργίας αντιγράφων, σε συνδυασμό με τα διάφορα ερωτήματα που θέτει ο κόμβος και τα αποτελέσματα που επιστρέφονται θα μας απασχολήσουν δύο στοιχεία: *η ποιότητα των δεδομένων* (Quality of Data) και *η ποιότητα των υπηρεσιών* (Quality of Service) και πώς αυτά μπορούν να βελτιωθούν.

Η **ποιότητα των δεδομένων** αναφέρεται στα εξής:

Freshness (φρεσκάδα): η ημερομηνία λήξη (expiration time) του επιστρεφόμενου αποτελέσματος και

Recall: το ποσοστό των πλειάδων που επιστρέφονται στο αποτέλεσμα σε σχέση με όλες τις πλειάδες που υπάρχουν στο δίκτυο και θα έπρεπε να επιστραφούν.

Η **ποιότητα των υπηρεσιών** αναφέρεται στα εξής:

Response time: ο αριθμός των βημάτων (hops) για το επιθυμητό αποτέλεσμα

Message overhead: συνολικός αριθμός των μηνυμάτων μέχρι την περάτωση της ερώτησης.

Στο σύστημά μας προσπαθούμε να βελτιώσουμε την ποιότητα των δεδομένων, δηλαδή να έχουμε καλύτερο Freshness και Recall με όσο το δυνατόν λιγότερο αριθμό βημάτων και μηνυμάτων για την ολοκλήρωση της ερώτησης. Βέβαια όσο καλύτερη ποιότητα δεδομένων έχουμε, τόσο αυξάνονται τα βήματα και τα μηνύματα στο δίκτυο, δηλαδή επιβαρύνεται η ποιότητα υπηρεσιών. Έτσι προσπαθούμε να βρούμε μια χρυσή τομή μεταξύ των QoS και QoD ώστε να βελτιωθεί η απόδοση του συστήματος και η ικανότητα κλιμάκωσής του.

2.2.3. Τύποι Ερωτήσεων

Οι ερωτήσεις που έχουμε στο δίκτυο και μπορεί να τις υποβάλλει κάθε κόμβος είναι τύπου SQL. Μπορεί να έχουμε κάποιο απλό select του τύπου Select ή και ακόμα και πιο περίπλοκες λειτουργίες όπως join ή aggregation όπως average, minimum, maximum, sum και count. Εδώ θα αλλάξει ο τύπος της SQL ερώτησης για να υποστηριχθεί η ημερομηνία λήξης των δεδομένων καθώς και ο τρόπος επεξεργασίας της ερώτησης για να υποστηριχθεί η κατανεμημένου τύπου βάση δεδομένων που υπάρχει στο σύστημα.

Στο κεφάλαιο 3 υπάρχει πλήρης περιγραφή και ανάλυση των ερωτήσεων που μπορεί να θέτει κάποιος κόμβος καθώς και του τρόπου υπολογισμού της ημερομηνίας λήξης του επιστρεφόμενου αποτελέσματος.

2.3. Πιθανές Εφαρμογές

Στην παρούσα φάση της εργασίας τίθεται το ερώτημα αν κατά πόσο το προς μελέτη σύστημα είναι ρεαλιστικό και σε ποια περίπτωση μπορεί να χρησιμοποιηθεί και να βοηθήσει με εφαρμογές τον πραγματικό κόσμο.

Ας υποθέσουμε ότι οχήματα κινούνται σε ένα μεγάλο εθνικό δρόμο όπως στο σχήμα 2.6.

Σχήμα 2.6 Οχήματα σε Μεγάλο Εθνικό Δρόμο

Έστω ότι κάθε όχημα είναι συνδεδεμένο, με κάποιο τρόπο λ.χ. ασύρματο δίκτυο GPRS, στο διαδίκτυο και όλα μαζί αποτελούν ένα p2p δίκτυο αδόμητο, μη κεντρικοποιημένο, με πολλούς κόμβους-οχήματα που μπορεί να εισέρχονται ή να αποχωρούν από το δίκτυο συνεχώς και χωρίς προειδοποίηση. Επίσης, επειδή κάθε όχημα μπορεί να κινείται με διαφορετική ταχύτητα, μπορεί συχνά να βρίσκεται εκτός δικτύου για κάποιο διάστημα και έτσι να αλλάζει συνεχώς γείτονες. Αυτή η κατάσταση μας δίνει ένα δυναμικό δίκτυο όπως και αυτό που μελετάται. Κάθε όχημα μπορεί να κατέχει και να παρέχει στους γύρω του δυναμικά μεταβαλλόμενες πληροφορίες οι οποίες αναφέρουν για παράδειγμα την τοποθεσία του οχήματος, τη θερμοκρασία ή τις καιρικές συνθήκες μιας ορισμένης περιοχής και το τι μπορεί να υπάρχει στο δρόμο σε μικρή απόσταση από αυτό όπως βενζινάδικα, εστιατόρια, εμπορικά καταστήματα, κέντρα πρώτων βοηθειών κ.α. Για καθένα από αυτά μπορεί να παρέχει πληροφορίες οι οποίες ποικίλουν από πολύ απλές, όπως αναφορά ύπαρξης κάποιων υπηρεσιών, έως και περισσότερο πολύπλοκες όπως πληροφορίες σχετικά με την τιμή ή τη διαθεσιμότητα κάποιων αγαθών. Αυτές οι πληροφορίες αποτελούν και τη βάση δεδομένων του κάθε κόμβου-οχήματος αποτελούμενη από σχέσεις πχ καταστήματα και πλειάδες με διάφορα γνωρίσματα όπως προϊόντα, τιμές κ.α. Οι οδηγοί των υπόλοιπων οχημάτων έχουν την δυνατότητα να θέτουν ερωτήσεις στους παρακείμενους παρόχους πληροφοριών όπου τους αποτελούν τα οχήματα σε κοντινή αυτών απόσταση, με σκοπό ανακτήσουν, να συνδυάσουν και να χρησιμοποιήσουν τις

πληροφορίες που αυτοί κατέχουν. Έτσι όταν κάποιος οδηγός χρειάζεται μια πληροφορία σχετικά με καύσιμα και βενζινάδικα, θέτει την κατάλληλη ερώτηση στους γύρω του (πχ με τη μέθοδο της πλημμύρας), και αυτοί του απαντάνε με τις τοπικά αποθηκευμένες πληροφορίες τους.

Όμως σημαντικό είναι να ανακτηθούν μόνο οι χρήσιμες πληροφορίες, για παράδειγμα ο εν λόγω οδηγός χρειάζεται βενζινάδικα που να είναι σχετικά κοντά με την τοποθεσία του οχήματός του. Έτσι μαζί με τη πληροφορία που διαθέτει κάθε αυτοκίνητο για τα παρακείμενα βενζινάδικα, χρειάζεται και μια ιδιότητα που να απορρίπτει κάποιο βενζινάδικο για κάποιο λόγο, όπως ότι βρίσκεται σε μακρινή απόσταση, ή ότι πλέον η παροχή υπηρεσιών δεν είναι πλέον διαθέσιμη (έγινε έλεγχος πριν από αρκετό χρονικό διάστημα). Η ιδιότητα αυτή μεταφράζεται σαν την ημερομηνία λήξης της πλειάδας. Οποιαδήποτε πληροφορία μετά από ένα χρονικό διάστημα παύει να ισχύει όπως η τιμή της βενζίνης σε ένα γνωστό βενζινάδικο, που μετά από μια ώρα πορείας, πλέον βρίσκεται μακριά, μπορεί να έχει αλλάξει και αποτελεί άχρηστη πληροφορία για τα παρακείμενα οχήματα. Για το σκοπό αυτό πρέπει όχι μόνο να παρέχεται η κάθε πλειάδα με την ημερομηνία λήξης, αλλά είναι πάρα πολύ χρήσιμο να υπάρχουν τρόποι ώστε οι “άχρηστες” πληροφορίες να απορρίπτονται από την τοπική βάση δεδομένων, να ανανεώνονται με καινούριες και να διαδίδονται οι ενημερώσεις των βάσεων όσο γίνεται πιο γρήγορα. Με άλλα λόγια πρέπει σε ένα τέτοιο δίκτυο ομότιμων κόμβων, να υπάρχει τρόπος, με βάση διάφορα πρωτόκολλα που θα αναλυθούν, οι τοπικές βάσεις δεδομένων να παραμένουν ενημερωμένες, οι ενημερώσεις να διαδίδονται ταχύτατα και να διαγράφονται έγκαιρα οι “απαρχαιωμένες” πληροφορίες. Αυτός είναι εν γένει και ο σκοπός της παρούσας εργασίας.

ΚΕΦΑΛΑΙΟ 3. ΥΠΟΣΤΗΡΙΞΗ ΕΡΩΤΗΣΕΩΝ

Στο κεφάλαιο αυτό, παρουσιάζουμε τις ερωτήσεις που μπορούν να γίνουν στο σύστημα, δηλαδή τι και πως μπορεί να ρωτήσει κάποιος ομότιμος κόμβος τους υπόλοιπους ομότιμους στο δίκτυο. Αρχικά, στην ενότητα 3.1 γίνεται η περιγραφή ερωτήσεων με την υποενότητα 3.1.1 να παρουσιάζει την απλή ερώτηση τύπου select και την 3.1.3 τα διάφορα είδη των ερωτήσεων όπως είναι το aggregation και η ένωση (join). Στην υποενότητα 3.1.2 παρουσιάζεται η σημασιολογία της ερώτησης επιλογής καθώς και ένα παράδειγμα. Τέλος στην ενότητα 3.2 δίνονται οι αλγόριθμοι για το πώς μπορεί να γίνουν διάφορες λειτουργίες στο σύστημα όπως είναι το sum,min,max,join καθώς και παραδείγματα αυτών.

3.1. Περιγραφή Ερωτήσεων

Σε αυτή την ενότητα θα γίνει περιγραφή των ερωτήσεων που μπορεί να κάνει κάθε ομότιμος κόμβος στο σύστημα. Όπως αναφέρθηκε και προηγουμένως οι ερωτήσεις είναι τύπου SQL, αλλά με ορισμένες μετατροπές. Έτσι εισήχθησαν κάποιοι τελεστές για να υποστηριχθούν οι ιδιαιτερότητες του συστήματος, όπως ότι πλέον έχουμε ένα δίκτυο ομότιμων κόμβων και χρειάζεται μαζί με την ερώτηση να περιλαμβάνεται και το βάθος της ερώτησης (TTL) ή η συχνότητα που θα ανανεώνεται η ερώτηση κ.α. Επιπλέον, παρουσιάζεται αναλυτικά και ο υπολογισμός της ημερομηνίας λήξης του επιστρεφόμενου αποτελέσματος.

3.1.1. Select Query

Στην παρούσα υπό-ενότητα μελετάμε τη γενική μορφή που θα έχει ένα select query στο σύστημά μας.

Μια ερώτηση τύπου select μπορεί να γίνει ως εξής:

```
SELECT column
FROM table
WHERE statement
WITH
        [TTL t]
        [PERIOD p]
        [THRES.TIME thr]
```

Το παραπάνω μοντέλο με την απαιτούμενη επέκταση της SQL, δίνει τη δυνατότητα σε κάποιο peer να θέτει ερωτήσεις στο σύστημα που μελετάμε. Σημειώνουμε ότι μέσα σε αγκύλες ([…]), υπάρχουν προαιρετικά τμήματα της ερώτησης που μπορεί να παραληφθούν ή να χρησιμοποιηθούν ανάλογα με τις ανάγκες του peer.

Από ότι παρατηρούμε οι τρεις πρώτοι τελεστές (clauses) της ερώτησης (SELECT, FROM, WHERE) είναι ακριβώς οι ίδιοι, με την ίδια σημασία όπως στην κλασική SQL. Με την έκφραση “SELECT column” ορίζουμε τις στήλες προς προβολή στην απάντηση της ερώτησης, με την έκφραση “FROM table” ορίζουμε τον πίνακα ή τους πίνακες που συμμετέχουν στην ερώτηση και με την “WHERE statement” εκφράζουμε την συνθήκη που πρέπει να ικανοποιείται για την απάντηση της ερώτησης.

Η επέκταση έγινε στο τελεστή WITH όπου πλέον εισάγονται τρεις νέοι τελεστές: TTL, PERIOD, THRES.TIME:

TTL: Το TTL είναι μια παράμετρος στο σύστημά μας και έχει την έννοια “Time To Live” δηλαδή του βάθους της ερώτησης. Τον τελεστή TTL τον θέτει ο κόμβος που δημιουργεί την ερώτηση και σημαίνει το βάθος ή την ακτίνα της ερώτησης, δηλαδή

το πλήθος των κόμβων στους οποίους προωθείται η ερώτηση κατά μήκος του ίδιου μονοπατιού μέχρι να ανακτηθούν τα κατάλληλα αποτελέσματα. Σημαντικό είναι να αναφέρουμε ότι όταν το βάθος της ερώτησης είναι μικρό, τότε έχουμε λιγότερα μηνύματα στο δίκτυο, αλλά και περιορισμένα αποτελέσματα, ενώ σε αντίθετη περίπτωση έχουμε περισσότερα μηνύματα, αλλά με σημαντικά καλύτερα αποτελέσματα. Σκοπός είναι να εντοπιστεί η βέλτιστη παράμετρος TTL από τον αρχικό ομότιμο (peer), όσο το δυνατόν μικρότερη, με τα επιθυμητά αποτελέσματα.

Εδώ πρέπει να αναφερθεί ότι, καθώς το TTL δεν είναι αναγκαία παράμετρος στην ερώτηση, ο αρχικός κόμβος μπορεί να το παραλείψει. Γι' αυτό υπάρχει μια καθορισμένη τιμή *t* από το σύστημα έτσι ώστε όταν μια ερώτηση δεν έχει ορισμένο το TTL, η παράμετρος αυτή παίρνει την τιμή TTL= t. Το *t* το ορίζει το σύστημα εκ κατασκευής του.

PERIOD: Με τον τελεστή αυτό εισάγουμε την έννοια του χρονισμού της ερώτησης. Μια ερώτηση στο σύστημα μπορεί να είναι είτε ad-hoc (τίθεται μόνο μια φορά) είτε continuous (συνεχής ερώτηση). Στην πρώτη περίπτωση που χρειάζεται η ερώτηση να γίνει μια φορά, ο τελεστής PERIOD παραλείπεται. Σε ένα τέτοιο ενδεχόμενο η ερώτηση καλείται ad-hoc. Στην περίπτωση που μια ερώτηση που θέτει ο κόμβος είναι συνεχής (continuous), χρειάζεται να εισαχθεί μια παράμετρος που έχει σχέση με τη περίοδο της ερώτησης. Δηλαδή, το χρονικό διάστημα κατά το οποίο η ερώτηση αυτή θα επαναλαμβάνεται. Αυτή η παράμετρος είναι ο τελεστής PERIOD και στη περίπτωση που το PERIOD = *p*, η ερώτηση γίνεται συνεχώς και με τις ίδιες παραμέτρους κάθε *p* χρονικές στιγμές. Η παράμετρος ορισμένες φορές είναι αρκετά σημαντική γιατί κάθε πλειάδα στη τοπική βάση δεδομένων του κόμβου έχει ημερομηνία λήξης που δείχνει μέχρι πότε θα συνεχίσει να ισχύει.

THRES.TIME: Ο τελεστής αυτός δηλώνει το κατώφλι της ημερομηνίας λήξης του αποτελέσματος, δηλαδή μετά από τουλάχιστον πόσες χρονικές στιγμές θέλουμε το αποτέλεσμά μας να λήξει. Αν η παράμετρος THRES.TIME = *thr*, θέλουμε τα αποτελέσματα που θα πάρουμε ως απάντηση να λήγουν σε *thr* χρονικές στιγμές από τη στιγμή της ερώτησης ή αργότερα. Επειδή τα αποτελέσματα που θα λάβει ο κόμβος μπορεί με διάφορους τρόπους, χρησιμοποιώντας πρωτόκολλα που θα αναλυθούν

μετέπειτα, να είναι εγγυημένο ότι δεν θα έχουν λήξει, όμως δεν υπάρχει εγγύηση για το πόσο χρόνο θα ισχύουν ακόμα. Υπάρχει κίνδυνος να λήξει η απάντηση σε μια χρονική στιγμή οπότε το αποτέλεσμα θα είναι ουσιαστικά άχρηστο. Για το λόγο αυτό η παράμετρος ορισμένες φορές, ανάλογα με τις ανάγκες του κάθε κόμβου, μπορεί να είναι αρκετά χρήσιμη.

Σημαντικό είναι να αναφερθεί ότι στο σύστημά μας υπάρχει ένα καθολικό ρολόι. Για να μπορεί να προσομοιωθεί η έννοια της ημερομηνίας λήξης της πλειάδας, θα πρέπει όλοι οι κόμβοι να είναι συγχρονισμένοι σε αυτό. Αν και η έννοια του συγχρονισμού σε ένα σύστημα ομότιμων κόμβων, πάνω στο διαδίκτυο δεν υπάρχει, όμως πρέπει να υπάρχει κάποια έννοια του συγχρονισμού όσο αφορά την ημερομηνία λήξης των πλειάδων. Κάτι τέτοιο δεν είναι αδύνατο: μπορεί να είναι το ρολόι του πραγματικού κόσμου με ημερομηνία και ώρα, όπου όλοι οι κόμβοι να μπορούν εύκολα να συγχρονιστούν ανεξάρτητα από το πότε εισήλθαν στο σύστημα. Έτσι, όταν κάποια πλειάδα έχει ημερομηνία λήξης x σημαίνει ότι θα λήξει στην x χρονική στιγμή (πχ ώρα, λεπτά, δευτερόλεπτα). Αν το ρολόι δείχνει τ (τωρινός χρόνος), μια πλειάδα ισχύει αν x-τ>0. Στην περίπτωση που υπάρχει ο τελεστής THRES.TIME = *thr*, μια πλειάδα μπορεί να ληφθεί υπόψη στο αποτέλεσμα αν x- (τ + thr) > 0, δηλαδή θέτουμε όπου τ = τ + thr.

3.1.2. Select Query, Σημασιολογία

Στο σημείο αυτό, παρουσιάζεται η σημασιολογία της ερώτησης που αναλύθηκε στην ενότητα 3.1.1.

Έστω ότι ο κόμβος A θέτει ένα select query. Οι κόμβοι στους οποίους απευθύνεται είναι ένα υποσύνολο των κόμβων του δικτύου που βρίσκεται μέσα στο βάθος της ερώτησης (TTL= *t*) που θέτει ο κόμβος, όπως αυτό είχε αναλυθεί με τις δύο μεθόδους αναζήτησης στην ενότητα 2.1.3. Η ερώτηση ξεκινάει από τον κόμβο A και προωθείται με μήνυμα σε όλους τους κόμβους που βρίσκονται σε βάθος *t* από τον κόμβο. Το αποτέλεσμα που επιστρέφεται στον κόμβο A είναι οι πλειάδες που:

1. παιδία τους ορίζονται από τον τελεστή “SELECT column”

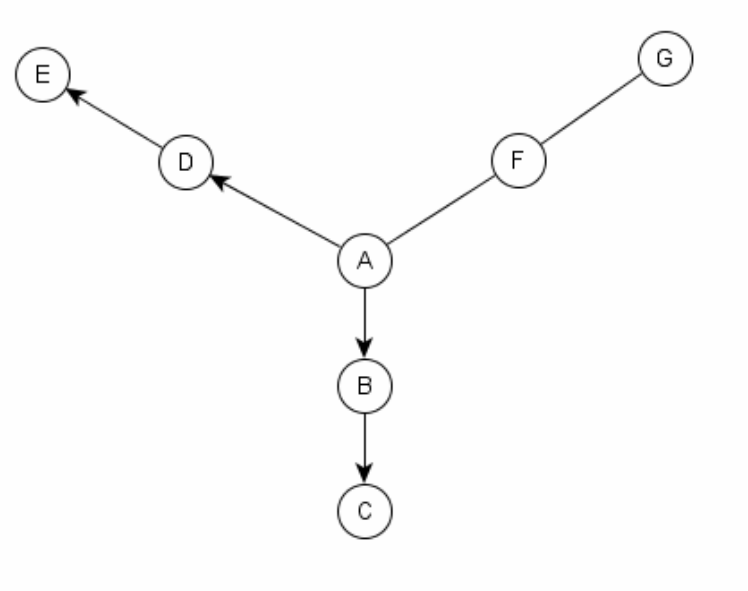
2. βρίσκονται σε κόμβο έως και βάθος t όπως αυτό ορίζεται από τη μέθοδο αναζήτησης. Δηλαδή η ερώτηση απευθύνεται σε ένα υποσύνολο των κόμβων του συστήματος
3. ανήκουν στον πίνακα “table”
4. δεν έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος). Αν υπάρχει ο τελεστής THRES.TIME = *thr* οι πλειάδες δεν πρέπει να έχουν λήξει τη χρονική στιγμή τ = τ + thr
5. ικανοποιούν τη συνθήκη που ορίζει ο τελεστής WHERE όπως γίνεται και στην παραδοσιακή SQL

Σημειώνουμε ότι αν στο επιστρεφόμενο αποτέλεσμα υπάρχουν πλειάδες με ίδιο κλειδί, τότε οι πλειάδες αυτές αποτελούν αντίγραφα της ίδιας πλειάδας (ενότητα 2.2.1). Τότε ο αρχικός κόμβος Α κρατάει σαν αποτέλεσμα το πιο ενημερωμένο αντίγραφο, δηλαδή την πλειάδα με τη μεταγενέστερη ημερομηνία λήξης.

Παράδειγμα ερώτησης: έστω ότι ο κόμβος Α θέτει την ερώτηση:

```
SELECT *
FROM R
WHERE attr>20
WITH
        TTL 2
```

Το μήνυμα της ερώτησης προωθείται με τη μέθοδο των τυχαίων διαδρομών με 2 τυχαίους περιπατητές. Στο σχήμα 3.1 απεικονίζονται 7 κόμβοι, υποσύνολο των κόμβων όλου του δικτύου. Με τη μέθοδο των τυχαίων διαδρομών το μήνυμα της ερώτησης συναντάει τους κόμβους A, B, C, D, E όπως φαίνεται στο σχήμα 3.1.



Σχήμα 3.1 Υποσύνολο των Κόμβων του Συστήματος που Σχετίζονται με την Ερώτηση

Έστω ότι κάθε κόμβος έχει μια βάση δεδομένων με πλειάδες που ανήκουν στον πίνακα R όπως δείχνει ο πίνακας 3.1.

Πίνακας 3.1 Βάση Δεδομένων των Κόμβων A, B, C, D, E

Node: A R		
id	attr	exp.time
AA	12	13
AB	22	16

Node: B R		
id	attr	exp.time
BA	53	18
BC	30	22

Node: C R		
id	attr	exp.time
CF	9	14
CG	21	29

Node: D R		
id	attr	exp.time
DE	23	12
DF	15	25

Node: E R		
id	attr	exp.time
EA	28	32
EG	22	9

Έστω τώρα, ότι το ρολόι δείχνει τ=15. Όπως αναφέρθηκε παραπάνω οι πλειάδες που επιστρέφονται σαν αποτέλεσμα πρέπει:

1. να ανήκουν στους κόμβους A, B, C, D, E
2. να ανήκουν στον πίνακα R
3. να έχουν exp.time > 15
4. να ικανοποιούν τη συνθήκη που ορίζει ο τελεστής WHERE, δηλαδή να έχουν attr > 20

Τελικά, οι πλειάδες που επιστρέφονται στο κόμβο A παρουσιάζονται στο πίνακα 3.2 και επιστρέφονται όλες οι στήλες της σχέσης R λόγω του τελεστή SELECT *

Πίνακας 3.2 Αποτέλεσμα της Ερώτησης

Result		
id	attr	exp.time
AB	22	16
BA	53	18
BC	30	22
CG	21	29
EA	28	32

3.1.3. Διάφοροι Τύποι Ερωτήσεων και Ημερομηνία Λήξης Αποτελέσματος

Σε αυτή τη φάση της εργασίας σκοπός μας είναι να εμπλουτίσουμε την ερώτηση (query) έτσι ώστε να υποστηρίξουμε διάφορους τύπους ερωτήσεων όπως είναι οι συνάθροιση (aggregation) και συνένωση (join), ως προς τον υπολογισμό της ημερομηνίας λήξης του αποτελέσματος.

Υπάρχουν 5 κυριότερες λειτουργίες συνάθροισης που συγκαταλέγονται στην παραδοσιακή SQL και είναι οι εξής: average, maximum, minimum, sum και count. Αυτές πρέπει να τις προσαρμόσουμε στο σύστημά μας, έτσι ώστε να έχουν ως είσοδο

και την ημερομηνία λήξης (expiration time) της κάθε πλειάδας αλλά και να υπολογίζουν σωστά την ημερομηνία λήξης του αποτελέσματος.

Ξεκινώντας, ο πίνακας 3.3 παρουσιάζει τις συντομεύσεις που θα χρησιμοποιήσουμε στο εξής καθώς και τη σημασία τους. Οι συντομεύσεις αυτές, σχετίζονται με την ημερομηνία λήξης των πλειάδων και του αποτελέσματος των λειτουργιών [1] που θα αναλυθούν παρακάτω.

Πίνακας 3.3 Επεξήγηση Συντομεύσεων

R	σχέση (relation)
r	πλειάδα (tuple) της σχέσης R
r(i)	το $i^{οστό}$ γνώρισμα της σχέσης R
$t_R^{\exp}(r)$	επιστρέφει την ημερομηνία λήξης της πλειάδας r της σχέσης R
$t^{\exp}(.)$	παίρνει μια έκφραση και επιστρέφει την ημερομηνία λήξης της
$\exp_\tau(R)$	επιστρέφει τις πλειάδες της R που δεν έχουν λήξει τη χρονική στιγμή τ (όπου τ είναι ο τωρινός χρόνος) δηλαδή: $\exp_\tau(R) = \{r \mid r \in \wedge \,\&\, t_R^{\exp}(r) > \tau\}$
φ	η λειτουργία που θέλουμε κάθε φορά να χρησιμοποιήσουμε πχ φ={ average, maximum, minimum, sum, count}
τ	ο τωρινός χρόνος (χρόνος του ρολογιού)

Έτσι ορίζουμε το query να έχει ως εξής:

```
SELECT φ(r(i))
FROM table
[WHERE statement]
WITH
        [TTL t]
        [PERIOD p]
        [THRES.TIME thr]
```

Το παραπάνω ερώτημα σημαίνει ότι θέλουμε να εφαρμόσουμε τη λειτουργία συνάθροισης φ στο $i^{οστό}$ γνώρισμα (r(i)) της σχέσης R, με TTL = *t* και PERIOD = *p* THRES.TIME = *thr* όπως έχουμε εξηγήσει παραπάνω.

Στο σημείο αυτό πρέπει να εξετάσουμε τι πρέπει να επιστραφεί σαν αποτέλεσμα της ερώτησης στην λειτουργία συνάθροισης (aggregation), και τι ημερομηνία λήξης θα έχει το επιστρεφόμενο αποτέλεσμα.

Έτσι, γενικά, το παραπάνω ερώτημα θα πρέπει να μας επιστρέφει τις πλειάδες (r) όπου:

1. δεν έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος)
2. έχει εφαρμοστεί η συνάρτηση συνάθροισης φ, στο γνώρισμα i της σχέσης “table” όπως γίνεται και στην παραδοσιακή SQL

Πρέπει όμως να υπολογιστεί και η ημερομηνία λήξης που θα έχει το αποτέλεσμα. Αρχικά, η ημερομηνία αυτή (t^{exp}) μπορούμε να πούμε ότι θα ήταν η ελάχιστη ημερομηνία λήξης των πλειάδων που παίρνουν μέρος στον υπολογισμό της λειτουργίας συνάθροισης φ.

Κάτι τέτοιο βέβαια δεν είναι απολύτως δίκαιο σε όλες τις λειτουργίες συνάθροισης. Για παράδειγμα για το άθροισμα (sum), μια πλειάδα που έχει r(i)=0, και ελάχιστη ημερομηνία λήξης, θα δώσει τη δική της ημερομηνία λήξης στο αποτέλεσμα πράγμα που δεν είναι ορθό, γιατί το γνώρισμά της δεν παίζει ρόλο στο τελικό αποτέλεσμα του αθροίσματος. Για το λόγο αυτό επαναπροσδιορίζουμε την ημερομηνία λήξης του αποτελέσματος.

Ο τρόπος υπολογισμού της ημερομηνίας λήξης του αποτελέσματος στις λειτουργίες συνάθροισης, διαφέρει στις περιπτώσεις που υπάρχουν ή όχι ενημερώσεις στο σύστημα. Ενημερώσεις, αναφέρονται στο γεγονός ότι οι κόμβοι μπορούν να ενημερώσουν τις πλειάδες τους όσο αφορά ένα ή περισσότερα γνωρίσματά τους ή την ημερομηνία λήξης τους ή να δημιουργήσουν νέες πλειάδες στην τοπική τους βάση

δεδομένων. Γενικά, έχουμε υποθέσει ότι στο σύστημα υπάρχουν ενημερώσεις, αλλά κάτι τέτοιο περιπλέκει, όπως θα διαπιστώσουμε, τον υπολογισμό της ημερομηνίας λήξης του τελικού αποτελέσματος.

Για το λόγο αυτό επανεξετάζουμε τις λειτουργίες με τους δύο τρόπους:

- Για το ελάχιστο (minimum):

Για το ελάχιστο οι πλειάδες που επιστρέφονται σαν αποτέλεσμα πρέπει:

1. να μην έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος)
2. να εφαρμοστεί ο τελεστής MIN στο $i^{οστό}$ γνώρισμα της σχέσης “table” στις πλειάδες που δεν έχουν λήξει όπως ειπώθηκε παραπάνω.

Σημειώνουμε ότι για κάθε μονοπάτι της ερώτησης στο δίκτυο, επιστρέφεται μια πλειάδα r με το ελάχιστο γνώρισμα r(i) και μεταγενέστερη ημερομηνία λήξης όπως θα αναλυθεί παρακάτω.

Για τη λειτουργία του ελαχίστου, αν δεν υπήρχαν ενημερώσεις στο σύστημα, η ημερομηνία λήξης του αποτελέσματος θα είναι η ημερομηνία λήξης που συνοδεύει την πλειάδα με το ελάχιστο γνώρισμα i. Οι υπόλοιπες πλειάδες αν δεν ενημερώνονται, τότε και να λήξουν δεν παίζουν ρόλο στο επιστρεφόμενο αποτέλεσμα. Αν περισσότερες από μια πλειάδες έχουν το ελάχιστο γνώρισμα το αποτέλεσμα θα είναι η πλειάδα με τη μεγαλύτερη ημερομηνία λήξης. Αν δεν υπάρχουν ενημερώσεις σημαίνει ότι το περιεχόμενο της βάσης δεδομένων όλων των ομότιμων είναι στατικό και δεν μεταβάλλεται, άρα η ημερομηνία λήξης ορίζεται σαν την ημερομηνία λήξης του αποτελέσματος. Έτσι, αν η ημερομηνία λήξης του αποτελέσματος είναι x, σημαίνει ότι το αποτέλεσμα που επιστρέφεται θα συνεχίσει να εκφράζει την πλειάδα με το ελάχιστο γνώρισμα i, μέχρι και την χρονική στιγμή x.

Διαφορετικά, αν στο σύστημα έχουμε ενημερώσεις ή δημιουργία νέων πλειάδων η προηγούμενη τεχνική δεν είναι αποδοτική: ανά πάσα στιγμή μπορεί να δημιουργηθεί μια πλειάδα που θα έχει ελάχιστο γνώρισμα i και θα έπρεπε να ήταν αυτή σαν αποτέλεσμα. Οπότε δεν υπάρχει πλέον η έννοια της ημερομηνίας λήξης του αποτελέσματος γιατί δεν υπάρχει εγγύηση για το χρονικό διάστημα που ισχύει το

αποτέλεσμα σαν ελάχιστο. Σαν ημερομηνία λήξης πάλι ενσωματώνεται η ημερομηνία λήξης που συνοδεύει την πλειάδα, αλλά εννοείται ότι η ημερομηνία λήξης εκφράζει το πότε θα λήξει η πλειάδα και όχι μέχρι πότε θα ισχύει το αποτέλεσμα σαν ελάχιστο.

- Για το μέγιστο (maximum):

Για το μέγιστο οι πλειάδες που επιστρέφονται σαν αποτέλεσμα πρέπει:

1. να μην έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος)
2. να εφαρμοστεί ο τελεστής MAX στο $i^{οστό}$ γνώρισμα της σχέσης “table”, στις πλειάδες που δεν έχουν λήξει όπως αναφέρθηκε παραπάνω

Σημειώνουμε ότι για κάθε μονοπάτι της ερώτησης στο δίκτυο, επιστρέφεται μια πλειάδα r με το μέγιστο γνώρισμα r(i) και μεταγενέστερη ημερομηνία λήξης όπως θα αναλυθεί παρακάτω.

Για την ημερομηνία λήξης ισχύει το αντίστοιχο που ισχύει και για το ελάχιστο. Αν δεν υπάρχουν ενημερώσεις στο σύστημα τότε η ημερομηνία λήξης του αποτελέσματος θα είναι η ημερομηνία λήξης της πλειάδας με το μέγιστο γνώρισμα i με την έννοια ότι η συγκεκριμένη πλειάδα θα εκφράζει το μέγιστο μέχρι την ημερομηνία λήξης της. Αν περισσότερες από μια πλειάδες έχουν το μέγιστο γνώρισμα το αποτέλεσμα θα είναι η πλειάδα με τη μεγαλύτερη ημερομηνία λήξης. Διαφορετικά, αν υπάρχουν ενημερώσεις η ημερομηνία λήξης θα σημαίνει ότι η πλειάδα λήγει κάποια δεδομένη χρονική στιγμή χωρίς να εκφράζει το μέγιστο γνώρισμα i.

- Για το μέσο όρο (average):

Για το μέσο όρο, οι πλειάδες που επιστρέφονται σαν αποτέλεσμα πρέπει:

1. να μην έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος)
2. να εφαρμοστεί ο τελεστής AVG στο $i^{οστό}$ γνώρισμα της σχέσης “table”, στις πλειάδες που δεν έχουν λήξει

Για την λειτουργία του μέσου όρου, αν δεν υπάρχουν ενημερώσεις η ημερομηνία λήξης του αποτελέσματος θα είναι η ελάχιστη ημερομηνία λήξης όλων των πλειάδων

που έχουν λάβει μέρος στον υπολογισμό του αποτελέσματος. Αν λοιπόν η ημερομηνία λήξης του αποτελέσματος είναι x, τότε το αποτέλεσμα θα δείχνει το μέσο όρο του i^{οστού} γνωρίσματος της σχέσης “table” μέχρι τη χρονική στιγμή x.

Διαφορετικά αν υπάρχουν ενημερώσεις στο σύστημα το αποτέλεσμα δεν μπορεί να έχει ημερομηνία λήξης. Δεν μπορεί να υπάρχει καμία εγγύηση για πόσο χρονικό διάστημα θα ισχύει σαν μέσος όρος. Σε αυτή την περίπτωση και επειδή δεν επιστρέφεται το γνώρισμα μιας πλειάδας (όπως στο MIN, MAX), αλλά ο μέσος όρος των γνωρισμάτων, δεν υπάρχει η έννοια της ημερομηνίας λήξης ούτε του αποτελέσματος ούτε της πλειάδας από όπου προήλθε.

- Για το άθροισμα (SUM):

Για το άθροισμα, οι πλειάδες που επιστρέφονται σαν αποτέλεσμα πρέπει:

1. να μην έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος)
2. να εφαρμοστεί ο τελεστής SUM στο i^{οστό} γνώρισμα της σχέσης “table”

Στην περίπτωση που δεν υπάρχουν ενημερώσεις στο σύστημά μας, το αποτέλεσμα θα έχει ημερομηνία λήξης την ελάχιστη ημερομηνία λήξης των πλειάδων που δεν έχουν λήξει τη χρονική στιγμή τ και που το $r(i) \neq 0$. Αυτό γίνεται γιατί αν σε κάποια πλειάδα το i^{οστό} γνώρισμά της έχει τιμή ίση με 0 και να λήξει αυτή πάλι το ίδιο άθροισμα τιμών θα έχει το αποτέλεσμα (δηλαδή δεν παίζει ρόλο στην τελική τιμή του αποτελέσματος).

Και στην περίπτωση του AVG, αν υπάρχουν ενημερώσεις στο σύστημα, η ημερομηνία λήξης του αποτελέσματος δεν μπορεί να υπάρχει, όπως και στην προηγούμενη περίπτωση του μέσου όρου, ούτε σαν ημερομηνία λήξης του αποτελέσματος, ούτε σαν ημερομηνία λήξης της πλειάδας από όπου προήλθε.

- Για την αρίθμηση (COUNT):

Για την αρίθμηση, όπως και στις προηγούμενες περιπτώσεις, οι πλειάδες που επιστρέφονται σαν αποτέλεσμα πρέπει:

1. να μην έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος)
2. να εφαρμοστεί ο τελεστής COUNT στο $i^{οστό}$ γνώρισμα της σχέσης

Στην περίπτωση της αρίθμησης η ημερομηνία λήξης του αποτελέσματος είναι όμοια με την περίπτωση του μέσου όρου. Αν δεν υπάρχουν ενημερώσεις, τότε η ημερομηνία λήξης του αποτελέσματος είναι η ελάχιστη ημερομηνία λήξης των πλειάδων που έλαβαν μέρος στον υπολογισμό..

Διαφορετικά αν υπάρχουν ενημερώσεις στο σύστημα το αποτέλεσμα δεν μπορεί και πάλι να έχει ημερομηνία λήξης.

- Λειτουργία Συνένωσης (Join operation)

Συνεχίζουμε τον εμπλουτισμό των ερωτήσεων με την υποστήριξη και της λειτουργίας της συνένωσης (join).
Έστω r(i) το $i^{οστό}$ γνώρισμα της σχέσης R και s(i) το $i^{οστό}$ γνώρισμα της σχέσης S. Ορίζουμε σαν συνένωση (join):

$$R \bowtie_{r(i)\theta s(i)} S = \{r \vee s : r \in \exp_{\tau}(R), s \in \exp_{\tau}(S), r(i)\theta s(i)\}$$

Όπου R,S οι δύο σχέσεις, r,s πλειάδες των R,S αντίστοιχα, θ={=,>.<,...} και τ είναι ο τωρινός χρόνος.
Το ερώτημα μπορεί να γίνει ως:

```
SELECT column
FROM R,S
WHERE R(i)θS(i)
WITH
        [TTL t]
        [PERIOD p]
        [THRES.TIME thr]
```

Στην περίπτωση της συνένωσης, η ημερομηνία λήξης του αποτελέσματος είναι η ίδια ανεξάρτητα αν στο σύστημα υπάρχουν ενημερώσεις ή όχι. Επειδή πρέπει να υπάρχει δικαιοσύνη η κάθε πλειάδα του αποτελέσματος έχει προέλθει από 2 πλειάδες των σχέσεων R,S. Άρα πρέπει να έχει την ημερομηνία λήξης της πλειάδας με τη μικρότερη ημερομηνία λήξης μεταξύ των δύο που συμμετέχουν στον υπολογισμό της κάθε πλειάδας του αποτελέσματος.

3.1.4. Εισαγωγή Κατωφλίου στην Ερώτηση

Μια τελευταία παρατήρηση που πρέπει να γίνει είναι το πώς θα αντιμετωπιστεί η περίπτωση που η ημερομηνία λήξης του αποτελέσματος είναι πολύ σύντομη. Για παράδειγμα όταν το τ=10 (τωρινός χρόνος) και το αποτέλεσμα έχει exp.time=11, σημαίνει ότι το αποτέλεσμα θα λήξει σε 1 χρονική μονάδα οπότε πρέπει να ξανά-υπολογιστεί σε 1 χρονική μονάδα αν χρειαζόμαστε πιο φρέσκα δεδομένα. Για το λόγο αυτό βάζουμε ένα κατώφλι (threshold) στην ερώτηση όπως παρουσιάσαμε προηγουμένως έτσι ώστε να είμαστε σίγουροι ότι το αποτέλεσμα θα λήξει σε τουλάχιστον όσες χρονικές μονάδες εμείς επιθυμούμε. Για παράδειγμα εισάγουμε ένα threshold=10 και εννοούμε ότι το αποτέλεσμα θα έχει ισχύει για τις επόμενες 10 χρονικές μονάδες.

Για να εισάγουμε το κατώφλι στην ερώτησή μας, προσθέτουμε τον τελεστή THRES.TIME στο WITH με το επιθυμητό κατώφλι. Η ερώτηση μένει ως έχει κατά τα άλλα, με τη διαφορά ότι για να μπορούμε να υποστηρίξουμε και αυτό το τελεστή όπου έχουμε υποθέσει ότι τ = current time (τωρινός χρόνος) ορίζουμε σαν:

$$\tau = \text{current time} + \text{thr}$$

Έτσι μια πλειάδα για να ληφθεί υπόψη στην συνένωση πρέπει να έχει ημερομηνία λήξης *x* όπου x – (current time + thr) > 0.

Τελικά, θα έχουμε το επιθυμητό αποτέλεσμα με εγγυημένα φρέσκα αποτελέσματα για τις επόμενες *thr* χρονικές μονάδες.

3.2. Αλγόριθμοι για τις Λειτουργίες sum, min, max, join

Στο σημείο αυτό, σημαντικό είναι να μελετηθούν αλγόριθμοι τις λειτουργίες aggregation όπως είναι το MIN, MAX, SUM, καθώς και join. Για τους αλγορίθμους που θα μελετήσουμε χρησιμοποιείται αναζήτηση με τη μέθοδο των τυχαίων διαδρομών (random walks).

3.2.1. Αλγόριθμος για την Εύρεση του Ελαχίστου (min)

Έστω ο κόμβος A (originator), ξεκινάει μια ερώτηση SQL για τον υπολογισμό του ελαχίστου (MIN), ως εξής:

```
SELECT MIN(r(i))
FROM R
[WHERE statement]
WITH
        [TTL t]
        [PERIOD p]
        [THRES.TIME thr]
```

Ο κόμβος A δημιουργεί ένα αντικείμενο MIN τύπου message όπου εκεί γράφεται το αποτέλεσμα και κάθε σημαντική πληροφορία για την εξέλιξη της ερώτησης. Ο αρχικός κόμβος βρίσκει την πλειάδα με την ελάχιστη τιμή στο συγκεκριμένο γνώρισμα από την τοπική βάση δεδομένων του και την ενσωματώνει στο μήνυμα μαζί με την ημερομηνία λήξης του αποτελέσματος. Στη συνέχεια η πορεία του μηνύματος ακολουθεί τη μέθοδο των τυχαίων διαδρομών. Αρχικά, το μήνυμα το στέλνει σε d γείτονές του (d τυχαίοι περιπατητές). Οι γείτονες υπολογίζουν το τοπικό τους αποτέλεσμα επεξεργάζοντας τις πλειάδες της τοπικής βάσης δεδομένων τους. Αν εντοπίσουν πλειάδα με γνώρισμα μικρότερο του αποτελέσματος του μηνύματος, τότε ενσωματώνουν το δικό τους αποτέλεσμα στο μήνυμα. Διαφορετικά δεν μεταβάλουν το περιεχόμενο του μηνύματος. Στη συνέχεια όσο το βάθος του μηνύματος είναι

μικρότερο από το TTL το στέλνουν σε ένα τυχαία επιλεγμένο γείτονά τους κάθε φορά. Όταν τα βάθος γίνει ίσο με TTL, το αποτέλεσμα επιστρέφεται από το ίδιο μονοπάτι πίσω στον κόμβο A.

Η ημερομηνία λήξης του αποτελέσματος όπως αναλύσαμε και σε προηγούμενο κεφάλαιο είναι η ημερομηνία λήξης που συνοδεύει την πλειάδα με το ελάχιστο γνώρισμα r(i). Αν υπάρχουν περισσότερες από μια τότε το αποτέλεσμα θα έχει ημερομηνία λήξης τη μέγιστη ημερομηνία λήξης των πλειάδων με το ελάχιστο γνώρισμα r(i).

Πιο συγκεκριμένα η πληροφορία που έχει το μήνυμα παρουσιάζεται στον πίνακα 3.4.

Πίνακας 3.4 Πληροφορία του Μηνύματος

Μεταβλητή	Πληροφορία
int exp	η ημερομηνία λήξης του αποτελέσματος (αρχικά είναι άπειρο)
int min	το αποτέλεσμα του ελαχίστου (αρχικά είναι άπειρο)
int attr	το γνώρισμα που θέλουμε να υπολογίσουμε το ελάχιστο
int hops	το βάθος που βρίσκεται το μήνυμα
int ttl	το τελικό βάθος της ερώτησης

Παρακάτω δίνεται ο αλγόριθμος για την εύρεση ελαχίστου (min):

Αλγόριθμος υπολογισμού ελαχίστου (MIN)

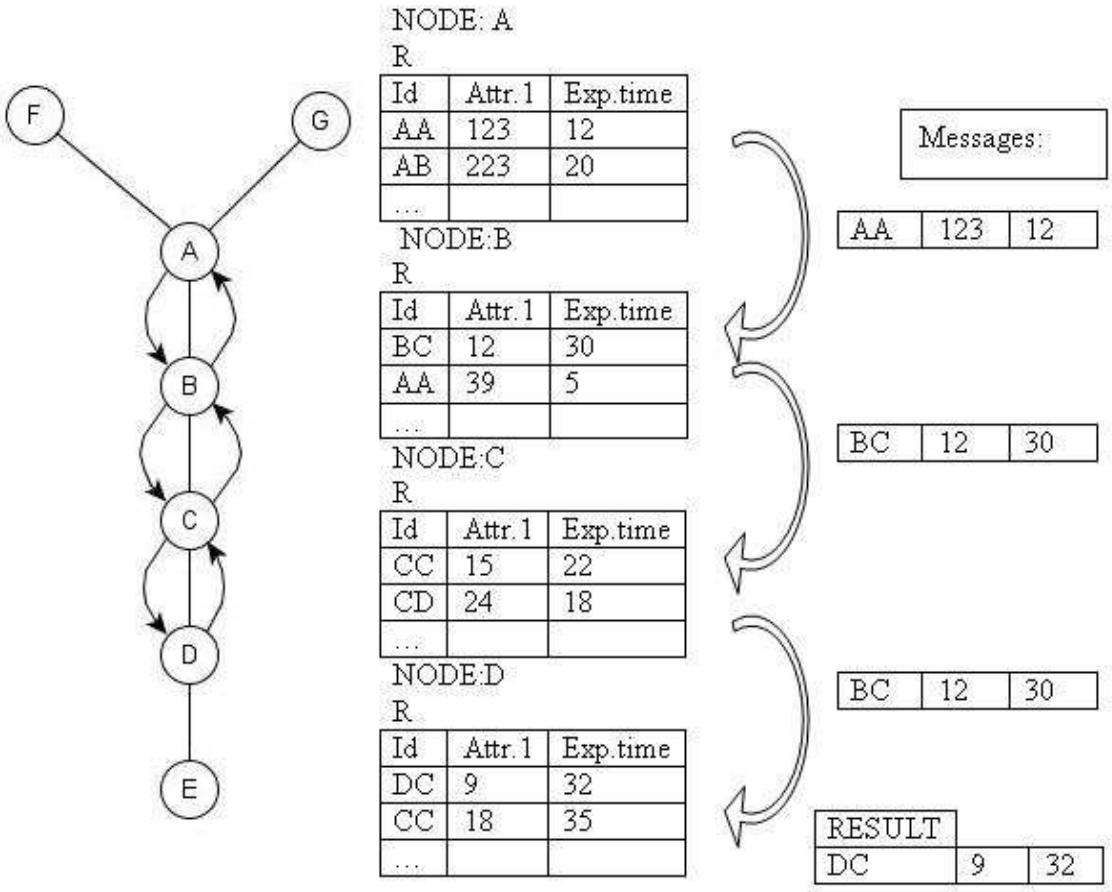
```
1.  if originator:
2.      create object min
3.      explore()
4.      send sum to d-neighbors

5.  upon receive min message:
6.      explore()
7.  if hops < TTL then
8.      send min message to a neighbor
9 .  else
10.     send message back to originator

11.  function explore()
12.   foreach tuple r in data
13.   if t_R^exp(r) > current time  then
14.       if min > r(attr) then
15.                 min = r(attr)
17.                 exp = t_R^exp(r)
18.       end if
19.       if min = = r(attr) AND exp < t_R^exp(r) then
20.                 exp = t_R^exp(r)
21.       endif
22.
23.    endif
24. end function
```

Στο σημείο αυτό πρέπει να σημειωθεί ότι ο αλγόριθμος για την εύρεση του μεγίστου (max) είναι παρόμοιος οπότε και παραλείπεται η ανάλυσή του.

Παράδειγμα εύρεσης ελαχίστου: Έστω κόμβος Α θέτει ένα ερώτημα εύρεσης ελαχίστου ως εξής:

```
SELECT MIN(Attr.1))
FROM R
WITH
        TTL 3
```

Στο σχήμα 3.2 παρουσιάζεται η πορεία του μηνύματος της εύρεσης ελαχίστου και το αποτέλεσμα που στο τέλος λαμβάνει ο κόμβος A.



Σχήμα 3.2 Παράδειγμα Εύρεσης Ελαχίστου

Στο σχήμα 3.2 φαίνεται η πορεία ενός τυχαίου περιπατητή που ξεκινάει από τον κόμβο A και επισκέπτεται τους κόμβους B, C και D. Κάνουμε την σύμβαση ότι καμιά πλειάδα από αυτές που έχουν οι κόμβοι δεν έχει λήξει. Έτσι από τον κόμβο A ξεκινάει το μήνυμα με την πλειάδα με id = AA και γνώρισμα 123 σαν ελάχιστο γνώρισμα της βάσης του. Έπειτα, ο κόμβος B αντικαθιστά το αποτέλεσμα με πλειάδα της τοπικής βάσης δεδομένων του με id = BC διότι έχει μικρότερο γνώρισμα attr.1. Η πορεία του μηνύματος συνεχίζεται μέχρι να βρεθεί η πλειάδα με το ελάχιστο γνώρισμα που τελικά την έχει ο κόμβος D. Σημειώνουμε ότι το αποτέλεσμα έχει την ημερομηνία λήξης που συνοδεύει την πλειάδα με id = CC που τελικά ανήκει στο

αποτέλεσμα. Τέλος το μήνυμα της απάντησης ακολουθώντας το ίδιο μονοπάτι γυρνάει πίσω στον κόμβο Α.

Η πορεία του μηνύματος που αναλύθηκε αφορούσε έναν τυχαίο περιπατητή. Αν ο κόμβος Α είχε θέσει περισσότερους από έναν τότε θα του επέστρεφαν περισσότερα από ένα αποτελέσματα. Όμως για την εύρεση ελαχίστου αρκεί να κρατούσε από αυτά το ελάχιστο.

3.2.2. Αλγόριθμος για την Εύρεση του Αθροίσματος (sum)

Έστω ένας κόμβος (originator) ξεκινάει μια ερώτηση SQL για τον υπολογισμό του αθροίσματος (sum) κάποιου γνωρίσματος:

```
SELECT SUM(r(i))
FROM R
[WHERE statement]
WITH
        [TTL t]
        [PERIOD p]
        [THRES.TIME thr]
```

Η διαδικασία έχει ως εξής:

Ο αρχικός κόμβος Α δημιουργεί ένα αντικείμενο sum τύπου message, όπου εκεί καταγράφεται το αποτέλεσμα και κάθε σημαντική πληροφορία για την εξέλιξη της ερώτησης. Ο αρχικός κόμβος υπολογίζει το άθροισμα των δεδομένων του στο γνώρισμα που χρειάζεται και την ημερομηνία λήξης του αποτελέσματος και το ενσωματώνει στο μήνυμα. Στη συνέχεια το μήνυμα το στέλνει σε d γείτονές του (d random walkers). Οι γείτονες υπολογίζουν το τοπικό τους αποτέλεσμα και στη συνέχεια όσο το βάθος του μηνύματος είναι μικρότερο από το TTL το στέλνουν σε ένα τυχαία επιλεγμένο γείτονά τους. Έπειτα κάθε ενδιάμεσος κόμβος συνεχίζει όμοια

μέχρι το βάθος να είναι ίσο με TTL τότε το αποτέλεσμα το στέλνει πίσω στον αρχικό κόμβο.

Το αποτέλεσμα θα έχει ημερομηνία λήξης την ελάχιστη ημερομηνία λήξης των πλειάδων που δεν έχουν λήξει τη χρονική στιγμή τ (τωρινός χρόνος) και που το $r(i) \neq 0$. Αυτό γίνεται γιατί αν κάποια πλειάδα έχει τιμή ίση με 0 και να λήξει αυτή πάλι το ίδιο άθροισμα τιμών θα έχει το αποτέλεσμα (δηλαδή δεν παίζει ρόλο στην τελική τιμή του αποτελέσματος).

Ένα σημαντικό πρόβλημα του αλγορίθμου είναι οι διπλές πλειάδες. Κάθε κόμβος πρέπει να υπολογίζει το sum (του $i^{οστού}$ γνωρίσματος) των πλειάδων που έχει χωρίς όμως να υπολογίζει τις ίδιες πλειάδες που έχει ήδη υπολογίσει κάποιος άλλος προηγούμενος κόμβος. Για το λόγο αυτό γίνεται χρήση των φίλτρων bloom. Κάθε κόμβος υπολογίζει στο άθροισμα μόνο τις πλειάδες που δεν ανήκουν στο bloom φίλτρο και τις προσθέτει σε αυτό. Στη συνέχεια ο κόμβος στέλνει μαζί με το αποτέλεσμα στους γείτονες και το φίλτρο bloom για να γνωρίζουν και αυτοί με τη σειρά τους ποιες πλειάδες έχουν υπολογιστεί.

BLOOM ΦΙΛΤΡΑ: Ένα φίλτρο bloom [9] είναι μία δομή που αναπαριστά ένα σύνολο στοιχείων $P = \{p_1, p_2,\ldots,p_n\}$ μέσω ενός πίνακα bit μήκους m συνοδευμένο από k συναρτήσεις κατακερματισμού, έστω $h_1, h_2,\ldots, h_k$. Συγκεκριμένα, τα bits του πίνακα μπορούν να λάβουν τις τιμές 0 ή 1 ενώ υποθέτουμε ότι όλες οι συναρτήσεις κατακερματισμού, κατακερματίζουν ένα τυχαίο στοιχείο x ομοιόμορφα στο διάστημα [0, m-1]. Αρχικά όλα τα bit του φίλτρου έχουν την τιμή 0. Για να εισάγουμε ένα στοιχείο p στο φίλτρο εφαρμόζουμε κάθε μία από τις k συναρτήσεις κατακερματισμού ξεχωριστά στο p και μεταβάλουμε την τιμή του bit στη θέση $h_i(p)$ του φίλτρου σε 1, για $1 \leq i \leq k$. Για να ταιριάζει ένα στοιχείο q στο φίλτρο θα πρέπει να ισχύει το εξής: Αν εφαρμόσουμε κάθε μία από τις k συναρτήσεις κατακερματισμού ξεχωριστά στο q, θα πρέπει τα bits στις θέσεις $h_i(q)$ του φίλτρου να έχουν τιμή ίση με 1, για $1 \leq i \leq k$.

Υπάρχει περίπτωση ένα στοιχείο q να ταιριάζει στο φίλτρο bloom αλλά να μην ανήκει στο σύνολο P το οποίο αναπαρίσταται από το φίλτρο. Όμως, ένα στοιχείο που

δεν ταιριάζει στο φίλτρο bloom δεν μπορεί σε καμία περίπτωση να ανήκει στο αντίστοιχο σύνολο P.

Η πιθανότητα τέτοιων σφαλμάτων κατά τη χρήση των φίλτρων bloom εξαρτάται από τα χαρακτηριστικά τους και έχει αποδεχθεί ότι είναι ίση με:

$$\varepsilon = (1 - (1 - \frac{1}{m})^{kn})^{k} \approx (1 - e^{-\frac{kn}{m}})^{k}$$

όπου n είναι ο αριθμός των δεδομένων που θα εισαχθούν στο φίλτρο.

Στην περίπτωσή μας, το σύνολο των στοιχείων P είναι οι πλειάδες της σχέσης R στο δίκτυο που τις αναπαριστούμε σαν ένα αλφαριθμητικό και κάθε συνάρτηση κατακερματισμού μας δίνει ένα ακέραιο από 0 έως m-1. Για να μειώσουμε πολύ την πιθανότητα λάθους, έστω ότι έχουμε n=1000 (πλειάδες), k=10 συναρτήσεις κατακερματισμού και m=100 bits τότε το λάθος είναι $O(10^{-3})$.

Ένα άλλο πρόβλημα που πρέπει να σημειώσουμε εδώ είναι ότι ο αρχικός κόμβος στέλνει το ερώτημα σε d γείτονές του και στο τέλος παίρνει d διαφορετικά αποτελέσματα. Από αυτά τα αποτελέσματα δεν μπορεί να εξάγει ένα και μόνο τελικό αποτέλεσμα. Αυτό συμβαίνει γιατί αν και υπάρχουν τα d bloom φίλτρα, δεν υπάρχουν όμως οι πλειάδες από όπου προήλθαν τα d αποτελέσματα. Θα ήταν λανθασμένη τακτική να άθροιζε τα d αυτά αποτελέσματα ο αρχικός κόμβος γιατί προήλθαν από ένα ποσοστό όμοιων πλειάδων και ένα άλλο ποσοστό διαφορετικών. Επίσης δεν είναι συμφέρον να μαζεύονται όλες οι πλειάδες σε ένα κόμβο για να βγει το αποτέλεσμα διότι σε τέτοια περίπτωση θα είχαμε μεγάλη μεταφορά δεδομένων χωρίς ιδιαίτερο λόγο. Έτσι ο αρχικός κόμβος μπορεί να λάβει d διαφορετικά αποτελέσματα, ή μπορεί να θέσει d=1 και με μεγαλύτερο βάθος δικτύου να έχει καλύτερο αποτέλεσμα.

Πιο συγκεκριμένα η πληροφορία που έχει το μήνυμα παρουσιάζεται στον πίνακα 3.5.

Πίνακας 3.5 Πληροφορία του Μηνύματος

Μεταβλητή	Πληροφορία
int exp	η ημερομηνία λήξης του αποτελέσματος (αρχικά είναι άπειρο)
int sum	το αποτέλεσμα του αθροίσματος (αρχικά είναι 0)
int attr	το γνώρισμα που θέλουμε να υπολογίσουμε το άθροισμα
int bloom	το φίλτρο bloom με περίπου 100 bits για ακρίβεια 0.01
int hops	το βάθος που βρίσκεται το μήνυμα
int ttl	το τελικό βάθος της ερώτησης

Παρακάτω δίδεται ο αλγόριθμος για το άθροισμα (sum):

Αλγόριθμος υπολογισμού αθροίσματος (SUM)

```
1. if originator:
2.         create object sum
3.         explore()
4.         send sum to d-neighbors

5.  upon receive sum message:
6.  explore()
7.  if hops < TTL then
8.         send sum to a neighbor
9 .  else
10.         sum back to originator

11.  function explore()
12.  foreach tuple r in data
13.       if r not in bloom AND t_R^exp(r) > current time  then
14.                 sum = sum +  r(attr)
15.                 if exp > t_R^exp(r) then
16.                           exp = t_R^exp(r)
17.                 endif
17.                 add r tuple in bloom filter
18.        endif
19. end function
```

3.2.3. Αλγόριθμος για Συνένωση (join)

Έστω ένας κόμβος (originator) ξεκινάει μια ερώτηση SQL για τον υπολογισμό της συνένωσης (join), ως εξής:

```
SELECT *
FROM R, S
WHERE R.i θ S.j
WITH
        [TTL t]
        [PERIOD p]
        [THRES.TIME thr]
```

Υποθέτουμε ότι κάνουμε join R⋈ S όπου R, S δύο σχέσεις. R.i είναι το i γνώρισμα της σχέσης R και S.j είναι το j γνώρισμα της σχέσης S. R.i θ S.j είναι η συνθήκη που γίνεται η συνένωση με $\theta \in \{<, >, =, ...\}$.

Καθώς στο σύστημά μας οι σχέσεις είναι κατανεμημένες, κανένας κόμβος δεν έχει ολική ή έστω μερική γνώση των πλειάδων των υπολοίπων κόμβων. Για το λόγο αυτό το join δεν μπορεί να γίνει από τον κάθε κόμβο ξεχωριστά. Παρακάτω θα μελετηθεί μια προσέγγιση για τον υπολογισμό του join κατανεμημένα.

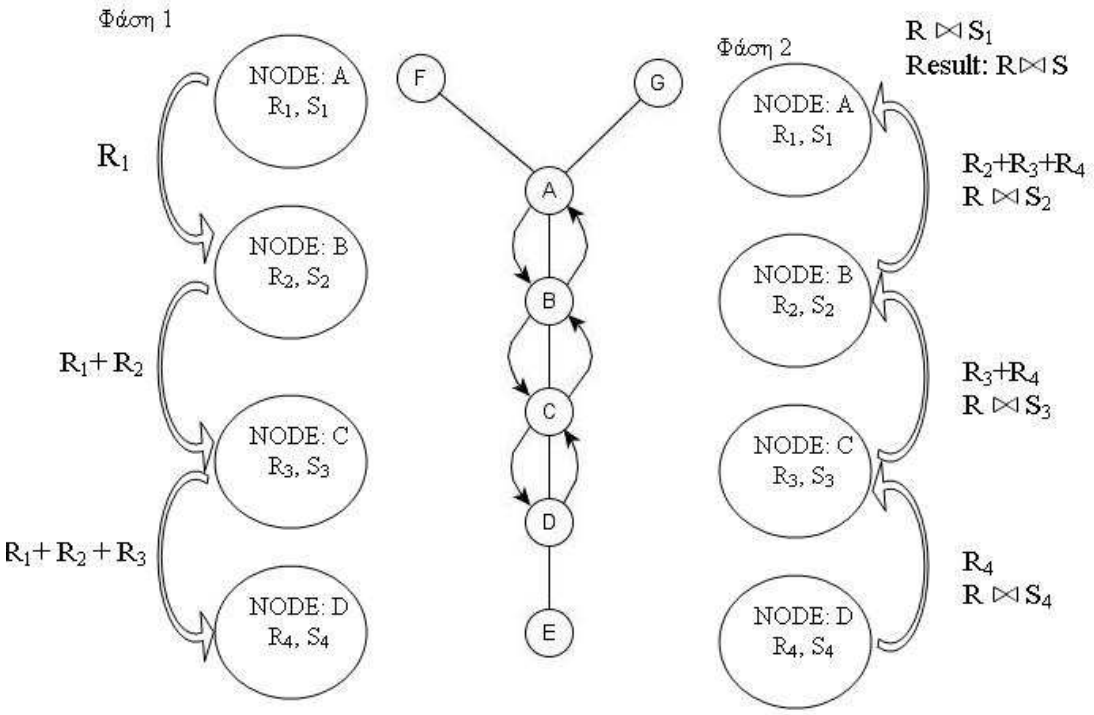
Προσέγγιση distributed join: Με αυτή τη μέθοδο ο κόμβος που ξεκινάει μια ερώτηση τύπου join δεν μπορεί να χρησιμοποιήσει ούτε πλημμύρα, ούτε τυχαίες διαδρομές. Η ερώτηση γίνεται μόνο κατά μήκος ενός μονοπατιού. Ο κόμβος στέλνει σε ένα γείτονα, ο γείτονας με τη σειρά του σε άλλον γείτονα κ.ο.κ. χρησιμοποιώντας φυσικά το βάθος της ερώτησης (TTL). Ο τελευταίος κόμβος στέλνει προς τα πίσω το join μέχρι να φτάσει στον αρχικό κόμβο.

Ο αλγόριθμος που μελετάμε έχει δύο φάσεις. Η πρώτη φάση είναι ξεκινάει από τον αρχικό κόμβο Α που δημιουργεί το μήνυμα της συνένωσης και συνεχίζει μέχρι το μήνυμα να φράσει στον κόμβο με βάθος TTL. Έπειτα αρχίζει η δεύτερη φάση όπου το μήνυμα ακολουθεί την ανίστροφη πορεία μέχρι τον αρχικό κόμβο Α.

Κατά την πρώτη φάση το μήνυμα μεταφέρει τις πλειάδες μιας σχέσης (έστω της R από τις R,S που γίνεται η συνένωση) των κόμβων που συναντάει. Στη φάση 2 ο τελευταίος κόμβος, έστω i, γνωρίζει όλες τις πλειάδες της σχέσης R των προηγουμένων του και αρχίζει τη συνένωση της σχέσης $S_ι$ ($S_ι$ είναι το υποσύνολο των πλειάδων της σχέσης S που διαθέτει ο κόμβος i) με τη σχέση R που πλέον διαθέτει. Έπειτα το μήνυμα με τις πλειάδες της σχέσης R καθώς και το αποτέλεσμα κάθε φορά συνεχίζει την αντίστροφη πορεία έως τον κόμβο Α. Εδώ πρέπει να ειπωθεί ότι το μήνυμα στη φάση 2 δεν χρειάζεται να περιέχει όλες τις πλειάδες της σχέσης R των κόμβων αλλά μόνο εκείνες που δεν γνωρίζει ο προηγούμενος κόμβος. Δηλαδή αν στον κόμβο 3 στη φάση 1 στάλθηκαν οι πλειάδες της σχέσης R των κόμβων 1,2 στη φάση 2 ο κόμβος 4 δεν του ξαναστέλνει αυτές τις πλειάδες, μονάχα τις πλειάδες των κόμβων 4 και έπειτα. Σημαντικό είναι να αναφέρουμε ότι η σχέση με τις λιγότερες πλειάδες εκ των σχέσεων που γίνεται η συνένωση, θα μεταφέρεται από κόμβο σε κόμβο και στις δύο φάσεις. Αν ο αρχικός κόμβος ζητάει τη συνένωση R⋈S ξεκινάει το μήνυμα με τις πλειάδες εκείνης της σχέσης από τις R, S που έχει τις λιγότερες σε σχέση με την άλλη.

Έτσι στο τέλος των δύο φάσεων ο αρχικός κόμβος Α έχει όλο το αποτέλεσμα της συνένωσης. Η ημερομηνία λήξης του αποτελέσματος ανατίθεται από τον κάθε κόμβο σε κάθε πλειάδα της συνένωσης ξεχωριστά. Αυτή η ημερομηνία είναι η ελάχιστη ημερομηνία λήξης των πλειάδων από όπου προήλθε η κάθε πλειάδα του αποτελέσματος της συνένωσης.

Οι δύο φάσεις απεικονίζονται στο σχήμα …όπου ο κόμβος Α ξεκινάει τη συνένωση με TTL=2.



Σχήμα 3.3 Οι δύο Φάσεις της Συνένωσης

Η παρούσα προσέγγιση έχει δύο κύρια μειονεκτήματα:

1. Το join μπορεί να γίνει μόνο κατά μήκος ενός μονοπατιού και

2. Έχουμε ανταλλαγή πολλών μηνυμάτων σε όλο το μονοπάτι (των πλειάδων R και του αποτελέσματος)

Έχει όμως και το βασικό πλεονέκτημα, ότι αν οι κόμβοι στο μονοπάτι δεν αποσυνδεθούν κατά τη διάρκεια του υπολογισμού, ο αρχικός κόμβος έχει το ακριβές αποτέλεσμα του join σε αυτό το μονοπάτι και χωρίς την ανταλλαγή όλων των R,S πλειάδων.

- Αλγόριθμος Distributed Join:

Η πληροφορία που περιέχει το μήνυμα της συνένωσης παρουσιάζεται στο πίνακα 3.6.

Πίνακας 3.6 Πληροφορία του Μηνύματος

Μεταβλητή	Πληροφορία
int phase	Η φάση του αλγορίθμου
list Rtuples	μια λίστα με τις πλειάδες της σχέσης R
list Result	μια λίστα με τις πλειάδες του αποτελέσματος
int hops	το βάθος που βρίσκεται το μήνυμα
int ttl	το τελικό βάθος της ερώτησης

Αλγόριθμος Distributed join (Κατανεμημένης Συνένωσης (Join)

```
1.  if originator:
2.      create object join
3.      explore()
4.      send join to 1-neighbor

5.  upon receive join message:
6.  explore()
7.  if hops < TTL && phase = = 0 then
8.      send join message to a neighbor
9 . else if phase = = 1 && !originator then
10.     send join message back to the previous node
11. endif

12. function explore()
13. if phase = = 0 then
14.   foreach tuple r in R
15.   Rtuples.add(r)
16.   if hops < TTL then
17.         hops=hops+1
18    else if hops = = TTL && phase = = 0 then
19.         phase = 1
20.    end if
21.    if phase = = 1 && !originator
22.
23.       foreach tuple in S
24.       foreach tuple in Rtuples
25.       if R.i θ S.i then
26.               join the tuples and add in Result list
27.        endif
28.
29.    else if phase = = 1 && originator
30.       foreach tuple in S
31.       foreach tuple in RTuples
32.       if R.i θ S.i then
33.               join the tuples and add in Result list     //the final result
34.    endif
```

Παράδειγμα συνένωσης: Έστω ο κόμβος Α θέτει την ερώτηση:

```
SELECT *
FROM R, S
WHERE R.attr1 = S.attr1
WITH
        TTL=2
```

Έστω ότι κατά τον αλγόριθμο της συνένωσης συναντιούνται οι κόμβοι Α, Β, C, κατά μήκος ενός μονοπατιού, με τη βάση δεδομένων τους να είναι όπως δείχνει το σχήμα 3.4.

NODE: A

R		
Id	Attr.1	Exp.time
AA	123	12
AB	12	20

NODE: B

R		
Id	Attr.1	Exp.time
BC	12	30
BA	39	5

NODE: C

R		
Id	Attr.1	Exp.time
CC	15	22
CD	24	18

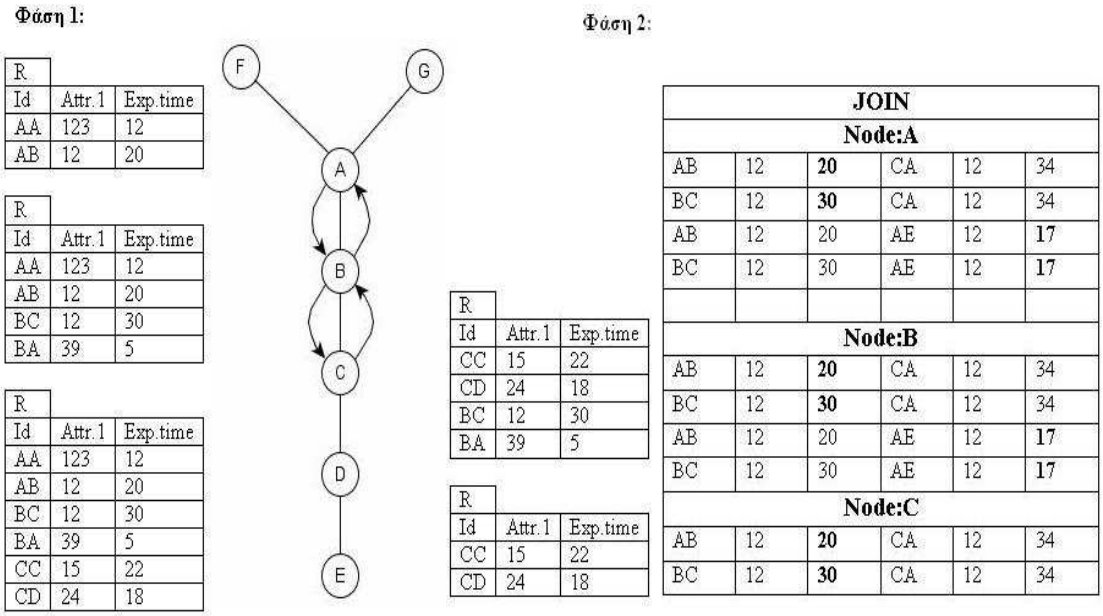
S		
Id	Attr.1	Exp.time
AF	311	5
AD	19	11

S		
Id	Attr.1	Exp.time
BF	34	4
AE	12	17

S		
Id	Attr.1	Exp.time
CG	15	27
CA	12	34

Σχήμα 3.4 Βάση Δεδομένων των Κόμβων Α, Β, C

Το σχήμα 3.5 δείχνει πως υπολογίζεται η συνένωση σύμφωνα με τον αλγόριθμο που μελετήσαμε παραπάνω.



Σχήμα 3.5 Παράδειγμα Υπολογισμού Συνένωσης

Στο σχήμα 3.5 παρατηρούμε ότι κάθε κόμβος προωθεί τις πλειάδες της σχέσης R που διαθέτει καθώς και των προηγουμένων του. Στη φάση 2 προωθούνται μόνο οι πλειάδες που δεν διαθέτουν οι προηγούμενοι κόμβοι καθώς και το αποτέλεσμα της συνένωσης.

ΚΕΦΑΛΑΙΟ 4. ΠΡΩΤΟΚΟΛΛΑ ΕΝΗΜΕΡΩΣΗΣ-ΑΝΤΙΓΡΑΦΗΣ ΜΕ ΠΡΟΣΔΙΟΡΙΣΜΟΥΣ ΠΟΙΟΤΗΤΑΣ

Στο κεφάλαιο αυτό, παρουσιάζονται και αναλύονται τα διαφορετικά πρωτόκολλα ενημέρωσης, αντιγραφής και λανθάνουσας μνήμης που μπορεί να ενεργοποιήσει κάθε κόμβος έτσι ώστε να διατηρηθεί ενημερωμένη η κατανεμημένη βάση του συστήματος και να διαδίδονται οι ενημερώσεις ταχύτερα. Πιο συγκεκριμένα στην ενότητα 4.1 παρουσιάζονται τα πρωτόκολλα ενημέρωσης και αντιγραφής που μπορεί να ενεργοποιήσει κάθε κόμβος καθώς και πως αυτά τα πρωτόκολλα συμβάλλουν στην ποιότητα υπηρεσιών και δεδομένων (QoS, QoD) στο δίκτυο. Τέλος στην ενότητα 4.2 γίνεται αναφορά στα πρωτόκολλα λανθάνουσας μνήμης και αντιγραφής που είναι και αυτά χρήσιμα για την διατήρηση της φρεσκάδας της κατανεμημένης βάσης δεδομένων στο σύστημα.

4.1. Πρωτόκολλα Ενημέρωσης (Update) και Αντιγραφής (Replication) Βασισμένα σε Φρεσκάδα

Στην παρούσα ενότητα θα αναλυθούν τα διαφορετικά πρωτόκολλα ενημέρωσης (update) και αντιγραφής (replication) που μπορεί να χρησιμοποιήσει κάθε κόμβος έτσι ώστε να έχει ενημερωμένη την τοπική του βάση δεδομένων, να διαγράφει “απαρχαιωμένες” πλειάδες και να συλλέγει πλειάδες οι οποίες είναι αρκετά

ενημερωμένες. Έτσι συμβάλλει στη γρήγορη διάδοσή τους και στην ενημέρωση της κατανεμημένης βάσης δεδομένων του συνολικού συστήματος.

Τα πρωτόκολλα που θα αναλυθούν, θα μπορούν να συνδυαστούν μεταξύ τους με στόχο την ενημέρωση της βάσης δεδομένων του κόμβου. Είναι σημαντικό κάθε πρωτόκολλο που μελετάται, να παρουσιάζεται και η σχέση του με την ποιότητα των δεδομένων (Quality of Data) που συγκαταλέγεται η φρεσκάδα (Freshness) και η ανάκληση (Recall), καθώς και με την ποιότητα υπηρεσιών (Quality of Service) που συγκαταλέγονται ο χρόνος απόκρισης (Response Time) και ο φόρτος μηνυμάτων (Message Overhead), όπως αναλύθηκαν σε προηγούμενο κεφάλαιο.

4.1.1. Πρωτόκολλα Ενημέρωσης (Update) Βασισμένα σε Φρεσκάδα

Στην παρούσα υποενότητα θα μελετηθούν τα πρωτόκολλα ενημέρωσης και αυτά είναι τα εξής:

- UP0: Όταν μια πλειάδα λήξει, απλά δεν γίνεται τίποτα

Σε αυτό το πρωτόκολλο, καθώς δεν ανανεώνονται οι πλειάδες που λήγουν, έχουμε εξαιρετικά χαμηλής απόδοσης ποιότητα δεδομένων (QoD) και η βάση δεδομένων τόσο η τοπική όσο και η ολική παραμένει “απαρχαιωμένη”.

Για την ποιότητα υπηρεσιών (QoS), δεν έχουμε μεταφορά μηνυμάτων (άρα ούτε και hops) άρα και ο φόρτος δικτύου που επιφέρει είναι μηδενικός. Επίσης δεν συμβάλλει στην ενημέρωση της βάσης δεδομένων ούτε στη διάδοση των ενημερώσεων

- UP1: Όταν μια πλειάδα λήξει, ο κόμβος την αποβάλλει απλά από τη βάση δεδομένων του

Αυτό το πρωτόκολλο δίνει τη δυνατότητα στο κόμβο να αποβάλλει πλειάδες στη βάση δεδομένων του που έχουν λήξει, οπότε και να μην έχει απαρχαιωμένα δεδομένα.

Υπάρχουν δύο τεχνικές [1] διαγραφής δεδομένων σε μια βάση, βασισμένοι στην ημερομηνία λήξης τους: χαλαρή διαγραφή (lazy removal) και ανυπόμονη διαγραφή (eager removal). Σύμφωνα με την τεχνική “lazy removal” τα δεδομένα διαγράφονται ανά τακτά χρονικά διαστήματα, για παράδειγμα κάθε t χρονικές μονάδες, ενώ στη δεύτερη τεχνική “eager removal” τα δεδομένα διαγράφονται όσο πιο σύντομα γίνεται, δηλαδή σε κάθε χρονική μονάδα. Αν και με την πρώτη τεχνική τα δεδομένα διαγράφονται ανά περιόδους και υπάρχει εγγύηση ότι τα ληγμένα δεδομένα δεν είναι ορατά στο χρήστη, παρόλα αυτά δεν γίνεται εξοικονόμηση αποθηκευτικού χώρου και λόγω του κατανεμημένου της βάσης δεδομένων που έχουμε προτιμήθηκε η τεχνική “eager removal”. Η τεχνική “eager removal” έγκειται στο ότι όταν ενεργοποιηθεί το πρωτόκολλο UP1 γίνεται αμέσως διαγραφή των ληγμένων πλειάδων.

Το συγκεκριμένο πρωτόκολλο δεν βοηθάει στην ενημέρωση των πλειάδων, αλλά προσφέρει την εγγύηση ότι στο επιστρεφόμενο αποτέλεσμα δεν θα υπάρχουν ληγμένα δεδομένα. Έτσι δεν μπορεί να γίνει και πάλι ανάλυση της ποιότητας των υπηρεσιών ή των δεδομένων, αφού δεν προσφέρει υπηρεσίες και δεδομένα, απλά βοηθάει στην απομάκρυνση πλειάδων που επιβαρύνουν το σύστημα σε μηνύματα και αποθηκευτικό χώρο, με κάθε προσπάθεια ανάκτησής τους. Εδώ πρέπει να προστεθεί ότι και πάλι ο φόρτος δικτύου που επιφέρει το πρωτόκολλο είναι μηδενικός αφού δεν έχουμε ανταλλαγή μηνυμάτων.

- UP2: Όταν μια πλειάδα λήξει, ο κόμβος ψάχνει με νέο ερώτημα για όμοια πλειάδα που δεν έχει ακόμα λήξει

Το παραπάνω πρωτόκολλο συγκαταλέγεται στα πρωτόκολλα ενημέρωσης διότι δίνει τη δυνατότητα στο κόμβο, όχι μόνο να αποβάλλει πλειάδες που έχουν λήξει (σε συνδυασμό και με το UP1) αλλά και να ενημερώνει τις απαρχαιωμένες πλειάδες με νέες περισσότερο ενημερωμένες που δεν έχουν ακόμα λήξει.

Γνωρίζουμε ότι κάθε πλειάδα έχει ένα κλειδί και διάφορα γνωρίσματα όπου ένα από αυτά είναι η ημερομηνία λήξης της. Δύο πλειάδες είναι όμοιες όταν έχουν το ίδιο κλειδί και αλλάζουν στα γνωρίσματα. Με το πρωτόκολλο UP2 όταν μια πλειάδα λήξει, ο κόμβος ψάχνει με ερώτηση μια πλειάδα με ίδιο κλειδί που δεν έχει ακόμα

λήξει. Δηλαδή, ενδεχομένως να εντοπίσει μια πλειάδα που αναφέρεται στο ίδιο κλειδί με διαφορετικά ένα ή περισσότερα γνωρίσματα όπου ένα από αυτά είναι η ημερομηνία λήξης. Βέβαια, επειδή αυτό γίνεται με νέα αναζήτηση στο δίκτυο, το πρωτόκολλο έχει και το ανάλογο κόστος μηνυμάτων.

Επειδή, λοιπόν, η βάση δεδομένων του κόμβου ενδέχεται να είναι αρκετά μεγάλη, το πρωτόκολλο αυτό παράγει πολλές ερωτήσεις κάθε χρονική στιγμή. Το πόσο εύκολα μπορεί να βρεθεί μια πλειάδα που δεν έχει λήξει, έγκειται στο πόσο διαδεδομένη είναι. Επειδή όμως όλοι οι κόμβοι συμβάλλουν σε αυτό αυξάνεται η διαθεσιμότητα των φρέσκων δεδομένων. Το πρόβλημα εδώ είναι ο αριθμός των μηνυμάτων και τα hops της ερώτησης που επιβαρύνουν το δίκτυο. Για να τα ελαττώσουμε, αρκεί να μειώσουμε το βάθος της αναζήτησης, γιατί όπως θα αναλύσουμε παρακάτω, τα μηνύματα αυξάνονται εκθετικά ως προς το βάθος δικτύου (TTL), ενώ αυξάνονται γραμμικά ως προς τη συχνότητα των ερωτήσεων. Δεδομένου ότι η συχνότητα παραγωγής των ερωτήσεων είναι μεγάλη, μειώνουμε το TTL.

Τελικά, με δεδομένο ότι το παρόν πρωτόκολλο θα το εφαρμόζουν όλοι οι κόμβοι, η ποιότητα των δεδομένων (QoD) που θα προσφέρει θα είναι αυξημένη, καθώς θα βελτιώνεται η φρεσκάδα (freshness) των αποτελεσμάτων, τα φρέσκα δεδομένα θα διαδίδονται γρήγορα, οπότε βελτιώνεται και το ποσοστό των επιτυχών αποτελεσμάτων σε σχέση με αυτά που υπάρχουν στο δίκτυο (Recall). Η ποιότητα των υπηρεσιών (QoS) θα είναι επίσης βελτιωμένη για μια απλή ερώτηση στο δίκτυο: πλέον τα φρέσκα δεδομένα θα είναι πιο διαδεδομένα, οπότε θα χρειάζονται λιγότερα βήματα για το επιθυμητό αποτέλεσμα (Response Time). Το πρωτόκολλο αυτό όμως, έχει και κάποιο κόστος σε μηνύματα που είναι αρκετά σημαντικό αν γίνεται για κάθε πλειάδα που λήγει και θα αυξάνεται περισσότερο όσο το TTL αυξάνει. Γενικά το πρωτόκολλο επιφέρει έναν μεγάλο φόρτο δικτύου (message overhead).

Καθώς όμως χρησιμοποιώντας το πρωτόκολλο έχουμε επιβάρυνση δικτύου με μηνύματα, πρέπει να μελετηθεί πειραματικά πότε υπάρχει χρυσή τομή μεταξύ των μηνυμάτων του πρωτοκόλλου και των μηνυμάτων της ερώτησης αν δεν υπήρχε το πρωτόκολλο και γινόταν η ερώτηση ρητώς από κάποιους κόμβους που χρειάζονταν αυτά τα δεδομένα. Είναι αναμφισβήτητο όμως ότι με το πρωτόκολλο ο φόρτος

μηνυμάτων κατανέμεται σε όλο το μήκος του δικτύου, ενώ με την ρητή ερώτηση με μεγάλο βάθος δικτύου, αυξάνεται ο φόρτος εκθετικά σε ορισμένους κόμβους κάθε φορά. Οπότε συμβάλλει στο να κατανέμεται ο φόρτος δικτύου αρκετά σωστά (load balancing).

- UP2-Lazy: Σε κάθε n χρονικές στιγμές ο κόμβος αρχικοποιεί το update protocol για όλες τις ληγμένες πλειάδες στην τοπική βάση δεδομένων του

Το πρωτόκολλο αυτό είναι ένα περιοδικό πρωτόκολλο, συγκαταλέγεται στα πρωτόκολλα ενημέρωσης γιατί μπορεί να συνδυαστεί με αυτά, όμως μπορεί να συνδυαστεί και με τα πρωτόκολλα αντιγραφής. Έχει σχεδόν την ίδια επίδραση με τα πρωτόκολλα με τα οποία συνδυάζεται, με τη διαφορά ότι δεν αρχικοποιείται το συνδυαζόμενο πρωτόκολλο αμέσως, αλλά ενεργοποιείται κάθε *n* χρονικές στιγμές. Αν συνδυαστεί με το Up2 έχει το μειονέκτημα ότι μπορεί, αν το *n* είναι μεγάλο, να υπάρχουν πλειάδες στη βάση που να έχουν λήξει. Το πρόβλημα ξεπερνιέται με την εισαγωγή ενός threshold (=*n*) κάθε φορά που γίνεται έλεγχος πλειάδων. Έτσι κάθε *n* χρονικές στιγμές αρχικοποιείται το update πρωτόκολλο για τα πλειάδες που έχουν λήξει ή θα λήξουν σε λιγότερο από *n* χρονικές μονάδες. Αυτό το πρωτόκολλο συγκαταλέγεται στη lazy [1] τεχνική όπως αναφέρθηκε παραπάνω.

Ένα επιπλέον μειονέκτημα είναι ότι αν όλοι οι κόμβοι αρχικοποιήσουν τα πρωτόκολλα ενημέρωσης ταυτόχρονα, τα μηνύματα που θα παράγονται στο δίκτυο θα είναι πάρα πολλά και θα παρουσιαστεί μεγάλος φόρτος εργασίας. Για το λόγο αυτό θα πρέπει οι *n* χρονικές μονάδες να μην συμπίπτουν.

Τελικά αν ρυθμιστεί σωστά το παραπάνω πρωτόκολλο θα έχει ευεργετικά αποτελέσματα ισάξια με το προηγούμενο ως προς το QoS, QoD, φορτώνοντας το δίκτυο με λιγότερα μηνύματα αν συνδυαστεί κατάλληλα και με άλλα πρωτόκολλα που θα αναλυθούν παρακάτω.

4.1.2. Πρωτόκολλα Αντιγραφής (Replication) Βασισμένα σε Φρεσκάδα

Στην συγκεκριμένη υποενότητα θα μελετήσουμε τα πρωτόκολλα αντιγραφής. Στα πρωτόκολλα που θα αναφερθούν, χρησιμοποιούνται οι τεχνικές push και pull [5]. Οι τεχνικές αυτές αποσκοπούν στην δημιουργία ενημερωμένων αντιγράφων κατά μήκος του δικτύου. Κατά την τεχνική push ο κόμβος που έχει ένα ενημερωμένο αντίγραφο το προωθεί σε έναν ή περισσότερους γείτονές του. Ενώ κατά την τεχνική pull, ένας κόμβος ζητάει από έναν ή περισσότερους γείτονες να του προωθήσουν τα δικά τους ενημερωμένα αντίγραφα. Με τις τεχνικές αυτές επιτυγχάνεται γρήγορη διάδοση των ενημερώσεων όπου χρειάζεται, αφού απαιτούνται μόνο ελάχιστα βήματα και μηνύματα για να ολοκληρωθούν.

- UP2-Push: για κάθε πλειάδα που έχει ημερομηνία λήξης μεγαλύτερη από ένα κατώφλι, ο κόμβος που διαθέτει τη πλειάδα την προωθεί στους γείτονές του

Το πρωτόκολλο αφορά πρωτόκολλα δημιουργίας αντιγράφων και χρησιμοποιεί την τεχνική push που αναφέρθηκε παραπάνω. Συμβάλλει στη διάδοση πολύ πρόσφατων ενημερώσεων σε πλειάδες, με σκοπό τη γρήγορη διάδοσή τους και τη συνολική φρεσκάδα της κατανεμημένης βάσης δεδομένων του συστήματος. Διαδίδει, δεδομένα που είναι απολύτως απαραίτητα για τη συντήρηση-ενημέρωση του δικτύου, με πολύ μικρό φόρτο μηνυμάτων: αρκεί ένα μήνυμα για να φτάσει η ενημέρωση από ένα κόμβο σε κάποιον γείτονά του. Έτσι συμβάλλει και στη φρεσκάδα (Freshness) του δικτύου και στη εύκολη ανάκτηση (Recall) των αποτελεσμάτων αφού διαδίδει τις ενημερώσεις στους γείτονες, οπότε η ποιότητα των δεδομένων που προσφέρει είναι αρκετά σημαντική. Τέλος ως προς την ποιότητα των υπηρεσιών (QoS), επιβαρύνει με πολύ λίγα μηνύματα κάθε φορά (όσους γείτονες έχει, τόσα μηνύματα στέλνει), με στόχο καλύτερα αποτελέσματα σε μελλοντικές ερωτήσεις που γίνονται στο δίκτυο. Το κέρδος των μηνυμάτων που έχει το δίκτυο γίνεται αντιληπτό στην περίπτωση που κάποιος κόμβος αναζητάει ένα δεδομένο και μπορεί να το βρει πλέον σε σημαντικά λιγότερα βήματα.

- UP3-Push: Ένα push πρωτόκολλο όπου κάθε κόμβος κάνει push τα hot δεδομένα (πλειάδες) που δεν έχουν λήξει στους γείτονες. Τα hot δεδομένα

είναι τα πιο φημισμένα δεδομένα στο δίκτυο, με πολλές αιτήσεις στο κόμβο για ανάκτησή τους.

Ένα παρόμοιο push πρωτόκολλο με το UP2-Push είναι το UP3-Push που προσφέρει στους γείτονες δεδομένα που είναι πολύ δημοφιλή (hot). Ο κόμβος γνωρίζει ότι ένα δεδομένο είναι hot αν δεχθεί πάνω από ένα αριθμό αιτήσεων για το δεδομένο αυτό σε ένα ορισμένο χρονικό διάστημα που το θέτει το σύστημα.

Το πρωτόκολλο UP3-Push προσφέρει παρόμοια ποιότητα δεδομένων και υπηρεσιών (QoD, QoS) και έχουμε το ίδιο κέρδος μηνυμάτων με το προηγούμενο UP2-Push. Η μόνη διαφορά τους είναι ότι με αυτό το πρωτόκολλο γίνονται push τα φημισμένα δεδομένα, δηλαδή τα πιο δημοφιλή δεδομένα της βάσης δεδομένων του κόμβου, ενώ στο UP2-Push γινόταν push τα πιο φρέσκα δεδομένα ανεξάρτητα δημοφιλίας. Λόγω του μεγάλου κέρδους μηνυμάτων που έχουμε με τη push μέθοδο προστέθηκε το πρωτόκολλο αυτό έτσι ώστε να μειωθεί ο φόρτος εργασίας στο δίκτυο για τα πολύ φημισμένα δεδομένα όπου οι αναζητήσεις τους θα είναι αρκετά συχνές.

- UP4-push: Ένα push πρωτόκολλο όπου κάποιος κόμβος κάνει push τα πιο πρόσφατα δεδομένα (πλειάδες) που δεν έχουν λήξει, σε γειτονικό κόμβο που δεν έχει ξαναγίνει push ούτε του έχει ζητήσει pull για μεγάλο χρονικό διάστημα

Το παρών push πρωτόκολλο είναι παρόμοιο με τα προηγούμενα push πρωτόκολλα, με τη διαφορά ότι το push δεν εξαρτάται από τα δεδομένα (αν είναι αρκετά δημοφιλή ή φρέσκα), αλλά από τον κόμβο-γείτονα. Το πρωτόκολλο ενεργοποιείται σε περίπτωση που κάποιος κόμβος μείνει για αρκετή ώρα εκτός δικτύου, λόγω κάποιου προβλήματος σύνδεσης ή έχει μόλις συνδεθεί και χρειάζεται τα πρόσφατα δεδομένα τα οποία κατά μεγάλη πιθανότητα δεν θα τα διαθέτει. Η έλλειψη του πρωτοκόλλου θα αναγκάσει το κόμβο που επανασυνδέεται να αναζητήσει πρόσφατα δεδομένα με ρητή αναζήτηση που είναι αρκετά πιο δαπανηρή, αν υποτεθεί ότι η push στρατηγική χρειάζεται μόνο ένα μήνυμα για κάθε δεδομένο και έτσι υπάρχει σημαντικό κέρδος σε μηνύματα. Η ποιότητα υπηρεσιών και δεδομένων που προσφέρει είναι όμοια με τα

άλλα push πρωτόκολλα, αλλά αυτό που ενδιαφέρει είναι η μείωση του συνολικού φόρτου μηνυμάτων στο δίκτυο.

- UP5-pull: Ένα pull πρωτόκολλο όπου ένας κόμβος που μόλις συνδέεται, ή είχε πρόβλημα επικοινωνίας με άλλους κόμβους για ένα χρονικό διάστημα, ή δεν έχει λάβει ενημερωμένα δεδομένα για *n* χρονικές μονάδες τότε ζητάει νέα δεδομένα από τους γείτονες.

Το πρωτόκολλο συγκαταλέγεται στα πρωτόκολλα δημιουργίας αντιγράφων. Ενεργοποιείται όταν κάποιος κόμβος έχει πρόβλημα επικοινωνίας ή δικτύου για κάποιο διάστημα και όταν ανακάμψει, μπορεί να ζητήσει είτε ορισμένα δεδομένα που τον ενδιαφέρουν, είτε τα πιο φημισμένα, είτε τα πιο πρόσφατα ενημερωμένα.

Το UP5-pull πρωτόκολλο βοηθάει στην ενημέρωση της τοπικής βάσης δεδομένων επιβαρύνοντας το δίκτυο με ελάχιστα μηνύματα: αρκούν δύο μηνύματα (μια αίτηση pull και μια απάντηση) για κάθε νέα πλειάδα που λαμβάνει από κάποιον γείτονα. Έτσι έχει θετική επίπτωση στο επιστρεφόμενο αποτέλεσμα και σε μετέπειτα αναζητήσεις αυτών των δεδομένων από άλλους κόμβους. Τελικά βοηθάει στην φρεσκάδα της βάσης δεδομένων του κόμβου, αλλά και του συνολικού συστήματος.

Στην συνέχεια παραθέτουμε ένα πίνακα (πίνακας 4.1) όπου και γίνεται σύγκριση των διαφόρων πρωτοκόλλων ως προς το QoS, QoD που προσφέρουν. Πολλά εξαρτώνται από το TTL ή από το πόσο ενημερωμένος είναι ο κόμβος, ή από το πόσα αντίγραφα υπάρχουν στο δίκτυο κ.α. και μπορούν να βρεθούν μόνο πειραματικά με διάφορες παραμέτρους. Για τον πίνακα έγινε μια προσθήκη στο QoD που μετράει τη φρεσκάδα της τοπικής βάσης δεδομένων του κόμβου.

Πίνακας 4.1 Συγκεντρωτικός Πίνακας Πρωτοκόλλων Ενημέρωσης και QoS, QoD

Update protocols	QoD			QoS	
	Freshness	Recall	Freshness of the local database	Response time	Message overhead
UP0	- δεν βοηθάει	- δεν βοηθάει	- “απαρχαιωμένη» local database	- δεν βοηθάει	+ κανένα μήνυμα
UP1	- δεν βοηθάει	- δεν βοηθάει	- δεν ενημερώνεται	- δεν βοηθάει	+ κανένα μήνυμα
UP2	+ βοηθάει στη διάδοση ενημερώσεων	+ βοηθάει στη διάδοση ενημερώσεων	+ Έχουμε ενημερωμένη local database πάντα	- εξαρτάται από το TTL	- εξαρτάται από το TTL
UP2-Lazy	+ βοηθάει στη διάδοση ενημερώσεων	+ βοηθάει στη διάδοση ενημερώσεων	+ Έχουμε ενημερωμένη local database	+ συμβάλλει στη μείωση των βημάτων	+ μειώνονται τα μηνύματα
UP2-push	+ βοηθάει στη διάδοση ενημερώσεων	+ βοηθάει στη διάδοση ενημερώσεων	+ Έχουμε ενημερωμένη local database των γειτόνων	++ Μειώνονται τα hops για το επιθ. αποτέλεσμα	++ 1 μήνυμα για κάθε πλειάδα
UP3-push	+ βοηθάει στη διάδοση των hot δεδομένων	+ βοηθάει στη διάδοση των hot δεδομένων	+ Έχουμε ενημερωμένη local database των γειτόνων	++ Μειώνονται τα hops για το επιθ. αποτέλεσμα	++ 1 μήνυμα για κάθε πλειάδα
UP4-push	+ Ενημερώνεται local database των γειτόνων	+ βοηθάει στη διάδοση ενημερώσεων	+ Έχουμε ενημερωμένη local database των γειτόνων	++ Μειώνονται τα hops για το επιθ. αποτέλεσμα	++ 1 μήνυμα για κάθε πλειάδα
UP5-pull	+ Ενημερώνεται καλά η local database	+ βοηθάει στη διάδοση ενημερώσεων	+ Έχουμε ενημερωμένη local database	++ Μειώνονται τα hops για το επιθ. αποτέλεσμα	++ 2 μηνύματα για κάθε πλειάδα

Ο πίνακας 4.1 αναλύει τη σχέση της ποιότητας δεδομένων και υπηρεσιών με τα διάφορα πρωτόκολλα που χρησιμοποιούνται στο δίκτυο. Είναι πολύ χρήσιμο όμως να εξετάσουμε τα πρωτόκολλα αυτά πως συμβάλλουν στην ευρωστία του δικτύου, αν συνδυαστούν με αναζητήσεις δεδομένων, εφαρμόζοντας κάποια από τις μεθόδους

αναζήτησης όπως παρουσιάστηκαν σε προηγούμενο κεφάλαιο. Δηλαδή, αν παράλληλα με τα παραπάνω πρωτόκολλα ενημέρωσης, τυχαίοι κόμβοι στο δίκτυο αναζητούν τυχαία δεδομένα, όπως συμβαίνει σε ένα πραγματικό peer-to-peer σύστημα, χρησιμοποιώντας δεδομένο TTL, έστω TTL = *t*.

Στο πίνακα 4.2 γίνεται σύγκριση των πρωτοκόλλων ενημέρωσης συνδυασμένα με ερωτήσεις δεδομένων από τους κόμβους του συστήματος με δεδομένο TTL σε σχέση με την ποιότητα δεδομένων και υπηρεσιών που προσφέρουν. Για λόγους απλότητας παραλείφτηκε η έννοια της φρεσκάδας (freshness, freshness of the local database) γιατί ο συνδυασμός πρωτοκόλλου και αναζήτησης δεν έχει καμία διαφορετική επίπτωση σε αυτό. Η σύγκριση έγκειται στο Recall, Response time και Message overhead.

Πίνακας 4.2 Συγκεντρωτικός Πίνακας Πρωτοκόλλων Ενημέρωσης-Αναζήτησης και QoS, QoD

Update protocols-search	QoD	QoS	
	Recall	Response time	Message overhead
UP0-search	- δεν βελτιώνεται	- δεν βελτιώνεται	- δεν βελτιώνεται
UP1-search	- δεν βελτιώνεται	- δεν βελτιώνεται	- δεν βελτιώνεται
UP2-search	+ βελτιώνεται καθώς οι ενημερώσεις διαδίδονται	+ βελτιώνεται καθώς οι ενημερώσεις διαδίδονται	- εξαρτάται από τη σχέση TTL πρωτοκόλλου και αναζήτησης
UP2-push -search	+ βελτιώνεται καθώς τα φρέσκα δεδομένα διαδίδονται περισσότερο	+ βελτιώνεται καθώς με ένα hop βρίσκουμε το δεδομένο με μικρό TTL	+ βελτιώνεται καθώς με ένα μήνυμα μειώνονται τα μηνύματα της αναζήτησης
UP3-push - search	+ βελτιώνεται καθώς τα φημισμένα δεδομένα διαδίδονται περισσότερο	+ βελτιώνεται καθώς με ένα hop βρίσκουμε το δεδομένο με μικρό TTL	+ βελτιώνεται καθώς με ένα μήνυμα μειώνονται τα μηνύματα της αναζήτησης
UP4-push – search	+ βελτιώνεται καθώς τα φρέσκα δεδομένα διαδίδονται περισσότερο	+ βελτιώνεται καθώς με ένα hop βρίσκουμε το δεδομένο με μικρό TTL	+ βελτιώνεται καθώς με ένα μήνυμα μειώνονται τα μηνύματα της αναζήτησης
UP5-pull - search	+ βοηθάει στη διάδοση ενημερώσεων	+ βελτιώνεται καθώς με 2 hops βρίσκουμε το δεδομένο με μικρό TTL	+ βελτιώνεται καθώς με δύο μήνυματα μειώνονται τα μηνύματα της αναζήτησης

4.2. Πρωτόκολλα Λανθάνουσας Μνήμης και Αντιγραφής (Caching-replication)

Εκτός όμως από τα πρωτόκολλα ενημέρωσης, χρησιμοποιούμε και τη λανθάνουσα μνήμη του κόμβου (cache), για να αντιγράψουμε πλειάδες που είναι χρήσιμες για την ανανέωση της τοπικής βάσης δεδομένων του κόμβου. Τα πρωτόκολλα caching-replication είναι τα παρακάτω:

- Pr0: Δεν κρατάμε cache ούτε κάνουμε replication

Αυτό το πρωτόκολλο δεν μας δίνει καμία βελτίωση στο σύστημα.

- Pr1-cache (ή path replication): Κάθε κόμβος κρατάει σε μια cache τα αποτελέσματα από τα *m* τελευταία queries που έχουν περάσει από αυτόν.

Το παραπάνω πρωτόκολλο αναφέρεται στην path replication τακτική [8]. Καθώς το αποτέλεσμα μιας επιτυχημένης αναζήτησης περνάει από όλο το μονοπάτι της αναζήτησης, από τον κόμβο που διαθέτει το δεδομένο μέχρι τον αρχικό κόμβο, οι ενδιάμεσοι κόμβοι αποθηκεύουν το αποτέλεσμα στην λανθάνουσα μνήμη τους. Η συγκεκριμένη τεχνική για το λόγο αυτό ονομάζεται path replication και έχει στόχο να μπορούν οι κόμβοι να απαντάνε αμέσως σε παρόμοια queries που θα του γίνουν. Έτσι χωρίς επιπλέον κόστος μηνυμάτων δημιουργούνται περισσότερα αντίγραφα των δεδομένων κατά μήκος του δικτύου και μειώνεται το κόστος δικτύου σε μηνύματα λόγω πολλαπλών αναζητήσεων όμοιων δεδομένων.
Επειδή η λανθάνουσα μνήμη δεν μπορεί να είναι απεριόριστη, γίνεται η σύμβαση ότι κάθε κόμβος αποθηκεύει τα *m* (χρονολογικά) τελευταία ερωτήματα που θα του γίνουν και τα παλαιότερα τα διαγράφει.

- Pr2-replication: Ο κόμβος αντιγράφει τα πιο hot δεδομένα της cache του στην τοπική του βάση δεδομένων

Κάθε κόμβος στην cache του μπορεί να έχει κάποια δεδομένα συνέχεια, λόγω πολλαπλών όμοιων ερωτημάτων. Τα δεδομένα αυτά, τα ορίζει σαν hot και τα γράφει στην τοπική του βάση δεδομένων. Αυτό σημαίνει ότι τα δεδομένα αποθηκεύονται

πλέον τοπικά και ο κόμβος δεν τα διαγράφει τουλάχιστον μέχρι να λήξουν. Έτσι έχουμε περισσότερα αντίγραφα, που δεν έχουν λήξει, σε περισσότερους κόμβους στο δίκτυο.

Εδώ δεν μπορούμε να μετρήσουμε την ποιότητα υπηρεσιών και δεδομένων (QoS, QoD) γιατί δεν έχουμε ερώτηση ούτε ενημέρωση πλειάδων, δεν έχουμε καμία είδους αίτηση-απάντηση οπότε δεν μπορούμε να μετρήσουμε TTL, hops, μηνύματα στο δίκτυο κλπ. Όμως τέτοιες τεχνικές είναι πολύ ωφέλιμες για το δίκτυο 1) για να υπάρχουν περισσότερα δεδομένα που δεν έχουν λήξει σε περισσότερους κόμβους, 2) για να απαντάται μια ερώτηση όσο το δυνατόν πιο γρήγορα από ένα κόμβο (χωρίς επιπλέον φόρτο εργασίας) και 3) για να γνωρίζουν ποια είναι τα φημισμένα (hot) δεδομένα όλο και περισσότεροι κόμβοι και να τα διαδίδουν.

ΚΕΦΑΛΑΙΟ 5. ΑΝΑΛΥΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ

Στο κεφάλαιο αυτό θα παρουσιάσουμε μια θεωρητική μελέτη του συστήματός μας ως προς το κόστος και την απόδοση της αναζήτησης καθώς και των διαφόρων πρωτοκόλλων που μπορούν να χρησιμοποιηθούν. Πιο συγκεκριμένα στην ενότητα 5.1 γίνεται θεωρητική προσέγγιση της αναζήτησης με πλημμύρα όσο αφορά το φόρτο δικτύου και την απόδοσή της. Στην ενότητα 5.2 γίνεται μια παρόμοια μελέτη για την αναζήτηση με βάση τις τυχαίες διαδρομές (random walks). Τέλος στην ενότητα 5.3 γίνεται ανάλυση της απόδοσης των πρωτοκόλλων που χρησιμοποιούνται στο δίκτυο όσο αφορά το κόστος και την απόδοσή τους.

5.1. Θεωρητικό Μοντέλο Αναζήτηση με Πλημμύρα

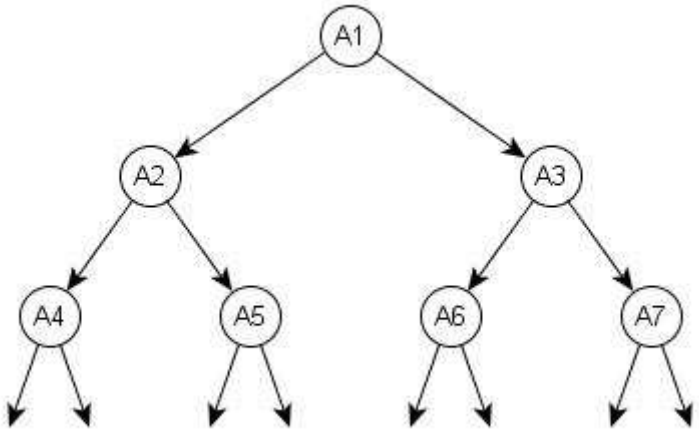
Στην παρούσα ενότητα θα γίνει θεωρητική ανάλυση της αναζήτησης με τη μέθοδο της πλημμύρας, όσον αφορά τα μηνύματα που χρειάζονται, το φόρτο δικτύου και την απόδοση αναζήτησης στο δίκτυο που μελετάμε.

Οι αναζητήσεις αυτές δεν σχετίζονται με ερωτήσεις τύπου SELECT, ή λειτουργίες συνάθροισης και συνένωσης όπως μελετήθηκαν στο κεφάλαιο 3. Οι αναζητήσεις είναι αναζητήσεις δεδομένων σε ένα δίκτυο ομότιμων κόμβων όπου αρκεί ο εντοπισμός ενός δεδομένου με βάση το όνομα για να ολοκληρωθεί η αναζήτηση. Μπορούμε να σχετίσουμε αυτές τις αναζητήσεις με την εργασία μας σαν ένα

ερώτημα τύπου SELECT, που αναζητάμε πλειάδες απλά με το κλειδί (id). Η πρώτη πλειάδα που εντοπίζεται και δεν έχει λήξει επιστρέφει σαν αποτέλεσμα. Αυτό συμβαίνει για να διαπιστωθεί κατά πόσο ο κόμβος που αναζητάει μια πλειάδα είναι σε θέση να την εντοπίσει ποιο είναι ο φόρτος δικτύου που επιφέρει η κάθε μέθοδος αναζήτησης καθώς και η απόδοση της.

Η αναζήτηση όπως περιγράφεται σε αυτό το κεφάλαιο μπορεί να αποτελέσει τη βάση για την υλοποίηση των ερωτημάτων του κεφαλαίου 3. Για παράδειγμα, για την υλοποίηση του ελαχίστου (MIN) με βάση κάποια συνθήκη μπορεί να χρησιμοποιηθεί είτε η αναζήτηση με τη μέθοδο της πλημμύρας είτε με τη μέθοδο των τυχαίων διαδρομών που αναλύεται στη συνέχεια.

Έστω ότι έχουμε *N* κόμβους στο δίκτυο και έστω ότι κάθε κόμβος έχει *d* συνδέσεις, δηλαδή επικοινωνεί με *d* γειτονικούς κόμβους. Το σύστημά μας είναι τυχαίο, αδόμητο, δυναμικό και μη κεντρικοποιημένο. Γίνεται όμως η σύμβαση ότι κάθε κόμβος έχει *d* γείτονες (*d*: μέσο πλήθος συνδέσεων). Έτσι όταν κάποιος κόμβος αναζητάει ένα δεδομένο x επικοινωνεί με όλους τους γείτονές του. Οι γείτονές του αναζητούν το δεδομένο x στην τοπική βάση δεδομένων και αν δεν υπάρχει τότε και αυτοί με τη σειρά τους επικοινωνούν με όλους τους γείτονές τους εκτός από τον γείτονα από τον οποίο έγινε η λήψη του μηνύματος. Καθώς συμβαίνει αυτό, υπάρχει ο κίνδυνος να δημιουργηθούν κύκλοι, δηλαδή κάποιος κόμβος να δεχθεί την αίτηση για το δεδομένο x πάνω από μία φορά. Όταν συμβεί κάτι τέτοιο, ο κόμβος που δέχεται το αίτηση πάνω από μια φορά σταματάει την προώθησή της. Τελικά, η αναζήτηση σταματάει όταν κάποιος κόμβος βρει το δεδομένο που αναζητείται ή όταν η αναζήτηση φτάσει σε κάποιο συγκεκριμένο βάθος από τον αρχικό κόμβο. Η παράμετρος TTL (time to live) δηλώνει το βάθος δικτύου μέχρι το οποίο φτάνει η αναζήτηση. Έστω TTL=*t*. Για οπτικοποίηση της αναζήτησης [10] θεωρούμε ότι κάθε κόμβος έχει d γείτονες που δεν αλληλεπικαλύπτονται, οπότε αν θεωρήσουμε τον κόμβο που αναζητάει το δεδομένο x ως ρίζα, η πορεία της αναζήτησης οπτικοποιείται σε δέντρο όπως φαίνεται στο σχήμα 5.1, με d=2.



Σχήμα 5.1 Οπτικοποίηση Αναζήτησης με τη Μέθοδο Flooding

Το δέντρο χωρίζεται σε t επίπεδα όσο είναι και το βάθος της ερώτησης TTL. Κάθε επίπεδο στο δέντρο αντικατοπτρίζει την προώθηση της ερώτησης από ένα κόμβο στους γείτονές του. Στο $i^{οστό}$ επίπεδο του δέντρου αριθμούνται d^i κόμβοι, οπότε και η αναζήτηση με πλημμύρα, θεωρητικά συναντάει d^i κόμβους σε κάθε επόμενο βήμα της. Γενικότερα με TTL=t συναντάμε $d^1+d^2+...+d^t$ κόμβους συνολικά.

Έστω τώρα ότι κάποιο δεδομένο x το έχουν n_x κόμβοι στο δίκτυο από τους N. Θέτοντας $a = n_x/N$, το ποσοστό των κόμβων που έχουν το δεδομένο x, σκοπός μας είναι να βρούμε το μικρότερο a έτσι ώστε η αναζήτηση να έχει καλή απόδοση με όσο το δυνατό μικρή παράμετρο *t* στο TTL.

Θεώρημα 5.1: Θεωρούμε ότι έχουμε ομοιόμορφη και τυχαία αναζήτηση δεδομένων [11] (uniform random search), όπου όλοι οι κόμβοι έχουν ίδια πιθανότητα να λάβουν μέρος στην αναζήτηση (η αναζήτηση είναι “τυφλή”). Τότε το προσδοκόμενο μέγεθος αναζήτησης (ESS-Expected Search Size [8]), που ορίζεται ως ο προσδοκόμενος αριθμός από κόμβους που πρέπει να ερωτηθούν έτσι ώστε η αναζήτηση κάποιου δεδομένου i, από τον κόμβο j, να είναι επιτυχής είναι:

$$ESS_{ij} = \frac{N}{n_i}$$

Απόδειξη:

Η πιθανότητα να βρεθεί το δεδομένο i από τον κόμβο j είναι:

$$P_{ij} = \frac{n_i}{N} = \mathrm{a}$$

Έτσι από γνωστό τύπο πιθανοτήτων το ESS_{ij} είναι:

$$ESS_{ij} = \sum_{k=1}^{\infty} k\mathrm{a}(1-\mathrm{a})^{\mathrm{k-1}} \Rightarrow$$

$$ESS_{ij} = \mathrm{a}(\sum_{\mathrm{k}=1}^{\infty} k(1-\mathrm{a})^{\mathrm{k-1}})$$

Όπου: $\sum_{\mathrm{k}=1}^{\infty} k(1-\mathrm{a})^{\mathrm{k-1}} = -\left[\sum_{k=1}^{\infty}(1-\mathrm{a})^{\mathrm{k}}\right]'$

Οπότε: $ESS_{ij} = \mathrm{a}(-\left[\sum_{k=1}^{\infty}(1-\mathrm{a})^{\mathrm{k}}\right]')$

$$ESS_{ij} = \mathrm{a}(-\left[\frac{1}{\mathrm{a}}\right]') = \mathrm{a}\frac{1}{\mathrm{a}^2} = \frac{1}{\mathrm{a}} = \frac{N}{n_i}$$

Και αποδείκτηκε το ζητούμενο.

Από το θεώρημα 5.1 διαπιστώνουμε ότι στο σύστημά μας κατά μέσο όρο πρέπει να συναντήσουμε 1/a κόμβους έτσι ώστε να βρούμε το αρχείο που αναζητείται.

Η μέθοδος της πλημμύρας, θεωρητικά, μπορεί να ρωτήσει $d^1+d^2+...+d^t$ κόμβους συνολικά [10] στην χειρότερη περίπτωση της αναζήτησης με TTL = *t*. Μπορούμε όμως εύκολα να παρατηρήσουμε ότι ποτέ δεν θα είναι τόσοι γιατί υπάρχουν αποτυχίες κόμβων, αρκετοί κύκλοι στην αναζήτηση, ή ακόμα και να βρεθεί το αρχείο νωρίτερα. Μπορεί όμως να εκτιμηθεί ότι το ESS με τη μέθοδο της πλημμύρας είναι ανάλογο του $d^1+d^2+...+d^t$. Άρα για να βρεθεί το δεδομένο x, πρέπει να ισχύει ότι το ESS με τη μέθοδο της πλημμύρας να είναι ανάλογο του 1/a:

$$d^1+d^2+...+d^t \sim 1/\mathrm{a} \quad (5.5)$$

Σκοπός στην πρώτη φάση του θεωρητικού μοντέλου είναι να σχετιστούν οι παράμετροι a,*t* με όσο δυνατόν μικρότερες τιμές έτσι ώστε να ικανοποιείται η (5.5). Ο πίνακας 5.1 παρουσιάζει συνοπτικά τα παραπάνω.

Πίνακας 5.1 Συγκεντρωτικός Πίνακας Συμβολισμών για τη Μέθοδο Flooding

Κόμβοι	N
Συνδέσεις (γείτονες)	d
TTL	t
Αναζήτηση με πλημμύρα	d^i κόμβοι σε κάθε βήμα
Αναζήτηση με πλημμύρα	$d^1+d^2+...+d^t$ κόμβοι συνολικά στη χειρότερη περίπτωση
Κόμβοι που έχουν το δεδομένο x	n_x
Ποσοστό κόμβων με το δεδομένο x	a ($=n_x/N$)
Πιθανότητα ένας κόμβος να έχει το δεδομένο x	a
Κατά μέσο όρο η πλημμύρα για να βρει το αρχείο πρέπει να ισχύει	$d^1+d^2+...+d^t \sim 1/a$

Στη συνέχεια θα αναφερθούμε στο κόστος της αναζήτησης με τη μέθοδο της πλημμύρας, όσο αφορά τα μηνύματα που χρειάζεται για την περάτωσή της και το φόρτο δικτύου που επιφέρει.

Μηνύματα: Ο αριθμός των μηνυμάτων που μεταδίδονται στο δίκτυο μέσω της αναζήτησης είναι: *d* μηνύματα από κάθε κόμβο στο δίκτυο καθώς προωθεί την αίτηση αναζήτησης στους *d* γείτονές του. Όπως παρατηρούμε και από τον πίνακα 5.1 τα μηνύματα που μεταδίδονται με TTL= *t* στη χειρότερη περίπτωση είναι $d+d^2+d^3+...+d^t$. Όταν βρεθεί το αρχείο ο κόμβος που το έχει στέλνει μήνυμα προς τα πίσω στον προηγούμενό του κόμβο ακολουθώντας το μονοπάτι της ερώτησης μέχρι η απάντηση να φθάσει στον αρχικό κόμβο ή στέλνονται μηνύματα τερματισμού από

τους τελικούς κόμβους της αναζήτησης προς τον αρχικό κόμβο. Τα μηνύματα αυτά είναι ίσα με τα βήματα της αναζήτησης για να φθάσει στα φύλλα του δέντρου, δηλαδή αν η αναζήτηση έκανε i βήματα τα μηνύματα είναι: $d+d^2+d^3+...+d^i$. Άρα συνολικά και στη χειρότερη περίπτωση που γίνονται *t* (=TTL) βήματα, στέλνονται $2d^1+2d^2+\ldots+2d^t$ μηνύματα στο δίκτυο. Η χειρότερη περίπτωση αναφέρεται στην εύρεση ή όχι του δεδομένου στο τελευταίο βήμα της αναζήτησης, έτσι ώστε η αναζήτηση (flooding) να κάνει *t* (=TTL) βήματα.

Φόρτος δικτύου: Έστω ότι κάθε κόμβος παράγει μια ερώτηση ανά c χρονικές μονάδες (γύρους) και έστω ότι στη χειρότερη περίπτωση όλες οι αναζητήσεις κάνουν t (=TTL) βήματα. Δηλαδή κάθε αναζήτηση παράγει $2d^1+2d^2+..+2d^t$ μηνύματα συνολικά στο δίκτυο στη χειρότερη των περιπτώσεων. Έτσι κάθε χρονική στιγμή έχουμε στο δίκτυο t/c αναζητήσεις ανά κόμβο με συνολικά μηνύματα που παράγουν $N(t/c)(2d^1+2d^2+..+2d^t)$ ανά t χρονικές μονάδες. Οπότε, κάθε χρονική στιγμή υπάρχουν στο δίκτυο $(N/c)(2d^1+2d^2+..+2d^t)$ μηνύματα κατανεμημένα σε όλους τους κόμβους. Τελικά έχουμε $(1/c)(2d^1+2d^2+..+2d^t)$ μηνύματα σε κάθε κόμβο του συστήματος (δηλαδή ο φόρτος εργασίας σε κάθε κόμβο).

Ο πίνακας 5.2 παρουσιάζει συνοπτικά τα μηνύματα και το φόρτο στο δίκτυο και στον κάθε κόμβο ξεχωριστά όπως αναλύθηκαν παραπάνω.

Πίνακας 5.2 Συγκεντρωτικός Πίνακας για τα Μηνύματα και το Φόρτο Εργασίας για τη Μέθοδο Flooding

Μηνύματα στο δίκτυο	$2d^1+2d^2+..+2d^t$ συνολικά στη χειρότερη περίπτωση
Συχνότητα παραγωγής αναζήτησης κάθε κόμβου	1/c αναζητήσεις 1 αναζήτηση ανά c χρονικές στιγμές
Αναζητήσεις στο δίκτυο κάθε χρονική στιγμή	t/c αναζητήσεις ανά κόμβο N*t/c συνολικά
Φόρτος εργασίας στο σύστημα κάθε χρονική στιγμή	$(N/c)*(2d^1+2d^2+..+2d^t)$ μηνύματα
Φόρτος εργασίας κάθε κόμβου κάθε χρονική στιγμή	$(1/c)(2d^1+2d^2+..+2d^t)$ μηνύματα

Απόδοση αναζήτησης: Για να γίνει μια εμπεριστατωμένη ανάλυση της απόδοσης της αναζήτησης με πλημμύρα, χρησιμοποιούμε τη σχέση (5.5) για να βρούμε τη συσχέτιση του TTL με το a, δηλαδή πόσο μεγάλο πρέπει να είναι το βάθος της ερώτησης για να βρεθεί ένα αρχείο που υπάρχει σε κάποιο κόμβο με πιθανότητα a. Έχουμε:

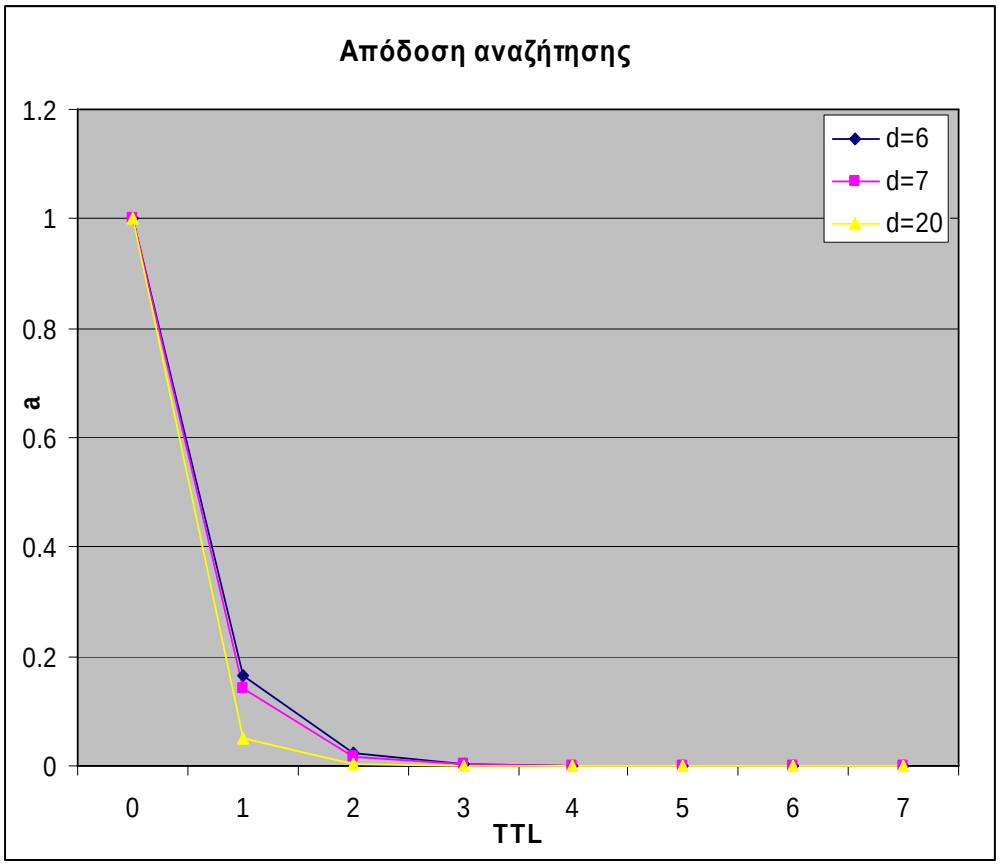
$$d^1 + d^2 + ... + d^t \cong \frac{1}{a}$$

Ο τύπος της γεωμετρικής προόδου μας δίνει:

$$\frac{d^t d - d}{d - 1} \cong \frac{1}{a} \Rightarrow$$

$$a \cong \frac{d - 1}{d^{t+1} - d}, t \neq 0 \qquad (5.6)$$

Το σχήμα 5.2 παρουσιάζει τις γραφικές παραστάσεις για τις διάφορες τιμές του TTL, από 0 έως 7 (όταν t=0 το a=1), το ποσοστό των κόμβων που μπορεί να έχουν το αρχείο που αναζητείται, αν υποθέσουμε ότι οι συνδέσεις σε κάθε κόμβο είναι d=6,7,20. Για να γίνει ανάλυση του τύπου (5.6) θεωρούμε ότι έχουμε ισότητα



Σχήμα 5.2 Συσχέτιση του TTL με το a (Flooding)

Από το σχήμα 5.2, παρατηρούμε ότι η αναζήτηση μας δίνει ικανοποιητικά αποτελέσματα χωρίς να απαιτείται μεγάλο d, παρόλο που η αναζήτηση προωθείται σε λίγους γείτονες κάθε φορά. Έτσι τα βήματα της αναζήτησης για ένα δημοφιλές αρχείο είναι λίγα (1-3), ενώ για ένα μη δημοφιλές είναι 4-7.

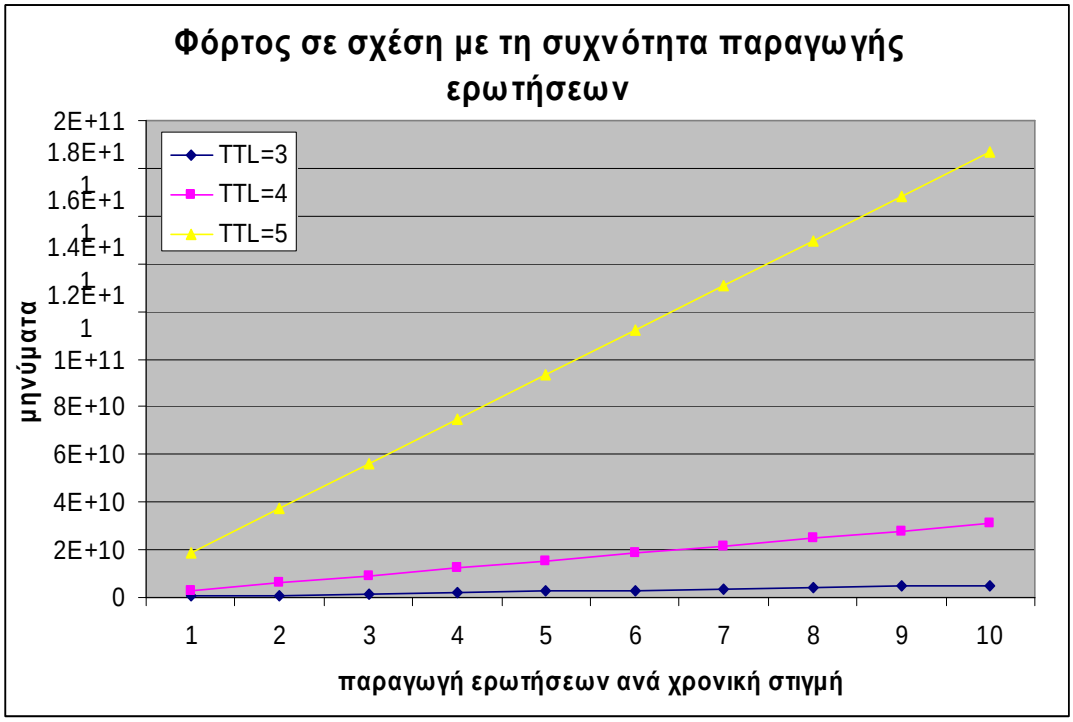
Για του λόγου το αληθές ο πίνακας 5.3 δίνει με αριθμούς το ποσοστό των κόμβων που πρέπει να διαθέτουν το αρχείο και το πλήθος των αντιγράφων που πρέπει να έχει αυτό για τις διάφορες τιμές του TTL, αν υποθέσουμε ότι d=6, $N=10^6$.

Πίνακας 5.3 Σχέση TTL και Πλήθος Αντιγράφων του Αρχείου

TTL	Ποσοστό κόμβων που έχουν το αρχείο	Πλήθος αντιγράφων του αρχείου στο δίκτυο
7	0.00001-0.00002	10-20
6	0.00002-0.00011	20-100
5	0.00011-0.000650	110-650
4	0.000650-0.0038	650-3880
3	0.0038-0.0238	3880-23810
2	0.0238-0.1667	23810-166700
1	0.1667-1	166700-10^6

Από τον παραπάνω πίνακα παρατηρούμε ότι, μια αναζήτηση με TTL από 1 έως 3 μπορεί να ανακτήσει ένα αρχείο με αντίγραφα έως 3880 που σημαίνει ότι το αρχείο είναι αρκετά δημοφιλές. Διαφορετικά το βάθος της ερώτησης πρέπει να είναι μεγαλύτερο. Βέβαια σε αυτούς τους υπολογισμούς δεν έχουν προβλεφθεί οι κύκλοι, και οι αποτυχίες κόμβων που είναι αρκετοί αλλά όχι προβλέψιμοι. Έτσι το βάθος δικτύου πρέπει να είναι μεγαλύτερο από το θεωρητικό κατά ένα παράγοντα που εξαρτάται από το σύστημα.

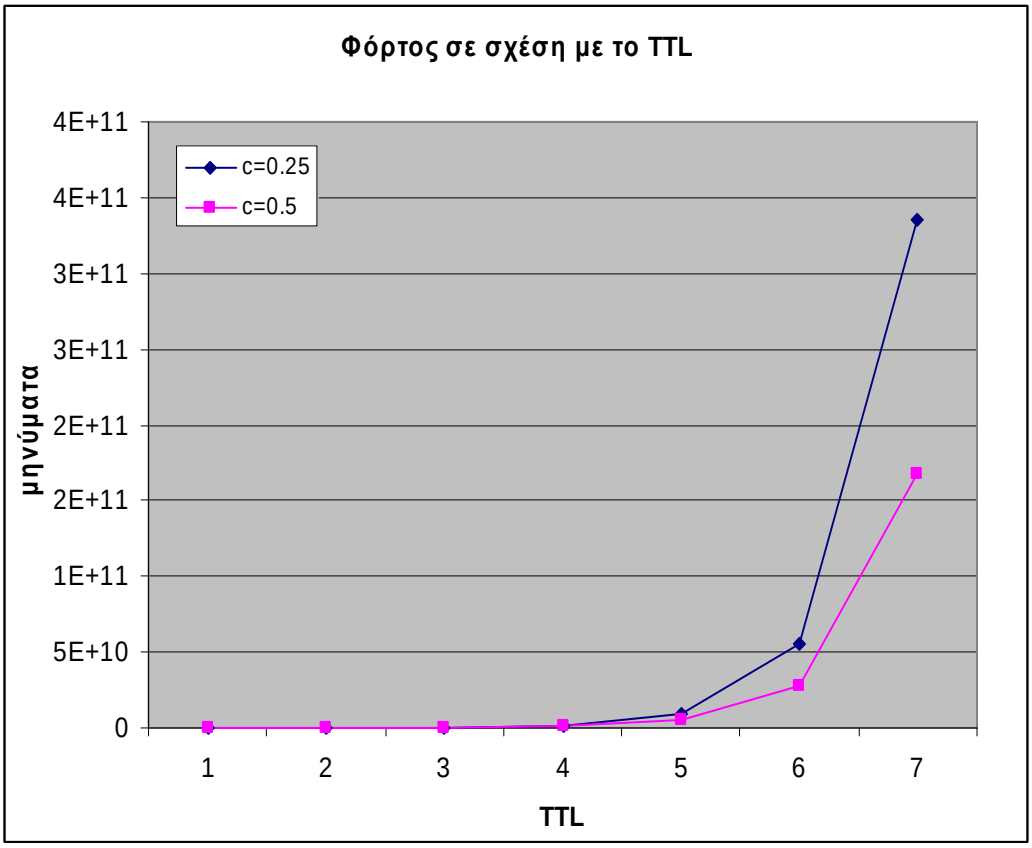
Φόρτος εργασίας: Όπως αναφέρθηκε παραπάνω ο φόρτος εργασίας σε μηνύματα όλων των κόμβων του δικτύου για τις διάφορες τιμές του TTL είναι $(N/c)*(2d^1+2d^2+..+2d^t)$ μηνύματα. Το σχήμα 4.3 δίνει τις γραφικές παραστάσεις για διάφορες τιμές του TTL (θέτουμε TTL=3,4,5), πώς σχετίζεται το c, δηλαδή κάθε πόσες χρονικές στιγμές γίνεται μια ερώτηση από κάθε κόμβο, με τα μηνύματα που παράγονται στο δίκτυο. Θεωρούμε ότι $N= 10^6$ κόμβοι, d=6 και c=1,1/2,1/3,1/4,…,1/10, δηλαδή 1 αναζήτηση κάθε χρονική στιγμή μέχρι 10 αναζητήσεις ανά χρονική στιγμή.



Σχήμα 5.3 Φόρτος Εργασίας σε Σχέση με τη Συχνότητα Παραγωγής Ερωτήσεων (Flooding)

Από το σχήμα 5.3 παρατηρούμε ότι τα μηνύματα που παράγονται αυξάνονται γραμμικά, ανάλογα με τη συχνότητα παραγωγής ερωτήσεων από τους κόμβους.

Στο σχήμα 5.4 παραθέτουμε το διάγραμμα των μηνυμάτων που παράγονται στο δίκτυο ανάλογα με το TTL, για παραγωγή αναζητήσεων σε κάθε κόμβο της τάξης των 2 αναζητήσεων ανά χρονική στιγμή (c=0.5) και 4 αναζητήσεων ανά χρονική στιγμή (c=0.25).



Σχήμα 5.4 Φόρτος Εργασίας σε Σχέση με το TTL (Flooding)

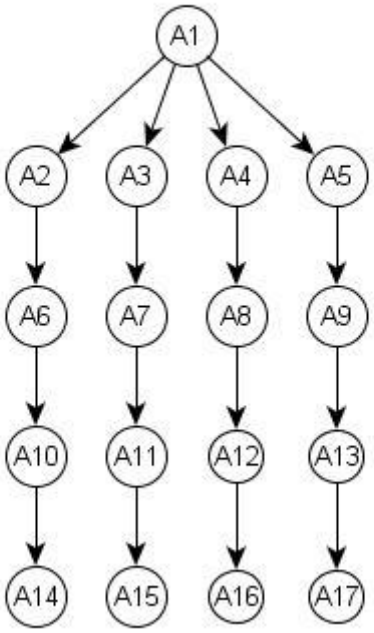
Από το παραπάνω διάγραμμα παρατηρούμε ότι τα μηνύματα αυξάνονται εκθετικά σύμφωνα με το TTL.

Τελικά παρατηρούμε, ότι όσο αφορά τα μηνύματα παραγωγής, μεγαλύτερο κόστος υπάρχει όταν αυξάνεται το TTL παρά όταν γίνονται πολλές αναζητήσεις ανά κόμβο κάθε χρονική στιγμή και αυτό γιατί τα μηνύματα αυξάνονται εκθετικά σε σχέση με το TTL, ενώ αυξάνονται γραμμικά σε σχέση με την συχνότητα παραγωγής ερωτήσεων. Έτσι, στόχος είναι να αυξήσουμε τη συχνότητα παραγωγής της ίδιας ερώτησης αν δεν έχουμε τα επιθυμητά αποτελέσματα, παρά να αυξήσουμε το βάθος αναζήτησης TTL. Ανάλογο φόρτο εργασίας έχει και ο κάθε κόμβος χωριστά, αφού η διαφορά στα μηνύματα του κόμβου σε σχέση με το δίκτυο είναι της τάξης του 1/N.

5.2. Θεωρητικό Μοντέλο Αναζήτηση με Τυχαίες Διαδρομές

Στην παρούσα ενότητα θα γίνει θεωρητική ανάλυση της αναζήτησης με τη μέθοδο των τυχαίων διαδρομών (random walks). Για την θεωρητική ανάλυση παρουσιάζονται οι διαφορές της μεθόδου από την αναζήτηση με πλημμύρα.

Σύμφωνα με αυτή τη μέθοδο των τυχαίων διαδρομών ο κόμβος υποβάλει μια ερώτηση σε k τυχαία επιλεγμένους γείτονές του. Αυτά τα μηνύματα ονομάζονται walkers και το καθένα ακολουθεί τυχαία τη δική του διαδρομή όπου κάθε κόμβος τα προωθεί σε ένα τυχαίο γείτονά του σε κάθε βήμα. Η αναζήτηση σταματάει όταν κάποιος κόμβος βρει το δεδομένο που αναζητείται ή όταν η αναζήτηση φτάσει σε κάποιο βάθος από τον αρχικό κόμβο (παράμετρος TTL). Για οπτικοποίηση της αναζήτησης θεωρούμε ότι ο αρχικός κόμβος προωθεί το μήνυμα σε k γείτονες του και κάθε κόμβος το προωθεί σε 1 γείτονα. Αν θεωρήσουμε τον κόμβο που αναζητάει το δεδομένο x σαν ρίζα, η πορεία της αναζήτησης οπτικοποιείται σε δέντρο όπως φαίνεται στο σχήμα 5.5, με k=4, TTL=4.



Σχήμα 5.5 Οπτικοποίηση Αναζήτησης με τη Μέθοδο των Τυχαίων Διαδρομών

Η διαφορά της μεθόδου αυτής είναι ότι σε κάθε επίπεδο του δέντρου αριθμούνται ακριβώς k κόμβοι, οπότε σε κάθε επόμενο βήμα της αναζήτησης συναντιούνται ακριβώς k κόμβοι. Πλέον οι κόμβοι είναι σταθεροί σε κάθε βήμα της αναζήτησης και δεν αυξάνονται εκθετικά όπως με τη μέθοδο της πλημμύρας. Γενικά για TTL=t, συναντάμε, θεωρητικά, k+k+…+k κόμβους ή t*k κόμβους (εκτός του αρχικού).

Κατ' αντιστοιχία με τη σχέση (5.5) για να ανακτήσουμε ένα αρχείο x που το διαθέτει κάποιος κόμβος με πιθανότητα a πρέπει να ισχύει:

$$t*k \sim 1/a \qquad (5.7)$$

Αντίστοιχα βρίσκουμε και τις άλλες σχέσεις που ισχύουν στη μέθοδο των τυχαίων διαδρομών και παρουσιάζονται στο πίνακα 5.4.

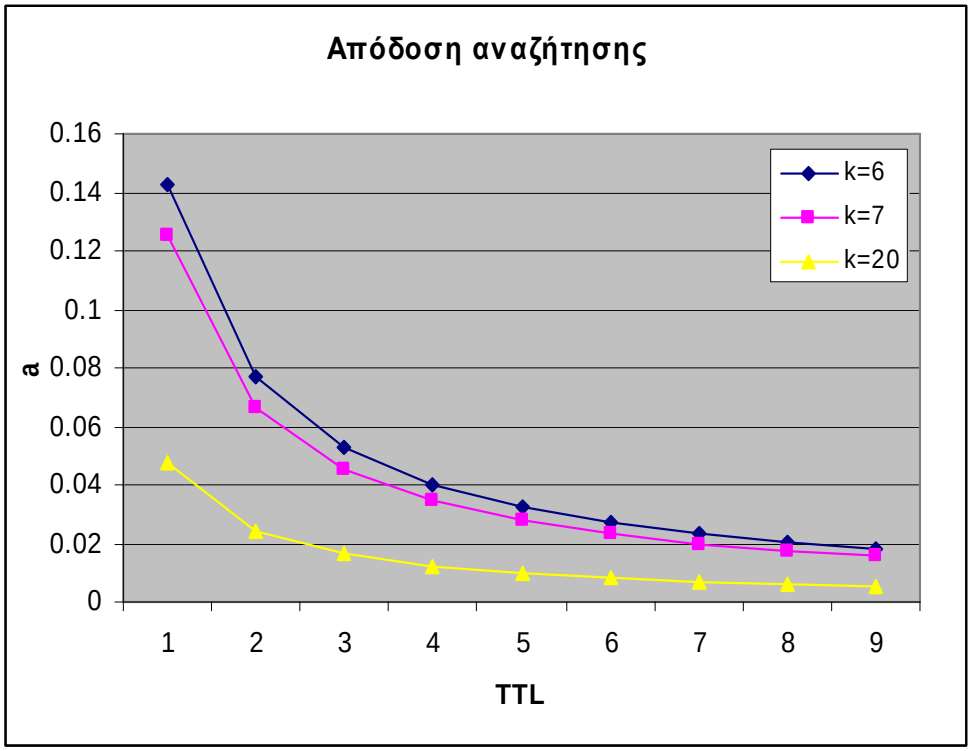
Πίνακας 5.4 Συγκεντρωτικός Πίνακας για τη Μέθοδο Random Walks

Αναζήτηση με τη μέθοδο των τυχαίων διαδρομών	k κόμβοι σε κάθε βήμα
	t*k κόμβοι συνολικά
Κόμβοι που διαθέτουν το δεδομένο x	n_x κόμβοι
Ποσοστό κόμβων με το δεδομένο x	a(=n_x/N)
Πιθανότητα ένας κόμβος να διαθέτει το x	a
Προσδοκόμενος αριθμός βημάτων	t*k~1/a
Μηνύματα στο δίκτυο	2*t*k συνολικά στη χειρότερη περίπτωση
Συχνότητα παραγωγής αναζήτησης κάθε κόμβου	1/c αναζητήσεις 1 ανά c χρονικές στιγμές
Αναζητήσεις στο δίκτυο κάθε χρονική στιγμή	t/c αναζητήσεις ανά κόμβο N*t/c συνολικά
Φόρτος εργασίας στο σύστημα κάθε χρονική στιγμή	(N/c)*(2*t*k) μηνύματα
Φόρτος εργασίας ανά κόμβο κάθε χρονική στιγμή	(1/c)*(2*t*k) μηνύματα

Απόδοση αναζήτησης: Αντίστοιχα με την απόδοση αναζήτησης με τη μέθοδο της πλημμύρας, στη μέθοδο των τυχαίων διαδρομών πρέπει να ισχύει:

$$k + k + ... + k \cong \frac{1}{a} \Rightarrow t * k \cong \frac{1}{a} \Rightarrow a \cong \frac{1}{t * k} \quad (5.8)$$

Το σχήμα 5.6 παρουσιάζει τις γραφικές παραστάσεις για τις διάφορες τιμές του TTL, από 0 έως 9 (όταν t=0 το a=1), το ποσοστό των κόμβων που μπορεί να έχουν το αρχείο που αναζητείται (a), αν υποθέσουμε ότι οι k=6,7,20, όσο και το d για ευκολότερη σύγκριση. (Θεωρούμε ότι στον τύπο 5.8 έχουμε ισότητα)

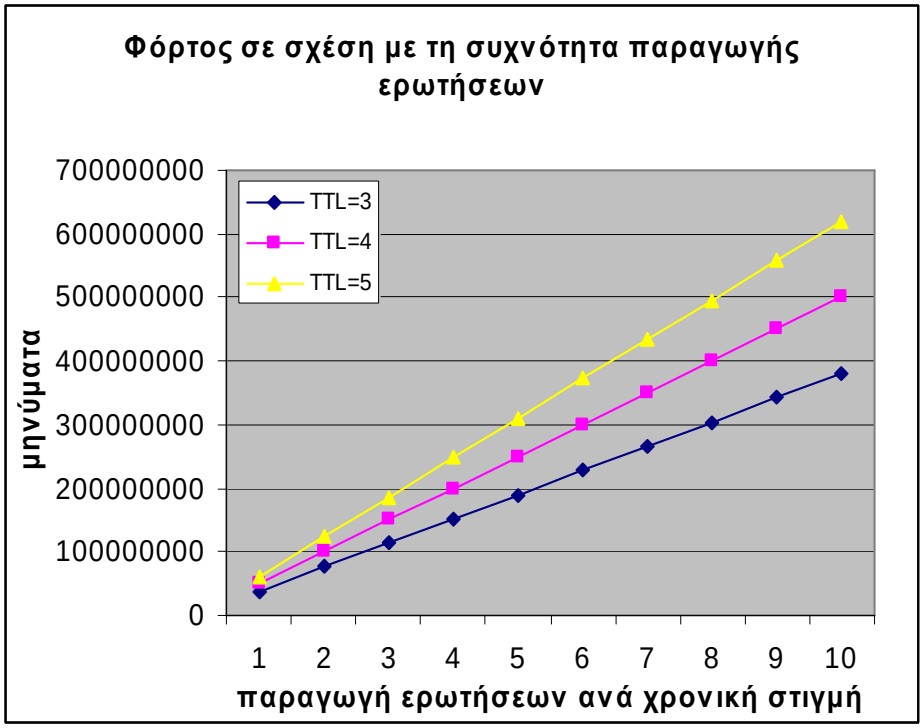


Σχήμα 5.6 Συσχέτιση του TTL με το a (Τυχαίες Διαδρομές)

Από το σχήμα 5.6 παρατηρούμε ότι πλέον οι συνδέσεις (το k) παίζει ρόλο στο τελικό αποτέλεσμα. Οι κόμβοι που συναντά η αναζήτηση δεν είναι εκθετικά αυξανόμενοι αλλά ανάλογοι με το k. Άρα όσο μεγαλύτερο k τόσο καλύτερα αποτελέσματα. Τέλος η γραφική παράσταση τείνει πολύ αργά στο 0, πράγμα που σημαίνει ότι και με μεγάλο TTL δεν είμαστε σίγουροι ότι το αρχείο θα βρεθεί. Η διαφορά είναι ότι με μεγάλο TTL οι κόμβοι που συναντάμε είναι πολλές φορές λιγότεροι σε πλήθος από αυτούς που συναντάμε με το ίδιο TTL με τη μέθοδο της πλημμύρας.

Φόρτος εργασίας: Ο φόρτος εργασίας σε μηνύματα όλων των κόμβων του δικτύου για τις διάφορες τιμές του TTL είναι (N/c)*(2*t*k) μηνύματα. Το σχήμα 5.7 δίνει τις

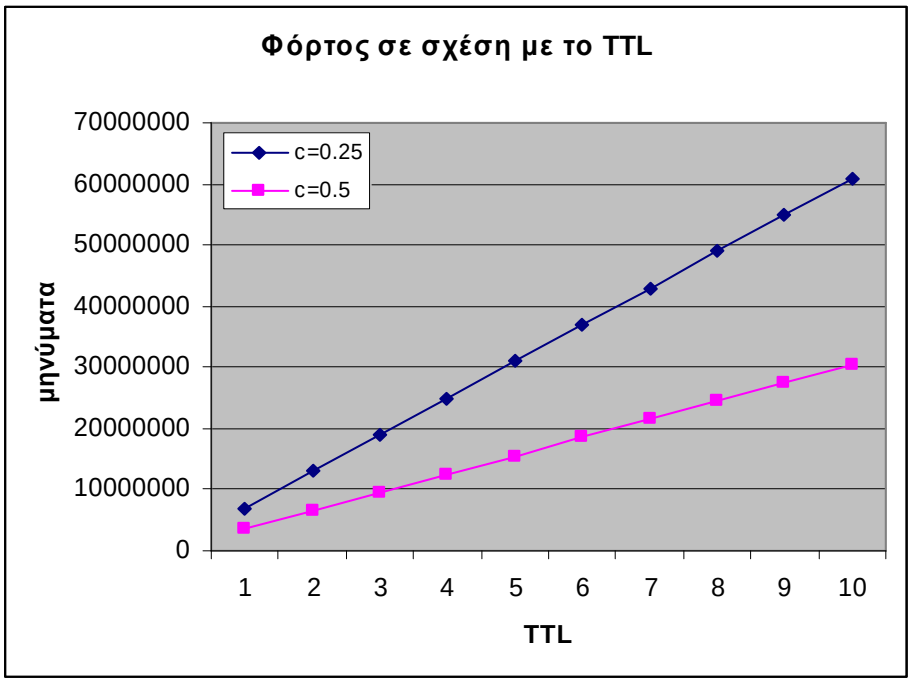
γραφικές παραστάσεις για διάφορες τιμές του TTL (θέτουμε TTL=3,4,5), πως σχετίζεται το c με τα μηνύματα που παράγονται στο δίκτυο. Θεωρούμε ότι N= 10^6 κόμβοι, k=6 και c=1,1/2,1/3,1/4,…,1/10, δηλαδή 1 αναζήτηση κάθε χρονική στιγμή μέχρι 10 αναζητήσεις ανά χρονική στιγμή.



Σχήμα 5.7 Φόρτος Εργασίας σε Σχέση με τη Συχνότητα Παραγωγής Ερωτήσεων (Random Walks)

Από το σχήμα 5.7 παρατηρούμε ότι τα μηνύματα που παράγονται αυξάνονται γραμμικά, ανάλογα με τη συχνότητα παραγωγής ερωτήσεων από τους κόμβους.

Στο σχήμα 5.8 παραθέτουμε το διάγραμμα των μηνυμάτων που παράγονται στο δίκτυο ανάλογα με το TTL, για παραγωγή αναζητήσεων σε κάθε κόμβο της τάξης των 2 αναζητήσεων ανά χρονική στιγμή (c=0.5) και 4 αναζητήσεων ανά χρονική στιγμή (c=0.25).



Σχήμα 5.8 Φόρτος Εργασίας σε Σχέση με το TTL (Random Walks)

Από το παραπάνω διάγραμμα παρατηρούμε ότι τα μηνύματα αυξάνονται γραμμικά σε σχέση με το TTL και αυτό αποτελεί τη κυριότερη διαφορά με τη μέθοδο της πλημμύρας.

Έτσι παρατηρώντας τα σχήματα 4.7 και 4.8 όσο αφορά τα μηνύματα παραγωγής, έχουμε ίδιο κόστος ανεξάρτητα αν αυξάνουμε το TTL ή αν γίνονται πολλές αναζητήσεις ανά κόμβο κάθε χρονική στιγμή, με τη μέθοδο random walks.

Διαφορές των δύο μεθόδων αναζήτησης: Οι διαφορές των δύο μεθόδων αναζήτησης (flooding, random walks) μπορούν να συνοψιστούν στις παρακάτω:

- Με τη πλημμύρα συναντάμε εκθετικά αυξανόμενους κόμβους σε κάθε βήμα της αναζήτησης ενώ τις τυχαίες διαδρομές σταθερούς
- Με τη πλημμύρα τα μηνύματα αυξάνονται εκθετικά σε σχέση με το βάθος της αναζήτησης ενώ με τις τυχαίες διαδρομές γραμμικά

- Όσο αυξάνει το βάθος της αναζήτησης, η πλημμύρα μας δίνει περισσότερα αποτελέσματα (αυξάνεται η απόδοση αναζήτησης δηλαδή) από τις τυχαίες διαδρομές γιατί συναντάμε περισσότερους κόμβους
- Με τη πλημμύρα έχουμε πολύ περισσότερα μηνύματα στο δίκτυο όσο αυξάνει η συχνότητα παραγωγής ερωτήσεων
- Ο φόρτος εργασίας με τη πλημμύρα είναι πολύ μεγαλύτερος από ότι με τις τυχαίες διαδρομές με ίδιο TTL
- Γενικά με σταθερό TTL, η πλημμύρα δίνει καλύτερα και περισσότερα αποτελέσματα, αλλά έχουμε πολύ μεγάλο κόστος μηνυμάτων στο δίκτυο
- Με την πλημμύρα συμφέρουν (ως προς το κόστος μηνυμάτων) πολλές αναζητήσεις με μικρό TTL αντί για λίγες με μεγάλο TTL, ενώ στο random walks έχουμε το ίδιο κόστος

5.3. Απόδοση των Πρωτοκόλλων Ενημέρωσης και Δημιουργίας Αντιγράφων

Στην παρούσα ενότητα θα αναλύσουμε τα πρωτόκολλα ενημέρωσης και δημιουργίας αντιγράφων που παρουσιάστηκαν σε προηγούμενο κεφάλαιο και τι επίπτωση έχουν αυτά στο δίκτυο (μηνύματα, απόδοση κλπ), κυρίως ως προς την ποιότητα δεδομένων και υπηρεσιών.

- UP0: Όταν ένα item λήξει, απλά δεν γίνεται τίποτα

Σε αυτό το πρωτόκολλο όπως αναλύθηκε και σε προηγούμενο κεφάλαιο, δεν ανανεώνονται τα ληγμένα δεδομένα, ούτε και αποβάλλονται από την βάση δεδομένων. Το συγκεκριμένο πρωτόκολλο έχει εξαιρετικά χαμηλής απόδοσης ποιότητα δεδομένων και υψηλή ποιότητα υπηρεσιών (δεν υπάρχουν μηνύματα που επιβαρύνουν το δίκτυο, ούτε βήματα για το επιθυμητό αποτέλεσμα).

- UP1: Όταν ένα item λήξει, ο κόμβος το διαγράφει από τη βάση δεδομένων του

Το συγκεκριμένο πρωτόκολλο δεν βοηθάει στην ενημέρωση των πλειάδων, αλλά συμβάλλει στην απομάκρυνση των “απαρχαιωμένων” πλειάδων από το σύστημα. Δεν έχουμε νέο ερώτημα σε αυτό το πρωτόκολλο, άρα ούτε βήματα (hops), ούτε επιπλέον μηνύματα στο δίκτυο. Μπορεί, όμως, το πρωτόκολλο αυτό να αποτελέσει μια βάση για την μελέτη της απόδοσης των υπόλοιπων πρωτοκόλλων στο δίκτυο. Έτσι ως προς την ποιότητα των δεδομένων το πρωτόκολλο δεν προσφέρει καλύτερη φρεσκάδα (Freshness) ούτε και ανάκληση αποτελεσμάτων (Recall. Ως προς την ποιότητα υπηρεσιών παραμένει υψηλή, όπως το UP0, και το Response Time και το Message Overhead είναι μηδενικά.

- UP2: Όταν ένα item λήξει, ο κόμβος ψάχνει με νέο ερώτημα για item που δεν έχει ακόμα λήξει

Το πρωτόκολλο αυτό εντάσσεται στα πρωτόκολλα ενημέρωσης των αντιγράφων διότι δεν δημιουργούμε νέα αντίγραφα αλλά ενημερώνουμε τα αντίγραφα που υπάρχουν αλλά είναι “απαρχαιωμένα”. Το πρωτόκολλο, αν συνδυαστεί και με το UP1 μπορεί όχι μόνο να αποβάλει τα ληγμένα δεδομένα από τη βάση δεδομένων του κόμβου, αλλά και να ανανεώνει τα δεδομένα που έχουν λήξει, ψάχνοντας για νέα, φρέσκα δεδομένα στο δίκτυο με νέο ερώτημα. Βέβαια, επειδή γίνεται νέα ερώτηση, υπάρχει και το ανάλογο κόστος που εδώ είναι αρκετά μεγάλο λόγω του κόστους της αναζήτησης. Το θέμα είναι να διαπιστωθεί κατά πόσο είναι συμφέρον να γίνεται η διαδικασία αυτή σε σχέση με ερωτήματα που θα γίνονται αν δεν υπήρχε αυτό το πρωτόκολλο.

Επειδή η βάση δεδομένων του κόμβου ενδέχεται να είναι αρκετά μεγάλη, το παρών πρωτόκολλο παράγει αρκετές ερωτήσεις ανά χρονική στιγμή. Το πρόβλημα εδώ είναι ο αριθμός των μηνυμάτων και τα βήματα (hops) της ερώτησης στο δίκτυο. Για να τα ελαττώσουμε αρκεί να μειώσουμε το βάθος της αναζήτησης, γιατί όπως αναλύθηκε παραπάνω τα μηνύματα αυξάνονται εκθετικά ως προς το TTL ενώ αυξάνονται γραμμικά ως προς τη συχνότητα των ερωτήσεων (για flooding). Δεδομένου ότι η

συχνότητα παραγωγής των ερωτήσεων είναι μεγάλη μειώνουμε το TTL. Για να έχουμε “φρέσκα” δεδομένα αρκεί το δεδομένο να το έχουν αρκετοί (και όχι πάρα πολλοί) κόμβοι. Έτσι αν το TTL=3, τότε για να βρεθεί αρκεί να το προσφέρουν 3880-23810 κόμβοι στους 10^6 συνολικά και αυτό έχει επιβάρυνση της τάξης του $O(d^3)$ μηνύματα σε όλο το δίκτυο (~474). Παρόλο που το TTL είναι μικρό πρέπει να για να βρεθεί το αρχείο πρέπει να το προσφέρουν ένα μικρό ποσοστό κόμβων (~1.5%) αλλά αν το ίδιο πρωτόκολλο χρησιμοποιηθεί από όλους τους κόμβους η αναζήτηση θα είναι σε ένα βαθμό αποδοτικότερη.

Τελικά για το QoS:
Response time: Δεδομένου ότι το πρωτόκολλο το εφαρμόζουν όλοι οι κόμβοι τα βήματα θα είναι λίγα 3-4 (ή και περισσότερα δεδομένου ότι θα υπάρχουν αποτυχίες) .
Message overhead: ο συνολικός αριθμός μηνυμάτων θα είναι $N*O(d^3)$ για όλους τους κόμβους ανά ερώτηση, αν η αναζήτηση γίνεται με πλημμύρα

Για το QoD:
Freshness: Ο μέσος όρος της ημερομηνίας λήξης του επιστρεφόμενου αποτελέσματος θα είναι αρκετά καλός γιατί με αυτό τον τρόπο διαδίδονται τα μη-ληγμένα δεδομένα αρκετά γρήγορα
Recall: Το ποσοστό των αποτελεσμάτων που επιστράφηκαν σε σχέση με αυτά που θα έπρεπε να επιστραφούν, εξαρτάται από το πόσο δημοφιλές είναι το δεδομένο. Ένα όχι και τόσο δημοφιλές δεδομένο, αν δεν έχει διαδοθεί δεν θα επιστραφεί γρήγορα.

Για να διαπιστώσουμε κατά πόσο είναι συμφέρον ένα τέτοιο πρωτόκολλο αναλογιζόμαστε τα εξής: Για κάθε δεδομένο που αναζητείται, έστω ότι γίνεται αναζήτηση με πλημμύρα με TTL=3. Αυτό επιφέρει φόρτο στο δίκτυο της τάξης του $O(d^3)$ μηνύματα. Έστω ότι για το ίδιο δεδομένο γίνονται αναζητήσεις από ένα πλήθος κόμβων a. Άρα τα μηνύματα είναι $O(ad^3)$. Κάτι τέτοιο όμως εκτός των μηνυμάτων που φορτώνει το δίκτυο, έχει το θετικό ότι αυξάνει τη διαθεσιμότητα του δεδομένου αρκετά καλά. Αν έλλειπε το πρωτόκολλο, ένας κόμβος για να ψάξει το δεδομένο θα χρειαζόταν μια μεγαλύτερου βάθους αναζήτηση της τάξης του TTL=5. Αυτό θα είχε σαν φόρτο $O(d^5)$ μηνύματα. Τα μηνύματα αυτά είναι πολύ περισσότερα αν αναλογιστούμε ότι $\alpha << d^2$. Αυτό συμβαίνει γιατί όσο αυξάνεται η διαθεσιμότητα του

δεδομένου τόσο μικρότερο TTL χρειάζεται η αναζήτηση για να το ανακτήσει. Όπως αναλύθηκε παραπάνω, πολλές αναζητήσεις μικρού TTL συμφέρουν περισσότερο από μια αναζήτηση μεγαλύτερου TTL. Άρα το παραπάνω πρωτόκολλο υπό τέτοιες συνθήκες πετυχαίνει και διαθεσιμότητα και λιγότερα μηνύματα στο δίκτυο σε βάθος χρόνου.

Στην περίπτωση όμως όπου η αναζήτηση γίνεται με τυχαίες διαδρομές, όσο μεγάλου βάθους και να είναι επιφέρει φόρτο σε μηνύματα της τάξης του O(d). Έτσι το πρωτόκολλο δεν απαλλάσσει το δίκτυο από μηνύματα αλλά έχει δύο πλεονεκτήματα. Το πρώτο είναι ότι αυξάνει τη διαθεσιμότητα του δεδομένου (πολύ καλύτερο recall), καθώς και δίνει καλύτερη εξισορρόπηση φόρτου εργασίας (load balance). Αυτό συμβαίνει, γιατί οι κόμβοι που θα ενεργοποιήσουν το πρωτόκολλο θα είναι κατά βάση διασκορπισμένοι στο δίκτυο οπότε θα υπάρχει ισορροπία στο φόρτο του κάθε κόμβου ξεχωριστά από το να γίνει η ερώτηση από ένα κόμβο στο δίκτυο με μεγάλο TTL όπου φορτώνονται οι κόμβοι που βρίσκονται σε αυτή την περιοχή συνεχώς.

- UP2-Lazy: Για κάθε n βήματα ο κόμβος αρχικοποιεί το update protocol για όλα τα ληγμένα items στην τοπική βάση δεδομένων του

Όπως αναλύθηκε και σε προηγούμενο κεφάλαιο το πρωτόκολλο UP2-Lazy, έχει την ίδια επίδραση με τα υπόλοιπα πρωτόκολλα, ανάλογα με ποιο συνδυάζεται, όσο αφορά τα μηνύματα και τα βήματα (hops) στο δίκτυο και την απόδοση αναζήτησης. Η μόνη διαφορά του είναι ότι δεν αρχικοποιείται το update πρωτόκολλο αμέσως μόλις λήξει κάποιο item, αλλά γίνεται έλεγχος κάθε n χρονικές στιγμές. Αν λοιπόν ρυθμιστεί σωστά το παραπάνω πρωτόκολλο θα έχει τα ίδια αποτελέσματα που με το πρωτόκολλο ενημέρωσης που συνδυάζει με τη διαφορά ότι αρκετές φορές τα μηνύματα θα μειώνονται, αλλά αυτό εξαρτάται από διάφορους παράγοντες που θα αναλυθούν και πειραματικά οπότε δεν χρειάζεται περαιτέρω ανάλυση.

- Ανάλυση των push πρωτοκόλλων

Στην μελέτη μας έχουμε συμπεριλάβει τρία πρωτόκολλα push που εντάσσονται στην δημιουργία αντιγράφων. Ένα push πρωτόκολλο δεν ενημερώνει, αλλά δημιουργεί

αντίγραφα ενημερωμένα κατά μήκος του δικτύου, ώστε να συμβάλλει στη γρήγορη διάδοσή τους αλλά και στην φρεσκάδα της κατανεμημένης βάσης δεδομένων.

Καθώς έχει γίνει ανάλυση των πρωτοκόλλων αυτών, σκοπός εδώ είναι να αναλυθεί τη επίπτωση έχουν αυτά στο δίκτυο (μηνύματα και απόδοση) κυρίως ως προς την ποιότητα υπηρεσιών και δεδομένων. Επειδή όμως και τα τρία έχουν την ίδια επίπτωση στο δίκτυο, στην ενότητα αυτή μελετούνται συνολικά.

Τα πρωτόκολλα push έχουν για την ποιότητα υπηρεσιών (QoS):

Response time: Χρειάζεται ένα βήμα μόνο για να ολοκληρωθεί η διαδικασία push.
Message overhead: Για κάθε δεδομένο που γίνεται push, χρειάζεται ένα μήνυμα για κάθε γείτονα που προωθείται. Έτσι αν το δεδομένο προωθηθεί σε ένα γείτονα απαιτείται ένα μήνυμα, ενώ αν γίνεται σε όλους τους d γείτονές του απαιτούνται d μηνύματα συνολικά.

Για την ποιότητα δεδομένων (QoD):

Freshness: Δεν υπάρχει επιστρεφόμενο αποτέλεσμα άρα δεν μπορεί κάτι τέτοιο να μετρηθεί, αλλά βοηθάει στη φρεσκάδα της βάσης δεδομένων ολόκληρου του δικτύου.
Recall: Ομοίως δεν μπορεί να εκτιμηθεί διότι δεν υπάρχει επιστρεφόμενο αποτέλεσμα. Βοηθάει όμως συνολικά στο επιστρεφόμενο αποτέλεσμα των αναζητήσεων στο δίκτυο, καθώς αυξάνεται ο αριθμός των αντιγράφων και μειώνεται σημαντικά το βάθος και τα μηνύματα της αναζήτησης.

Από τα παραπάνω διαπιστώνεται ότι τα push πρωτόκολλα με λίγα μηνύματα ανά δεδομένο που γίνεται push, βοηθάνε στην φρεσκάδα της κατανεμημένης βάσης δεδομένων και επιπλέον συμβάλλουν στην γρηγορότερη διάδοση κάποιου νέου δεδομένου, έτσι ώστε οι κόμβοι να το βρίσκουν ευκολότερα. Τέλος πρέπει να επισημανθεί ότι με τα push πρωτόκολλα μειώνονται αισθητά τα μηνύματα που μεταδίδονται στο δίκτυο κατά τη διάρκεια κάποιου πρωτοκόλλου ενημέρωσης αντιγράφων, καθώς και των αναζητήσεων από κόμβους με τις μεθόδους που αναλύσαμε παραπάνω. Αυτό γίνεται γιατί αν θεωρηθεί ότι κάποιος κόμβος αναζητάει

ένα δεδομένο που το βρίσκει σε i βήματα, πλέον μπορεί να το βρει σε αρκετά λιγότερα βήματα. Το όφελος είναι μεγάλο γιατί έχουμε περισσότερα αντίγραφα των ενημερωμένων δεδομένων οπότε και ανοχή σφαλμάτων (fault tolerance), όπου συγκαταλέγονται οι αποτυχίες κόμβων, κύκλοι αναζήτησης κ.α.

Έστω ότι αναζητούμε κάποιο δεδομένο με τη μέθοδο της πλημμύρας. Για να διαπιστώσουμε το μέγεθος του κέρδους μηνυμάτων που έχουμε από ένα και μόνο βήμα της αναζήτησης, δηλαδή αντί να μεταδοθούν d^i μηνύματα στο τελευταίο βήμα της αναζήτησης η αναζήτηση να σταματάει στο βήμα d^{i-1}, αρκεί να παρατηρήσουμε τον πίνακα 5.5. Ο πίνακας 5.5 παρουσιάζει το κέρδος των μηνυμάτων ανά βήμα της αναζήτησης όπου το κέρδος είναι d^i και το κόστος του ενδεχόμενου push είναι d.

Πίνακας 5.5 Κέρδος σε μηνύματα ανάλογα με το TTL της αναζήτησης

TTL	ΚΕΡΔΟΣ ΣΕ ΜΗΝΥΜΑΤΑ (d^i-d , d=6)
1	0
2	30
3	210
4	1290
5	7770
6	46650
7	279930

Από τον παραπάνω πίνακα διαπιστώνουμε το μεγάλο κέρδος που έχουμε με το πρωτόκολλα push όσο αφορά το κέρδος σε μηνύματα. Έτσι, τα πρωτόκολλα push έχουν υψηλής απόδοσης ποιότητα υπηρεσιών, και παρότι δεν μπορεί να γίνει ανάλυση ως προς την ποιότητα δεδομένων, είναι σίγουρο ότι συμβάλλουν στο συνολικό freshness, recall του συστήματος σε ενδεχόμενες ερωτήσεις για δεδομένα από τους κόμβους.

Σημειώνουμε ότι το κέρδος σε μηνύματα όταν η αναζήτηση γίνεται με τη μέθοδο των τυχαίων διαδρομών, είναι μικρότερο γιατί σε κάθε βήμα της μεταδίδονται d μηνύματα και όχι d^i όπως προηγουμένως. Επομένως αν μειωθούν τα βήματα κατά ένα παράγοντα s το κέρδος θα είναι s*d σαφώς λιγότερο από d^i. Υπάρχουν όμως τα

επιπλέον πλεονεκτήματα όπως παρουσιάστηκαν και παραπάνω, όπως ότι αυξάνεται η διαθεσιμότητα των δεδομένων στο δίκτυο (πολύ καλύτερο recall), ανοχή σε σφάλματα (fault tolerance) καθώς και υπάρχει καλύτερη εξισορρόπηση φόρτου εργασίας (load balance) και δεν υπερφορτώνονται ορισμένοι μόνο κόμβοι στο δίκτυο.

- Ανάλυση των pull πρωτοκόλλων

Το ίδιο κέρδος μηνυμάτων έχουμε και στα πρωτόκολλα pull καθώς η φιλοσοφία παραμένει η ίδια. Στην συγκεκριμένη περίπτωση χρειάζονται (για το message overhead) δύο βήματα για να ολοκληρωθεί η διαδικασία pull για κάθε δεδομένο σε ένα γειτονικό κόμβο. Ένα μήνυμα για αίτηση και ένα για απόκριση.

Έτσι και σε αυτή την περίπτωση έχουμε πολύ καλή απόδοση ποιότητα υπηρεσιών και δεδομένων, όμοια με αυτήν που αναλύσαμε παραπάνω για τα πρωτόκολλα push. Η μόνη διαφορά είναι ότι για να ολοκληρωθεί το pull χρειάζονται 2 μηνύματα για κάθε γειτονικό κόμβο οπότε συνολικά 2*d μηνύματα. Τέλος σημειώνουμε ότι και σε αυτή την περίπτωση υπάρχει εξισορρόπηση του φόρτου εργασίας αλλά και ανοχή σε σφάλματα.

ΚΕΦΑΛΑΙΟ 6. ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Στο κεφάλαιο αυτό παρουσιάζεται η υλοποίηση που δημιουργήθηκε για τις ανάγκες της παρούσας μελέτης καθώς και αναλύονται τα πειραματικά αποτελέσματα για την ανάγκη αξιολόγησης των πρωτοκόλλων ενημέρωσης και δημιουργίας των αντιγράφων. Πιο συγκεκριμένα στην ενότητα 5.1 γίνεται περιγραφή της υλοποίησης, παρουσιάζοντας εκτενώς ποια θα είναι τα βήματα και η πορεία της υλοποίησης, η μέθοδος αναζήτησης και τα πρωτόκολλα που ενεργοποιούνται από τους κόμβους. Στην ενότητα 5.2 παρουσιάζονται τα πειραματικά αποτελέσματα της υλοποίησης. Γίνεται μελέτη κάθε πρωτοκόλλου ξεχωριστά με τους παράγοντες που τα επηρεάζουν για την εξαγωγή χρήσιμων συμπερασμάτων και τέλος ο συνδυασμός όλων των πρωτοκόλλων για πλήρη εικόνα αποτελεσμάτων.

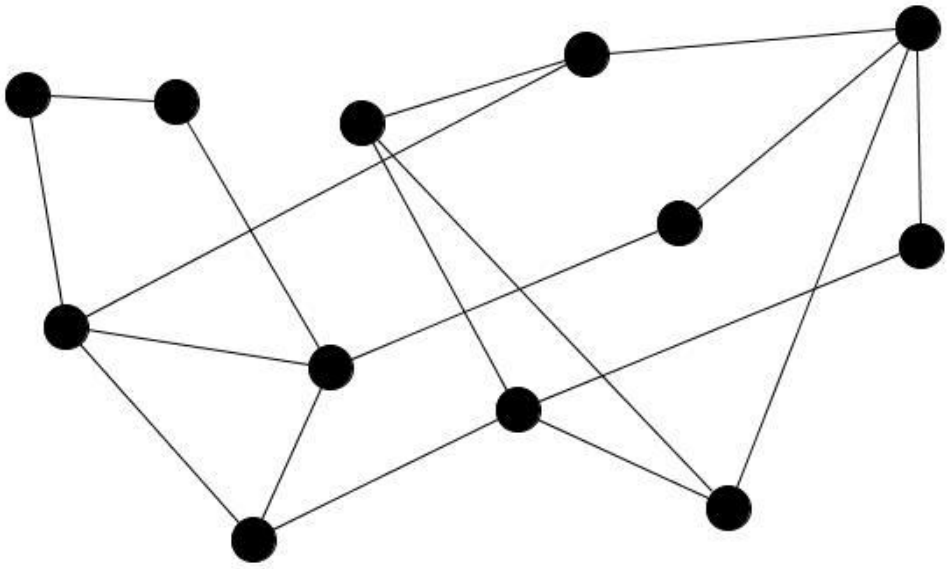
6.1. Περιγραφή Υλοποίησης

Στην ενότητα αυτή παρουσιάζουμε την υλοποίηση που δημιουργήθηκε για το σκοπό της μελέτης. Η υλοποίηση αποτελεί μια προσομοίωση ενός δικτύου ομότιμων κόμβων και κατασκευάστηκε εξαρχής χρησιμοποιώντας την αντικειμενοστραφή γλώσσα προγραμματισμού Java. Η υλοποίηση κατασκευάστηκε από την αρχή και αυτό συνέβαλλε όχι μόνο στην όσο γίνεται καλύτερη προσομοίωση ενός υπάρχοντος δικτύου ομότιμων κόμβων χρησιμοποιώντας έναν υπολογιστή, αλλά και στην πλήρη

παραμετροποίηση του δικτύου έτσι ώστε να πλησιάζει στις δικιές μας ανάγκες και περιορισμούς που έχουμε θέσει.

Όπως έχουμε προαναφέρει το σύστημα που μελετάμε αποτελεί ένα αδόμητο, μη κεντρικοποιημένο δίκτυο, όπου όλοι οι κόμβοι είναι ισάξιοι-ομότιμοι μεταξύ τους και τέλος δυναμικό δίκτυο όπου δηλαδή οι κόμβοι μπορούν να εισέρχονται και να αποχωρούν αυτοβούλως, όσες φορές και όποτε το επιθυμούν. Ένα τέτοιο δίκτυο προσομοιώθηκε μέσω της υλοποίησής μας.

Το δίκτυο που προσομοιώθηκε αποτελείται από N κόμβους και μπορεί να περιγραφεί από ένα τυχαίο γράφο G=(V,E) (σχήμα 6.1) όπου οι κορυφές V αποτελούν τους κόμβους του δικτύου και οι ακμές E αποτελούν τις συνδέσεις επικοινωνίας μεταξύ τους. Κάθε κόμβος είναι άμεσα συνδεδεμένος με d άλλους κόμβους που αποτελούν τους γείτονές του. Η τιμή d αποτελεί παράμετρο του συστήματος αλλά δεν είναι πάντα σταθερός αριθμός, όπως ακριβώς συμβαίνει και στα πραγματικά δίκτυα. Επίσης πρέπει να προσθέσουμε ότι οι ακμές του γράφου δεν είναι κατευθυνόμενες, δηλαδή οι συνδέσεις επικοινωνίας μεταξύ των κόμβων είναι διπλής κατεύθυνσης: ένας κόμβος μπορεί να επικοινωνήσει με τους γείτονές του, αλλά και οι γείτονές του μπορούν να επικοινωνήσουν με αυτόν.



Σχήμα 6.1 Απεικόνιση του P2P Δικτύου σαν Γράφος

Οι κόμβοι στο δίκτυο συνεργάζονται μεταξύ τους διαμοιράζοντας πόρους. Στην περίπτωσή μας διαμοιράζονται τα δεδομένα που διαθέτει ο κάθε κόμβος, όπου και τα προσφέρει στους ομότιμούς του. Ο κόμβος διατηρεί μια τοπική βάση δεδομένων με μια σχέση R και πλειάδες που την απαρτίζουν. Οι πλειάδες αποτελούνται από ένα αλφαριθμητικό id, ένα γνώρισμα τύπου ακεραίου και μια χρονοσφραγίδα που υποδηλώνει την ημερομηνία λήξης της. Η ημερομηνία λήξης τη χρονική στιγμή που θα λήξει η πλειάδα. Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο, το σύστημα έχει ένα καθολικό ρολόι που ξεκινάει από το μηδέν και αυξάνεται προσομοιώνοντας τον πραγματικό χρόνο. Έτσι μια πλειάδα που έχει ημερομηνία λήξης 10 υποδηλώνει ότι θα λήξει σε 10 χρονικές μονάδες και έπειτα θα πάει να ισχύει.

Η υλοποίηση που πραγματοποιήθηκε αποτελείται από ένα αντικείμενο τον “προσομοιωτή”, που εκτελεί κυρίως τις βασικές λειτουργίες της σύνθεσης του δικτύου και επιπλέον επιτελεί χρέη παρατηρητή όσων συμβαίνουν στο δίκτυο έτσι ώστε να εξάγονται τα αποτελέσματα των πειραματικών μετρήσεων. Επίσης η υλοποίηση αποτελείται και από άλλα αντικείμενα: “κόμβοι”, “μηνύματα” που ανταλλάσουν αυτοί και “πλειάδες”, αποτελούν αντικείμενα που αλληλεπιδρούν για την πλήρη προσομοίωση του δικτύου.

Το δίκτυο αποτελείται από Ν κόμβους, όπου και αποτελεί παράμετρο του συστήματος. Η τοπολογία του δικτύου είναι ένας τυχαίος γράφος. Ο κόμβος διαθέτει d γείτονες το πολύ, καθώς και μια τοπική βάση δεδομένων με πλειάδες. Επίσης περιέχονται μέθοδοι που επιτρέπουν στον κόμβο την αρχή μιας διαδικασίας αναζήτησης δεδομένων, την επεξεργασία και προώθηση των μηνυμάτων αναζήτησης από τους γείτονες, την ενημέρωση ή την δημιουργία νέων πλειάδων και τέλος την ενεργοποίηση διαφόρων πρωτοκόλλων που θα αναλύσουμε παρακάτω.

Αφού ο προσομοιωτής δημιουργήσει Ν κόμβους, αναθέτει τυχαία d γείτονες στον κάθε κόμβο και έναν αριθμό από πλειάδες στον καθένα. Κάθε πλειάδα έχει και μια ημερομηνία λήξης σαν γνώρισμα. Αυτό το γνώρισμα το δίνει αρχικά ο προσωμοιωτής τυχαία. Αρχικά δίνονται τιμές στην ημερομηνία λήξης από 2-31 χρονικές στιγμές (κύκλοι ρολογιού). Στην συνέχεια αυτές οι πλειάδες μπορούν να ενημερωθούν και να αυξήσουν την ημερομηνία λήξης τους.

Στη συνέχεια ο προσομοιωτής θέτει το κεντρικό ρολόι που αρχίζει από το 0 και σε κάθε κύκλο ρολογιού αυξάνεται κατά ένα. Σε κάθε κύκλο ρολογιού μπορούν να γίνουν διάφορες διεργασίες: Να ξεκινήσουν τυχαίες αναζητήσεις από τυχαίους κόμβους για δεδομένα, να επεξεργαστεί κάποιος κόμβος τα εισερχόμενα μηνύματα, να δημιουργήσει κάποιος κόμβος εξερχόμενα μηνύματα προς τους γείτονες, και να ενεργοποιηθούν τα διάφορα πρωτόκολλα στο δίκτυο. Η προσομοίωση ολοκληρώνεται όταν το κεντρικό ρολόι φτάσει σε κάποιο μεγάλο αριθμό όπου αποτελεί και αυτός παράμετρο του συστήματος. Στο τέλος ο παρατηρητής μας ενημερώνει για διάφορα ενδιαφέροντα αποτελέσματα όπως το πόσα ερωτήματα ήταν επιτυχημένα και πόσα όχι, τη φρεσκάδα των επιτυχημένων απαντήσεων, τη φρεσκάδα των πλειάδων των κόμβων του συστήματος κ.α.

Εδώ πρέπει να επισημανθεί ότι σε κάθε κύκλο ρολογιού ξεκινάνε ένας αριθμός από αναζητήσεις σε τυχαίους κόμβους του συστήματος για τυχαία δεδομένα. Αυτές οι αναζητήσεις θα μας δώσουν χρήσιμα συμπεράσματα ως προς την επιτυχία των αναζητήσεων, την διάδοση των ενημερώσεων κατά μήκος του δικτύου, τη φρεσκάδα της απάντησης κ.α. Οι αναζητήσεις αυτές δεν έχουν σχέση με ερωτήσεις τύπου

SELECT, ή λειτουργίες συνάθροισης και συνένωσης όπως μελετήθηκαν στο κεφάλαιο 3. Οι αναζητήσεις είναι αναζητήσεις των πλειάδων απλά με το κλειδί id. Σύμφωνα με αυτές τις αναζητήσεις ο αρχικός κόμβος αναζητάει με τη μέθοδο των τυχαίων διαδρομών μια πλειάδα σύμφωνα με το κλειδί της. Η πρώτη πλειάδα που εντοπίζεται και δεν έχει λήξει επιστρέφει σαν αποτέλεσμα. Αυτό συμβαίνει για να διαπιστωθεί αν ο κόμβος που αναζητάει μια πλειάδα είναι σε θέση να την εντοπίσει και να ενημερώσει μια «απαρχαιωμένη» δικιά του ή απλά να πάρει μια πληροφορία από το δίκτυο που του είναι χρήσιμη.

Επιπλέον ο προσομοιωτής ρίχνει, ή επανασυνδέει έναν αριθμό από κόμβους κάθε φορά. Όταν ένας κόμβος “πέφτει” τότε δεν συμμετέχει σε καμία πορεία αναζήτησης πετάει τα εισερχόμενα μηνύματά του, δεν μπορεί να δώσει εξερχόμενα μηνύματα και τέλος χάνει και τους κόμβους-γείτονες του. Αντιθέτως όταν επανασυνδέεται, αποκτά γείτονες και συμμετέχει ενεργά στο δίκτυο. Η τελευταία λειτουργία είναι πολύ χρήσιμη και συμβαίνει όπως ακριβώς και στο πραγματικό δίκτυο: κάθε κόμβος μπορεί να αποχωρήσει βίαια, χωρίς προειδοποίηση αλλά και το αντίθετο, δηλαδή μπορεί να επανασυνδεθεί και να συμμετέχει ενεργά στις διαδικασίες του δικτύου.

Συνοψίζοντας ο πίνακας 6.1 δίνει τις μέχρι στιγμής παραμέτρους του συστήματος. Στις παραμέτρους δίνονται και ενδεικτικές τιμές.

Πίνακας 6.1 Συνοπτικά οι αρχικές παράμετροι του συστήματος

Παράμετρος	Τιμή
Πλήθος κόμβων *N*	10 000 κόμβοι το μέγιστο
Πλήθος γειτόνων *d*	5
Πλήθος διαφορετικών πλειάδων του δικτύου	400
Πλήθος πλειάδων του κάθε κόμβου	4
Ημερομηνία λήξης των πλειάδων	2-31 χρονικές στιγμές από την έναρξη
Κύκλοι ρολογιού	200
Πλήθος κόμβων που αποσυνδέονται/επανασυνδέονται στο δίκτυο	10 κόμβοι σε κάθε κύκλο ρολογιού

Στη συνέχεια για να έχουμε μια ολοκληρωμένη και πλήρη άποψη της εφαρμογής και τις λειτουργίες της, πρέπει να αναφερθούμε στην μέθοδο αναζήτησης που

χρησιμοποιήθηκε, τα πρωτόκολλα που μπορούν να ενεργοποιήσουν οι κόμβοι για τη διάδοση και ενημέρωση των αντιγράφων και τέλος τι ακριβώς γίνεται σε κάθε βήμα της εφαρμογής, δηλαδή σε κάθε κύκλο ρολογιού.

6.1.1. Μέθοδος Αναζήτησης

Η μέθοδος αναζήτησης που χρησιμοποιήθηκε στην υλοποίηση είναι η μέθοδος των τυχαίων διαδρομών [6] (random walks). Οι λόγοι της χρήσης αυτής της τεχνικής σε αντίθεση με τη μέθοδο της πλημμύρας ήταν αρκετοί. Αρχικά αποφεύγουμε τα εκθετικά αυξανόμενα μηνύματα σε κάθε βήμα αναζήτησης καθώς σε κάθε $i^{οστό}$ βήμα της αναζήτησης η πλημμύρα παράγει d^i μηνύματα ενώ η μέθοδος random walks παράγει ακριβώς d. Επίσης λόγω των περιορισμένων μηνυμάτων ανά βήμα περιορίζονται τα διπλά μηνύματα και οι κύκλους που αποτελούν έναν σημαντικό αρνητικό παράγοντα στην αναζήτηση σε δίκτυο ομότιμων κόμβων. Τέλος με τη μέθοδο αυτή επιτυγχάνουμε μεγαλύτερη ικανότητα κλιμάκωσης του δικτύου.

Στην υλοποίησή μας η διαδικασία αναζήτησης με χρήση των τυχαίων διαδρομών έχει ως εξής: ο κόμβος που επιθυμεί κάποιο δεδομένο δημιουργεί ένα μήνυμα αναζήτησης με τα εξής στοιχεία: το δεδομένο που επιθυμεί, το TTL της αναζήτησης και ένα μοναδικό αναγνωριστικό της αναζήτησης. Αρχικά το μήνυμα αυτό προωθείται σε k γείτονές του (k random walkers). Όταν ένας ενδιάμεσος κόμβος λάβει το μήνυμα της αναζήτησης, ελέγχει το αναγνωριστικό της. Αν έχει περάσει το μήνυμα ξανά από αυτόν το αποβάλλει και δεν το επεξεργάζεται. Διαφορετικά εξετάζει την τοπική του βάση δεδομένων για το αν διαθέτει το ζητούμενο δεδομένο. Αν το δεδομένο υπάρχει και δεν έχει λήξει (δηλαδή αν η ημερομηνία λήξης της πλειάδας>0), τότε η αναζήτηση είναι επιτυχημένη, ο κόμβος στέλνει θετική απάντηση στον αρχικό κόμβο και το μήνυμα παύει να προωθείται. Αν ο κόμβος έχει το δεδομένο αλλά αυτό έχει λήξει, δηλαδή η ημερομηνία λήξης του είναι αρνητική, τότε συνεχίζεται η προώθηση του μηνύματος σαν να μην είχε βρεθεί. Αν δεν διαθέτει το δεδομένο, αυξάνει τα βήματα της αναζήτησης κατά ένα και αν τα βήματα είναι μικρότερα του TTL τότε το μήνυμα προωθείται σε έναν τυχαίο γείτονα. Διαφορετικά η αναζήτηση τερματίζεται και μια αρνητική απάντηση στέλνεται στον αρχικό κόμβο που υπέβαλλε την ερώτηση.

Όταν κάποια αναζήτηση είναι επιτυχημένη, εκτός από την θετική απάντηση, στον αρχικό κόμβο στέλνεται και το δεδομένο που βρέθηκε με την ημερομηνία λήξης που έχει και αυτός το αποθηκεύει στην τοπική βάση δεδομένων του, αφού σίγουρα δεν έχει λήξει. Παράλληλα το δεδομένο στέλνεται σε όλους τους κόμβους του μονοπατιού που έλαβαν μέρος στην αναζήτηση και οι κόμβοι αυτοί αποθηκεύουν το δεδομένο στην λανθάνουσα μνήμη (cache) που διαθέτουν.

Με την παραπάνω τακτική έχουμε αντιγραφή (replication) των δεδομένων που αναζητούνται για μεγαλύτερη διαθεσιμότητα και ανοχή σφαλμάτων. Οι κόμβοι που διαθέτουν το δεδομένο στην λανθάνουσα μνήμη τους πλέον μπορούν να απαντήσουν αμέσως σε μελλοντικές αναζητήσεις αυτών των δεδομένων. Η παραπάνω τακτική ονομάζεται path replication [8] και αποτελεί τον πιο εύκολα υλοποιήσιμο αλγόριθμο για square root replication [8]. Η square-root replication τακτική, είναι αρκετά αποδοτική, μειώνει σημαντικά τα βήματα των επιτυχημένων αναζητήσεων, δίνοντας καλούς χρόνους αναζήτησης και σε μη-δημοφιλή δεδομένα αλλά και σε δημοφιλή δεδομένα. Στην square-root τεχνική, ο αριθμός αντιγράφων κάθε αρχείου είναι ανάλογος της τετραγωνικής ρίζας της δημοφιλίας του, δηλαδή αν r_i είναι ο αριθμός αντιγράφων του δεδομένου i, και q_i η δημοφιλία του τότε : $r_i \propto \sqrt{q_i}$

6.1.2. Πρωτόκολλα Ενημέρωσης και Δημιουργίας Αντιγράφων στο Δίκτυο

Όπως έχουμε αναλύσει σε προηγούμενο κεφάλαιο, κάθε κόμβος μπορεί να ενεργοποιήσει διάφορα πρωτόκολλα ενημέρωσης των “απαρχαιωμένων” πλειάδων της τοπικής βάσης δεδομένων του ή και δημιουργίας αντιγράφων των φρέσκων δεδομένων ώστε να συμβάλλει στη διάδοση των ενημερώσεων των “φρέσκων” πλειάδων του συστήματος, άρα και στην φρεσκάδα της κατανεμημένης βάσης δεδομένων του δικτύου.

Τα πρωτόκολλα που μπορεί κάθε κόμβος να ενεργοποιήσει είναι τα ακόλουθα:

1. UP1: Όταν μια πλειάδα λήξει, ο κόμβος την αποβάλλει απλά από τη βάση δεδομένων του

2. UP2: Όταν μια πλειάδα λήξει, ο κόμβος τη διαγράφει και ψάχνει με νέο ερώτημα για πιο ενημερωμένη πλειάδα που δεν έχει ακόμα λήξει
3. UP2-Push: για κάθε πλειάδα x που έχει ημερομηνία λήξης μεγαλύτερη από ένα κατώφλι, ο κόμβος που κρατάει το x το προωθεί σε όλους τους γείτονές του. Το κατώφλι αυτό αποτελεί παράμετρο του συστήματος
4. UP2-pull: Ένα pull πρωτόκολλο όπου ένας κόμβος, αν έχει όλες τις πλειάδες του να λήγουν κάτω από ένα κατώφλι, ζητάει με pull από τους γείτονες την πιο πρόσφατη πλειάδα τους. Το κατώφλι και εδώ αποτελεί παράμετρο

Μπορούμε ακόμα να συνδυάσουμε με διάφορους τρόπους τα πρωτόκολλα όπως τα παρακάτω:

1. UP1,UP2,UP2-Push: συνδυασμός πρωτοκόλλων, ένα για δημιουργία ερωτήσεων για ενημέρωση των δεδομένων και ένα για δημιουργία αντιγράφων (push)
2. UP1,UP2,UP2-Pull: συνδυασμός πρωτοκόλλων, ένα για δημιουργία ερωτήσεων για ενημέρωση των δεδομένων και ένα για δημιουργία αντιγράφων (pull)

Τέλος ένα ακόμα πρωτόκολλο που συνδυάζουμε με τα 3,1 (Up2-Push, Up2-Pull), είναι το Lazy πρωτόκολλο που αναλύσαμε στα κεφάλαια 2 και 3. Δηλαδή μπορεί να μην γίνει push ή pull από τον κόμβο κάθε χρονική στιγμή αλλά ανά τακτά διαστήματα. Έτσι μειώνεται ο άσκοπος φόρτος δικτύου σε μηνύματα.

Στα πειράματα που θα ακολουθήσουν κάθε κόμβος ενεργοποιεί ένα από τα παραπάνω πρωτόκολλα ή συνδυασμό τους, ίδια για όλους τους κόμβους. Τα αποτελέσματα θα δείξουν τα θετικά και τα αρνητικά του κάθε πρωτοκόλλου, από ποιες παραμέτρους επηρεάζεται το κάθε πρωτόκολλο και πως μπορούμε να τα συνδυάσουμε για καλύτερα αποτελέσματα.

6.1.3. Τα Βήματα της Προσομοίωσης

Για μια πιο ξεκάθαρη εικόνα της υλοποίησης εδώ παρουσιάζουμε βήμα-βήμα τι ακριβώς γίνεται στην εφαρμογή και τι ακριβώς μετράτε σαν αποτέλεσμα:

- Αρχικά δημιουργείται το δίκτυο με N=10000 κόμβους όπου ο κάθε κόμβος έχει d γείτονες συνήθως 5 και έναν αριθμό από πλειάδες
- Έπειτα ο προσομοιωτής ξεκινάει ένα καθολικό ρολόι από το μηδέν όπου σε κάθε κύκλο ρολογιού αυξάνεται κατά ένα. Η προσομοίωση σταματάει μόλις το ρολόι δείξει κάποιο αριθμό που έχουμε θέσει σαν παράμετρο
- Σε κάθε κύκλο ρολογιού δημιουργείται ένας αριθμός από τυχαίες ερωτήσεις τυχαίων δεδομένων (όπου ακολουθούν τη zipf κατανομή που είναι πιο κοντά στην πραγματικότητα) από τυχαίους κόμβους του συστήματος. Το πόσες ερωτήσεις γίνονται αποτελεί παράμετρο του συστήματος και συνήθως τις θέτουμε στις 25 συνολικά ανά κύκλο ρολογιού. Αυτό σημαίνει ότι δεν αναζητάνε όλοι οι κόμβοι δεδομένα σε κάθε κύκλο ρολογιού, γιατί αυτό θα είχε μεγάλο φόρτο δικτύου
- Επίσης ο προσομοιωτής αλλάζει την κατάσταση ενός αριθμού από κόμβους στο δίκτυο. Ο αριθμός αυτός αποτελεί παράμετρο του συστήματος και το θέτουμε στο 0.001% των κόμβων (στην περίπτωσή μας 10 κόμβους) σε κάθε κύκλο ρολογιού και τους θέτει εκτός δικτύου, ή αν είναι εκτός τους επαναφέρει σε αυτό. Αν ο κόμβος βγει εκτός δικτύου χάνει τα δεδομένα της cache του, χάνει τα εισερχόμενα μηνύματα και δεν μπορεί να δημιουργήσει νέα ερωτήματα, μέχρι να επανασυνδεθεί
- Ο κάθε κόμβος σε κάθε κύκλο ρολογιού κάνει τις εξής λειτουργίες:

  1. ενεργοποιεί κάποιο ή κάποια από τα πρωτόκολλα που έχουμε θέσει στο σύστημα, φυσικά όλοι οι κόμβοι ενεργοποιούν τα ίδια πρωτόκολλα

  2. μπορεί να ενημερώνει ή όχι τις πλειάδες του ανά χρονικά διαστήματα, σαν να δημιουργούσε δικά του δεδομένα. Είναι όμοιο με κάποιο κόμβο σε ένα πραγματικό δίκτυο να ενημέρωνε, ή να δημιουργούσε κάποιο αρχείο και να το διέθετε στους ομότιμού του. Το κάθε πότε ίσως να ενημερώνει κάποια από τις πλειάδες του ο κόμβος, αποτελεί παράμετρο που εμείς θέτουμε

3. μπορεί να δημιουργεί κάποιο μήνυμα για αναζήτηση πλειάδας που δεν την διαθέτει στην βάση του

4. μπορεί να επεξεργαστεί τα εισερχόμενα μηνύματα και να ενημερώσει τους κόμβους αν διαθέτει το δεδομένο που αναζητείται ή να το προωθήσει όπως εξετάσαμε προηγουμένως

- Τέλος η εφαρμογή μας, μας ενημερώνει για το πόσες αναζητήσεις ήταν επιτυχημένες και πόσες όχι, τα βήματα των αναζητήσεων, για τη φρεσκάδα της τοπικής βάσης δεδομένων του κάθε κόμβου (δηλαδή το μέσο όρο των expiration times των πλειάδων στη βάση), τα πόσα αρχεία υπάρχουν κάθε φορά στο σύστημα και για τα μηνύματα που φορτώνει το κάθε πρωτόκολλο το σύστημα. Όλα αυτά θα μας δώσουν χρήσιμα συμπεράσματα για το ποιο πρωτόκολλο ή συνδυασμούς τους, μας δίνουν καλύτερα αποτελέσματα και πότε.

6.2. Πειραματικά Αποτελέσματα

Στην ενότητα αυτή παρουσιάζονται τα αποτελέσματα της υλοποίησής μας όπως αυτή αναλύθηκε παραπάνω. Για αρχή πρέπει να δοθούν όλες οι παράμετροι και οι τιμές όπως αυτές χρησιμοποιήθηκαν. Ο πίνακας 5.2 δίνει αυτές τις αρχικές τιμές των παραμέτρων και όπου θα γίνουν αλλαγές θα υπάρχει σαφής αναφορά.

Πίνακας 6.2 Πλήρης Πίνακας Παραμέτρων-Τιμών του Συστήματος

Παράμετρος	Τιμή
Πλήθος κόμβων *N*	10 000 κόμβους το μέγιστο
Πλήθος γειτόνων *d*	5
Πλήθος διαφορετικών πλειάδων του δικτύου	400
Πλήθος πλειάδων του κάθε κόμβου	4
Ημερομηνία λήξης των πλειάδων	2-31 χρονικές στιγμές από την έναρξη
Κύκλοι ρολογιού	120
Πλήθος κόμβων που αποσυνδέονται/επανασυνδέονται στο δίκτυο	10 κόμβοι σε κάθε κύκλο ρολογιού
Τυχαίες ερωτήσεις ανά κύκλο ρολογιού	25
Default TTL	10 στα τυχαία ερωτήματα 5 για το πρωτόκολλο ερωτημάτων UP2
Update 1 πλειάδας κόμβου	Με πιθανότητα 1/200 συνήθως
Χρόνος Update	3-12 χρονικές μονάδες
Κατώφλι push (πρωτόκολλο UP2-Push)	Ανάλογα με το πείραμα
Κατώφλι pull (πρωτόκολλο UP2-Pull)	Ανάλογα με το πείραμα

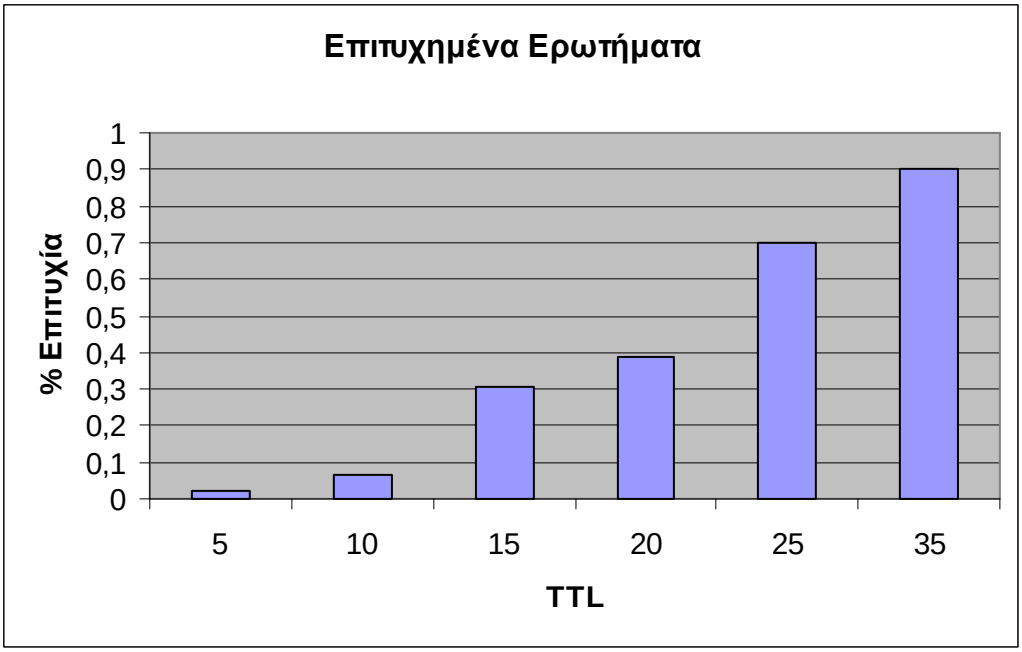
Στις παρακάτω υποενότητες θα γίνουν διάφορα πειράματα ως προς τα επιτυχημένα ερωτήματα και ο παράγοντας του βάθους δικτύου της ερώτησης κυρίως στο πρωτόκολλο Up1, οι παράγοντες που επηρεάζουν το κάθε πρωτόκολλο καθώς και πως συμπεριφέρονται συνδυασμένα τα πρωτόκολλα στο δίκτυο.

6.2.1. Επιτυχημένα Ερωτήματα

Το πρώτο πείραμα που θα πραγματοποιήσουμε είναι αυτό των επιτυχημένων ερωτημάτων. Αναφέρθηκε παραπάνω ότι σε κάθε κύκλο ρολογιού ένας αριθμός από κόμβους του συστήματος αρχίζουν αναζήτηση για τυχαία δεδομένα στο σύστημα. Στο σύστημα γίνονται 25 τέτοιες ερωτήσεις σε κάθε κύκλο ρολογιού και σκοπός μας είναι να εντοπίσουμε αν και κατά πόσο μας βοηθάνε τα πρωτόκολλα για την επιτυχή ολοκλήρωση αυτών. Όσο πιο πολλά ερωτήματα είναι τελικά επιτυχημένα, τόσο πιο πολύ βοηθάει το πρωτόκολλο που χρησιμοποιείται κάθε φορά στο να κρατήσει φρέσκιες τις πλειάδες αλλά και να τις διαδώσει όσο περισσότερο γίνεται. Σημειώνουμε ότι οι αναζητήσεις πλειάδων είναι αναζητήσεις πλειάδων με βάση το κλειδί (id) και η αναζήτηση σταματάει μόλις εντοπιστεί μια μη ληγμένη πλειάδα με αυτό το κλειδί. Αυτό γίνεται για να διαπιστωθεί αν οι ενημερωμένες πλειάδες είναι

διαδεδομένες κατά μήκος του δικτύου και αν κάποιος κόμβος που τις αναζητάει είναι σε θέση να τις εντοπίσει.

Όπως αναφερθήκαμε και παραπάνω στις τυχαίες αναζητήσεις, θέσαμε TTL=10. Είναι γνωστό ότι όσο μεγαλύτερο είναι το TTL τόσο πιο πολλές επιτυχημένες αναζητήσεις έχουμε. Εμείς προσπαθούμε να έχουμε λίγες επιτυχημένες αναζητήσεις χωρίς ενεργοποιημένο κάποιο πρωτόκολλο έτσι ώστε να διαπιστώσουμε αν βοηθάνε ή όχι όταν είναι ενεργοποιημένα. Αρχικά αναφέρουμε ότι τα διαφορετικά δεδομένα του δικτύου είναι 400 και το δίκτυο έχει 10000*4=40000 δεδομένα. Άρα το κάθε δεδομένο έχει 40000/400=100 αντίγραφα στο δίκτυο. Για να μπορέσει μια αναζήτηση να εντοπίσει το δεδομένο πρέπει να ισχύει (όπως αναφέραμε σε προηγούμενο κεφάλαιο): $k * t \cong \frac{N}{n_x}$ όπου k περιπατητές της αναζήτησης που τους θέσαμε k=d (=5 το μέγιστο), t = TTL και $\frac{N}{n_x} = \frac{10000}{100} = 100$ άρα t=100/5=20. Οπότε, θεωρητικά, με TTL=20 αν δεν υπάρχουν κύκλοι, αποτυχίες κόμβων ή αν κάθε κόμβος είχε πάντα 5 γείτονες, πράγμα που δεν συμβαίνει, το δεδομένο αναμένεται να βρεθεί. Το σχήμα 6.2 μας δείχνει τα επιτυχημένα ερωτήματα ανάλογα με το TTL όταν στο δίκτυο ενεργοποιείται μόνο το UP1, δηλαδή να υπάρχουν μόνο οι πλειάδες που δεν έχουν λήξει.



Σχήμα 6.2 Επιτυχημένα Ερωτήματα σε Σχέση με το TTL Χωρίς Ενεργοποιημένο Πρωτόκολλο

Από το σχήμα 6.2 παρατηρούμε πράγματι ότι όσο μεγαλύτερο το TTL τόσο περισσότερα ερωτήματα θα είναι επιτυχημένα. Επίσης είναι αναμενόμενο ότι με TTL=20 λόγω των διπλών μηνυμάτων, των αποτυχιών και των κύκλων το 40% της επιτυχίας είναι πάλι αναμενόμενο.

Εμείς για την υλοποίησή μας θέλαμε να κρατήσουμε σε χαμηλά ποσοστά το TTL άρα και την επιτυχία των ερωτημάτων χωρίς ενεργοποιημένα πρωτόκολλα ενημέρωσης-δημιουργίας αντιγράφων, ώστε να διαπιστώσουμε αν τα πρωτόκολλα συμβάλλουν ή όχι στην διάδοση των ενημερώσεων. Αν αναλογιστούμε και το γεγονός ότι όσο μικρότερο TTL τόσο λιγότερα μηνύματα άρα και φόρτο δικτύου, μας ώθησε στο να θέσουμε TTL=10 στα τυχαία ερωτήματα. Όσο αφορά το TTL στο πρωτόκολλο UP2 κρατήθηκε αρχικά σε μικρά επίπεδα με TTL=5 για να μην υπάρχει μεγάλος φόρτος σε μηνύματα κατά μήκος του δικτύου. Όσο μεγαλώνει το TTL θα έχουμε πολύ περισσότερα μηνύματα αλλά και επιτυχημένα ερωτήματα, αλλά κάτι τέτοιο θα το εξετάσουμε στη συνέχεια.

6.2.2. Επίπτωση Παραμέτρων στα Διάφορα Πρωτόκολλα

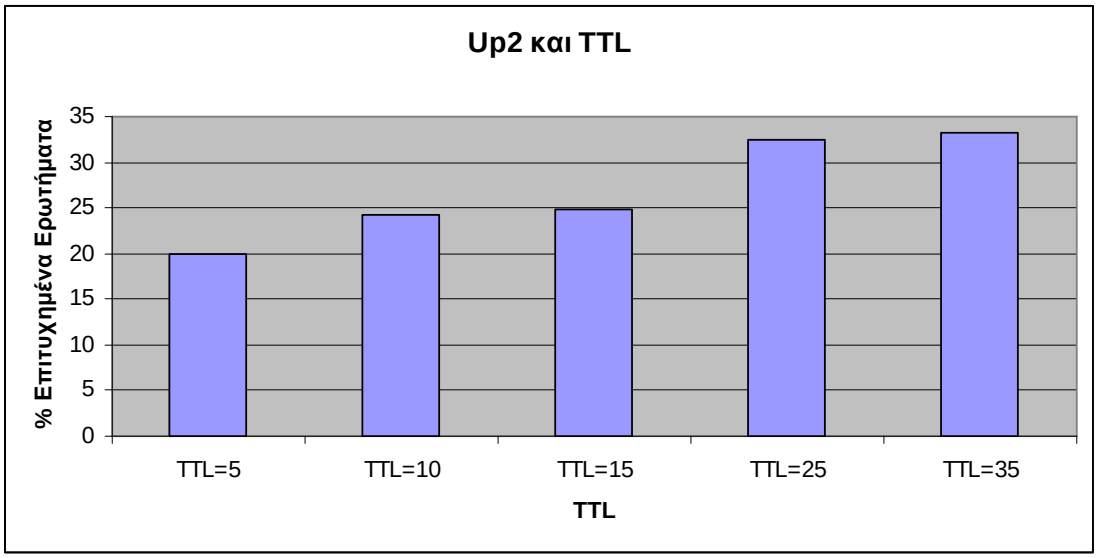
Στην παρούσα ενότητα θα ασχοληθούμε με το ποιες παραμέτρους επηρεάζουν την απόδοση των 3 βασικών πρωτοκόλλων ενημέρωσης και αντιγραφής δεδομένων στο σύστημα. Η αρχή θα γίνει με το πρωτόκολλο Up2 και στη συνέχεια με το Up2-Push και Up2-Pull. Σκοπός είναι να διαπιστώσουμε κατά πόσο επηρεάζονται τα πρωτόκολλα από τις παραμέτρους και να βρούμε τις τιμές των παραμέτρων που δίνουν καλά αποτελέσματα με όσο το δυνατόν μικρότερο φόρτο σε μηνύματα.

- Πρωτόκολλο Up2 και επίπτωση παραμέτρων

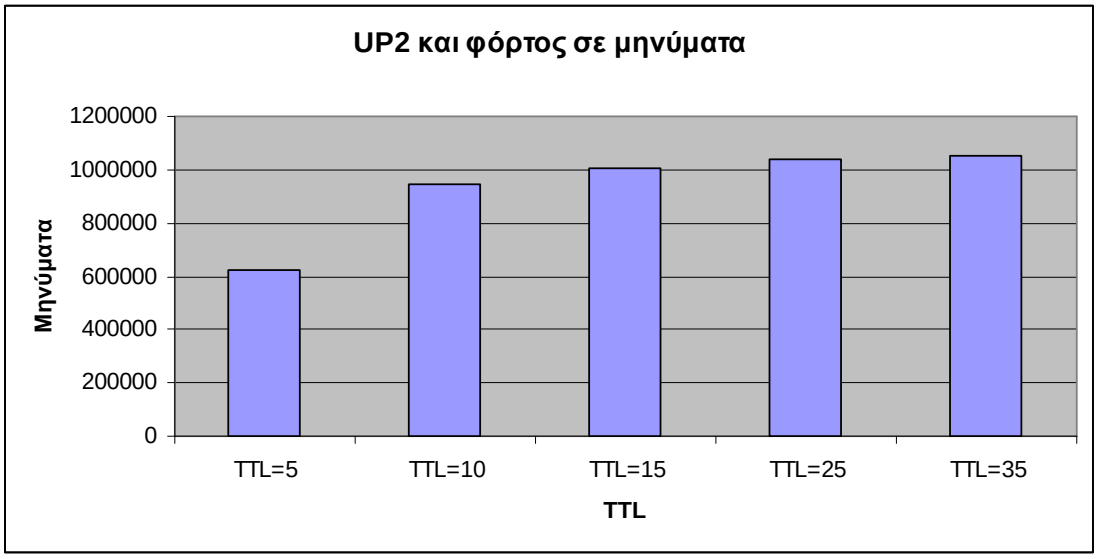
Για το πρωτόκολλο Up2 δύο παράγοντες που ενδεχομένως το επηρεάζουν είναι το βάθος δικτύου (TTL) των αναζητήσεων που δημιουργεί και το πόσο συχνά ενημερώνονται οι πλειάδες από τους ίδιους τους κόμβους.

Παράμετρος TTL: Γι' αυτήν την παράμετρο (το TTL) χρειαζόμαστε το βάθος δικτύου να μην είναι μεγάλο γιατί το δίκτυο φορτώνεται με πολλά μηνύματα ανά πλειάδα. Αν το δίκτυο έχει 10000*4=40000 πλειάδες, αυτές κάποια στιγμή θα λήξουν και συνολικά θα δημιουργηθούν 40000 αναζητήσεις για να βρεθούν ενημερώσεις όπου και αυτές θα λήξουν και αυτό συνεχίζεται επ' άπειρο.

Για το πρώτο πείραμα ενεργοποιούμε στο δίκτυο μόνο το UP2, και αλλάζουμε την παράμετρο TTL. Υπενθυμίζουμε ότι όποιος κόμβος έχει ληγμένη μια πλειάδα τότε δημιουργεί μια ερώτηση με κάποιο TTL για να βρει ίδια πλειάδα που να μην έχει λήξει. Προηγουμένως θέσαμε TTL=5, κάτι όμως που είναι πολύ μικρό. Τι θα γίνει αν το μεγαλώσουμε; Για να μελετήσουμε κάτι τέτοιο, αρχικά στο σχήμα 6.3 μετράμε το ποσοστό % των επιτυχημένων ερωτημάτων των τυχαίων ερωτημάτων που οι κόμβοι θέτουν ανεξάρτητα από το πρωτόκολλο. Όσο πιο ψηλό το ποσοστό της επιτυχίας τόσο πιο εύκολα βρίσκει ένας κόμβος το δεδομένο που επιθυμεί, άρα βγαίνει το συμπέρασμα ότι τα δεδομένα εξακολουθούν να υπάρχουν και να εξαπλώνονται στο δίκτυο. Όμως όσο το TTL αυξάνει αυξάνεται και ο φόρτος του δικτύου σε μηνύματα.
Το σχήμα 6.4 δείχνει το φόρτο σε μηνύματα που φορτώνεται το δίκτυο από το πρωτόκολο.



Σχήμα 6.3 Up2 ως προς τα Επιτυχημένα Ερωτήματα



Σχήμα 6.4 Up2 ως προς τα Μηνύματα που Φορτώνεται το Δίκτυο

Όπως παρατηρούμε από τα δύο σχήματα, όσο το TTL αυξάνεται, αυξάνεται και η επιτυχία των ερωτημάτων, όχι ανάλογα όμως. Επίσης αυξάνεται και ο φόρτος

δικτύου πάλι όχι γραμμικά (θα περιμέναμε να είναι γραμμικά για το λόγο ότι έχουμε αναζήτηση random walks και όχι flooding που δίνει εκθετικά αυξανόμενα μηνύματα). Για τα μηνύματα είναι λογικό να αυξάνονται αλλά όχι γραμμικά γιατί όσο μεγαλύτερο TTL έχουμε, τόσο δημιουργούνται περισσότεροι κύκλοι, και όσο πιο πολύ μένει στο δίκτυο η ερώτηση τόσο μεγαλύτερη πιθανότητα έχει να σταματήσει λόγω αποτυχίας κόμβων.

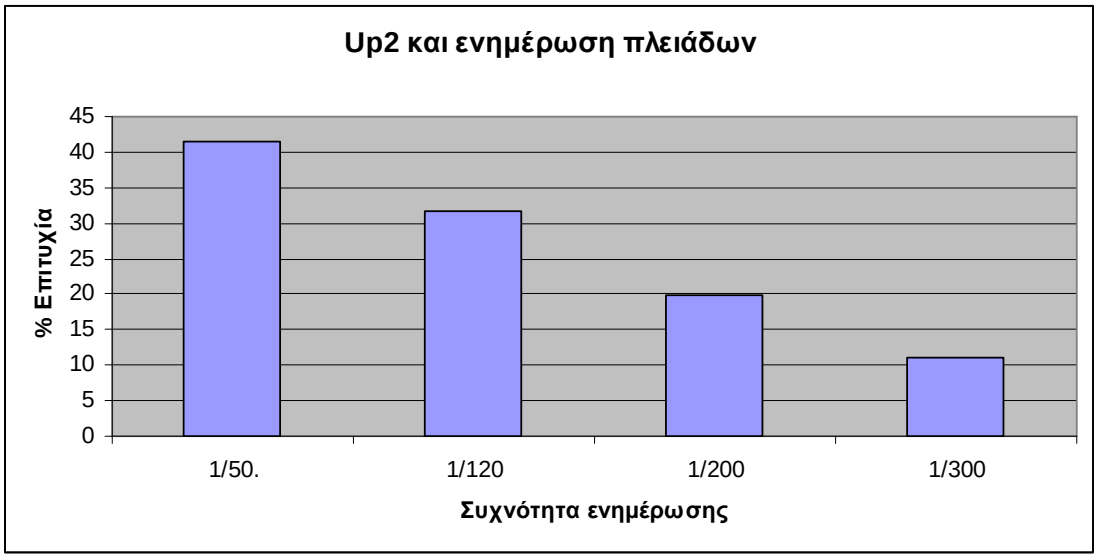
Όσο αφορά την επιτυχία των ερωτημάτων, αυτή είναι όμοια με τα μηνύματα. Αυξάνεται όσο αυξάνονται τα μηνύματα.

Άρα τελικά συμφέρει όπως αναλύθηκε και στο θεωρητικό μέρος της μελέτης να κρατάμε το TTL μικρό όσο γίνεται. Το TTL=5 στην περίπτωση του δικού μας δικτύου είναι αρκετά αποδοτικό συγκρινόμενο με την επιτυχία των ερωτημάτων (20% έναντι 6% αν δεν υπήρχε το Up2) και το φόρτο δικτύου.

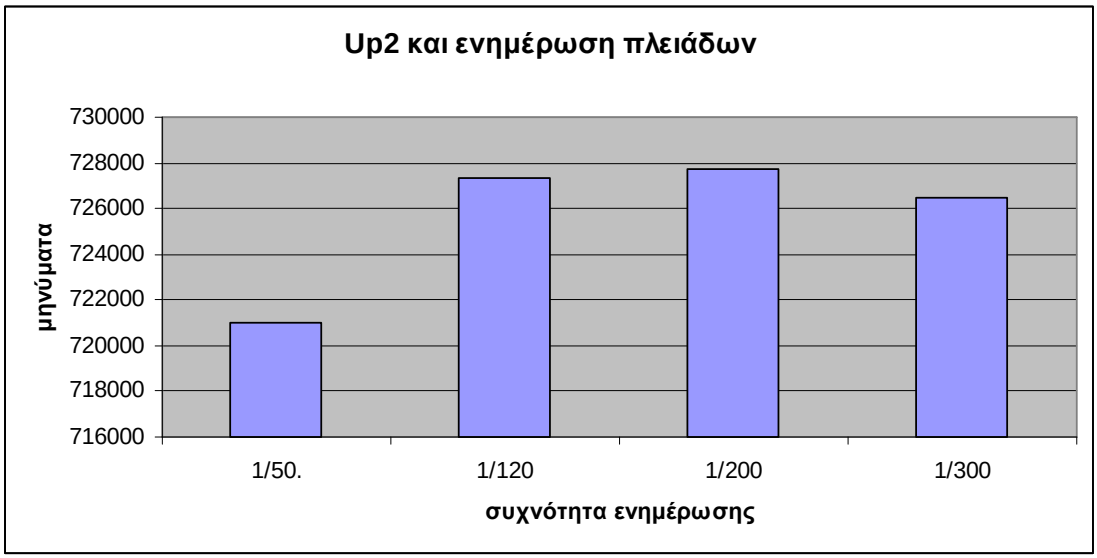
Συχνότητα ενημερώσεων των πλειάδων: Άλλη παράμετρος που πρέπει να αλλάξουμε για να διαπιστώσουμε το αντίκτυπο που έχει το πρωτόκολλο Up2 είναι το πόσο συχνά ανανεώνουμε (ή δημιουργούμε) πλειάδες στο σύστημα. Μέχρι τώρα οι πλειάδες ανανεώνονται με πιθανότητα 1/200 ανά κόμβο, δηλαδή αν έχουμε 10000 κόμβους αρχικά, ανανεώνονται 50 πλειάδες, κάτι που μειώνεται καθώς μειώνονται οι πλειάδες και οι κόμβοι στο σύστημα. Όσο πιο σπάνια ανανεώνουμε τις πλειάδες τόσο πιο γρήγορα (χωρίς ενεργοποιημένα πρωτόκολλα) θα λήξουν όλες οι πλειάδες.

Σκοπός μας εδώ είναι να διαπιστώσουμε αν όσο αυξάνουμε ή μειώνουμε τον αριθμό των πλειάδων που ανανεώνουμε, έχει επίπτωση στα επιτυχημένα μηνύματα και στο φόρτο του δικτύου σε μηνύματα προερχόμενα από το Up2.

Τα πειράματα έγιναν για ανανέωση πλειάδων του κόμβου 1 φορά κάθε 50 κύκλους ρολογιού, 1 κάθε 120, 1 κάθε 200 και 1 κάθε 300. Το σχήμα 5.5 δείχνει την % επιτυχία των τυχαίων ερωτημάτων και το σχήμα 5.6 το φόρτο του δικτύου σε μηνύματα:

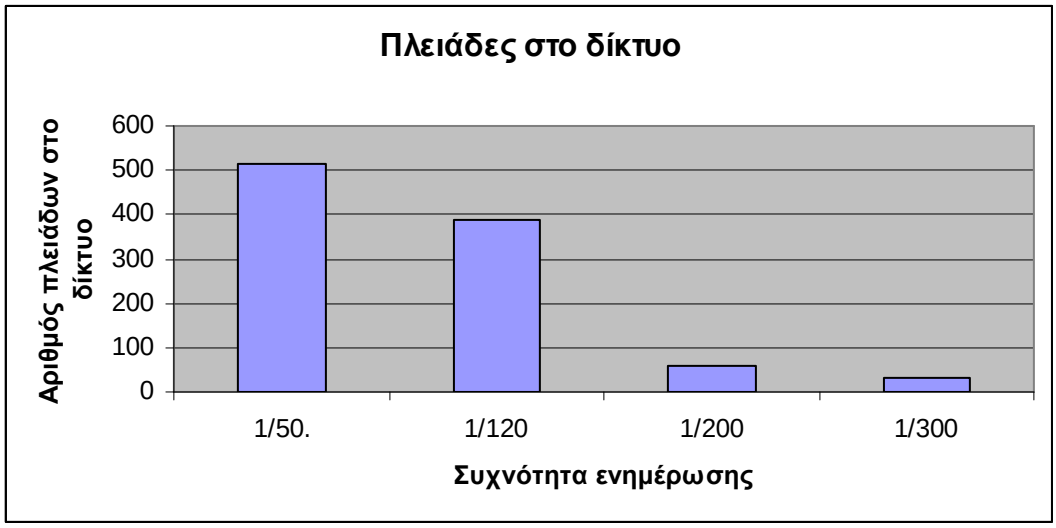


Σχήμα 6.5 Up2 και Επιτυχία Ερωτημάτων σε Σχέση με τη Συχνότητα Ενημέρωσης Πλειάδων



Σχήμα 6.6 Up2 και Μηνύματα σε Σχέση με τη Συχνότητα Ενημέρωσης των Πλειάδων

Το σχήμα 6.5 δείχνει ότι όταν ενημερώνονται συχνά οι πλειάδες τότε έχουμε καλύτερα αποτελέσματα αλλά, δεν έχει παίζει και τόσο ρόλο το πρωτόκολλο καθώς τα μηνύματα είναι πάνω κάτω παρόμοια:72100-728000 (σχήμα 6.6). Παίζει ρόλο όμως το πόσες πλειάδες παραμένουν μετά από 120 γύρους στο σύστημα και αυτό φαίνεται στο σχήμα 6.7:



Σχήμα 6.7 Πλειάδες που Παραμένουν στο Δίκτυο σε Σχέση με τη Συχνότητα Ενημέρωσης

Παρατηρούμε ότι όσο πιο συχνά ενημερώνονται οι πλειάδες τόσο πιο πολλές πλειάδες παραμένουν στο δίκτυο. Είναι καλό στο δίκτυο να ενημερώνονται οι πλειάδες συχνά όμως αυτή η παράμετρος δεν επηρεάζει το Up2 γιατί όσο πιο συχνά αυτές ενημερώνονται τόσο πιο σπάνια ενεργοποιείται αυτό.

Τελικά τα συμπεράσματα για το πρωτόκολλο Up2 είναι:

1. Όσο πιο μεγάλο TTL τόσο καλύτερα αποτελέσματα ως προς την επιτυχία ερωτήσεων έχουμε, αλλά όχι ανάλογα με τα μηνύματα. Για το λόγο αυτό το κρατάμε χαμηλό
2. Όσο πιο συχνά ενημερώνουμε τις πλειάδες τόσο καλύτερα αποτελέσματα έχουμε αλλά αυτό δεν έχει σχέση με το πρωτόκολλο Up2 γιατί παρατηρούμε ίσο αριθμό μηνυμάτων που φορτώνει το δίκτυο κάθε φορά. Ίσως αυτή η παράμετρος θα μας βοηθήσει σε άλλα πρωτόκολλα

3. Αυτό που πρέπει να γίνει είναι, δυναμικά όταν ένας κόμβος έχει ενημερωμένες πλειάδες, κατά μεγάλη πιθανότητα και οι άλλοι κόμβοι θα τις έχουν ενημερωμένες, οπότε πρέπει να κρατηθεί χαμηλό το TTL. Αλλιώς πρέπει να το αυξήσουμε για καλύτερα αποτελέσματα.

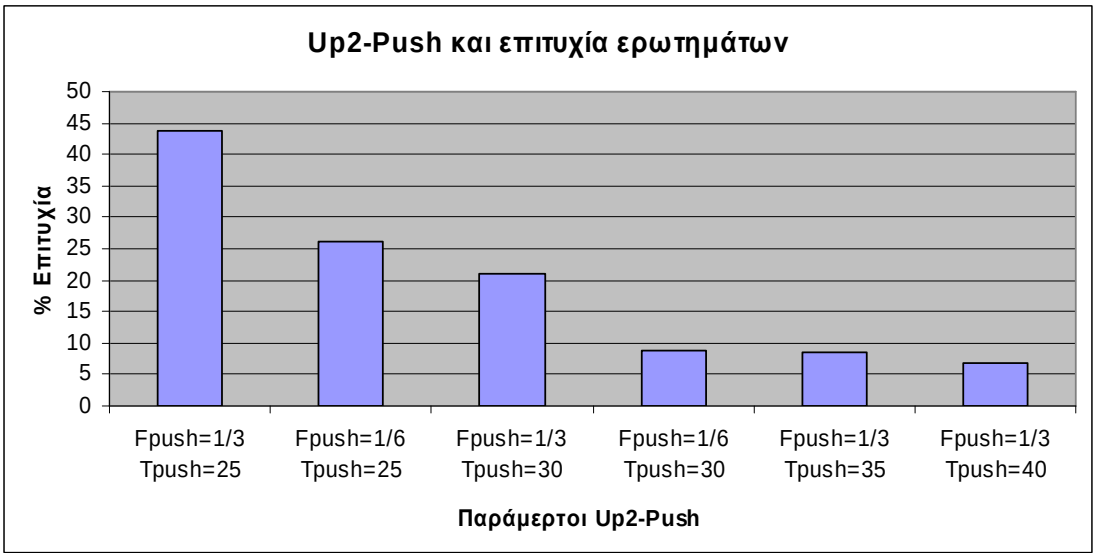
- Πρωτόκολλο Up2-Push και επίπτωση παραμέτρων

Όπως έγινε και στο πρωτόκολλο Up2 έτσι και στο Up2-Push θα δούμε πως επηρεάζεται από τις παραμέτρους του συστήματος έτσι ώστε να βγάλουμε χρήσιμα συμπεράσματα. Οι παράμετροι που επηρεάζουν το πρωτόκολλο Push είναι: Το threshold που έχουμε βάλει, δηλαδή πάνω από πόσες χρονικές μονάδες στο exp.time η πλειάδα να γίνεται Push από τον κόμβο, κάθε πότε θα ενεργοποιείται το πρωτόκολλο και τέλος πως επηρεάζεται από τη συχνότητα ενημέρωσης των πλειάδων στο σύστημα.

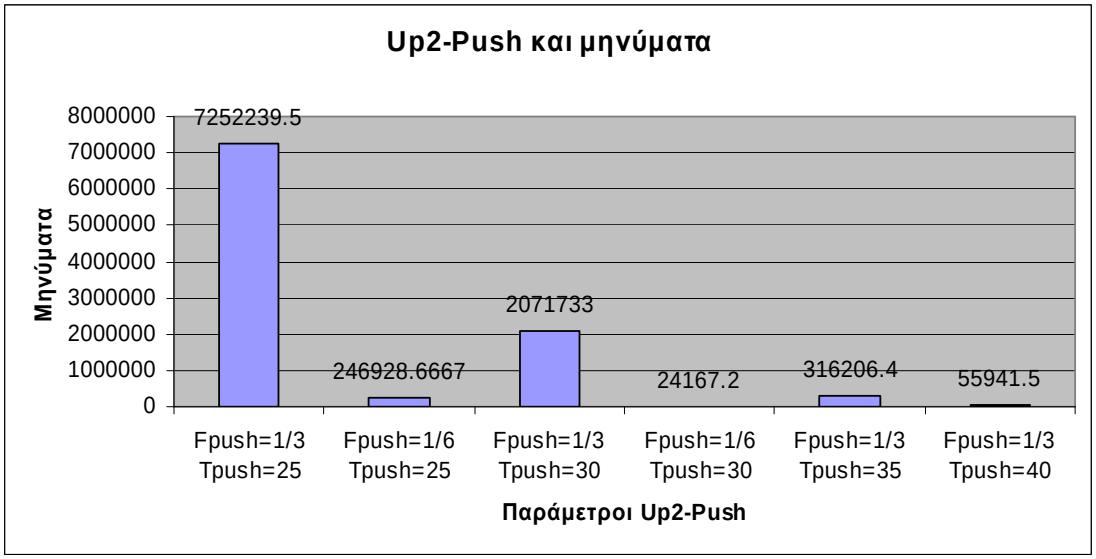
Από δω και στο εξής θα χρησιμοποιήσουμε τις συντομεύσεις:
Tpush: το threshold του push δηλαδή αν Tpush=20 αν η πλειάδα λήγει σε 20 χρονικές μονάδες να γίνεται Push από τον κόμβο
Fpush: η συχνότητα που ενεργοποιείται το πρωτόκολλο. Αν Fpush=1/3 το πρωτόκολλο θα ενεργοποιείται 1 φορά τις 3 χρονικές μονάδες. Αυτό γίνεται γιατί αν το πρωτόκολλο είναι ενεργοποιημένο συνεχώς η φρέσκια πλειάδα, θα κάνει ουσιαστικά πλημμύρα, δηλαδή πολλά μηνύματα στο δίκτυο και αρκετοί κύκλοι.. Θέλουμε απλά να γίνει κάποιο είδος replication χωρίς πλημμύρα.

Ουσιαστικά η εισαγωγή του Fpush, όπως και του Fpull όπως θα δούμε παρακάτω μαρτυράει τον συνδυασμό του Lazy πρωτοκόλλου που αναλύσαμε στο θεωρητικό μοντέλο με το πρωτόκολλο Up2-Push και Up2-Pull. Χρησιμοποιήθηκε αυτός ο συνδυασμός γιατί διαφορετικά υπάρχει άσκοπη ανταλλαγή μηνυμάτων συνεχώς.

<u>Το πρωτόκολλο Up2-Push και οι παράμετροι Tpush, Fpush:</u> Σταθεροποιώντας την συχνότητα ενημέρωσης στο 1/200, αλλάξαμε τις παραμέτρους Fpush και Tpush. Τα σχήματα 6.8 και 6.9 δείχνουν τα αποτελέσματα σε επιτυχία των μηνυμάτων και φόρτο εργασίας, για τις διάφορες παραμέτρους του Up2-Push:



Σχήμα 6.8 Up2-Push, Παράμετροι και Επιτυχία Ερωτημάτων



Σχήμα 6.9 Up2-Push, Παράμετροι και Μηνύματα

Από το σχήμα 6.8 παρατηρούμε ότι όσο πιο μικρό είναι το Tpush τόσο μεγαλύτερη επιτυχία στα ερωτήματα έχουμε. Αυτό είναι φυσικό γιατί ολοένα και περισσότερες

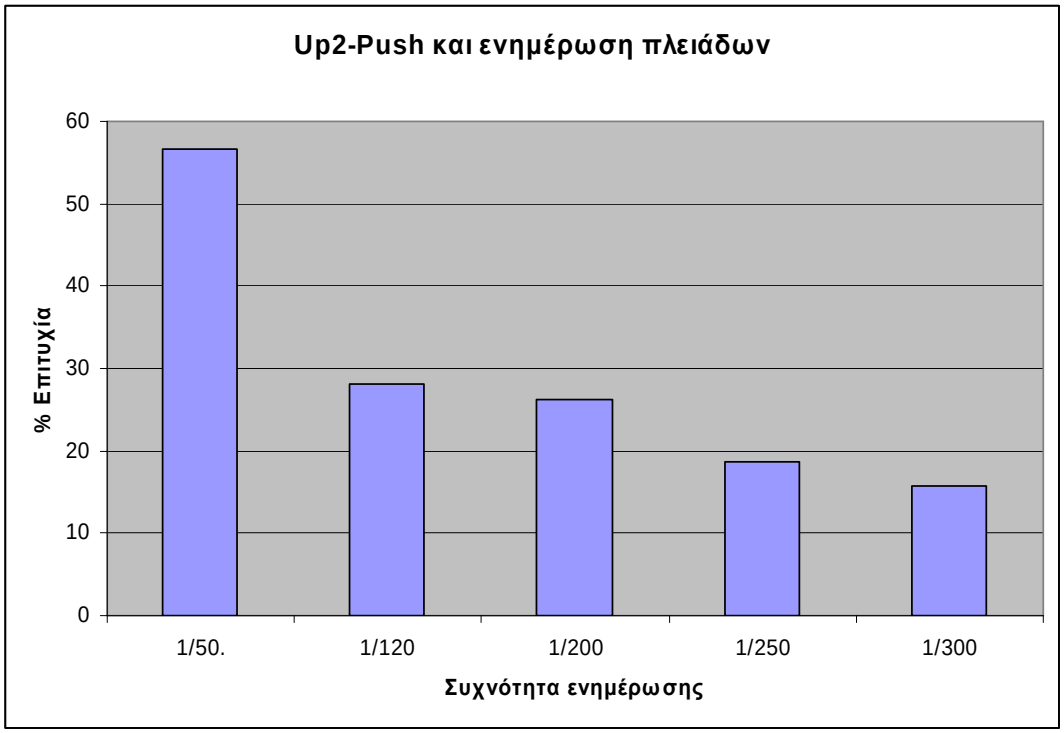
πλειάδες μπορεί να γίνονται Push και έτσι οι φρέσκιες πλειάδες εξαπλώνονται γρήγορα στο δίκτυο. Από τη στιγμή που θέτουμε τις πλειάδες με exp.time 2-32 χρονικές μονάδες αρχικά υπάρχουν πολλές πλειάδες πάνω από 25 χρονικές μονάδες, λίγες πάνω από 30 και καμία πάνω από 35. Γι' αυτό και το ποσοστό δεν βελτιώνεται με Tpush>30

Από το σχήμα 6.9 παρατηρούμε ότι τα μηνύματα είναι πάρα πολύ περισσότερα όσο μεγαλύτερο είναι το Fpush. Κάτι τέτοιο είναι λογικό γιατί όσο πιο μεγάλο είναι το Fpush τόσο περισσότερο η διάδοση των πλειάδων φτάνει σε πλημμύρα. Τα μηνύματα με Fpush=1/3 και Tpush=25 είναι τριπλάσια από ότι Fpush=1/6 και Tpush=25 αλλά η επιτυχία συνεχίζει να βρίσκεται σε υψηλά επίπεδα (~26%). Το ίδιο παρατηρούμε και με Tpush=1/30.

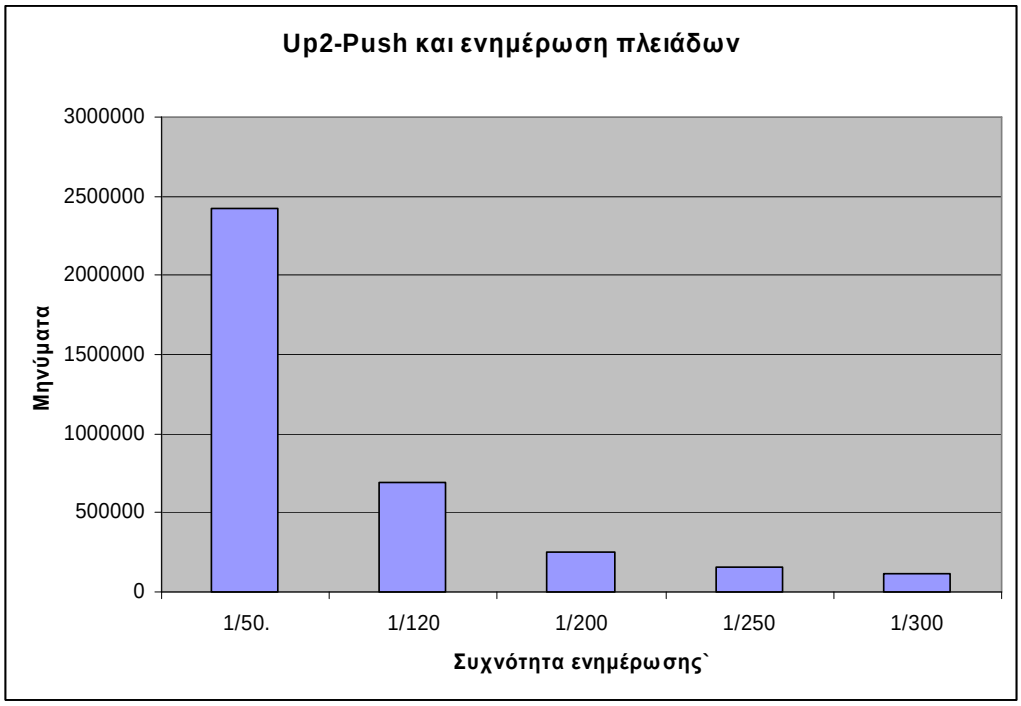
Τελικά, μπορούμε να πούμε ότι κρατάμε χαμηλό το Tpush, παραπάνω από το μέσο όρο των exp.time που είναι ~16 χρονικές μονάδες και επίσης χαμηλό το Fpush στο 1/6 (αν γίνει 1/8 είναι το ίδιο με Fpush=1/3 και Tpush 5 μονάδες παραπάνω). Στο δικό μας δίκτυο μια βέλτιστη λύση είναι να θέσουμε Fpush=1/6 και Tpush=25 που μας δίνει υψηλή επιτυχία με λίγα μηνύματα. Επαναλαμβάνουμε ότι η επιτυχία θα ανέβει αν το πρωτόκολλο συνδυαστεί με άλλα όπως γίνεται στη συνέχεια.

Το πρωτόκολλο Up2-Push και η συχνότητα ενημερώσεων των πλειάδων: Άλλη παράμετρος που πρέπει να αλλάξουμε για να διαπιστώσουμε το αντίκτυπο που έχει το πρωτόκολλο Up2-Push είναι το πόσο συχνά ανανεώνουμε (ή δημιουργούμε) πλειάδες στο σύστημα.

Τα πειράματα έγιναν για ανανέωση με συχνότητα 1/50, 1/120, 1/200, 1/250, 1/300 και με παραμέτρους Tpush=25 και Fpush=1/6. Το σχήμα 6.10 δείχνει την επιτυχία των ερωτημάτων που είχαμε και το 6.11 τα μηνύματα που φορτώνεται το δίκτυο κάθε φορά.



Σχήμα 6.10 Up2-Push και Ενημέρωση Πλειάδων ως προς την Επιτυχία των Μηνυμάτων



Σχήμα 6.11 Up2-Push και Ενημέρωση Πλειάδων ως προς το Φόρτο του Δικτύου σε Μηνύματα

Από τα δύο σχήματα παρατηρούμε ότι όσο η συχνότητα ενημέρωσης είναι μεγάλη, η επιτυχία των ερωτημάτων είναι μεγάλη, όμως είναι και πάρα πολλά τα μηνύματα που φορτώνεται το δίκτυο από το πρωτόκολλο. Αυτό συμβαίνει γιατί όσο πιο συχνά ενημερώνουμε τις πλειάδες τόσο πιο πολλές πλειάδες έχουμε με exp.time>25 άρα γίνονται push.

Τελικά αυτό που πρέπει να γίνει είναι να κρατάμε το Tpush λίγο μεγαλύτερο από το μέσο όρο των exp.time της βάσης αλλά και το Fpush να προσαρμόζεται ανάλογα με τις ενημερώσεις. Αν οι ενημερώσεις είναι συχνές τότε το Fpush πρέπει να είναι μικρό για να κρατηθεί ο φόρτος σε μηνύματα χαμηλός.

Οπότε κάθε κόμβος δυναμικά θα μπορούσε να αλλάζει το Tpush ανάλογα με τι πλειάδες έχει ή τι υπάρχει στο δίκτυο αν έχει κάποιου είδους ενημέρωση. Το Fpush καλό είναι να το θέσουμε χαμηλό περίπου στο Fpush=1/6 στο σύστημά μας.

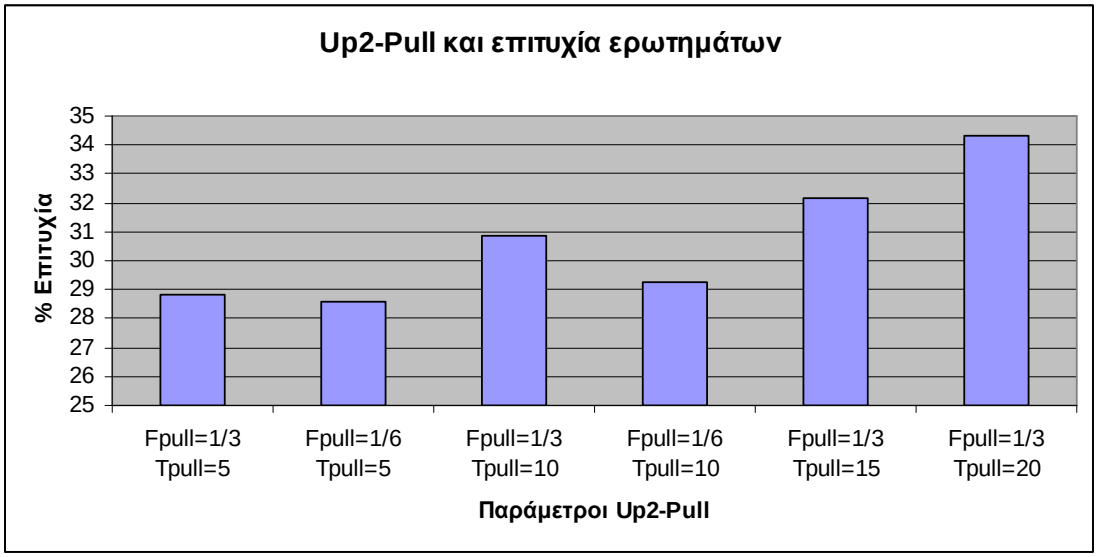
- Πρωτόκολλο Up2-Pull και επίπτωση παραμέτρων

Όπως και στο πρωτόκολλο Up2-Push έτσι και εδώ θα δούμε πώς το πρωτόκολλο Up2-Pull επηρεάζεται από τις παραμέτρους του συστήματος. Οι παράμετροι που επηρεάζουν το πρωτόκολλο Pull είναι οι ίδιοι με το Push: Το κατώφλι που έχουμε θέσει, δηλαδή το πότε ο κόμβος να ζητάει Pull από τους γείτονες και κάθε πότε θα ενεργοποιείται το πρωτόκολλο. Τέλος όπως θα δούμε το Up2-Pull επηρεάζεται και από τη συχνότητα ενημέρωσης των πλειάδων στο σύστημα.

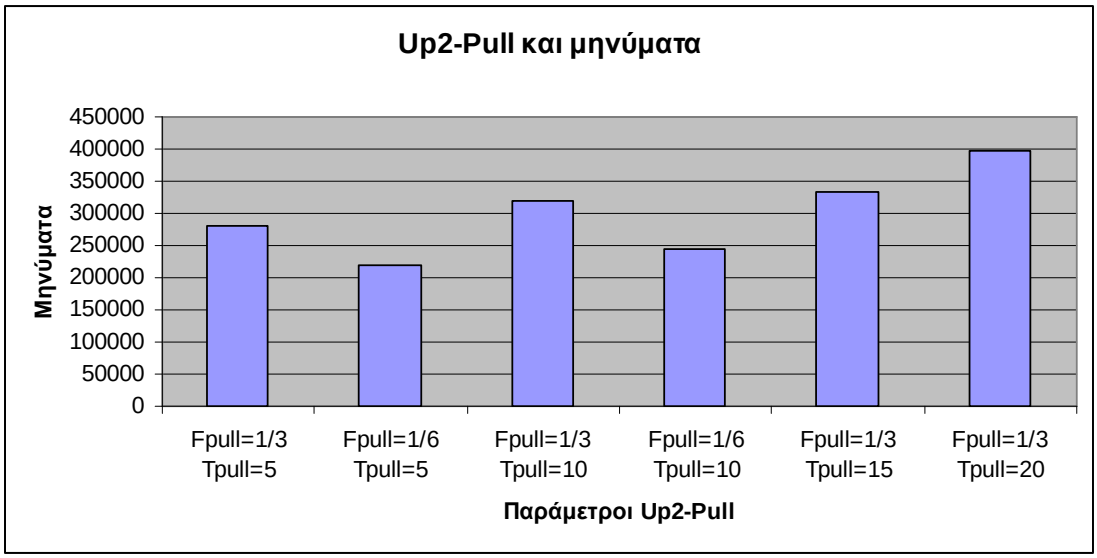
Από δω και στο εξής θα χρησιμοποιήσουμε τις συντομεύσεις όμοιες με αυτές του Up2-Push:
Tpull: το κατώφλι του pull. Δηλαδή ο κόμβος ζητάει Pull από τους γείτονες αν οι πλειάδες του έχουν ημερομηνία λήξης κάτω από το Tpull.
Fpull: η συχνότητα που ενεργοποιείται το πρωτόκολλο Up2-Pull.

Το πρωτόκολλο Up2-Pull και οι παράμετροι Tpull, Fpull: Τα σχήματα 6.12 και 6.13 δείχνουν τα αποτελέσματα σε επιτυχία των μηνυμάτων και φόρτο εργασίας, για τις

διάφορες παραμέτρους του Up2-Pull. Σταθεροποιήσαμε τη συχνότητα ενημέρωσης των πλειάδων σε 1/200 και αλλάξαμε τις παραμέτρους Fpull, Tpull.



Σχήμα 6.12 Up2-Pull Παράμετροι και Επιτυχία Ερωτημάτων

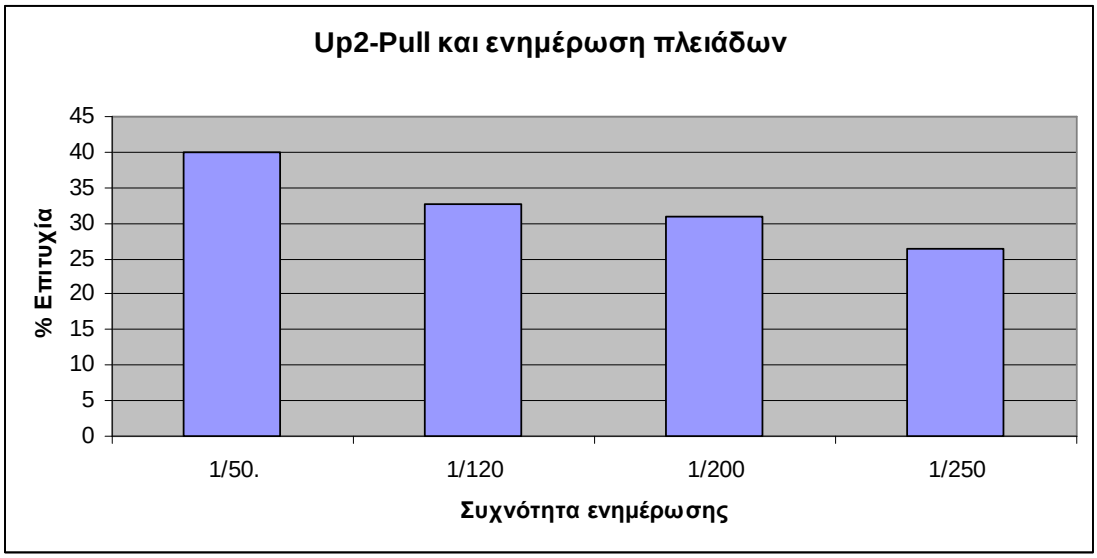


Σχήμα 6.13 Up2-Pull Παράμετροι και Μηνύματα

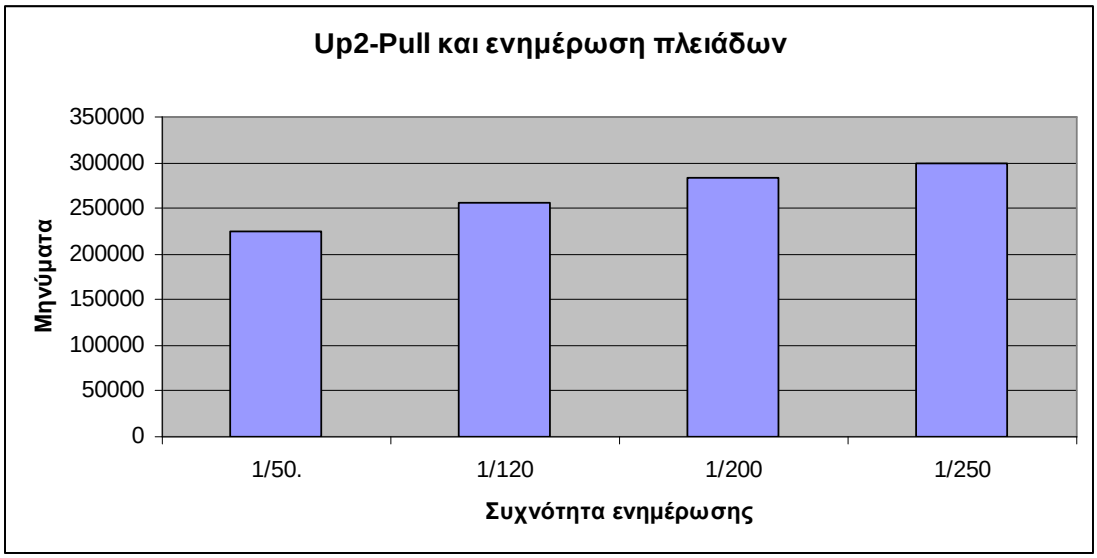
Από τα σχήματα 6.12 και 6.13 βγαίνουν τα εξής συμπεράσματα:

- Όσο αυξάνεται το Tpull η επιτυχία των ερωτημάτων μεγαλώνει, καθώς και τα μηνύματα αλλά όχι ανάλογα
- Όσο το Fpull μειώνεται, η επιτυχία μειώνεται ελάχιστα αλλά μειώνονται και τα μηνύματα του πρωτοκόλλου
- Τα μηνύματα δεν είναι πάρα πολλά: ένας κόμβος κάνει pull αν δεν έχει πλειάδες ή αν οι πλειάδες του λήγουν σύντομα. Όμως δεν παίρνει απάντηση αν οι γείτονες του δεν έχουν πλειάδες. Αν έχουν τότε τις παίρνει και είναι λογικό να μην ζητάει για λίγες χρονικές μονάδες άλλα pull
- Το πρωτόκολλο βοηθάει αρκετά στην επιτυχία των ερωτημάτων (~28-31% με Tpull<20). Δεν χρειάζεται να αυξήσουμε πολύ την παράμετρο Tpull: με Tpull>20 οι κόμβοι ζητάνε συνεχώς pull για το λόγο ότι ο μέσος όρος της ημερομηνίας λήξης των πλειάδων αρχικά είναι ~16 μονάδες
- Στην περίπτωσή μας μπορούμε να πούμε ότι Tpush=10 και Fpull=1/3 ή 1/6 είναι πολύ καλή επιλογή χωρίς πολλά μηνύματα

Το πρωτόκολλο Up2-Pull και η συχνότητα ενημερώσεων των πλειάδων: Για τη συχνότητα ενημέρωσης των πλειάδων, όμοια με το πρωτόκολλο Up2-Push έγιναν τα πειράματα του σχήματος 6.14, 6.15 για ανανέωση με συχνότητα 1/50, 1/120, 1/200, 1/250, 1/300 και με παραμέτρους Tpush=10 και Fpull=1/3.



Σχήμα 6.14 Up2-Pull και Ενημέρωση Πλειάδων ως προς την Επιτυχία των Μηνυμάτων



Σχήμα 6.15 Up2-Pull και Ενημέρωση Πλειάδων ως προς το Φόρτο Δικτύου σε Μηνύματα

Παρατηρούμε ότι όπως και στο προηγούμενο πρωτόκολλο Up2-Push, όταν η συχνότητα ενημέρωσης είναι μεγάλη, η επιτυχία των ερωτήσεων είναι επίσης μεγάλη. Όμως, όσο η συχνότητα ενημέρωσης μειώνεται τα μηνύματα αυξάνονται και αυτό γιατί όσο μικρότερη είναι, τόσο πιο γρήγορα λήγουν οι πλειάδες και ο κόμβος τότε ζητάει συχνότερα pull.

Τελικά η επιτυχία των ερωτήσεων δεν έχει μεγάλη σχέση με την αποδοτικότητα του πρωτοκόλλου. Όσο η συχνότητα ενημέρωσης αυξάνει, οι πλειάδες στο δίκτυο παραμένουν και αυξάνουν την ημερομηνία λήξης τους. Έτσι γίνονται πιο σπάνια Pull αλλά η επιτυχία των ερωτημάτων είναι μεγάλη.

Συμπερασματικά, το πρωτόκολλο Up2-Pull βοηθάει στην επιτυχία των ερωτημάτων και στην φρεσκάδα της βάσης δεδομένων, αλλά η απόδοσή του έχει επηρεάζεται περισσότερο από τις παραμέτρους Tpull, Fpull.

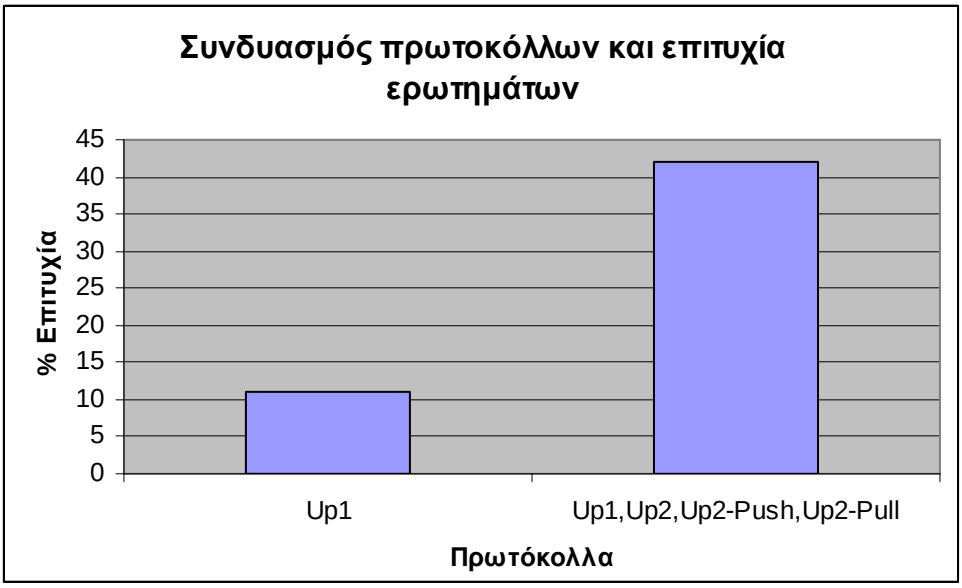
6.2.3. Συνδυασμός Όλων των Πρωτοκόλλων

Στην παρούσα φάση σκοπός είναι να συνδυάσουμε όλα τα πρωτόκολλα με τις παραμέτρους που βρήκαμε στην ενότητα 6.2.1. Ο πίνακας 6.3 δίνει αυτές τις παραμέτρους.

Πίνακας 6.3 Παράμετροι των Πρωτοκόλλων

Παράμετρος	Τιμή
TTL στο Up2	5 αν ο κόμβος έχει 3 ή 4 πλειάδες 10 αν ο κόμβος έχει 2 πλειάδες 15 αν ο κόμβος έχει 0 ή 1 πλειάδα
Tpush	25 χρονικές μονάδες
Fpush	1/6
Tpull	10 χρονικές μονάδες
Fpull	1/3

Οι παράμετροι του πίνακα 6.3 δώσανε καλά αποτελέσματα σε μεμονωμένα πειράματα των πρωτοκόλλων με λίγα μηνύματα κάθε φορά. Σκοπός είναι να συνδυαστούν όλα τα πρωτόκολλα μεταξύ τους και να διαπιστωθεί αν τελικά έχουμε καλύτερα αποτελέσματα σε σύγκριση με το πρωτόκολλο Up1. Το Up1 δεν βοηθάει στην ενημέρωση και στην αντιγραφή των φρέσκων δεδομένων στο σύστημα και άρα τα μηνύματα που φορτώνει το δίκτυο είναι μηδενικά. Τελικά παίρνουμε τα εξής αποτελέσματα του σχήματος 6.16 ως προς την επιτυχία των ερωτημάτων.



Σχήμα 6.16 Συνδυασμός Όλων των Πρωτοκόλλων και Επιτυχία Ερωτημάτων

Από το σχήμα 5.16 διαπιστώνουμε τη διαφορά της επιτυχίας μεταξύ των δύο παραμέτρων: το Up1 έχει 12% ενώ ο συνδυασμός των πρωτοκόλλων έχει 42% επιτυχία. Οπότε δεδομένου ότι το TTL των ερωτημάτων είναι χαμηλό, δεν υπάρχει δημιουργία αλλά μόνο ενημέρωση των πλειάδων, όπου και αυτή κυμαίνεται σε αρκετά χαμηλό επίπεδο, η αύξηση της επιτυχίας σε 42% κρίνεται ικανοποιητική.

Βέβαια υπάρχει και το ανάλογο κόστος σε μηνύματα όπου τα πρωτόκολλα επιβαρύνουν το δίκτυο. Στο συνδυασμό όλων των πρωτοκόλλων ανταλλάσσονται 5156583 μηνύματα που είναι αρκετά, αλλά αν αναλογιστούμε ότι έχουμε 10000 κόμβους και 120 κύκλους ρολογιού δηλαδή 5156583/1200000=4,3 μηνύματα ο κάθε κόμβος σε κάθε κύκλο ρολογιού κατά μέσο όρο.

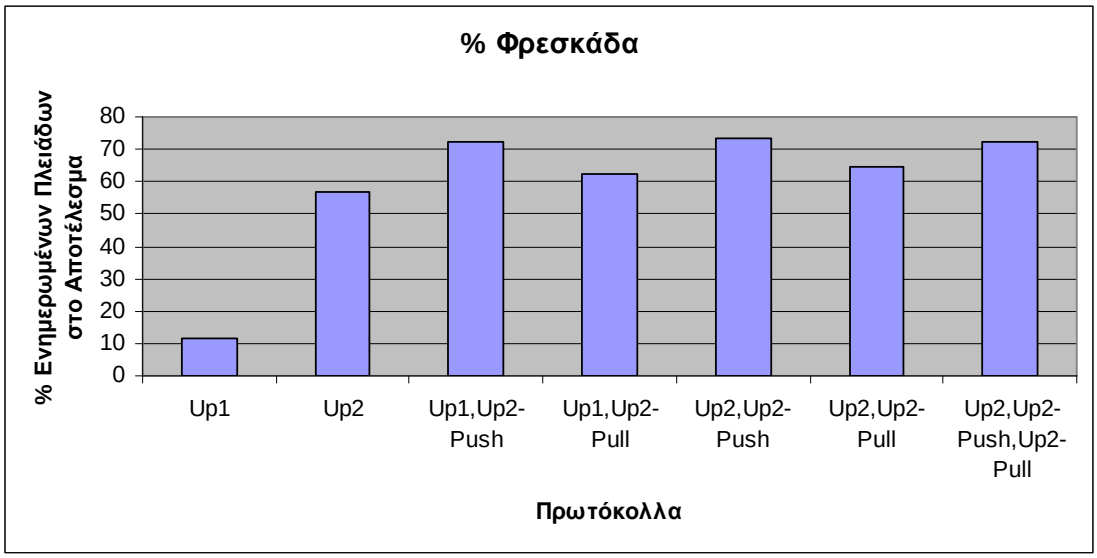
6.2.4. Συνδυασμός Πρωτοκόλλων και Ποιότητα Υπηρεσιών και Δεδομένων

Στην ενότητα αυτή θα μελετήσουμε την ποιότητα υπηρεσιών (QoS) και δεδομένων (QoD) των πρωτοκόλλων στο δίκτυο. Θα μελετήσουμε κάθε πρωτόκολλο ξεχωριστά καθώς και κάθε συνδυασμός τους. Οι παράμετροι των πρωτοκόλλων είναι αυτοί που δείχνει ο πίνακας 6.3 ενώ στις παραμέτρους του δικτύου δεν έγινε καμία αλλαγή.

- Πειραματικά αποτελέσματα ως προς την ποιότητα δεδομένων (QoD)

Για την μελέτη της ποιότητας δεδομένων, συνδυάζουμε κάθε πρωτόκολλο (και κάθε συνδυασμό πρωτοκόλλων) με την απλή αναζήτηση δεδομένων που έχουμε στο σύστημα και μετράμε τη φρεσκάδα των αποτελεσμάτων που επιστράφηκαν σαν αποτέλεσμα των αναζητήσεων. Πιο συγκεκριμένα μετράμε σαν φρεσκάδα το ποσοστό επί τοις εκατό εύρεσης της πιο «φρέσκιας» πλειάδας στο δίκτυο.

Τα πρωτόκολλα που εξετάστηκαν είναι τα πρωτόκολλα που παρουσιάστηκαν στο 6.1.2, οι συνδυασμοί τους καθώς και συνολικά όλα μαζί (Up1,Up2,Up2-Push,Up2-Pull). Σημειώνουμε, ότι παντού χρησιμοποιήθηκε το Up1 με την έννοια ότι στο δίκτυο δεν υπάρχουν πλειάδες που έχουν λήξει για να μην επιστρέφονται στο αποτέλεσμα. Το σχήμα 6.17 μας δείχνει το ποσοστό φρεσκάδας του αποτελέσματος.



Σχήμα 6.17 Ποιότητα Δεδομένων: Η Φρεσκάδα του Επιστρεφόμενου Αποτελέσματος

Από το σχήμα 6.17 παρατηρούμε ότι το Up1 δεν δίνει καλό ποσοστό ενημερωμένων πλειάδων. Αυτό συμβαίνει γιατί δεν έχουμε ενημέρωση πλειάδων ούτε και

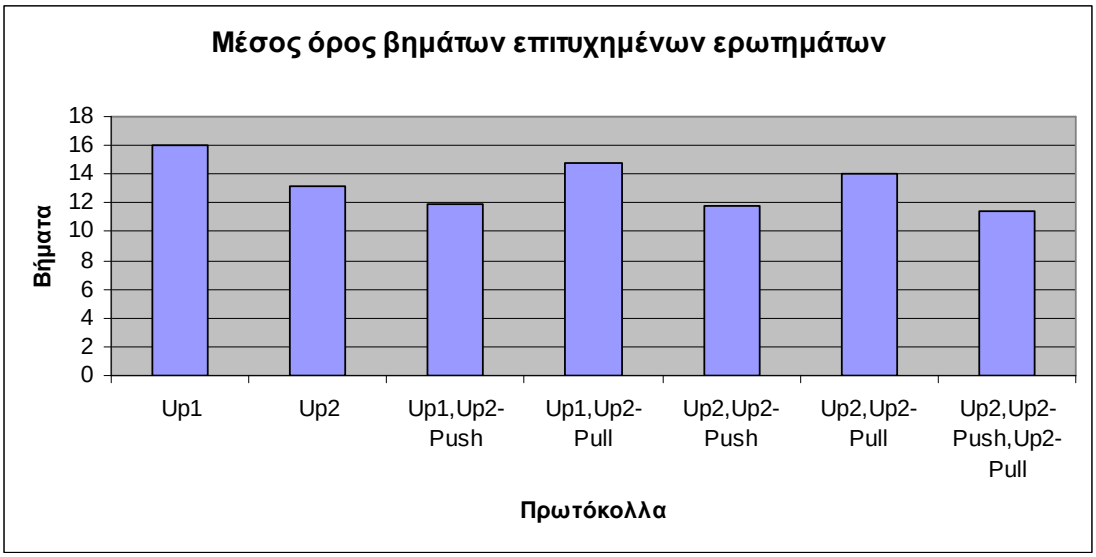
δημιουργία αντιγράφων. Το Up2 δίνει πολύ καλύτερα αποτελέσματα και ακόμα καλύτερα δίνουν τα πρωτόκολλα Up2-Push, Up2-Pull. Ο συνδυασμός πρωτοκόλλων δίνει όμοια αποτελέσματα που δεν βελτιώνουν σημαντικά τη φρεσκάδα του επιστρεφόμενου αποτελέσματος.

Τελικά παρατηρούμε ότι τα πρωτόκολλα έχουν σημαντική επίπτωση στην φρεσκάδα του επιστρεφόμενου αποτελέσματος. Το Up2 δίνει 5 φορές πιο φρέσκα δεδομένα από το Up1 ενώ η κατάσταση βελτιώνεται ελάχιστα στο συνδυασμό των πρωτοκόλλων, όπου τελικά όλα τα πρωτόκολλα μαζί δίνουν ~15% πιο φρέσκα αποτελέσματα από το Up2.

- Πειραματικά αποτελέσματα ως προς την ποιότητα υπηρεσιών (QoS)

Για την ποιότητα υπηρεσιών μετρήθηκε ο μέσος όρος βημάτων των επιτυχημένων ερωτημάτων στις τυχαίες αναζητήσεις (Message Overhead of Query Protocol) καθώς και τα μηνύματα που φορτώνουν το δίκτυο τα διάφορα πρωτόκολλα (Message Overhead of Protocol).

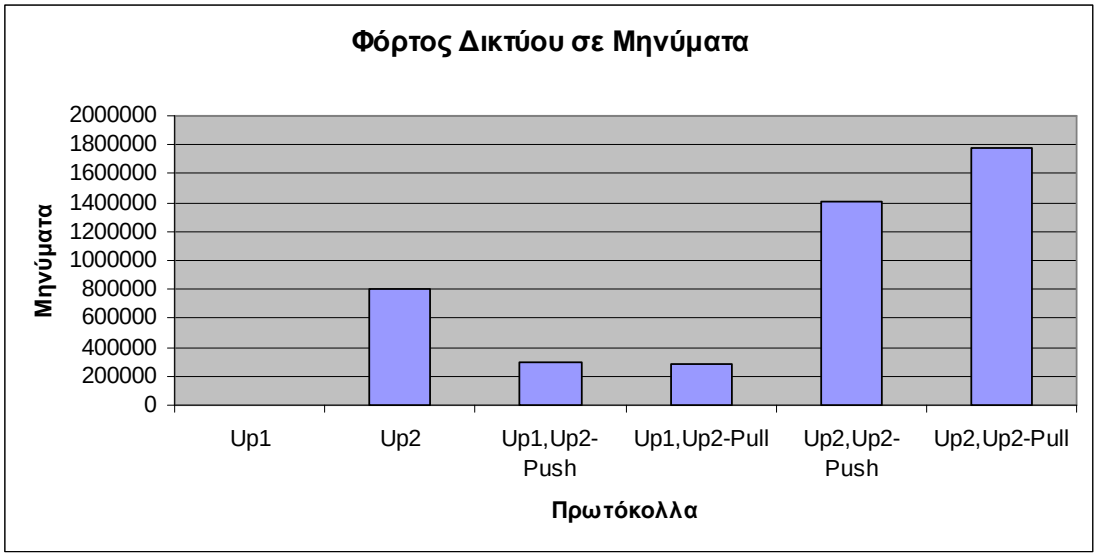
Message Overhead of Query Protocol: Μετράμε το μέσο όρο των βημάτων που χρειάζονται τα επιτυχημένα ερωτήματα κατά την περάτωση της ερώτησης καθώς στο δίκτυο είναι ενεργοποιημένα κάποιο από τα 4 διαφορετικά πρωτόκολλα ή οι τους 3 συνδυασμοί τους. Με τα βήματα μετράμε τα συνολικά βήματα που κάνουν οι περιπατητές στο δίκτυο, μέχρι την ολοκλήρωση της αναζήτησης. Το σχήμα 6.18 δίνει τα αποτελέσματα του πειράματος:



Σχήμα 6.18 Ποιότητα Υπηρεσιών: Μέσος Όρος Βημάτων Επιτυχημένων Ερωτημάτων

Από το σχήμα 6.18 παρατηρούμε τη μείωση του αριθμού των βημάτων καθώς ενεργοποιούνται τα διάφορα πρωτόκολλα στο δίκτυο. Ενδεικτικά το Up1 έχει μέσο όρο βημάτων στα επιτυχημένα ερωτήματα 16, ενώ όλα τα πρωτόκολλα συνδυασμένα έχουν 11. Αν αναλογιστούμε ότι παράγουμε 3000 ερωτήματα στους 120 γύρους του πειράματος τα συνολικά βήματα είναι κατά πολύ λιγότερα στο συνδυασμό των πρωτοκόλλων.

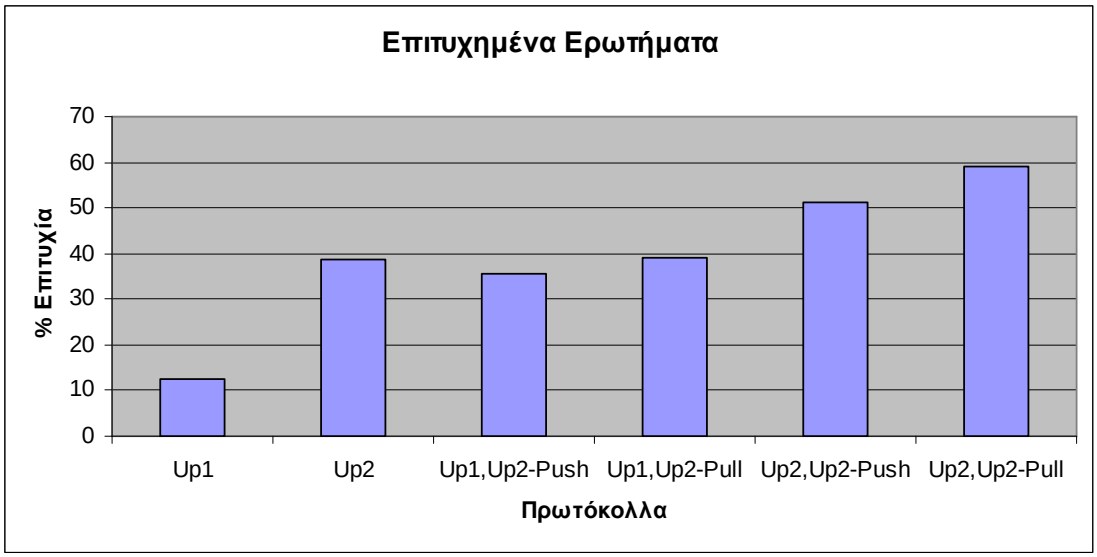
Message Overhead of Protocols: Μετράμε τα συνολικά μηνύματα που φορτώνουν το δίκτυο τα διάφορα πρωτόκολλα. Στο σχήμα 6.19 δίνεται ο φόρτος δικτύου σε μηνύματα.



Σχήμα 6.19 Ποιότητα Υπηρεσιών: Φόρτος Δικτύου σε Μηνύματα που Φορτώνουν τα Πρωτόκολλα

Από το σχήμα 6.19, παρατηρούμε ότι τα πρωτόκολλα Up2-Push, Up2-Pull παράγουν λίγα μηνύματα με το συνδυασμό παραμέτρων που έχουμε θέσει ενώ το Up2 παράγει αρκετά περισσότερα. Αυτό συμβαίνει διότι τα Up2-Push και Up2-Pull δεν ενεργοποιούνται συνεχώς και απαιτούν λίγα μηνύματα κάθε φορά. Αντιθέτων το Up2 δημιουργεί συνεχώς αναζητήσεις στο δίκτυο που η κάθε μια παράγει O(N) μηνύματα. Ο κάθε συνδυασμός των πρωτοκόλλων επιφέρει πολύ μεγάλο φόρτο δικτύου σχεδόν αθροιστικό με τα επιμέρους πρωτόκολλα. Σημειώνουμε εδώ ότι όλοι οι συνδυασμοί των πρωτοκόλλων παράγουν 5.176.848 μηνύματα, υπερδιπλάσια από τον συνδυασμό Up2, Up2-Pull. Για το λόγο αυτό δεν ενσωματώνεται ο συνδυασμός όλων των πρωτοκόλλων με τις παραμέτρους που έχουμε θέσει από εδώ και στο εξής.

Τελευταίο πείραμα είναι το πείραμα των επιτυχημένων ερωτημάτων για όλα τα πρωτόκολλα και τους συνδυασμούς τους (6 αποτελέσματα) όπως κάναμε και στις ενότητες 6.2.1, 6.2.2. Το σχήμα 6.20 δίνει τα αποτελέσματα.



Σχήμα 6.20 Πρωτόκολλα και Επιτυχημένα Ερωτήματα

Από το σχήμα 6.20 παρατηρούμε ότι τα πρωτόκολλα βοηθάνε αρκετά στην επιτυχία των ερωτημάτων. Τα πρωτόκολλα Up2, Up2-Push, Up2-Pull έχουν σχεδόν 4 φορές μεγαλύτερη επιτυχία από το Up1. Ο συνδυασμός Up2,Up2-Push αυξάνει επιπλέον την επιτυχία κατά 10% ενώ ο Up2, Up2-Pull κατά 20%.

Συμπερασματικά, μπορούμε να πούμε ότι τα πρωτόκολλα ενημέρωσης και δημιουργίας αντιγράφων σε συνδυασμό με απλά ερωτήματα αναζητήσεων στο δίκτυο προσφέρουν υψηλή ποιότητα δεδομένων με πάνω από 5 φορές πιο φρέσκα αποτελέσματα. Επίσης αυξάνονται αρκετά και τα επιτυχημένα ερωτήματα. Βέβαια, υπάρχει και το κόστος των μηνυμάτων που επιφέρουν στο δίκτυο που είναι αρκετά υψηλό. Ο συνδυασμός των πρωτοκόλλων, ενώ συμβάλλει στην υψηλότερη ποιότητα δεδομένων και στην επιτυχία των ερωτημάτων, δημιουργεί πάρα πολλά μηνύματα. Για το λόγο αυτό κάθε κόμβος πρέπει να χρησιμοποιεί δυναμικά και ακόμα πιο σπάνια το συνδυασμό των πρωτοκόλλων και μόνο όταν το χρειάζεται, προς αποφυγή μεγάλης κίνησης μηνυμάτων στο δίκτυο.

ΚΕΦΑΛΑΙΟ 7. ΣΧΕΤΙΚΗ ΕΡΓΑΣΙΑ

Στην παρούσα ενότητα θα αναλυθούν κάποια επιστημονικά άρθρα που σχετίζονται με την εργασία. Τα άρθρα αυτά ομαδοποιούνται σε κατηγορίες που είναι ποιότητα δεδομένων, ενημέρωση και αντιγραφή δεδομένων και τεχνικές ενημέρωσης χρησιμοποιώντας δειγματοληψία.

7.1. Ποιότητα Δεδομένων

Η ποιότητα των δεδομένων εκφράζεται από την ημερομηνία λήξης τους. Έτσι στα [1],[2] ορίζεται η έννοια της ημερομηνίας λήξης και μελετάται το πώς μια βάση δεδομένων μπορεί να τα υποστηρίξει και να διαχειριστεί τέτοια δεδομένα. Στο [3] δίνεται η ιδέα της προσωρινής αποθήκευσης δεδομένων “data caching” για βελτίωση της αποδοτικότητας του δικτύου.

Στο [1] παρουσιάζεται η έννοια της ημερομηνίας λήξης, που ορίζεται σαν τον χρόνο ζωής των πλειάδων που εισάγονται σε μια βάση δεδομένων, καθώς και ένα αφαιρετικό μοντέλο για το πως αυτή μπορεί να υποστηριχθεί από τη βάση έτσι ώστε ο χρήστης να μπορεί να εισάγει, να διαγράφει πλειάδες στη βάση και να κάνει ερωτήσεις χωρίς να έχει επίγνωση της ημερομηνίας λήξης. Σκοπός είναι ο χρήστης να μπορεί να παίρνει αποτελέσματα που να μην είναι “απαρχαιωμένα”. Για να γίνει αυτό η εργασία επεκτείνει το σχεσιακό μοντέλο και την σχεσιακή άλγεβρα που

χρησιμοποιείται στο σύστημα διαχείρισης βάσης δεδομένων (DBMS), έτσι ώστε να υποστηρίζει την ημερομηνία λήξης των πλειάδων.

Η εργασία αρχικά επαναπροσδιορίζει τις βασικές αρχές της βάσης δεδομένων όπως είναι η σχέση (R), οι πλειάδες (r), και τα γνωρίσματα των πλειάδων (attributes) και στη συνέχεια ενσωματώνει την ημερομηνία λήξης σαν μια χρονοσφραγίδα που προστίθεται σε κάθε πλειάδα της βάσης σαν ξεχωριστό γνώρισμα. Η χρονοσφραγίδα δηλώνει το χρόνο ζωής της πλειάδας, δηλαδή το πότε η πλειάδα παύει να είναι έγκυρη στην βάση. Έπειτα εισάγεται η έννοια της συνάρτησης $exp_{\tau}(R)$ που επιστρέφει τις πλειάδες της βάσης που δεν έχουν λήξει την τρέχουσα χρονική στιγμή. Μέσω αυτής της συνάρτησης στη συνέχεια ορίζει την σχεσιακή άλγεβρα για τους διάφορους χειριστές (selection, projection, union, aggregation, difference κλπ), το πώς θα υποστηρίξουν την ημερομηνία λήξης των πλειάδων και το πως θα ορίσουν την ημερομηνία λήξης του αποτελέσματος έτσι ώστε να υποστηριχθεί η κλειστότητα των χειριστών. Σημαντικό στοιχείο στην εργασία αυτή είναι ότι αναφέρεται μόνο στην διαχείριση πλειάδων με ημερομηνία λήξης, στον επαναυπολογισμό των αποτελεσμάτων αλλά και στην διαγραφή αυτών που έχουν λήξει, αλλά δεν αναφέρεται στην ενημέρωση πλειάδων. Η διαγραφή των πλειάδων στη βάση μπορεί να γίνει με δύο τεχνικές: eager removal όπου οι πλειάδες διαγράφονται όσο πιο σύντομα γίνεται εφόσον λήξουν και lazy removal όπου οι πλειάδες διαγράφονται ανά τακτά χρονικά διαστήματα.

Η εργασία έχει αρκετά κοινά σημεία με την παρούσα μελέτη, λόγω της έννοιας ημερομηνίας λήξης των πλειάδων που εισάγει. Στο [1] αναλύεται η διαχείριση δεδομένων με ημερομηνία λήξης και τεχνικές για την διατήρηση και διαγραφή των πλειάδων στη βάση δεδομένων. Εκτός από τις τεχνικές αυτές, χρησιμοποιήθηκε και η σχεσιακή άλγεβρα που προτάθηκε, με μερικές αλλαγές όπου ήταν χρήσιμο. Το [1] είναι η βασική εργασία όπου και στηρίχθηκε η μελέτη μας, σε σχέση με την ημερομηνία λήξης των πλειάδων, τη διαχείρισή τους, τη διαγραφή και την υποστήριξη της σχεσιακής άλγεβρας σε τέτοια δεδομένα.

Στο [2] παρουσιάζονται αλγόριθμοι και δομές δεδομένων για διαχείριση πλειάδων με ημερομηνία λήξης σε μια βάση δεδομένων. Όπως και στο [1] τα δεδομένα (οι

πλειάδες) έχουν χρονοσφραγίδα που δηλώνει το χρόνο ζωής τους. Σε ένα σύστημα όπου ο χρόνος ζωής των πλειάδων είναι μικρός και η ενημέρωσή τους πολύ συχνή, υπάρχει η ανάγκη της κατακράτησης των δεδομένων σε δομές όπου η εισαγωγή των πλειάδων να γίνεται γρήγορα και με τέτοιο τρόπο ώστε οι απαραίτητοι υπολογισμοί να ολοκληρώνονται σε σύντομο χρονικό διάστημα, ώστε το αποτέλεσμα να είναι συνεχώς ενημερωμένο.

Για το σκοπό αυτό εισάγεται μια νέα δομή δεδομένων με την ονομασία treap που συνδυάζει τη δομή του δέντρου και της σωρού. Οι αλγόριθμοι που προτείνονται είναι ένας συνδυασμός των δέντρων δυαδικής αναζήτησης ως προς το πρωτεύον γνώρισμα, αλλά και της χρήσης της σωρού ως προς το δευτερεύον γνώρισμα (ημερομηνία λήξης) δίνοντας προτεραιότητα σε πλειάδες που λήγουν. Σε αυτές τις δομές οι ερωτήσεις αναζήτησης τύπου look-up είναι πολύ γρήγορες (λογαριθμικού χρόνου) και χρησιμοποιείται η τεχνική eager διαγραφή (όπως προηγουμένως) και έτσι έχουμε μια διεργασία για λήξη πλειάδας και διαγραφή της. Επιπλέον οι δομές αυτές είναι persistent δηλαδή η ενημέρωση των δεδομένων παράγει καινούρια έκδοση της δομής χωρίς να διαφοροποιεί την προηγούμενη (άρα έχουμε κέρδος χρόνου).

Οι αλγόριθμοι που αναλύονται διεξοδικά στην εργασία είναι η εισαγωγή πλειάδων, διαγραφή και λήξη πλειάδων και η ισορροπία της δομής. Τέλος γίνεται μια αξιολόγηση των αλγορίθμων για πιο εμπεριστατωμένη μελέτη αποδοτικότητάς τους.
Στο [2] υπάρχει και πάλι η έννοια της ημερομηνίας λήξης σαν χρονοσφραγίδα που προστίθεται σε κάθε πλειάδα κατά την προσάρτησή της στη βάση δεδομένων. Συμβάλλει στην υποστήριξη της βάσης δεδομένων (DBMS) σε τέτοιες πλειάδες αλλά και στην αποδοτική διαχείρισή της, με γρήγορους αλγορίθμους εισαγωγής, διαγραφής και αναζήτησης. Αν και η μελέτη μας δεν επικεντρώθηκε στην αποδοτική διαχείριση των πλειάδων από πλευράς κάθε κόμβου ξεχωριστά, τέτοιοι αλγόριθμοι κρίνονται αναγκαίοι για την αποδοτική λειτουργία των κόμβων και τη γρήγορη ανταπόκρισή τους σε θέματα διάδοσης των ενημερώσεων, απάντησης των αιτήσεων των άλλων κόμβων του συστήματος κ.α.

Στο [3] παρουσιάζονται τεχνικές caching σε δίκτυο τύπου πελάτης-εξυπηρετητής (client-server). Σε συστήματα όπου υπάρχει μια κεντρική μονάδα στην οποία οι

χρήστες συνδέονται ώστε να ανακτήσουν τα επιθυμητά δεδομένα, υπάρχει πάντα ο κίνδυνος υπερφόρτωσης δικτύου. Πρόβλημα μπορεί να δημιουργηθεί 1) σε περίπτωση χαμηλού εύρους δικτύου, 2) όταν τα δεδομένα προστίθενται με μεγάλο ρυθμό 3) όταν οι χρήστες που συνδέονται με ένα εξυπηρετητή είναι πολλοί σε αριθμό και 4) όταν ο υπολογισμός πάνω στα δεδομένα ανά μονάδα χρόνου είναι μεγάλος. Για τη λύση τέτοιων προβλημάτων χρησιμοποιούνται πολλές τεχνικές, όπως την τεχνική caching που εστιάζεται αυτή η εργασία. Σύμφωνα με αυτήν, ο χρήστης μπορεί να αποθηκεύει τοπικά τα δεδομένα της κεντρικής μονάδας όπου χρειάζεται. Η τεχνική caching βελτιστοποιεί την απόδοση του συστήματος ως προς το φόρτο δικτύου γιατί μειώνει τις πολλαπλές αιτήσεις για ίδια δεδομένα με το ανάλογο όμως κόστος κάθε φορά: πρέπει τα δεδομένα που γίνονται cache να ενημερώνονται γρήγορα όταν αλλάξει η τιμή τους στην κεντρική μονάδα.

Η εργασία εστιάζεται στην τεχνική quasi-caching όπου τα δεδομένα γίνονται cache στο τοπικό δίσκο του χρήστη αλλά με ένα ασθενή τύπο συνέπειας: η ενημέρωσή τους δεν γίνεται από την κεντρική μονάδα άμεσα, αλλά το συντομότερο δυνατόν. Ο χρήστης διαχειρίζεται τα αντίγραφα των δεδομένων χρησιμοποιώντας δύο συνθήκες: 1)συνθήκη επιλογής, δηλαδή το να προσδιορίσει ποια δεδομένα θα κάνει cache και 2)συνθήκη συνοχής, δηλαδή το ποσοστό παρέκκλισης των τοπικών δεδομένων με τα ενημερωμένα δεδομένα στην κεντρική μονάδα.

Όσο αφορά τις συνθήκη επιλογής η καλύτερη λύση είναι να επιλέγει ο χρήστης ποια δεδομένα να γίνουν cache, πχ. με ένα απλό selection όπως είναι γνωστό στο σχεσιακό μοντέλο με επιπλέον (προαιρετικούς) τελεστές: 1.Add/Drop: που προσδιορίζει αν τα δεδομένα θα προστεθούν ή θα αφαιρεθούν από την cache. 2.Enforcement: με δύο τύπους: υποχρεωτικός όπου το σύστημα εγγυάται ότι τα δεδομένα θα γίνουν cache όπως ζητήθηκε και συμβουλευτικός όπου τα δεδομένα θα γίνουν cache μόνο όταν το σύστημα με αυτόν τον τρόπο θα αυξήσει την αποδοτικότητά του. 3.Static/Dynamic: δηλαδή στατικά και μοναδικά όταν το απαιτήσει ο χρήστης ή δυναμικά δηλαδή οι ενημερώσεις των δεδομένων πρέπει να εισχωρήσουν στην cache του χρήστη και το αποτέλεσμα να επαναπροσδιοριστεί. 4.Triggering delay: δηλαδή ο χρόνος καθυστέρησης από τη στιγμή της ενημέρωσης των δεδομένων στην κεντρική μονάδα

μέχρι να ενημερωθεί η τοπική βάση του χρήστη, σε περίπτωση της δυναμικής επιλογής.

Όσο αφορά τη συνθήκη συνοχής αποτελεί την περίοδο που τα τοπικά δεδομένα είναι ενημερωμένα (ή σχεδόν ενημερωμένα). Οι περιορισμοί που γίνονται για την ενημέρωση των δεδομένων είναι: 1.Συνθήκη καθυστέρησης δηλαδή τον χρόνο που θα κάνει ένα τοπικό αντίγραφο να ενημερωθεί σε σχέση με το αρχικό αντίγραφο του εξυπηρετητή. 2.Συνθήκη έκδοσης, όπου αντί για χρόνο εισάγουμε την έννοια της έκδοσης. Κάθε φορά που ενημερώνεται ένα δεδομένο η έκδοσή του αυξάνεται κατά ένα. Ο περιορισμός έγκειται για παράδειγμα στο ότι η τοπική βάση μπορεί να έχει δεδομένα έκδοσης έως 2 φορές πριν την πιο πρόσφατη στην κεντρική μονάδα. 3.Περιοδική συνθήκη, όπου τα δεδομένα της τοπικής βάσης μπορούν να ενημερώνονται περιοδικά οπότε γίνεται ο καθορισμός της περιόδου που πρέπει να ενημερωθούν τα δεδομένα. 4.Αριθμητική συνθήκη, δηλαδή αν τα δεδομένα είναι αριθμοί προσδιορίζεται η απόκλιση (πιο συγκεκριμένα η αφαίρεση) της τοπικής τιμής με αυτής της ενημερωμένης και πρέπει αυτή να είναι μικρότερη από ένα κατώφλι. Επιπλέον οι περιορισμοί αυτοί μπορούν να συνδυαστούν με τελεστές AND/OR/NOT.

Τέλος γίνεται μια αξιολόγηση της quasi-caching τεχνικής όπου αποδεικνύεται ότι αυξάνει την απόδοση του συστήματος και τη διαθεσιμότητα των δεδομένων.

Στο [3] μας αναλύεται η ιδέα της προσωρινής κατακράτησης δεδομένων σε μνήμη του συστήματος (“data caching”) όπου μπορεί να βελτιώσει την αποδοτικότητα του συστήματος με δύο τρόπους: μειώνει τις πολλαπλές αιτήσεις για ίδια δεδομένα και μειώνει το φόρτο μιας κεντρικής μονάδας αποθηκεύοντας τοπικά τα δεδομένα και αναλύοντας τα όπως ο χρήστης επιθυμεί. Αν και στο [3] αναλύεται η τεχνική caching σε συστήματα πελάτη-εξυπηρετητή, μια τέτοια ιδέα είναι πολύ χρήσιμη και στο σύστημα που μελετάμε χρησιμοποιώντας τη λανθάνουσα μνήμη του κόμβου για προσωρινή αποθήκευση δεδομένων. Τέτοια δεδομένα μπορεί να είναι πρόσφατες ερωτήσεις άλλων κόμβων, αν ο κόμβος βρίσκεται στο μονοπάτι της ερώτησης, για μελλοντική χρησιμοποίησή τους και βελτίωση απόδοσης του συστήματος. Επιπλέον δίνονται και τεχνικές για ενημέρωση της μνήμης cache για να είναι τα δεδομένα ενημερωμένα όπως χρειάζεται.

7.2. Ενημέρωση και Αντιγραφή Δεδομένων

Οι τεχνικές και αλγόριθμοι για ενημέρωση και αντιγραφή των δεδομένων αναλύονται στα [5], [6], [7], [8]. Δίνονται τεχνικές για διάδοση των ενημερώσεων, μελετάται το πλήθος των αντιγράφων των ενημερώσεων που πρέπει να γίνουν καθώς και παρουσιάζονται αλγόρίθμοι για να γίνει η ενημέρωση των κόμβων στο σύστημα.

Στο [5] περιγράφονται στρατηγικές ενημέρωσης των δεδομένων σε ένα μη κεντρικοποιημένο και αδόμητο δίκτυο ομότιμων κόμβων, το οποίο είναι σε μεγάλο βαθμό αναξιόπιστο (δυναμικό): οι κόμβοι εισέρχονται και εξέρχονται ακατάπαυστα, έχουν μόνο τοπική γνώση του συστήματος και όχι ολική και η πιθανότητα κάποιος κόμβος να πέσει είναι πολύ μεγαλύτερη από αυτήν να είναι διαθέσιμος. Τα δεδομένα ενημερώνονται από το κάθε κόμβο ξεχωριστά και αυθαίρετα οπότε σκοπός είναι να γίνουν όσο περισσότερα αντίγραφα γίνεται στο δίκτυο (replication) για λόγους διαθεσιμότητας και ανοχής σφαλμάτων. Η λύση που προτείνεται είναι ένας επιδημικός push/pull rumor spreading αλγόριθμος με σκοπό να γίνει αποδοτική ενημέρωση του δικτύου με όσο το δυνατό μικρότερο φόρτο εργασίας. Στόχος είναι μια πιθανοτική εγγύηση διάδοσης των ενημερωμένων αρχείων και όχι αυστηρή συνέπεια.

Ο αλγόριθμος push/pull rumor spreading βασίζεται σε δύο φάσεις push και pull. Κατά την push φάση ο κόμβος ο οποίος έχει μια ενημερωμένη έκδοση κάποιου αρχείου την προωθεί σε γείτονές του με μέθοδο της πλημμύρας. Παράλληλα προωθούνται και άλλες πληροφορίες όπως η έκδοση του αρχείου ποιοι κόμβοι έχουν το αρχείο κ.α. Έτσι όταν κάποιος κόμβος παίρνει αυτή την ενημερωμένη αίτηση την προωθεί σε γειτονικούς κόμβους που δεν την έχουν ακόμα λάβει, με μια πιθανότητα PF(t). Στην pull φάση παίρνουν μέρος κόμβοι οι οποίοι είχαν αποσυνδεθεί από το δίκτυο για αρκετό καιρό, κόμβοι οι οποίοι για κάποιο λόγο δεν έχουν λάβει ενημερωμένα αρχεία για μεγάλο χρονικό διάστημα ή κόμβοι οι οποίοι είχαν κάνει pull αίτηση αλλά δεν έλαβαν το πιο ενημερωμένο αρχείο. Στη φάση αυτή οι κόμβοι κάνουν αίτηση σε έναν ή περισσότερους γειτονικούς κόμβους για να λάβουν τις ενημερώσεις, με δεδομένο ότι αυτοί είναι για μεγαλύτερο χρονικό διάστημα διαθέσιμοι άρα και πιο ενημερωμένοι.

Η εργασία συνεχίζει με μια θεωρητική μελέτη του μοντέλου και ανάλυση του αλγορίθμου παρουσιάζοντας κάποια αποτελέσματα.

Το σύστημα που περιγράφεται στο [5] έχει αρκετές ομοιότητες με το σύστημα που μελετάμε: είναι αρκετά δυναμικό και συγκαταλέγεται στα αδόμητα, μη κεντρικοποιημένα peer-to-peer συστήματα. Έτσι και οι τεχνικές αντιγραφής (replication) και ενημέρωσης (updates) που προτείνονται μας είναι πολύ χρήσιμες. Οι τεχνικές push/pull διαδίδουν τις ενημερώσεις από κόμβο σε κόμβο με μικρό φόρτο δικτύου με αποτέλεσμα τη μεγάλη διαθεσιμότητα των ενημερώσεων για λόγους ανοχής σφαλμάτων και χρόνου απόκρισης. Επιπλέον όπως αναλύθηκε, στο σύστημά μας, τέτοιες τεχνικές μειώνουν κατά πολύ το φόρτο δικτύου σε πιθανές ερωτήσεις των ομότιμων γι’ αυτά τα δεδομένα.

Στο [6] περιγράφονται στρατηγικές αναζήτησης και δημιουργίας αντιγράφων σε μη κεντρικοποιημένα και αδόμητα δίκτυα ομότιμων κόμβων. Η τοπολογία δικτύου που χρησιμοποιούν είναι ο τυχαίος γράφος (random graph) όπου όλοι οι κόμβοι έχουν περίπου τον ίδιο αριθμό συνδέσεων, δηλαδή τον ίδιο αριθμό γειτονικών κόμβων. Σε αυτή την τοπολογία ο μέγιστος φόρτος εργασίας ενός κόμβου είναι λογαριθμικός του συνολικού αριθμού των κόμβων που συναντά η αναζήτηση.

Για την αναζήτηση προτείνει εναλλακτικές στρατηγικές από την μέθοδο της πλημμύρας (flooding-based query algorithm) που χρησιμοποιεί το δίκτυο Gnutella για διάφορους λόγους: 1) Δεν βοηθάει στην επεκτασιμότητα του δικτύου. 2) Δημιουργεί μεγάλο φόρτο στο δίκτυο γιατί χρησιμοποιεί το χρόνο ζωής της ερώτησης (TTL) που όταν αυτό είναι μεγάλο δημιουργεί φόρτο, ενώ όταν είναι μικρό δεν έχει τα επιθυμητά αποτελέσματα. 3) Δημιουργεί διπλά μηνύματα που σημαίνει ότι η ερώτηση μπορεί να περάσει πάνω από μια φορά από ένα κόμβο. Έτσι για να ξεπεραστούν αυτά τα προβλήματα προτείνονται δύο εναλλακτικές μέθοδοι αναζήτησης: Η μέθοδος expanding ring και random walks. Η μέθοδος expanding ring χρησιμοποιεί την μέθοδο της πλημμύρας αλλά με αυξανόμενο χρόνο ζωής (TTL) στην αναζήτηση. Η αναζήτηση αρχικά έχει TTL=1 και στην συνέχεια το TTL αυξάνει ανά δύο κάθε φορά μέχρι να έρθουν τα επιθυμητά αποτελέσματα. Έπειτα η

αναζήτηση τερματίζει και δεν έχουμε επιπλέον φόρτο στο δίκτυο. Έτσι αποδεικνύεται ότι έχουμε λιγότερο φόρτο στο δίκτυο από την μέθοδο της πλημμύρας, αλλά ούτε και αυτή η μέθοδος μειώνει τα διπλά μηνύματα. Με τη μέθοδο random walks το μήνυμα της αναζήτησης προωθείται σε κάποιον τυχαία επιλεγμένο γειτονικό κόμβο σε κάθε βήμα και ονομάζουμε αυτό το τύπο του μηνύματος “walker”, μέχρι να βρεθούν τα επιθυμητά αποτελέσματα. Επιπλέον μπορούμε να αυξήσουμε τον αριθμό των “walkers” με σκοπό πιο γρήγορα αποτελέσματα. Γι’ αυτό και η τεχνική ονομάζεται και multiple random walks. Σε αυτή ενσωματώνεται και ένας μηχανισμός τερματισμού που μπορεί να είναι είτε ο αριθμός των βημάτων (χρόνος ζωής TTL) του κάθε walker, είτε δυναμικά κάθε φορά πριν ο walker κάνει ένα βήμα να ρωτάει το κόμβο που δημιούργησε το μήνυμα της αναζήτησης για το αν θα συνεχίσει ή όχι.
Για την δημιουργία αντιγράφων προτείνονται τρεις μέθοδοι: uniform, proportional και square root με δεδομένο ότι ο συνολικός χώρος που διατίθεται για την αποθήκευση των αντιγράφων είναι σταθερός. Κατά την ομοιόμορφη (uniform) μέθοδο δημιουργούνται ίδιος αριθμός από αντίγραφα για κάθε αντικείμενο. Όμως κάτι τέτοιο δεν είναι αποτελεσματικό γιατί δημιουργούνται αντίγραφα ίσου αριθμού, ανεξάρτητα από τη συχνότητα που ζητούνται από τους κόμβους. Έτσι κατά την αναλογική (proportional) μέθοδο δημιουργούνται αντίγραφα ανάλογα με το ρυθμό ζήτησης του κάθε αντικειμένου από τα διάφορα queries. Οι δύο αυτές μέθοδοι έχουν τον ίδιο μέσο όρο αναζήτησης τελικά. Τέλος η μέθοδος της τετραγωνικής ρίζας (square root) είναι μια βελτιστοποίηση των προηγούμενων δύο μεθόδων με μικρότερο μέσο χρόνο αναζήτησης κάποιου αντικειμένου. Σύμφωνα με τη μέθοδο αυτή τα αντίγραφα που δημιουργούνται είναι ανάλογα της τετραγωνικής ρίζας της πιθανότητας του αντικειμένου να ζητηθεί από κάποιο query. Η τεχνική αυτή αποδεικνύεται ότι είναι πιο αποδοτική.

Το [6] παρουσιάζει τις πιθανές μεθόδους αναζήτησης σε ένα peer-to-peer δίκτυο με τα πλεονεκτήματα και τα μειονεκτήματα που αυτές έχουν. Στην μελέτη μας χρησιμοποιήθηκε η μέθοδος της πλημμύρας (flooding) και των τυχαίων διαδρομών (random walks) όπως αναλύονται στο [6] με σημαντικό θέμα την διαφορά των μηνυμάτων (φόρτο δικτύου) και τα διπλά μηνύματα που τελικά τα βελτιώνει η μέθοδος random walks που χρησιμοποιήθηκε στην εφαρμογή μας. Επίσης το πλήθος των αντιγράφων που χρειάζονται το κάθε δεδομένο στο δίκτυο και η αναγκαιότητα να

είναι συσχετισμένο με τη δημοφιλία του, φαίνεται χρήσιμο αν συνδυαστεί με το [8] για τον τρόπο που πρέπει να κατανεμηθούν τα αντίγραφα κατά μήκος του δικτύου σε ένα σύστημα όπως είναι αυτό που μελετάμε.

Στο [7] παρουσιάζονται επιδημικοί αλγόριθμοι για κατανομή ενημερώσεων και δημιουργία αντιγράφων μεταξύ κόμβων ενός δικτύου. Στο σύστημά μας έχουμε μια βάση δεδομένων που θέλουμε να δημιουργήσουμε αντίγραφά της σε ένα μεγάλο, ετερογενές, αναξιόπιστο και μεταβαλλόμενο δίκτυο εκατοντάδων/χιλιάδων κόμβων. Σκοπός είναι να δημιουργηθούν αντίγραφα της βάσης όσο το δυνατόν πιο ενημερωμένα κατά μήκος του δικτύου και σε όσο το δυνατόν συντομότερο χρονικό διάστημα με στόχο τη συνέπεια του συστήματος.

Οι μέθοδοι που προτείνουν για διάδοση των ενημερώσεων μεταξύ των κόμβων είναι:
α) Direct mail: Σύμφωνα με αυτή την μέθοδο κάθε καινούρια ενημέρωση αντικειμένου σε κάποιο κόμβο στέλνεται αμέσως σε όλους τους υπόλοιπους κόμβους του συστήματος. Αυτή η μέθοδος είναι αποτελεσματική αλλά υπάρχει η περίπτωση να χαθεί κάποια ενημέρωση στην περίπτωση που κάποιος κόμβος δεν έχει ολική γνώση του συστήματος δηλαδή δεν γνωρίζει όλους τους κόμβους του δικτύου. Σημαντικό είναι ότι οι ενημερώσεις γίνονται μέσω mail όπου τα mail αυτά κατακρατούνται αποτελεσματικά από τον mail-server και δεν υπάρχει καμία επίπτωση αν κάποιος κόμβος πέσει: μόλις ξανασυνδεθεί θα λάβει την ενημέρωση, αρκεί να συμβεί σε σύντομο χρονικό διάστημα έτσι ώστε να μην χαθούν οι ενημερώσεις λόγω υπερχείλισης.

β) Anti-entropy: Κάθε κόμβος τακτικά διαλέγει κάποιο άλλο κόμβο τυχαία, ανταλλάσσοντας τα περιεχόμενα της βάσης δεδομένων του με σκοπό να εντοπιστούν οι διαφορές και να ενημερωθούν οι βάσεις με τα πιο πρόσφατα δεδομένα μεταξύ των δύο κόμβων. Η ενημέρωση των κόμβων μπορεί να γίνει είτε προς τη μια κατεύθυνση ή προς την άλλη είτε αμφίδρομα. Αν ο αρχικός κόμβος είναι πιο ενημερωμένος έχουμε τη φάση push, το αντίθετο θα φέρει τη φάση pull, ενώ έχουμε και αμφίδρομη φάση push/pull. Η διεργασία αυτή συνεχίζεται μέχρι να ενημερωθούν όλοι οι κόμβοι. Η μέθοδος αυτή είναι αρκετά αξιόπιστη αλλά με μεγάλο κόστος καθώς πρέπει να ελεγχθούν όλα τα περιεχόμενα των βάσεων δεδομένων μεταξύ δύο κόμβων. Για το

λόγο αυτό δεν θα πρέπει να χρησιμοποιείται τόσο συχνά. Μια βελτιστοποίηση είναι να χρησιμοποιεί κάθε κόμβος ένα checksum το οποίο να αυξάνεται όταν ενημερώνονται τα δεδομένα του. Έτσι μπορεί να προλάβει μια ενδεχόμενη αργοπορημένη ενημέρωση. Επιπλέον με τη μέθοδο anti-entropy οι ενημερώσεις διαδίδονται πιο αργά από τη direct mail.

γ) Rumor mongering: αρχικά ο κόμβος είναι “ignorant”, δεν γνωρίζει δηλαδή την ύπαρξη ενημερώσεων. Όταν ένας κόμβος λάβει μια ενημέρωση γίνεται “hot rumor” και όση ώρα είναι σε αυτή τη φάση, περιοδικά επιλέγει τυχαία άλλο κόμβο κάθε φορά για να σιγουρευτεί ότι και οι άλλοι έχουν ενημερωθεί. Όταν δει ένα ικανοποιητικό αριθμό από κόμβους να είναι ενημερωμένοι τότε χάνει την ιδιότητα “hot rumor” και σταματάει να ενημερώνει κόμβους πια. Η μέθοδος αυτή είναι πιο γρήγορη από την anti-entropy αλλά και πάλι υπάρχει ο κίνδυνος να οι ενημερώσεις να μην διαδοθούν σε όλους τους κόμβους.

Η εργασία συνεχίζει με τον τρόπο διαγραφής “απαρχαιωμένων” δεδομένων από τη βάση όλων των κόμβων. Αυτό δεν μπορεί να γίνει με απλή διαγραφή στη τοπική βάση γιατί οι άλλοι κόμβοι θα το αντιληφθούν και θα επαναφέρουν το “χαμένο” αρχείο. Έτσι προτείνεται η μέθοδος των “death certificates”. Τα “death certificates” είναι αρχεία που περιέχουν χρονοσφραγίδες για τα δεδομένα της βάσης και διαδίδονται σαν κανονικά δεδομένα. Καθώς αυτά διαδίδονται όταν κάποιο απαρχαιωμένο αντίγραφο τα συναντήσει τότε διαγράφεται από τη βάση. Για το πρόβλημα του πότε θα διαγραφεί ένα death certificate έτσι ώστε να μην γεμίσουν οι αποθηκευτικοί χώροι των κόμβων, μια λύση είναι και αυτά να έχουν χρόνο ζωής για παράδειγμα 30 μέρες. Μια τροποποίηση των death certificates είναι τα “dormant death certificates” που βασίζονται στην παρατήρηση ότι αν τα death certificates έχουν χρόνο ζωής μεγαλύτερο από το χρόνο που χρειάζεται για να διαδοθούν σε όλο το σύστημα τότε όλα τα απαρχαιωμένα δεδομένα στο σύστημα θα διαγραφούν.
Στο [7] αναλύονται μέθοδοι διάδοσης και διαγραφής των ενημερώσεων που όμως δεν μπορούν να εφαρμοστούν στο δικό μας σύστημα. Χρησιμοποιούν μεγάλο εύρος δικτύου για τις διαδόσεις των ενημερώσεων, σε πολλές περιπτώσεις μεταφέρεται ολόκληρη η βάση δεδομένων του κόμβου ή ακόμα συνεργάζονται πάρα πολλοί κόμβοι για τη διάδοση μιας και μόνης ενημέρωσης. Κάτι τέτοιο στο σύστημά μας δεν

είναι εφικτό για το λόγο ότι οι κόμβοι είναι αρκετές χιλιάδες, δεν έχουν ολική γνώση του συστήματος και η κατανεμημένη βάση δεδομένων είναι πολύ μεγάλη. Ενδιαφέρει η διάδοση των ενημερώσεων όχι όμως χρησιμοποιώντας όλο το διαθέσιμο εύρος δικτύου, ούτε και η αντιγραφή τους σε όλους τους κόμβους: αρκεί ένα μικρό ποσοστό έτσι ώστε να μπορεί να ανακτήσει την ενημέρωση κάθε κόμβος που το επιθυμεί θέτοντας ερώτηση με μικρό βάθος δικτύου.

Στο [8] αναλύονται διάφορες στρατηγικές για replication σε μη κεντρικοποιημένα και αδόμητα δίκτυα ομότιμων κόμβων. Σκοπός είναι να βρεθεί ο καλύτερος τρόπος για δημιουργία αντιγράφων στο δίκτυο έτσι ώστε η αναζήτηση κάποιου δεδομένου να μπορεί να ολοκληρωθεί γρηγορότερα (με μικρό βάθος δικτύου “path-length”) και αποτελεσματικότερα, δοσμένου ότι ο κάθε κόμβος έχει σταθερό χώρο αποθήκευσης.
Οι τρεις στρατηγικές για replication που παρουσιάζονται αναλυτικά είναι η uniform, η proportional, και η square-root. Κατά τη uniform όλα τα δεδομένα αντιγράφονται στους κόμβους ομοιόμορφα και έτσι όλα τα δεδομένα έχουν τον ίδιο αριθμό από αντίγραφα ανεξάρτητα από την δημοφιλία τους. Ενώ κατά την proportional τα δεδομένα αντιγράφονται στους κόμβους ανάλογα με το ποσοστό της δημοφιλίας τους. Όσο περισσότερες αιτήσεις υπάρχουν για κάποιο δεδομένο τόσα περισσότερα αντίγραφα δημιουργούνται με στόχο την γρηγορότερη εξυπηρέτηση τους. Παρατηρούμε ότι με την πρώτη τεχνική όλες οι αναζητήσεις έχουν ίδιο μέσο χρόνο ολοκλήρωσης. Στην proportional ο χρόνος αναζήτησης δημοφιλών δεδομένων είναι πολύ μικρότερος από αυτόν για τα μη-δημοφιλή δεδομένα. Όμως και στις δύο στρατηγικές ο μέσος χρόνος αναζήτησης δεδομένων είναι ίδιος. Για το λόγο αυτό προτείνεται η square-root τεχνική, όπου βρίσκεται ανάμεσα στις δύο προηγούμενες, αλλά τα αποτελέσματά της είναι καλύτερα από αυτές, με το να μειώνει το χρόνο της χειρότερης επιτυχημένης αναζήτησης και να σταθεροποιεί το χρόνο της μικρότερης αναζήτησης. Αυτό επιτυγχάνεται με το να δημιουργούνται αντίγραφα ανάλογα με τη τετραγωνική ρίζα της δημοφιλίας του εκάστοτε δεδομένου.
Για την υλοποίηση της square root τεχνικής προτείνονται τρία πρωτόκολλα: path replication, replication with sibling-number memory και replication with probe memory. Το πρωτόκολλο “path replication” θέτει τον αριθμό των αντιγράφων να είναι ίσος με το μέγεθος της αναζήτησης δηλαδή με τον αριθμό των κόμβων που περιλαμβάνονται για την ολοκλήρωσή της. Με το replication with sibling-number

memory (SNM) για κάθε αντίγραφο που δημιουργείται κρατάμε πληροφορίες για τον αριθμό των αντιγράφων, το πότε αντιγράφτηκε και το χρόνο δημιουργίας του. Με το replication with probe memory κάθε κόμβος καταγράφει για κάθε δεδομένο που του έχει ζητηθεί έστω μια φορά, το πλήθος των αναζητήσεων που του έχουν γίνει και το μέγεθος της αναζήτησης που έχει το δεδομένο αυτό μέχρι την επιτυχημένη εύρεσή του.

Στο [8] όπως και στο [6] μελετάται το πλήθος των αντιγράφων που πρέπει να υπάρχει στο δίκτυο για να μπορεί μια ενημέρωση να είναι εύκολα ανακτήσιμη από οποιονδήποτε κόμβο. Δηλαδή, πόσα πρέπει να είναι τα αντίγραφα έτσι ώστε να μπορεί να τα ανακτήσει κάθε κόμβος εύκολα με αναζήτηση με μικρό βάθος δικτύου. Έτσι αφού καταλήγει στο συμπέρασμα ότι το πλήθος των αντιγράφων πρέπει να είναι ανάλογο της τετραγωνικής ρίζας της δημοφιλίας του (square-root), δίνονται και τρία πρωτόκολλα για την υλοποίηση αυτής της τεχνικής. Στην μελέτη μας χρησιμοποιήθηκε ο αλγόριθμος “path replication” που δίνει τη “square-root” και είναι εύκολα υλοποιήσιμος.

7.3. Τεχνικές Ενημέρωσης Χρησιμοποιώντας Δειγματοληψία

Στο [4] παρουσιάζεται ένα παρόμοιο μοντέλο όπου υπάρχουν πολλοί κόμβοι-εξυπηρετητές (servers) που ανανεώνουν τα δεδομένα τους, και άλλοι κόμβοι-πελάτες (χρήστες, servers) που χρειάζονται να κάνουν τοπικά αντίγραφα δεδομένων όπου χρειάζεται. Η πρόκληση είναι να εντοπίσει ο πελάτης ποια δεδομένα ανανεώθηκαν και ποιος εξυπηρετητής τα προσφέρει, έτσι ώστε να δημιουργήσει τοπικά αντίγραφα. Το πρόβλημα είναι ότι από πλευράς πελάτη οι υπολογιστικοί και οι δικτυακοί πόροι είναι περιορισμένοι οπότε πρέπει να βρεθεί μια τεχνική αποδοτικού εντοπισμού των ενημερωμένων δεδομένων. Η λύση που προτείνεται η δειγματοληψία (sapling): αρχικά ο πελάτης κατεβάζει ένα μικρό υποσύνολο δεδομένων από κάθε εξυπηρετητή και έπειτα αποφασίζει ποιά πηγή έχει τα περισσότερα ενημερωμένα δεδομένα έτσι ώστε να τα κατεβάσει από εκεί. Εναλλακτικές στρατηγικές όπου μπορούν να συνδυαστούν και ανά 2 είναι: 1)round-robin, όπου ο πελάτης κατεβάζει όλα τα δεδομένα από κάθε εξυπηρετητή σειριακά. Η στρατηγική αυτή έχει μεγάλο κόστος

και το κίνδυνο να περισσότερες ενημερώσεις μέχρι να κατέβουν όλα τα δεδομένα. 2)Change-frequency-based, όπου γίνεται εκτίμηση της συχνότητας ενημέρωσης των δεδομένων από τον κάθε εξυπηρετητή και δημιουργούνται αντίγραφα με την ίδια συχνότητα από κάθε πηγή.

Η εργασία βασίζεται στην τεχνική της δειγματοληψίας παρουσιάζει δύο πολιτικές: greedy και proportional sampling. Αρχικά ο πελάτης κατεβάζει ένα σύνολο δεδομένων από κάθε εξυπηρετητή και υπολογίζει το ποσοστό των ενημερωμένων δεδομένων του. Έπειτα κατά την proportional πολιτική κατεβάζει αναλογικά τα υπόλοιπα δεδομένα. Για παράδειγμα αν μπορεί να κατεβάσει 80 δεδομένα από 2 εξυπηρετητές, μπορεί να κατεβάσει 60 από τον πρώτο και 20 από τον δεύτερο αν υπολογίσει ότι ο πρώτος έχει ποσοστό ¾ των συνολικών ενημερωμένων δεδομένων. Κατά την greedy πολιτική ο εξυπηρετητής που έχει το μεγαλύτερο ποσοστό των ενημερωμένων δεδομένων κερδίζει, ο πελάτης κατεβάζει όλα τα υπόλοιπα δεδομένα από αυτόν και έπειτα να συνεχίσει με τους υπόλοιπους. Στο προηγούμενο παράδειγμα ο πελάτης θα κατεβάσει όλα τα δεδομένα από τον πρώτο και σε περίπτωση που τα δεδομένα που παρέχει είναι λιγότερα από 80 συνεχίζει με τον δεύτερο.

Η παρούσα εργασία παρουσιάζει σαν πιο αποδοτική την greedy πολιτική και έτσι προτείνει την βελτιστοποιημένη μέθοδο: ο πελάτης κάνει δειγματοληψία από κάθε πηγή ίσου αριθμού από τυχαία δεδομένα και έπειτα όλοι οι πόροι του συστήματος δεσμεύονται για κατέβασμα από τον πιο ενημερωμένο εξυπηρετητή, έπειτα από τον λιγότερο κλπ.

Τέλος η εργασία επικεντρώνεται στο πρόβλημα του ποσοστού των δεδομένων που θα αφιερώσει ο πελάτης στη δειγματοληψία σε σχέση με τους πόρους δικτύου που διαθέτει. Έτσι το ποσοστό μπορεί να είναι αναλογικό, για παράδειγμα το 1/10 των πόρων του συστήματος ή δυναμικό δηλαδή σταματάει η δειγματοληψία αν κάποια πηγή κερδίσει νωρίς.

Στο [4] παρουσιάζεται μια τεχνική ενημέρωσης των δεδομένων χρησιμοποιώντας δειγματοληψία. Αν και η τεχνική αυτή είναι αρκετά αποδοτική όταν υπάρχουν διάφοροι εξυπηρετητές και πελάτες, στο σύστημά μας δεν μπορεί να εφαρμοστεί. Το

σύστημά που μελετάμε ανήκει στα peer-to-peer όπου δεν υπάρχει ολική γνώση του συστήματος από τον κάθε ομότιμο και επιπλέον έστω και στη μερική γνώση που έχει κάθε κόμβος αν εφαρμοζόταν δειγματοληψία, θα επέφερε μεγάλο φόρτο δικτύου που θα επιβάρυνε το σύστημα με αρκετά μηνύματα.

ΚΕΦΑΛΑΙΟ 8. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Στα πλαίσια της διατριβής, εξετάσαμε την ενημερότητα των δεδομένων σε κατανεμημένες βάσεις δεδομένων. Το σύστημα που μελετήσαμε εντάσσεται στα αδόμητα, μη κεντρικοποιημένα δίκτυα ομότιμων κόμβων.

Ένας τομέας που ασχοληθήκαμε στην εργασία είναι η υποστήριξη SQL ερωτημάτων σε τέτοια δίκτυα. Αρχικά έγινε επέκταση των SQL ερωτημάτων και προστέθηκαν νέοι τελεστές που δηλώνουν το βάθος δικτύου της ερώτησης, το χρονισμό της ερώτησης (ad hoc ή continuous) και το κατώφλι της ημερομηνίας λήξης του αποτελέσματος που επιθυμούμε να έχουμε. Επίσης προτάθηκαν αλγόριθμοι έτσι ώστε να υπολογίζονται κατανεμημένα οι λειτουργίες συνάθροισης (aggregation) και συνένωσης (join). Ένα σημαντικό κομμάτι των αλγορίθμων αποτελούσε και ο υπολογισμός της ημερομηνίας λήξης του αποτελέσματος. Στον υπολογισμό αυτό, στις λειτουργίες συνάθροισης, πρέπει να ληφθεί υπόψη για το αν γίνονται ενημερώσεις πλειάδων στο σύστημα ή όχι. Όταν αναφερόμαστε σε ενημερώσεις υπονοείται το γεγονός ότι οι κόμβοι μπορούν να ενημερώσουν ένα υποσύνολο των γνωρισμάτων των πλειάδων και μαζί την ημερομηνία λήξης τους ή να δημιουργήσουν νέες πλειάδες στην τοπική βάση δεδομένων τους. Στην περίπτωση που υπάρχουν ενημερώσεις, δεν υπάρχει η έννοια της ημερομηνίας λήξης του αποτελέσματος για το λόγο ότι το αποτέλεσμα μπορεί να μεταβληθεί ανά πάσα στιγμή. Υπάρχει όμως η έννοια της ημερομηνίας λήξης της πλειάδας ή των πλειάδων που δίνονται στο αποτέλεσμα. Στην διαφορετική περίπτωση που δεν υπάρχουν ενημερώσεις στο σύστημα δίνεται ρητά ο

τρόπος υπολογισμού της ημερομηνίας λήξης του αποτελέσματος, δηλαδή, το χρονικό διάστημα που θα ισχύει το εν λόγω αποτέλεσμα.

Στην εργασία μας, για την προώθηση των SQL ερωτημάτων καθώς και της αναζήτησης πλειάδων, μελετήθηκε η χρήση της μεθόδου της πλημμύρας (flooding) και των τυχαίων διαδρομών (random walks). Λόγω των προβλημάτων που έχει η μέθοδος της πλημμύρας, ο εκθετικά αυξανόμενος αριθμός μηνυμάτων σε κάθε βήμα της αναζήτησης, η αύξηση των κύκλων και των διπλών μηνυμάτων καθώς και η αδυναμία υπολογισμού ορισμένων SQL ερωτημάτων σε διαφορετικά μονοπάτια, προτείνεται η μέθοδος των τυχαίων διαδρομών. Σε αυτές τις δύο μεθόδους έγινε μελέτη όσο αναφορά την απόδοσή τους, το φόρτο δικτύου σε μηνύματα και τον αριθμό βημάτων που χρειάζονται για την ολοκλήρωσή τους.

Ένας άλλος τομέας που μελετήθηκε στην εργασία είναι η δημιουργία αντιγράφων των ενημερωμένων πλειάδων κατά μήκος του δικτύου και η ενημέρωση των απαρχαιωμένων πλειάδων στο σύστημα, για λόγους φρεσκάδας της κατανεμημένης βάσης δεδομένων. Για το λόγο αυτό προτάθηκαν διάφορα πρωτόκολλα ενημέρωσης και αντιγραφής που ενεργοποιούνται από κάθε κόμβο ξεχωριστά και μελετήθηκε η απόδοσή τους. Για την απόδοση των πρωτοκόλλων μελετήθηκε η ποιότητα υπηρεσιών (QoS) και η ποιότητα δεδομένων (QoD) που προσφέρουν σε συνδυασμό με τα διάφορα ερωτήματα αναζήτησης που θέτουν οι κόμβοι. Στόχος των πρωτοκόλλων είναι να έχουμε καλύτερα αποτελέσματα όσο αφορά την ενημερότητα (freshness) και την ανάκληση (recall) των αποτελεσμάτων των ερωτημάτων, όσο αφορά την ποιότητα δεδομένων καθώς και τη μείωση του αριθμού των βημάτων (response time) καθώς και των μηνυμάτων (message overhead) που φορτώνεται το δίκτυο, όσο αφορά την ποιότητα υπηρεσιών.

Τέλος, στην πειραματική μελέτη που έγινε με βάση την υλοποίηση που δημιουργήθηκε, διαπιστώθηκε η απόδοση των διαφόρων πρωτοκόλλων στην φρεσκάδα της κατανεμημένης βάσης του συστήματος, στο φόρτο δικτύου που επιφέρουν και στην απόδοση αναζήτησης τυχαίων πλειάδων από τους κόμβους με μικρό βάθος ερώτησης (TTL) κάθε φορά. Όπως διαπιστώθηκε τα πρωτόκολλα συμβάλλουν σημαντικά στην διάδοση των ενημερωμένων πλειάδων στο σύστημα,

την ενημέρωση των απαρχαιωμένων πλειάδων καθώς και στην απόδοση της τυχαίας αναζήτησης πλειάδων από τους κόμβους του συστήματος. Όμως το δίκτυο φορτώνεται με αρκετά μηνύματα κάθε φορά που σκοπός μας είναι να τα μειώσουμε όσο γίνεται περισσότερο. Σε αυτό βοηθάει ο συνδυασμός των πρωτοκόλλων κατά τις ανάγκες του κόμβου κάθε φορά καθώς και η ενεργοποίησή τους ανά τακτά χρονικά διαστήματα (lazy) και όχι συνεχώς.

Σαν μελλοντική έρευνα, θα μπορούσε να υλοποιηθεί μια κατανεμημένη γλώσσα που θα παρέχει πλήρη υποστήριξη του συνόλου των SQL ερωτημάτων σε αδόμητα, μη κεντρικοποιημένα συστήματα ομότιμων κόμβων, καθώς και θα υπολογίζει την ημερομηνία λήξης του αποτελέσματος. Τα προβλήματα που αναζητούν λύση είναι ο υπολογισμός της ημερομηνίας λήξης στα αποτελέσματα των ερωτήσεων σε περίπτωση που στο σύστημα έχουμε ενημερώσεις καθώς και ο υπολογισμός του αποτελέσματος με σε πολλαπλά μονοπάτια αναζήτησης, δηλαδή με περισσότερους από έναν τυχαίους περιπατητές (random walkers).

Τέλος σαν μελλοντική εργασία θα μπορούσε να μελετηθεί η περίπτωση που όλοι οι κόμβοι θα έπρεπε να είχαν την ίδια τοπική βάση δεδομένων, ενώ θα μπορούσαν να γίνουν ενημερώσεις από τον καθένα ξεχωριστά. Σε αυτή την περίπτωση πρέπει να γίνεται αρχικά αντιγραφή ολόκληρης της βάσης δεδομένων από κόμβο σε κόμβο και έπειτα να κρατάτε η έκδοση της βάσης και οι επιπλέον ενημερώσεις που γίνονται σε αυτήν ξεχωριστά, ώστε αυτές να διαδίδονται σύντομα κατά μήκος του δικτύου. Πρέπει να μελετηθούν διάφορα πρωτόκολλα αντιγραφής και ενημέρωσης ώστε όλοι οι κόμβοι να έχουν μια συνεπή και πλήρης άποψη του συνόλου της βάσης δεδομένων. Κάτι τέτοιο θα είναι πολύ χρήσιμο σε πολλαπλούς εξυπηρετητές, συνδεδεμένους μεταξύ τους σαν δίκτυο ομότιμων κόμβων, όπου χρήστες να μπορούν να συνδέονται με κάποιον από αυτούς, να ενημερώνουν τα δεδομένα και οι ενημερώσεις να είναι ορατές από όλους τους εξυπηρετητές σε σύντομο χρονικό διάστημα.

ΑΝΑΦΟΡΕΣ

[1] Albrecht Schmidt, Christian S. Jensen, Simonas Saltenis, Expiration Times for Data Management, International Conference on Data Engineering (ICDE), 2006.

[2] A. Schmidt and C. S. Jensen. Efficient Management of Ephemeral Data, accepted at the International Conference on Database Systems for Advanced Applications (DASFAA), 2006.

[3] Rafael Alonso, Daniel Barbará, Hector Garcia-Molina: Data Caching Issues in an Information Retrieval System. ACM Trans. Database Syst. 15(3): 359-384(1990)

[4] J. Cho and A. Ntoulas., Effective change detection using sampling. In Proc. of the 28th Int. Conf. on Very Large Databases, pages 514--525, Sept. 2002.

[5] A. Datta, M. Hauswirth, K. Aberer, Updates in Highly Unreliable, Replicated Peer-to-Peer Systems, 23rd International Conference on Distributed Computing Systems (ICDCS 2003), 19-22 May 2003, Providence, USA

[6] Qin Lv, Pei Cao, Edith Cohen, Kai Li, Scott Shenker Search and Replication in Unstructured Peer-to-peer Networks, International Conference on Supercomputing: 84-95, 2002

[7] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, Howard Sturgis, D. Swinehart and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In Proceedings of the 6th annual ACM Symposium on Principles of distributed computing,1987

[8] E. Cohen, S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In Proccedings of ACM SIGCOMM'02, Aug. 2002.

[9] Bloom filters: http://en.wikipedia.org/wiki/Bloom_filter

[10] V. V. Dimakopoulos and E. Pitoura. On the Performance of Flooding-Based Resource Discovery, IEEE Transactions on Parallel and Distributed Systems, 17(11):1242-1252, 2006.

[11] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In Proceedings IEEE INFOCOM 2003, 2003.

[12] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In Proc. ACM SIGCOMM (San Diego, CA, August 2001)

[13] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In Proceedings ACM Sigcomm 2001, San Diego, CA, Aug. 2001

[14] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the Skype peer-to-peer VoIP system. In IPTPS, 2006

[15] Internet World Stats http://www.internetworldstats.com/stats.htm

[16] Ipoque´s 2007 P2P Survey to be presented at Technology Review´s Emerging Technologies Conference at MIT:
http://www.ipoque.com/media/news/ipoques_2007_p2p_survey_to_be_presented_at_technology_reviews_emerging_technologies_conference_at_mit.htm

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Κωνσταντίνος Αλεξάκης γεννήθηκε το 1982, στο Ηράκλειο της Κρήτης όπου και μεγάλωσε. Τελείωσε το 5^{ο} Γενικό Λύκειο Ηρακλείου το 1999. Το 2000 ξεκίνησε τις ανώτατες σπουδές του στο τμήμα Εφαρμοσμένων Μαθηματικών του Πανεπιστημίου Κρήτης όπου και αποφοίτησε τον Ιούνιο του 2005, με βαθμό 6,95. Από το Σεπτέμβριο του 2005 είναι μεταπτυχιακός φοιτητής στο τμήμα Πληροφορικής του Πανεπιστημίου Ιωαννίνων.