Skyline ερωτήσεις σε συστήματα ομότιμων

Φωτεινή Πεχλιβάνη, Αικατερίνη Φωτιάδου

Τμήμα Πληροφορικής, Σχολή Θετικών Επιστημών, Ιωάννινα

{fpechliv, kfotiado}@cs.uoi.gr

Φεβρουάριος, 2006

Περίληψη

Οι Skyline ερωτήσεις επιστρέφουν ενδιαφέροντα σημεία από ένα πιθανόν πολύ μεγάλο σύνολο δεδομένων. Η εφαρμογή των Skyline ερωτήσεων σε συστήματα ομότιμων δεν είναι κάτι το τετριμμένο. Αυτό συμβαίνει διότι οι αλγόριθμοι που έχουν προταθεί για συστήματα βάσεων δεδομένων βασίζονται σε παραδοχές οι οποίες δεν ισχύουν σε αυτά τα συστήματα. Σε αυτή την εργασία παρουσιάζουμε κατάλληλα ευρετήρια και αλγόριθμο δρομολόγησης για την αποδοτική εφαρμογή των Skyline ερωτήσεων σε συστήματα ομότιμων. Συγκεκριμένα παρουσιάζουμε τέτοιου είδους ευρετήρια ώστε να ελαχιστοποιούνται οι ερωτήσεις σε ομότιμους που δεν έχουν πληροφορία η οποία μπορεί να ανήκει στο Skyline. Η προσέγγιση μας δεν βασίζεται στην γνώση ολικής πληροφορίας και απαντά αποδοτικά στα Skyline ερωτήματα. Αποδεικνύουμε ότι έχει μικρό υπολογιστικό κόστος αφού i) ερευνά το δίκτυο μία μόνο φορά –δεν υπάρχουν κύκλοι, ii) τα ευρετήρια έχουν μικρό μέγεθος και είναι εύκολα ανανεώσιμα, iii) μερική παραλληλία στον υπολογισμό του Skyline, iv) μειώνει τον αριθμό των ερωτηθέντων ομότιμων κατά περίπου 25,5% σε σχέση με μία προσέγγιση που χρησιμοποιεί απλά ευρετήρια.

1 Εισαγωγή

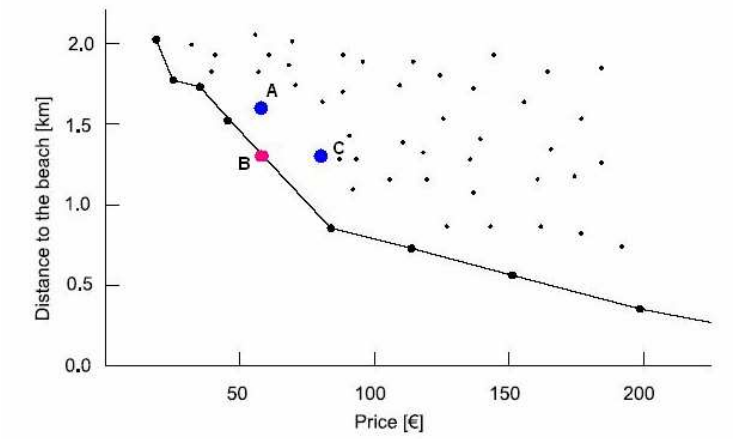
Η σημερινή υποδομή των πληροφοριακών συστημάτων παρέχει στους καταναλωτές της ένα τεράστιο πλήθος πληροφορίας. Οι χρήστες έχοντας πολλούς και διαφορετικούς στόχους θέλουν να λαμβάνουν αποφάσεις οι οποίες εξαρτώνται συνήθως από πολύ μεγάλα σύνολα δεδομένων. Μάλιστα είναι αρκετά συχνό φαινόμενο το να θέλουν να βελτιώσουν τις επιλογές τους αποφασίζοντας με γνώμονα πάνω από ένα χαρακτηριστικά των δεδομένων. Εδώ κάνουν την εμφάνιση τους τα Skyline ερωτήματα τα οποία επιστρέφουν ένα σύνολο από ενδιαφέρουσες απαντήσεις. Συγκεκριμένα το Skyline ενός συνόλου σημείων ορίζεται ως τα σημεία που δεν κυριαρχούνται (not dominated) από κανένα άλλο σημείο. Ένα σημείο x

κυριαρχεί ενός άλλου σημείου y αν το x δεν είναι χειρότερο σε καμία διάσταση(από αυτές που μας ενδιαφέρουν) από το y και είναι καλύτερο σε τουλάχιστον μια. Η σύνταξη των Skyline ερωτημάτων όπως ορίστηκε από τους [BoKS01] είναι η εξής:

```
SELECT … FROM … WHERE …
GROUP BY … HAVING …
SKYLINE OF [DISTINCT] d1 [MIN |MAX |DIFF], …, dm [MIN |MAX |DIFF]
ORDER BY …
```

Όπου όταν έχουμε MIN θέλουμε να ελαχιστοποιήσουμε το χαρακτηριστικό, MAX να το μεγιστοποιήσουμε και DIFF να το διαφοροποιήσουμε. Για λόγους απλότητας στη συγκεκριμένη εργασία θα υποθέσουμε ότι στόχος μας είναι μόνο να ελαχιστοποιήσουμε κάποια χαρακτηριστικά (έχουμε μόνο MIN). Επίσης για λόγους απλότητας θεωρούμε ότι τα attributes που μας ενδιαφέρουν έχουν ακέραιες τιμές.

Ένα παράδειγμα μιας Skyline ερώτησης φαίνεται στην Εικόνα 1. Σε αυτό το παράδειγμα ενδιαφερόμαστε για ξενοδοχεία τα οποία να βρίσκονται κοντά στην παραλία αλλά να είναι και φθηνά. Το Skyline περιέχει το σύνολο των απαντήσεων, δηλαδή όλα τα ξενοδοχεία που δεν κυριαρχούνται από κάποιο άλλο. Π.χ. το σημείο A δεν ανήκει στο Skyline αφού υπάρχει το σημείο B το οποίο είναι πιο κοντά στην παραλία από το A ενώ έχουνε την ίδια τιμή. Αντίστοιχα το C δεν ανήκει στο Skyline αφού είναι στην ίδια απόσταση με το B αλλά το B είναι πιο φθηνό.



Εικόνα 1,

```
SELECT * FROM Hotels WHERE city= “Nassau”
SKYLINE OF price MIN, distance MIN;
```

Πρόσφατα έχει παρατηρηθεί αυξημένο ενδιαφέρον για την εφαρμογή των Skyline ερωτήσεων σε συστήματα ομότιμων. Οι αλγόριθμοι που έχουν ήδη προταθεί για τον υπολογισμό του Skyline βασίζονται σε παραδοχές που δεν ισχύουν σε αυτά τα συστήματα. Η απευθείας εφαρμογή κάποιων από αυτών σε συστήματα ομότιμων θα οδηγούσε σε εξοντωτικές αναζητήσεις, πολλά round trips και την ανάγκη να υπάρχει κάπου κεντρικά ολική πληροφορία. Όλα αυτά είναι καταστροφικά για ένα τέτοιο σύστημα, μη αποδεκτά και συνήθως μη υλοποιήσιμα. Για αυτό το λόγο είναι σκόπιμο να προταθούν νέοι τρόποι για τη

λύση του προβλήματος οι οποίοι να λαμβάνουν υπόψη τα ειδικά χαρακτηριστικά των συστημάτων ομότιμων. Συγκεκριμένα έχουμε τις εξής απαιτήσεις:

- ✓ Ο αλγόριθμος να μην στηρίζεται σε γνώση ολικής πληροφορίας
- ✓ Αποδοτική εύρεση απαντήσεων όσον αφορά το υπολογιστικό κόστος. Δηλαδή:
  - o Ρωτάμε μόνο μερικούς ομότιμους
  - o Τους ρωτάμε μόνο μία φορά

Η προσέγγιση μας πληροί αυτές τις προϋποθέσεις. Εφαρμόζουμε Skyline ερωτήματα σε ένα αδόμητο σύστημα ομότιμων στο οποίο κάποιοι ομότιμοι έχουν περισσότερες αρμοδιότητες από τους άλλους. Οι ομότιμοι αυτοί έχουν υπό την κατοχή τους ένα ευρετήριο σύμφωνα με το οποίο μπορούν να επιλέγουν τους γειτονικούς τους ομότιμους, στους οποίους θα προωθήσουν το ερώτημα. Με αυτόν τον τρόπο ρωτώνται μόνο ομότιμοι οι οποίοι είναι πιθανό να έχουν πληροφορία που θα ανήκει στο Skyline της ερώτησης ενώ οι υπόλοιποι αγνοούνται. Αποδεικνύουμε ότι αυτή η τεχνική μπορεί να μειώσει τον αριθμό των ερωτηθέντων ομότιμων κατά περίπου 25,5% σε σχέση με μία τεχνική η οποία θα χρησιμοποιούσε απλά ευρετήρια π.χ. αυτά που ορίζονται στο [CrGa02].

Το υπόλοιπο της εργασίας είναι δομημένο με τον τρόπο που ακολουθεί: Αρχικά στην $2^{η}$ παράγραφο περιγράφονται συνοπτικά οι αλγόριθμοι που έχουν ήδη προταθεί για την επίλυση του προβλήματος και ο λόγος για τον οποίο δεν είναι αποδοτικοί σε συστήματα ομότιμων. Στην παράγραφο 3 περιγράφουμε το μοντέλο πάνω στο οποίο βασίζεται ο αλγόριθμος δρομολόγησης που παρουσιάζουμε, και πιο συγκεκριμένα η τοπολογία του δικτύου και το είδος των ευρετηρίων που υπάρχει στους κόμβους. Στην $4^{η}$ παράγραφο παρουσιάζεται αναλυτικά ο αλγόριθμος δρομολόγησης και ένα παράδειγμα αυτού. Για περαιτέρω βελτίωση του χρόνου που χρειάζεται για να απαντηθεί ένα ερώτημα παρουσιάζουμε στην παράγραφο 5 μία caching τεχνική. Στην παράγραφο 6 αποδεικνύουμε μέσω πειραμάτων την καλή λειτουργία του αλγορίθμου που προτείνουμε και ότι μάλιστα είναι καλύτερη από ότι χρησιμοποιώντας ένα απλό ευρετήριο. Τέλος στην παράγραφο 7 αναφέρουμε κάποια γενικά συμπεράσματα και κάποιες προτάσεις για μελλοντικές επεκτάσεις που μπορούν να γίνουν.

2 Σχετικές Εργασίες

Οι πρώτοι που εισήγαγαν το πρόβλημα του Skyline, γνωστό πρωτύτερα ως "The maximum vector problem", στο πλαίσιο των παραδοσιακών βάσεων δεδομένων είναι οι [BoKS01]. Στην εργασία τους προτείνουν μια επέκταση του SELECT της SQL κατά ένα SKYLINE

clause, έτσι ώστε να μπορεί να υποστηρίξει Skyline queries που να ελαχιστοποιούν, μεγιστοποιούν και/ ή διαφοροποιούν κάποια χαρακτηριστικά (attributes). Για την υλοποίηση της Skyline λειτουργίας παρουσιάζουν εναλλακτικούς αλγόριθμους και μετρούν την απόδοση τους, με γνώμονα τη διαθεσιμότητα της κύριας μνήμης, το εύρος εισόδου/ εξόδου κτλ. Οι πιο αποδοτικοί και διαδεδομένοι από αυτούς τους αλγορίθμους είναι οι BNL (Block Nested Loops Algorithm), ο D&C (Divide and Conquer) και κάποιες παραλλαγές αυτών. Ο BNL μπορεί να εφαρμοστεί για οσοδήποτε αριθμό διαστάσεων, αλλά προβλήματα παρουσιάζονται με το μέγεθος της κύριας μνήμης. Ο D&C είναι αποδοτικός για μικρά data set, και τελείως ακατάλληλος για online processing, αφού έχει μεγάλο κόστος εισόδου/ εξόδου.

Η ανάγκη για έναν online αλγόριθμο, ο οποίος θα αλληλεπιδρά με το χρήστη έκανε τους [KoRR02] να προτείνουν τον NN (Nearest Neighbor) αλγόριθμο ο οποίος βασίζεται στη Nearest Neighbor αναζήτηση. Το ζήτημα ήταν ο αλγόριθμος να επιστρέφει γρήγορα τα πρώτα αποτελέσματα χωρίς να διαβάσει όλη την πληροφορία στη βάση, να παράγει όλο και περισσότερα αποτελέσματα συνεχώς και να επιτρέπει στο χρήστη να εισάγει τις προτιμήσεις του ενώ το πρόγραμμα τρέχει, έτσι ώστε οι επιλογές του να επηρεάζουν τα αποτελέσματα. Ο NN φαίνεται να λύνει τα προβλήματα των προηγούμενων προσεγγίσεων, εισάγει όμως καινούρια όπως κακή αποδοτικότητα και απαγορευτικές απαιτήσεις σε χώρο όταν το Skyline αφορά πάνω από τρεις διαστάσεις.

Λόγω των μειονεκτημάτων του NN, οι [PTFS03] προτείνουν τον BBS (Branch and Bound Skyline) αλγόριθμο, ο οποίος είναι ένας επίσης online αλγόριθμος που βασίζεται στη Nearest Neighbor αναζήτηση. Ο BBS υπερπηδά όλα τα εμπόδια που ο NN δεν καταφέρνει να λύσει από τη στιγμή που είναι αποδοτικός και για progressive και για complete Skyline queries ανεξάρτητα από τα χαρακτηριστικά των δεδομένων, χειρίζεται εύκολα τις προτιμήσεις του χρήστη, έχει μικρές απαιτήσεις σε μνήμη κ.α.

Παρόλα αυτά αυτοί οι αλγόριθμοι δεν μπορούν να εφαρμοστούν σε συστήματα ομότιμων, τουλάχιστον όχι χωρίς τροποποιήσεις. Αυτό συμβαίνει επειδή δεν λαμβάνουν υπόψη τους βασικά χαρακτηριστικά αυτών των συστημάτων, όπως:

- Απουσία ολικής (global) πληροφορίας. Κάθε ομότιμος γνωρίζει μόνο πληροφορία την οποία κατέχει ο ίδιος και ίσως πληροφορία για κάποιους από τους γείτονες του.
- Όλοι οι ομότιμοι είναι ίσοι. Καθένας από αυτούς μπορεί να θέσει ένα ερώτημα.
- Το σύστημα είναι δυναμικό, δηλαδή οι ομότιμοι εισέρχονται και εξέρχονται κατά βούληση.

- Είναι σχεδόν αδύνατο να εγγυηθούμε ολοκληρωμένες και ακριβείς απαντήσεις χωρίς εξοντωτική αναζήτηση.

Μια υλοποίηση των Skyline queries σε αδόμητα P2P συστήματα, η οποία λαμβάνει υπόψη της τα ειδικά χαρακτηριστικά τους, παρουσιάζεται στο [Hose05]. Συγκεκριμένα η συγγραφέας παρουσιάζει μια προσέγγιση η οποία μειώνει τον αριθμό των ερωτηθέντων ομότιμων και δίνει πιθανοτικές εγγυήσεις για την ορθότητα των απαντήσεων. Για να το πετύχει αυτό χρησιμοποιεί advanced routing indexes στα οποία κάθε ομότιμος κρατάει για τους γείτονες του πληροφορία που έχει να κάνει με τα ερωτηθέντα attributes. Έχοντας αυτήν την πληροφορία κάθε ομότιμος δρομολογεί την ερώτηση μόνο σε γείτονες των οποίων τα στοιχεία είναι πιθανό να συμμετέχουν στο Skyline. Επειδή τα ευρετήρια γενικά παρέχουν μία "προσέγγιση" της πραγματικότητας, ποτέ ένας ομότιμος δεν μπορεί να είναι απολύτως σίγουρος για το αν ένας γείτονας του θα συμμετέχει ή όχι στο Skyline μιας συγκεκριμένης ερώτησης. Μια συντηρητική προσέγγιση θα ρωτούσε όλους τους ομότιμους για τους οποίους δεν είμαστε σίγουροι με αποτέλεσμα να φτάσει σε εξοντωτική αναζήτηση του δικτύου. Αντιθέτως στο [Hose05] κάθε ομότιμος ερωτάται σύμφωνα με το ευρετήριο του αλλά και κάποια πιθανότητα η οποία δίνεται από τον χρήστη. Με αυτήν την τεχνική καταφέρνει να δώσει έναν αποδοτικό αλγόριθμο ο οποίος δίνει σωστά αποτελέσματα με μεγάλη πιθανότητα (>90%).

Τέλος στο [WZF+06] παρουσιάζεται ένας αλγόριθμος που εφαρμόζεται στο δομημένο σύστημα ομότιμων CAN [RFH+01]. Ο αλγόριθμος αυτός στοχεύει στο να μεγιστοποιήσει την παραλληλία όσον αφορά τον υπολογισμό του Skyline. Υπολογίζει και εμφανίζει τα σημεία του Skyline σταδιακά χωρίς να χρειαστεί να προσπελάσει όλα τα δεδομένα. Αυτό μπορεί και το επιτυγχάνει εκμεταλλευόμενος το γεγονός ότι στο CAN κάθε κόμβος περιέχει δεδομένα μόνο από μία συγκεκριμένη ζώνη που διαχειρίζεται. Έτσι δεν έχουμε μόνο σημεία που γίνονται dominated από κάποια άλλα, αλλά και κόμβους που γίνονται dominated από άλλους κόμβους. Με βάση αυτήν την παρατήρηση βάζει τους κόμβους σε μια μερική διάταξη (partial-order) ξεκινώντας από τους κόμβους οι οποίοι δεν γίνονται dominated από κανέναν άλλο κόμβο και προσθέτοντας μετά κόμβους των οποίων ο υπολογισμός του Skyline, προϋποθέτει το αποτέλεσμα των προγόνων τους. Ξεκινώντας λοιπόν από την αρχή αυτής της διάταξης και υπολογίζοντας το Skyline των κόμβων με αυτή τη σειρά μπορεί και εμφανίζει το Skyline σταδιακά χωρίς να προσπελάσει όλους τους κόμβους.

Παρόλο που οι δύο τελευταίες προσεγγίσεις λαμβάνουν υπόψη τους τα χαρακτηριστικά των συστημάτων ομότιμων συνεχίζουν να έχουν κάποια μειονεκτήματα. Τα ευρετήρια του

[Hose05] είναι δύσκολα ενημερώσιμα από τη στιγμή που περιέχουν και ποσοτική πληροφορία για τα δεδομένα (πόσα δεδομένα υπάρχουν σε κάθε διάστημα). Επίσης το ότι είναι περιορισμένου ορίζοντα (limited-horizon) δεν σημαίνει ότι αποφεύγουν το πρόβλημα των κύκλων, αφού κύκλοι μπορεί να υπάρχουν σε αυτόν τον ορίζοντα. Το πρόβλημα των κύκλων επηρεάζει τόσο στον αλγόριθμο δρομολόγησης όσο και στην ενημέρωση των ευρετηρίων. Η προσέγγιση που περιγράφεται σε αυτήν την εργασία χειρίζεται αποδοτικά το πρόβλημα των κύκλων χρησιμοποιώντας μία συγκεκριμένη τοπολογία και έναν καλό αλγόριθμο δρομολόγησης. Επίσης στην προσέγγιση μας ευρετήρια δεν υπάρχουν σε όλους τους κόμβους και ανανεώνονται πολύ εύκολα. Τέλος με την δική μας προσέγγιση τα αποτελέσματα παύουν να είναι πιθανοτικά και είναι πλήρως αξιόπιστα (εκτός από την τροποποίηση με την cache, παράγραφος 5).

Το [WZF+06] βασίζει την καλή του απόδοση σε μία content-based τεχνική η οποία βασίζεται εξολοκλήρου στην CAN τοπολογία που χρησιμοποιεί. Η προσέγγιση που παρουσιάζουν είναι πολύ καλή, αλλά ενδιαφέρουσα μόνο από θεωρητική άποψη αφού τα δομημένα συστήματα ομότιμων δεν χρησιμοποιούνται στην πράξη. Σκοπός αυτής της εργασίας είναι να προτείνει μία τεχνική η οποία να είναι εύκολα υλοποιήσιμη και να μπορεί να χρησιμοποιηθεί στην πράξη. Για αυτόν τον λόγο χρησιμοποιούμε και την τοπολογία που περιγράφεται στην παράγραφο 3.1 η οποία έχει χρησιμοποιηθεί με επιτυχία σε ένα πραγματικό σύστημα ομότιμων.

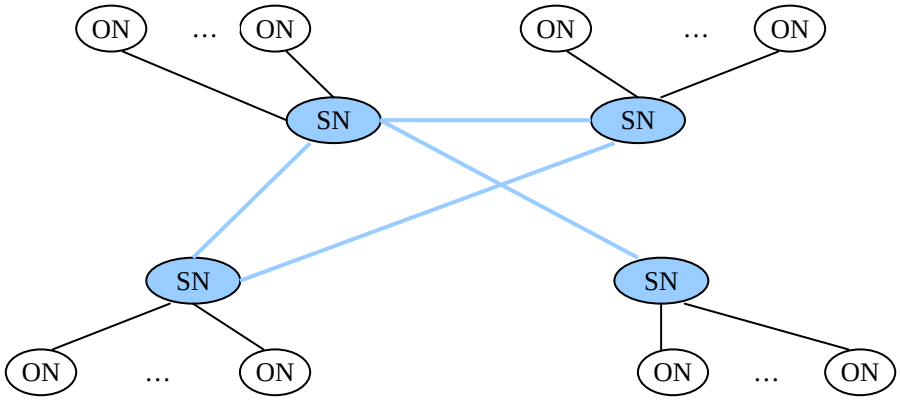
3 Περιγραφή μοντέλου

Για να περιγράψουμε πως δρομολογείται και τελικά εκτελείται ένα Skyline Query πρέπει πρώτα να περιγράψουμε την τοπολογία του δικτύου και στη συνέχεια να ορίσουμε τα ευρετήρια που υπάρχουν στους κόμβους, τα οποία θα χρειαστούμε στη δρομολόγηση.

3.1 Τοπολογία

Το δίκτυο στο οποία βασιζόμαστε είναι ένα αδόμητο P2P σύστημα στο οποίο υπάρχουν δύο κλάσεις ομότιμων. Η πρώτη κλάση αποτελείται από τους Ordinary Nodes (ONs) ενώ η δεύτερη από τους Super Nodes (SNs). Σε αυτήν τοπολογία δεν είναι όλοι οι ομότιμοι ίσοι, οι SNs έχουν περισσότερες αρμοδιότητες από τους ONs. Την κλάση των SNs την αποτελούν κόμβοι οι οποίοι έχουν μεγάλη υπολογιστική και αποθηκευτική ισχύ και παραμένουν στο δίκτυο για μεγάλο χρονικό διάστημα. Κάθε ON ανήκει σε έναν SN,

δηλαδή κάθε ON διατηρεί μια και μοναδική σύνδεση με τον SN στον οποίο ανήκει. Αντίθετα οι SN είναι συνδεδεμένοι με πολλούς ON αλλά και με αρκετούς SN. Γενικά ένας SN συνδέεται με περίπου 5% των συνολικών SNs που υπάρχουν στο δίκτυο και οι συνδέσεις αυτές αλλάζουν αρκετά συχνά. Κάθε SN διατηρεί πληροφορίες για τα δεδομένα που έχουν τα παιδιά του. Η τοπολογία εμφανίζει αρκετές ομοιότητες με την τοπολογία που εκτιμάται ότι έχει το Kazza σύμφωνα με τα [LiKR03] και [LiKR04].



Εικόνα 2, Η τοπολογία του δικτύου

Όπως γνωρίζουμε τα P2P συστήματα χαρακτηρίζονται για τις άναρχες εισόδους και εξόδους των κόμβων στο δίκτυο. Στη συνέχεια περιγράφουμε τον τρόπο με τον οποίο αυτές οι ενέργειες εκτελούνται. Όταν ένας SN φύγει από το δίκτυο τότε ενημερώνει τα παιδιά του και αυτά επαναλαμβάνουν εκ νέου την διαδικασία εισόδου στο δίκτυο. Όταν ένας ON αποφασίσει να φύγει από το δίκτυο τότε ενημερώνει τον πατέρα του για την απόφασή του και ο πατέρας του απλά ενημερώνει την πληροφορία που διατηρεί για τα δεδομένα των παιδιών του.

Στην περίπτωση κατά την οποία ένας κόμβος θέλει να εισέλθει στο δίκτυο αυτός θα μπει σίγουρα ως ON. Ακόμη πρέπει -για να μπει επιτυχώς- να γνωρίζει τουλάχιστον ένα κόμβο που εκείνη τη στιγμή είναι στο δίκτυο. Για να πετύχουμε καλό load balance ο κόμβος που θέλει να εισέλθει αρχικοποιεί μια προσωρινή επικοινωνία με πέντε SN του δικτύου, με τη βοήθεια του ενός τουλάχιστον κόμβου που γνωρίζει, και επιλέγει τελικά να συνδεθεί με τον SN που έχει τα λιγότερα παιδιά. Για την διατήρηση της τοπολογίας μετά από συνεχείς εισόδους και εξόδους εκτελείται μια περιοδική ρουτίνα ώστε αν σε κάποιους SNs έχουν ανατεθεί ON οι οποίοι να ξεπερνούν ένα άνω όριο τότε αποφασίζει και μετατρέπει έναν ή περισσότερους ONs σε SNs ώστε να εξισορροπηθεί το φορτίο του δικτύου.

Δεν πρέπει να αγνοήσουμε το ενδεχόμενο να καταρρεύσει κάποιος κόμβος, αν κάποιος ON για κάποιο λόγο πέσει τότε ο πατέρας του θα πρέπει να σβήσει την πληροφορία που

διατηρεί για αυτόν για να μην έχουμε ασυνέπεια στο σύστημα. Η περιοδική ρουτίνα που αναφέραμε παραπάνω επιτελεί και αυτήν την λειτουργία. Τα προβλήματα είναι σοβαρότερα αν πέσει ένας SN. Τότε όλα τα παιδιά του βρίσκονται εκτός δικτύου και όταν το αντιληφθούν πρέπει να αρχίσουν μόνα τους την είσοδο στο δίκτυο.

3.2 Ευρετήρια

Στο P2P σύστημα με την παραπάνω τοπολογία κάθε κόμβος N_i διατηρεί κάποια δεδομένα έστω Δ_i. Από αυτά τα δεδομένα με κάποιες στατιστικές μεθόδους έχουν προσδιοριστεί κάποια attributes τα οποία μας ενδιαφέρουν, έστω $A=\{a_j \mid 0 < j < m\}$. Το ευρετήριο που χρησιμοποιούμε έχει πληροφορίες για αυτά και μόνο τα attributes, τα οποία για ευκολία υποθέτουμε πως έχουν μόνο ακέραιες τιμές. Για κάθε a_j ορίζουμε τη μέγιστη τιμή την οποία μπορεί να πάρει έστω a_jMax. Τα ευρετήρια που θα περιγράψουμε με λεπτομέρεια στη συνέχεια υπάρχουν μόνο στους SNs και περιέχουν πληροφορία για τα παιδιά τους.

Το ευρετήριο είναι ένας πίνακας ο οποίος έχει τόσες στήλες όσα και τα attributes, δηλαδή m, και γραμμές ίσες με τους ONs που ανήκουν σε έναν SN. Για να γίνεται πιο αποτελεσματικά η δρομολόγηση δεν αρκεί να ξέρουμε αν κάποιος κόμβος έχει ή όχι πληροφορία για τα attributes που περιλαμβάνονται στην ερώτηση. Αυτό που χρειαζόμαστε είναι ποιοτική πληροφορία, όπως το αν έχει attributes τα οποία είναι μικρότερα ή μεγαλύτερα από κάποιες τιμές. Αυτήν την πληροφορία μπορούμε να τη χρησιμοποιήσουμε για να βρούμε αν ένας κόμβος έχει δεδομένα τα οποία μπορούν να ανήκουν στο Skyline. Με σκοπό να μειώσουμε τον όγκο του ευρετηρίου ομαδοποιούμε τις τιμές κάθε attribute σε το πολύ 16 buckets. Έτσι σε κάθε a_j αντιστοιχίζεται ο αριθμός των buckets έστω b_j, όπου $0<b_j<16$. Το b_j καθορίζεται όταν το a_j επιλεχθεί να συμμετέχει στο ευρετήριο. Το εύρος των τιμών κάθε bucket δίνεται από το $\lceil a_j\text{Max}/b_j \rceil$. Σύμφωνα με τα παραπάνω για κάθε τιμή του attribute a_j βρίσκουμε το bucket στο οποίο ανήκει και το καταχωρούμε στο ευρετήριο.

Κάθε πεδίο του πίνακα πρέπει να αποτελείται από 16 bits γιατί κάθε bucket αντιστοιχεί σε ένα bit, όπου αν είναι μηδέν σημαίνει πως ο κόμβος N_i δεν έχει πληροφορία για το a_j σε αυτό το bucket, ενώ αν είναι ένα ότι περιέχει τέτοια πληροφορία. Εύκολα βλέπει κανείς πως αν όλα τα bits είναι μηδενικά τότε ο κόμβος N_i δεν περιέχει καθόλου πληροφορία για το a_j.

Το μέγεθος του ευρετηρίου είναι m*CHILDS*2 bytes, δηλαδή το πλήθος των attributes που υπάρχουν στο ευρετήριο επί το πλήθος των παιδιών κάθε SN επί 2 bytes που απαιτούνται για τα buckets. Το μέγεθος του ευρετηρίου είναι αρκετά μικρό και μάλιστα

επειδή ευρετήρια υπάρχουν μόνο στους SNs αντιλαμβανόμαστε ότι καταλαμβάνουν πολύ μικρή χωρητικότητα συνολικά στο δίκτυο.

Για να είναι τα ευρετήρια συνεχώς ενημερωμένα πρέπει να συμβαίνουν τα εξής:

a) Όταν ένας ON τροποποιεί κάποια πληροφορία για τα attributes που ξέρει ότι υπάρχουν στο ευρετήριο πρέπει να ενημερώνει τον πατέρα του στέλνοντας του το νέο περιεχόμενο.
b) Όταν ένας ON εισέρχεται στο δίκτυο ο SN στον οποίο συνδέεται θα πρέπει να ενημερώνει το ευρετήριο του με τα νέα δεδομένα.
c) Όταν ένας ON αποχωρεί από το δίκτυο ειδοποιεί τον πατέρα του έτσι ώστε αυτός να αφαιρέσει οποιαδήποτε πληροφορία υπάρχει στο ευρετήριο και είναι σχετική με τον κόμβο που αποχώρησε.

Ένα παράδειγμα ευρετηρίου περιγράφεται παρακάτω:

Υποθέτουμε ότι SN_1 έχει παιδία του τους κόμβους ON_2 και ON_3 οι οποίοι περιέχουν τα δεδομένα που φαίνονται στον Πίνακα 1. Επίσης υποθέτουμε για την ευκολία του παραδείγματος πως έχουμε 16 buckets και πως η μέγιστη τιμή που μπορούν να πάρουν η τιμή του ξενοδοχείου και η απόσταση σε χιλιόμετρα είναι 160. Στο ευρετήριο το πρώτο bit από το τέλος αντιπροσωπεύει το bucket 0 το οποίο γίνεται μονάδα αν υπάρχει τιμή για το αντίστοιχο attribute μεταξύ του 0 και του 9. Όμοια το δεύτερο bit γίνεται μονάδα αν υπάρχει τιμή μεταξύ του 10 και του 19 κλπ. Έτσι λοιπόν το ευρετήριο του SN_1 είναι αυτό που φαίνεται στον Πίνακα 2. Π.χ. επειδή ο κόμβος ON_2 έχει την τιμή 20, η οποία αντιστοιχεί στο δεύτερο bucket, το τρίτο bit από το τέλος γίνεται 1, όμοια επειδή έχει την τιμή 55 το έκτο bit από το τέλος γίνεται 1 κλπ.

ΔΕΔΟΜΕΝΑ

	ΤΙΜΗ	**ΑΠΟΣΤΑΣΗ**
ON_2	80€	20km
	55€	25km
	20€	145km
ON_3	55€	18km
	30€	80km
	70€	100km

Πίνακας 1, Τα δεδομένα των κόμβων ON_2, ON_3

INDEX

ΚΟΜΒΟΣ	**ΤΙΜΗ**	**ΑΠΟΣΤΑΣΗ**
ON_2	00000001 00100100	01000000 00000100
ON_3	00000000 10101000	00000101 00000010

Πίνακας 2, Το ευρετήριο του SN_1

4 Δρομολόγηση

Σε αυτή την παράγραφο περιγράφεται ο αλγόριθμος που χρησιμοποιείται για τη δρομολόγηση των ερωτήσεων. Αρχικά στην παράγραφο 4.1 περιγράφουμε αναλυτικά τον

αλγόριθμο και την εύρεση των ομότιμων στους οποίους προωθούμε το ερώτημα. Στην παράγραφο 4.2 παρουσιάζουμε μια μικρή βελτίωση του αλγόριθμου ώστε να ρωτάει ακόμα λιγότερους ομότιμους.

4.1 Αλγόριθμος Δρομολόγησης

Θεωρούμε ότι κάθε ερώτημα έχει ένα μοναδικό id το οποίο το καθορίζεται από τα attributes της ερώτησης. Δηλαδή ερωτήσεις που ρωτάνε για τα ίδια attributes (το ίδιο Skyline) έχουν το ίδιο id.

Υπάρχουν 4 περιπτώσεις:

1. Ένας SN θέτει ένα ερώτημα
2. Ένας ON θέτει ένα ερώτημα
3. Ένας SN λαμβάνει ένα ερώτημα
4. Ένας ON λαμβάνει ένα ερώτημα

Στην $1^η$ περίπτωση ο SN προωθεί το μήνυμα σε όλους τους SN τους οποίους γνωρίζει μαζί με έναν χρόνο αναμονής t_2 (ότι δηλαδή θα περιμένει χρόνο t_2 για τα αποτελέσματα). Έπειτα υπολογίζει το τοπικό του Skyline και προωθεί το ερώτημα στους ON με τους οποίους συνδέεται, αλλά μόνο σε αυτούς των οποίων τα δεδομένα ενδέχεται να εμπεριέχονται στο Skyline. Αν δε λάβει αποτελέσματα από όλους μέσα στον καθορισμένο χρόνο t_2 υποθέτει ότι κάποιοι κόμβοι έχουνε πέσει (ή έχουνε στείλει σε άλλον SN τα αποτελέσματα) και υπολογίζει το Skyline από αποτελέσματα που έχει ήδη λάβει. Σε περίπτωση που ο SN δεν έχει τοπικό Skyline (δηλαδή δεν έχει σχετικά με την ερώτηση δεδομένα ή δεν περιλαμβάνει στα δεδομένα του όλα τα attributes της ερώτησης) τότε επιλέγει τυχαία έναν από τους ON του ευρετηρίου του, του θέτει το ερώτημα και χρησιμοποιεί το Skyline που ο ON του επιστρέφει για να επιλέξει τους ON στους οποίους θα προωθήσει το ερώτημα.

Στην $2^η$ περίπτωση ο ON που θέτει το ερώτημα υπολογίζει το τοπικό του Skyline και προωθεί την ερώτηση στον SN με τον οποίο είναι συνδεδεμένος. Έπειτα περιμένει ώσπου ο SN να του στείλει τα αποτελέσματα της ερώτησης, τα οποία συνυπολογίζει στο Skyline του ώστε να έχει το τελικό αποτέλεσμα. Αν ο χρόνος αναμονής υπερβεί έναν χρόνο t_1 τον οποίο καθορίζει ο χρήστης και ο ON δεν έχει λάβει ακόμα αποτελέσματα τότε ο ομότιμος αποφασίζει αν θα θέσει ξανά το ερώτημα ή όχι.

Στην $3^η$ περίπτωση ο SN ελέγχει αν έχει ξαναδεχθεί ερώτημα με ίδιο id σε συγκεκριμένο χρονικό διάστημα t_3. Αν το ερώτημα είναι το πρώτο με αυτό το id τότε ο SN συμπεριφέρεται ακριβώς όπως στην $1^η$ περίπτωση μόνο που στο τέλος στέλνει τα

αποτελέσματα του Skyline στον SN ο οποίος των ρώτησε. Επίσης ο χρόνος αναμονής μειώνεται κατά μία σταθερά c, περιμένει δηλαδή για αποτελέσματα χρόνο t_2 - c. Αν το ερώτημα έχει ληφθεί ξανά μέσα στο διάστημα t_3, ο SN απλά το αγνοεί.

Στην 4η περίπτωση ο ON υπολογίζει το τοπικό του Skyline και έπειτα στέλνει τα αποτελέσματα στον SN ο οποίος τον ρώτησε.

Σημειώστε ότι $t_1 >> t_2$.

Παρακάτω δίνεται ο ακριβής αλγόριθμος:

```
QD: query definition; id: unique id of QD; SN's= the set of Super Nodes
that p is aware  of; RF= Routing Filter of p;

//Ένας κόμβος θέτει ένα ερώτημα
pose_query(QD, id)
      //αν ο p είναι ordinary node
      if p==ON then
                  //στέλνει το ερώτημα στον Super Node που γνωρίζει
            send_query(QD,id,SN's)
                  //περιμένει χρόνο ίσο με t1
            wait(t1)
                  //συλλέγει τις απαντήσεις
            AN=collect_answers()
                  //υπολογίζει το ολικό Skyline
            SF= compute_Skyline(QD,AN)
      //αλλιώς αν ο p είναι Super Node
      else if p==SN then
                  //στέλνει το ερώτημα στους Super Nodes που γνωρίζει
            send_query(QD,id,SN's,t2)
                  //υπολογίζει το τοπικό Skyline
            SL=compute_local_Skyline(QD)
                        //αν το Skyline του είναι κενό
                  if SL==∅ then
                              //επιλέγει έναν ON που έχει τα attributes
                        της ερώτησης
                        ON1=choose_ON(id)
                              //στέλνει το ερώτημα στον ON1
                        send_query(QD,id,ON1)
                              //συλλέγει τις απαντήσεις
                        SL=collect_answers()
                  end if
                  //βρίσκει τους γειτονικούς ordinary nodes που
                      μπορεί τα στοιχεία τους να ανήκουν στο Skyline
            RN=find_relevant_neighbors(QD,RF,SL)
                  //και τους στέλνει το ερώτημα
            send_query(QD,id,RN)
                  //περιμένει χρόνο ίσο με t2
            wait(t2)
                  //συλλέγει τις απαντήσεις
            AN=collect_answers()
                  //υπολογίζει το ολικό Skyline
            SF= compute_Skyline(QD,AN)
            end if
      end if
```

```
//Ένας κόμβος λαμβάνει ένα ερώτημα
receive_query(QD, id, t2) from M

      //αν ο p είναι ordinary node
      if p==ON then
                  //υπολογίζει το τοπικό Skyline
            SL=compute_local_Skyline(QD)
                  //στέλνει τα αποτελέσματα σε αυτόν που τον ρώτησε
            if SL<>∅ then
                  send_results_to_M(SL)
            end if
      //αλλιώς αν ο p είναι Super Node
      else if p==SN then
            //αν το ερώτημα με το συγκεκριμένο id είναι το πρώτο που
λαμβάνω μέσα σε χρόνο t3
            if check_id(id)==1 then
                        //θέτω το χρόνο της ερώτησης σε μηδέν
                  set_time(id,0)
                        //στέλνει το ερώτημα στους Super Nodes που
                                                      γνωρίζει
                  send_query(QD,id,SN's,t2-c)
                        //υπολογίζει το τοπικό Skyline
                  SL=compute_local_Skyline(QD)
                  //αν το Skyline του είναι κενό
                  if SL==∅ then
                              //επιλέγει έναν ON που έχει τα
                        attributes της ερώτησης
                        ON1=choose_ON(id)
                              //στέλνει το ερώτημα στον ON1
                        send_query(QD,id,ON1)
                              //συλλέγει τις απαντήσεις
                        SL=collect_answers()
                  end if
                  //βρίσκει τους γειτονικούς ordinary nodes που
                  μπορεί τα στοιχεία τους να ανήκουν στο Skyline
                  RN=find_relevant_neighbors(QD,RF,SL)
                        //και τους στέλνει το ερώτημα
                  send_query(QD,id,RN)
                        //περιμένει χρόνο ίσο με t2-c
                  wait(t2-c)
                        //συλλέγει τις απαντήσεις
                  AN=collect_answers()
                        //υπολογίζει το ολικό Skyline
                  SF= compute_Skyline(QD,AN)
                        //στέλνει το αποτέλεσμα σε αυτόν που τον
                                                      ρώτησε
                  send_results_to_M(SF)
            else do nothing
            end if
      end if
```

Τρόπος υπολογισμού των ομότιμων που έχουν σχετική πληροφορία:

```
find_relevant_neighbors(QN,RF,S_L)
      repeat for all ON's
            if ∃ p ∈ ON_i: p may dominates at least one point q ∈ S_L then
                  ON_i ∈ RN
            end if

      return RN
```

Αρχικά γίνεται έλεγχος για τον αν ο κόμβος περιέχει σχετική με το ερώτημα πληροφορία ή πιο συγκεκριμένα αν το attribute για το οποίο ρωτάμε υπάρχει καταχωρημένο στον εν λόγω κόμβο. Αν υπάρχει προχωρά στον έλεγχο, αλλιώς απορρίπτει τον ON. Για να ελέγξει αν ένα στοιχείο κάνει dominate κάποιο άλλο, ελέγχει αν υπάρχουν στον ON στοιχεία που να είναι μικρότερα τουλάχιστον από ένα attribute της ερώτησης για τουλάχιστον ένα σημείο που ανήκει στο ήδη υπολογισμένο Skyline. Αν ισχύει αυτό μπορεί να υπάρχει σημείο στον ON τέτοιο ώστε να ανήκει στο Skyline και πρέπει να πάει στον ON για να το βρει. Διαφορετικά σίγουρα δεν υπάρχει τέτοιο σημείο και ο ON δεν λαμβάνει καθόλου το ερώτημα. Παρακάτω δίνεται πιο φορμαλιστικά η συνθήκη αυτή.

Έστω A{a_1, a_2,…, a_n}τα attributes τα οποία είναι σχετικά με την ερώτηση και ανήκουν στον κόμβο N τον οποίο θέλουμε να ελέγξουμε αν θα επισκεφτούμε ή όχι. Επίσης έστω h_i, i ∈ [0,k] τα στοιχεία που ανήκουν στο ήδη υπολογισμένο Skyline και h_i{a_{i1}, a_{i2},…, a_{in}}τα attributes αυτών. Τότε ισχύει το εξής:

Ο κόμβος N μπορεί να περιέχει στοιχεία του Skyline και πρέπει να τον επισκεφθούμε αν:

Υπάρχει τουλάχιστον ένα i για το οποίο να ισχύει:

Υπάρχουν τιμές στο A τέτοιες ώστε:

$a_1 <= a_{i1}$ or $a_2 <= a_{i2}$ … or $a_n <= a_{in}$

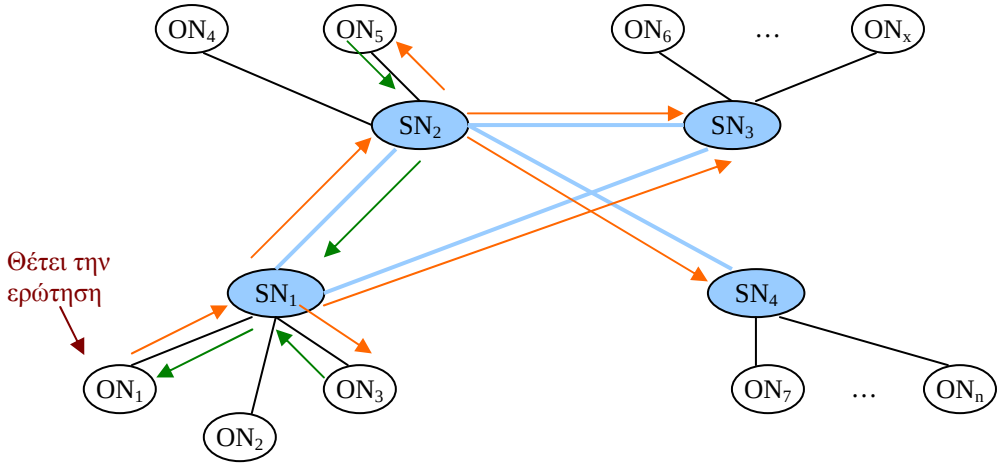
Ένα παράδειγμα δρομολόγησης φαίνεται στην Εικόνα 3. Ο ON_1 του SN_1 θέτει το κλασικό ερώτημα που θέλει να μάθει για ξενοδοχεία της πόλης Nassau που βρίσκονται κοντά στην παραλία αλλά είναι και φθηνά. Σε SQL το ερώτημα είναι το εξής:

```
SELECT *
FROM Hotels
WHERE city= "Nassau"
SKYLINE OF price MIN, distance MIN;
```

Για λόγους ευκολίας και συντομίας παραλείπουμε μερικά βήματα θεωρώντας ότι οι περισσότεροι κόμβοι δεν έχουν σχετική πληροφορία. Συγκεκριμένα θεωρούμε ότι όλοι οι

κόμβοι που σχετίζονται με τους SN_3 και SN_4 καθώς και οι ON_1 και ON_4 δεν έχουν καθόλου πληροφορία για ξενοδοχεία (Εικόνα 3).

Οι κόμβοι που έχουν σχετική πληροφορία καθώς και η πληροφορία αυτή φαίνονται στον Πίνακα 3. Στον Πίνακα 4 φαίνονται τα ευρετήρια που διατηρούν οι κόμβοι SN_1 και SN_2. Στο ευρετήριο δεν περιλαμβάνονται τα δεδομένα των ίδιων των SN_1, SN_2 αλλά μόνο αυτά των ON κόμβων για τους οποίους είναι υπεύθυνοι.



Εικόνα 3,

```
SELECT * FROM Hotels WHERE city= "Nassau"
SKYLINE OF price MIN, distance MIN;
```

ΚΟΜΒΟΣ	HOTEL	ΤΙΜΗ	ΑΠΟΣΤΑΣΗ
SN_1	H1	50€	19km
	H2	55€	15km
	H3	20€	70km
ON_2	H4	80€	90km
	H5	76€	100km
	H6	60€	150km
ON_3	H7	55€	18km
	H8	30€	70km
	H9	70€	100km
SN_2	H10	130€	20km
	H11	145€	10km
	H12	80€	50km
ON_5	H13	40€	18km
	H14	35€	20km
	H15	155€	15km

Πίνακας 3, Τα ξενοδοχεία που υπάρχουν σε κάθε κόμβο

	ΚΟΜΒΟΣ	ΤΙΜΗ	ΑΠΟΣΤΑΣΗ
SN_1	ON_2	00000001 11000000	10000110 00000000
	ON_3	00000001 10101000	00000100 10000010

	ΚΟΜΒΟΣ	ΤΙΜΗ	ΑΠΟΣΤΑΣΗ
SN_2	ON_5	10000000 00011000	00000000 00000110

Πίνακας 4, Τα ευρετήρια των SN_1, SN_2

Η δρομολόγηση φαίνεται στην Εικόνα 2 όπου με πορτοκαλί βελάκι φαίνεται η ερώτηση, ενώ με πράσινο οι απαντήσεις που στέλνονται. Στα παρακάτω συμβολίζουμε με $S_i\{\}$ τα επιμέρους Skyline.

Η ερώτηση ξεκινάει από τον κόμβο ON_1. Ο κόμβος αυτός θέτει το ερώτημα στον SN_1 ο οποίος το προωθεί στους κόμβους SN_2 και SN_3 τους οποίους γνωρίζει. Έπειτα υπολογίζει το τοπικό του Skyline το οποίο εδώ τυχαίνει να είναι όλα τα ξενοδοχεία τα οποία έχει καταχωρημένα S_1{H1, H2, H3} (Πίνακας 3). Ελέγχει στο ευρετήριο του αν οι ON_2 και ON_3 που συνδέονται με αυτόν έχουν σχετική πληροφορία, με τον τρόπο που περιγράφηκε παραπάνω. Για το ON_2 δε βρίσκει σχετική πληροφορία και δεν το ρωτά καθόλου. Για το ON_3 βρίσκει ότι υπάρχουν στοιχεία (σύμφωνα με το ευρετήριο) τα οποία έχουν τιμές τέτοιες ώστε ΤΙΜΗ<50 (παρόλα αυτά παρατηρούμε ότι κανένα στοιχείο του ON_3 δεν θα ανήκει στο Skyline). Ο ON_3 υπολογίζει το τοπικό του Skyline το οποίο είναι το S_2{H7, H8} και το στέλνει στον SN_1.

Εν τω μεταξύ ο SN_2 όταν έλαβε την ερώτηση υπολόγισε το τοπικό του Skyline το οποίο είναι το S_3{ H10, H11, H12} και με τον έλεγχο στο ευρετήριο του αποφάσισε να πάει στον ON_5. Ο ON_5 στέλνει το τοπικό του Skyline, S_4{H13, H14, H15} στον SN_2 ο οποίος υπολογίζει το ολικό Skyline, S_5{H11, H13, H14}. Στέλνει το S_5 στον SN_1 ο οποίος υπολογίζει το ολικό Skyline το οποίο είναι το S_6{H2, H3, H11, H13, H14} και το στέλνει στον ON_1. Το S_6 είναι το τελικό Skyline επειδή θεωρήσαμε ότι ο ON_1 δεν έχει σχετική με ξενοδοχεία πληροφορία.

4.2 Περαιτέρω βελτίωση της εύρεσης σχετικών γειτόνων

Σε αυτή την παράγραφο περιγράφουμε έναν τρόπο βελτίωσης του παραπάνω αλγορίθμου από την άποψη ότι θα ρωτάει ακόμα λιγότερους ομότιμους. Στην προηγούμενη προσέγγιση ο έλεγχος που κάνει ο SN για τους ONs που έχουνε σχετική πληροφορία εξαρτάται μόνο από το δικό του τοπικό Skyline. Είναι πιθανό το Skyline του SN να περιέχει μεγάλες τιμές και έτσι να μην μπορεί από το ευρετήριο να αποκλείσει πολλούς ONs. Για να βελτιωθεί λίγο λοιπόν αυτός ο έλεγχος, ο SN βρίσκει από το ευρετήριο του τον ON ο οποίος έχει τις μικρότερες τιμές για τα attributes που υπάρχουν στο ευρετήριο. Ρωτάει αυτόν τον ON και συνυπολογίζει το Skyline που του επιστρέφει μαζί με το δικό του τοπικό Skyline. Έτσι με γνώμονα πλέον αυτά το νέο Skyline, που πιθανολογείται ότι θα έχει μικρότερες τιμές από αυτό της προηγούμενης προσέγγισης, υπολογίζει τους ONs στους οποίους θα προωθήσει την ερώτηση. Στην παράγραφο 6.2 φαίνεται πειραματικά ότι με αυτήν την τεχνική μειώνονται ακόμη περισσότερο οι ONs τους οποίους ρωτάμε.

5 Caching

Caching γίνεται μόνο στους Super Nodes. Για κάθε ερώτηση που κάνει cache ο Super Node αποθηκεύει το id της ερώτησης, το Skyline της, καθώς και το ποιοι κόμβοι συνέβαλαν για τον υπολογισμό του. Κάθε Super Node έχει έναν cache manager, ο οποίος είναι υπεύθυνος για την διατήρηση της cache του. Υπάρχουν δύο συναρτήσεις η CreateCache(query) και η DeleteCache(query). Για να λειτουργήσει αποδοτικά η cache λειτουργία γίνεται μία μικρή αλλαγή στον αλγόριθμο δρομολόγησης. Πλέον κάθε Super Node μαζί με τα αποτελέσματα που λαμβάνει από έναν άλλον κόμβο λαμβάνει και πληροφορία για το ποιων κόμβων τα δεδομένα χρησιμοποιήθηκαν για τον υπολογισμό του. Για την κάθε ερώτηση που λαμβάνει ένας SN αν αποφασίσει να κάνει cache, κάνει το Skyline που έχει υπολογίσει από τη στιγμή που έλαβε όλες τις απαντήσεις από τους γείτονες του μέσα στον αναμενόμενο χρόνο t_2. Έτσι αν ο SN ξαναδεχθεί αργότερα την ίδια ερώτηση, θα ανατρέξει στην cache του για να στείλει τα αποτελέσματα, χωρίς να χρειαστεί να δρομολογήσει το ερώτημα. Ο cache manager κρίνει για το κατά πόσον μία ερώτηση θα γίνει cache ή όχι καθώς και για το κάθε πόσο η cache ενημερώνεται.

Π.χ. στο παράδειγμα της παραγράφου 3 ο SN_1 θα λάβει τα αποτελέσματα που του στέλνει ο SN_2 γνωρίζοντας ότι μόνο πληροφορία του SN_2 χρησιμοποιήθηκε. Έπειτα θα υπολογίσει το τελικό Skyline και θα αποθηκεύσει στην cache του το αποτέλεσμα μαζί με το ότι χρησιμοποιήθηκαν μόνο οι SN_1 και SN_2. Αν και ο SN_4 είχε πληροφορία τότε ο SN_2 θα έστελνε στον SN_1 το Skyline λέγοντας του ότι υπολογίστηκε από τους SN_2 και SN_4. Έτσι ο SN_1 θα αποθηκεύσει στην cache του το Skyline των SN_1, SN_2 και SN_4. Αν ο SN_1 λάβει ξανά το ίδιο ερώτημα θα ανατρέξει στην cache του και βλέποντας ότι έχει αποθηκευμένο το Skyline των κόμβων SN_1, SN_2 και SN_4 θα δρομολογήσει το ερώτημα μόνο στον κόμβο SN_3, κάνοντας του γνωστό ότι έχει ήδη πληροφορία από τους SN_1, SN_2 και SN_4 έτσι ώστε ο SN_3 να μην τους ρωτήσει. Αφού λάβει τα αποτελέσματα από τον SN_3 θα ενημερώσει την cache του με το νέο Skyline.

5.1 Επιλογή ερωτήσεων που θα γίνουν cache

Δύο είναι οι συνήθεις τύποι ερωτήσεων για τις οποίες καλείται η CreateCache(query):

a) Συχνές Ερωτήσεις: Οι Super Nodes έχουν κάποιους μετρητές οι οποίοι ανάλογα με το χρόνο μετράνε τη συχνότητα των ερωτήσεων. Αν ένας Super Node αντιληφθεί ότι δέχεται συχνά ένα ερώτημα αποθηκεύει τα αποτελέσματα του. Η αποθήκευση

του Skyline ερωτήσεων που γίνονται πολύ συχνά στο δίκτυο είναι αποδοτική επειδή γλιτώνουμε πολύ χρόνο. Αντίθετα το να αποθηκεύσουμε ερωτήσεις που γίνονται σπάνια είναι ασύμφορο.

b) Ερωτήσεις με πολλά attributes: Οι ερωτήσεις με πολλά attributes έχουν πολύ μεγάλο υπολογιστικό κόστος όσον αφορά το Skyline. Έτσι ακολουθούμε ως τεχνική την άμεση αποθήκευση των αποτελεσμάτων ώστε να μη χρειαστεί οι υπολογισμοί αυτοί να γίνουν ξανά.

5.2 Ενημέρωση της cache

Η καλή και έγκαιρη ενημέρωση της cache είναι απαραίτητη έτσι ώστε να δίνονται πάντα τα σωστά («φρέσκα») ή τουλάχιστον «σχεδόν σωστά» δεδομένα ως απάντηση στις ερωτήσεις. Ενημέρωση γίνεται στις παρακάτω περιπτώσεις:

a) Το Skyline του ερωτήματος υπάρχει ήδη στην cache αλλά λαμβάνοντας υπόψη μόνο μερικούς κόμβους. Ο κόμβος λαμβάνει το Skyline ενός άλλου κόμβου για την ίδια ερώτηση που δεν τον έχει συνυπολογίσει στα αποτελέσματα της cache (π.χ. στο προηγούμενο παράδειγμα όταν ο SN_1 λάβει το Skyline του SN_3). Τότε ο κόμβος ανανεώνει το αποτέλεσμα της cache του λαμβάνοντας υπόψη και τα νέα δεδομένα.

b) Αν αντιληφθεί ότι ένας από τους γνωστούς του Super Node δεν είναι πλέον στο δίκτυο (αποχώρησε εθελοντικά ή έχει πέσει). Σε αυτή την περίπτωση ο κόμβος διαγράφει από την cache του όλα τα Skyline στα οποία η πληροφορία αυτού του κόμβου έχει ληφθεί υπόψη.

Κανονικά η cache πρέπει να ανανεώνεται κάθε φορά που γίνεται εισαγωγή κάποιου κόμβου στο δίκτυο, έτσι ώστε τα δεδομένα της να αντιστοιχούν στα πραγματικά. Επειδή όμως αυτό θα έκανε το κόστος για τη διατήρηση της cache απαγορευτικό χρησιμοποιούνται οι παρακάτω τεχνικές οι οποίες μας εξασφαλίζουν ότι τα δεδομένα της cache θα είναι «λίγο» μη συνεπή για μικρό χρονικό διάστημα.

c) Όταν ένας ordinary node συνδεθεί σε έναν Super Node τότε αυτός ο Super Node ανανεώνει τα Skyline που έχει αποθηκευμένα στην cache του.

d) Αν το ερώτημα δεν ζητηθεί καθόλου για έναν χρόνο t τότε καλείται η DeleteCache(query) και το ερώτημα σβήνεται από την cache.

e) Αν το ερώτημα έχει συνεχή ζήτηση τότε ο κόμβος ανανεώνει σταδιακά την cache του για αυτό το ερώτημα. Για παράδειγμα, μετά από χρόνο t_1 ρωτάει έναν γείτονα του και ανανεώνει, μετά από χρόνο t_2 ρωτάει κάποιον άλλον κτλ. Με αυτόν τον

τρόπο εξασφαλίζει ότι η cache του δεν θα έχει δεδομένα που δεν υπάρχουν στο δίκτυο ή δεν αντιστοιχούν στην πραγματικότητα για μεγάλο χρονικό διάστημα.

6 Πειραματικά Αποτελέσματα

Υλοποιήσαμε ένα δίκτυο το οποίο έχει την τοπολογία που περιγράψαμε στο 3.1 και στο οποίο οι SNs έχουν το ευρετήριο που περιγράφηκε στο 3.2. Για να έχουμε ένα μέτρο σύγκρισης υλοποιήσαμε και ένα άλλο ευρετήριο το οποίο περιέχει μόνο πληροφορία για το αν ο κόμβος έχει πληροφορία για το κάθε attribute ή όχι, αυτό το αποκαλούμε simple index. Ο στόχος αυτού του πειράματος είναι να δείξουμε πως το ευρετήριο που προτείνουμε βοηθάει στο να γίνεται καλύτερη δρομολόγηση, ότι δηλαδή μειώνονται οι ερωτήσεις σε ομότιμους οι οποίοι δεν περιέχουν δεδομένα τα οποία να μπορούν να συμμετέχουν στο Skyline.

6.1 Το δικό μας index σε σχέση με το simple index

Για να πάρουμε μετρήσεις ήταν απαραίτητο να δώσουμε τιμές σε κάποιες παραμέτρους του συστήματος. Θεωρούμε ότι

- ✓ κάθε ευρετήριο περιέχει πληροφορία για πέντε attributes,
- ✓ οι κόμβοι έχουν πληροφορίες για 100 δεδομένα,
- ✓ τα δεδομένα τα παράγει μια ομοιόμορφη κατανομή,
- ✓ κάθε SN μπορεί να έχει από 75 έως 125 ONs που να ανήκουν σε αυτόν,
- ✓ οι SNs έχουν κατά μέσο όρο συνδέσεις με περίπου το 5% των συνολικών SNs,
- ✓ τα δεδομένα για όλες τις μετρήσεις είναι χωρισμένα σε 16 buckets

Επίδραση του αριθμού των παιδιών κάθε SN. Σε αυτό το πείραμα, μεταβάλλαμε τον αριθμό των παιδιών κάθε SN από 75 έως 125 διατηρώντας τον αριθμό των συνολικών κόμβων στο δίκτυο σταθερό. Η αλλαγή αυτή επηρέασε το πλήθος των SNs στο δίκτυο, π.χ. αν υπάρχουν συνολικά 1500 ONs στο δίκτυο και αν κάθε SN έχει 75 παιδιά τότε υπάρχουν 20 SNs ενώ αν κάθε SN έχει 125 παιδιά τότε υπάρχουν 12 SNs. Επιπλέον η μεταβολή του αριθμού των παιδιών επηρέασε και το μέγεθος του ευρετηρίου κάθε SN, αυτό συνέβη γιατί όπως έχουμε αναφέρει στην παράγραφο 3.2 το μέγεθος του ευρετηρίου κάθε SN εξαρτάται από το πλήθος των παιδιών του. Τελικά όμως παρατηρήσαμε πως η μεταβολή του πλήθους των παιδιών δεν επηρέασε καθόλου τον αριθμό των κόμβων που ρωτήθηκαν. Αυτό

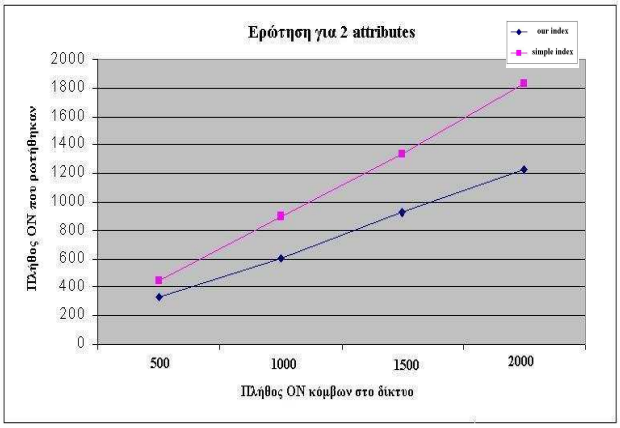
συμβαίνει γιατί η δρομολόγηση μιας ερώτησης εξαρτάται από το Skyline κάθε SN και από τα δεδομένα των ONs.

Επίδραση του αριθμού των συνολικών ONs στο δίκτυο. Όπως φαίνεται στις γραφικές παραστάσεις που ακολουθούν τρέξαμε το πείραμα ώστε να υπάρχουν συνολικά 500, 1000, 1500 και 2000 κόμβοι. Στις γραφικές βλέπουμε πως την μεγαλύτερη μείωση την πετυχαίνουμε όταν έχουμε 2000 κόμβους αυτό είναι λογικό γιατί τότε ρωτάμε χρησιμοποιώντας το simple index ακόμη περισσότερους κόμβους. Όμως το ποσοστό των κόμβων που μειώνεται είναι σχεδόν ίδιο αν έχουμε 1000, 1500 ή 2000 κόμβους. Εμφανίζεται λίγο χαμηλότερο μόνο στην περίπτωση κατά την οποία έχουμε 500 κόμβους αλλά δεν μας προβληματίζει γιατί στα συστήματα ομότιμων έχουμε πολύ περισσότερους κόμβους.

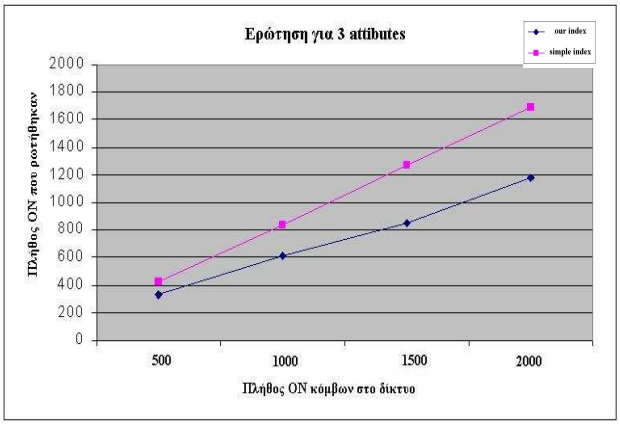
Επίδραση του αριθμού των attributes που έχει η Skyline ερώτηση. Εκτελέσαμε το πείραμα για ερωτήσεις οι οποίες περιείχαν 2, 3 και 4 attributes. Όπως φαίνεται στις γραφικές παραστάσεις 1, 2 και 3 το πλήθος των κόμβων που ρωτώνται παρουσιάζει μια μικρή αύξηση όσο αυξάνονται τα attributes αλλά παρόλα αυτά η απόδοση είναι αρκετά καλύτερη από αυτήν του simple index. Η μείωση που επιτυγχάνουμε, όταν στο Skyline έχουμε 3 attributes και στο δίκτυο έχουμε 2000 κόμβους, είναι κατά μέσο όρο 30% σε σχέση με το πλήθος των κόμβων που ρωτώνται με το simple index.

6.2 Περαιτέρω βελτίωση της απόδοσης

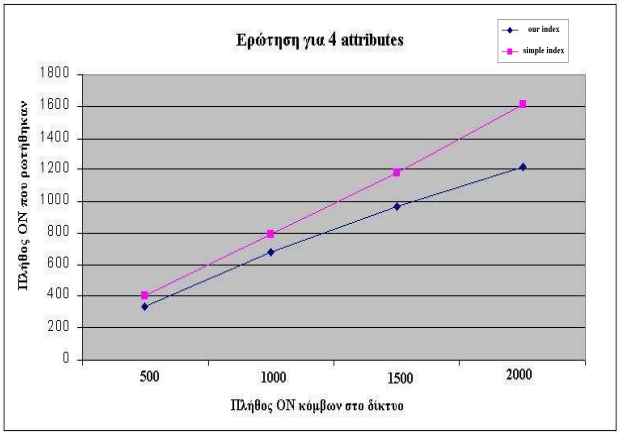
Στο πείραμα αυτό υλοποιούμε την προσέγγιση της παραγράφου 4.2 γιατί όπως έχουμε αναφέρει η απόδοση της δρομολόγησης εξαρτάται αρκετά από το αρχικό Skyline που θα υπολογίσουν οι SNs. Τα αποτελέσματα φαίνονται στη Γραφική παράσταση 4, όπου είναι εμφανής η βελτίωση της απόδοσης όταν κάθε SN έχει υπολογίσει ένα "καλό" Skyline πριν δρομολογήσει την ερώτηση στα παιδιά του. Με βάση τις τιμές της Γραφικής παράστασης 4, υπολογίσαμε πως η βελτίωση που περιγράφεται στην παράγραφο 4.2 προκαλεί μια επιπλέον μείωση κατά περίπου 20% στο πλήθος των κόμβων που ρωτώνται σε σχέση με τα αποτελέσματα που πήραμε από το πρώτο πείραμα. Τελικά η μείωση που επιτυγχάνουμε σε σχέση με το simple index είναι περίπου 35%.



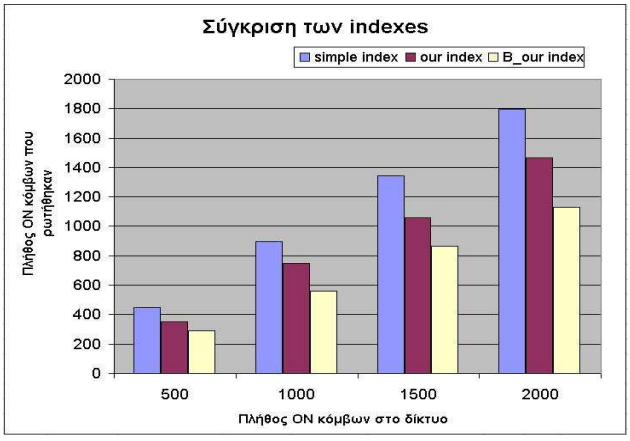
Γραφική παράσταση 1



Γραφική παράσταση 2



Γραφική παράσταση 3



Γραφική παράσταση 4

7 Συμπεράσματα – Μελλοντικές Επεκτάσεις

Στην εργασία αυτή παρουσιάσαμε κατάλληλα ευρετήρια και αλγόριθμο δρομολόγησης για την αποδοτική εφαρμογή των Skyline ερωτήσεων σε συστήματα ομότιμων. Συγκεκριμένα παρουσιάσαμε τέτοιου είδους ευρετήρια τα οποία ελαχιστοποιούν τις ερωτήσεις σε ομότιμους που δεν έχουν πληροφορία η οποία μπορεί να ανήκει στο Skyline. Η προσέγγιση μας έχει μικρό υπολογιστικό κόστος από της στιγμή που: i) ερευνά το δίκτυο μία μόνο φορά –δεν υπάρχουν κύκλοι, ii) τα ευρετήρια έχουν μικρό μέγεθος και είναι εύκολα ανανεώσιμα, iii) μερική παραλληλία στον υπολογισμό του Skyline, iv) μειώνει τον αριθμό των ερωτηθέντων ομότιμων. Συγκεκριμένα αποδείξαμε πειραματικά ότι μειώνει τον αριθμό των ερωτηθέντων ομότιμων κατά περίπου 25,5% σε σχέση με μία προσέγγιση που χρησιμοποιεί απλά ευρετήρια. Η μείωση δεν επηρεάζεται σημαντικά από τον αριθμό των

attributes της ερώτησης, ούτε από τον αριθμό των ON που αντιστοιχούν σε κάθε SN. Επίσης παρατηρήσαμε ότι η προσέγγιση αποδίδει όλο και καλύτερα όσο αυξάνει ο αριθμός των κόμβων στο δίκτυο (δηλαδή δουλεύει καλά για μεγάλα συστήματα ομότιμων που είναι και αυτά που υπάρχουν στην πραγματικότητα). Τέλος δοκιμάσαμε μία μικρή βελτίωση στην εύρεση των ομότιμων που θα ρωτήσουμε και είδαμε ότι παρουσιάζεται επιπλέον βελτίωση κατά 20%.

Η έρευνα μπορεί να συνεχισθεί έτσι ώστε το παραπάνω μοντέλο να μπορεί να απαντήσει σε constrained και ranked skyline queries. Στα constrained ερωτήματα ο χρήστης δεν ρωτάει για τα ξενοδοχεία που είναι φθηνά και είναι κοντά στην παραλία, αλλά μπορεί να καθορίσει τις τιμές που τον ενδιαφέρουν π.χ. το ξενοδοχείο να είναι φθηνότερο από 100€. Στα ranked ερωτήματα ο χρήστης ζητάει τα Top-K σημεία προσδιορίζοντας το K και προσδιορίζοντας μία συνάρτηση για να καθορίσει την βαρύτητα που δίνει σε κάθε attribute π.χ. η απόσταση από την παραλία έχει βαρύτητα 5 ενώ η τιμή του ξενοδοχείου 2, έτσι ώστε να επιλέγονται τα K καλύτερα σημεία. Μια ακόμη τροποποίηση που μπορεί να γίνει είναι να δημιουργηθεί ένας πρόσθετος μηχανισμός έτσι ώστε να υποστηρίζονται categorical attributes.

Αναφορές

[BoKS01] S. Borzsonyi, D. Kossmann and K. Stocker. "The Skyline Operator". ICDE 2001.

[CrGa02] A. Crespo and H. Garcia-Molina. "Routing Indices For Peer-to-Peer Systems". ICDCS 2002.

[Hose05] K. Hose. "Processing Skyline Queries in P2P Systems". VLDB 2005 PhD Workshop, 2005.

[KoRR02] D. Kossmann, F. Ramsak and S. Rost. "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries". VLDB 2002.

[LiKR03] J. Liang, R. Kumar, K. W. Ross. "Understanding KaZaA". August 2003.

[LiKR04] J. Liang, R. Kumar, K. W. Ross. "The KaZaA Overlay: A Measurement Study". September 2004.

[PTFS03] D. Papadias, Y. Tao, G. Fu and B. Seeger. "An Optimal and Progressive Algorithm for Skyline Queries". ACM SIGMOD 2003.

[RFH+01] S. Ratnasamy, P. Fransis, M. Handley, R. Karp, S. Shenker. "A Scalable Content-Addressable Network". SIGCOMM'01 August 27-31 2001, San Diego, California, USA.

[WZF+06] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal and A. El Abbadi. "Parallelizing Skyline Queries for Scalable Distribution". EDBT 2006.