ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ

Διπλωματική Εργασία

ΜΕΛΕΤΗ ΔΟΜΩΝ ΕΥΡΕΤΗΡΙΟΥ ΠΕΡΙΟΧΗΣ ΣΕ ΔΙΚΤΥΑ P2P

υπό

ΒΑΓΙΑΣ ΜΠΟΥΡΑ

Υπεβλήθη για την εκπλήρωση μέρους των απαιτήσεων για την απόκτηση του Διπλώματος Μηχανικού Η/Υ Τηλεπικοινωνιών & Δικτύων

Μάρτιος 2007

Εγκρίθηκε από τα μέλη της Διμερούς Εξεταστικής Επιτροπής

Πρώτος Εξεταστής (Επιβλέπων Καθηγητής)	Δρ. Παναγιώτης Μποζάνης Λέκτορας, Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών & Δικτύων, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Δρ. Δημήτριος Κατσαρός Διδάσκων ΠΔ 407/80 , Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών & Δικτύων, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Θα ήθελα να ευχαριστήσω, καταρχήν, τον επιβλέποντα καθηγητή μου Λέκτορα κ. Παναγιώτη Μποζάνη για τη συνεργασία μας και τη βοήθεια που μου προσέφερε για την εκπόνηση της διπλωματικής μου εργασίας, όπως επίσης και τον δεύτερο υπεύθυνο καθηγητή για τη διπλωματική μου εργασία, κ. Δημήτρη Κατσαρό. Επίσης, είμαι ευγνώμων σε όλους τους καθηγητές μου, στο Τμήμα Μ.Η.Υ.Τ.Δ, για τη γνώση και τις διδαχές που μου προσέφεραν τα χρόνια που φοίτησα στο Πανεπιστήμιο.

Ένα μεγάλο ευχαριστώ στην οικογένεια μου, τους γονείς μου και την αδερφή μου, που είναι πάντα δίπλα μου, με στηρίζουν σε κάθε μου απόφαση και ένα μεγάλο μέρος όσων έχω καταφέρει μέχρι τώρα, το χρωστώ σε εκείνους. Τέλος, ευχαριστώ τους φίλους μου και τους ανθρώπους που βρίσκονται κοντά μου, γιατί υπάρχουν στη ζωή μου και την κάνουν ομορφότερη με κάθε τρόπο.

Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή

1.1: Τι είναι τα P2P δίκτυα;

Τα Peer-to-Peer δίκτυα (ή P-to-P, ή P2P) είναι δίκτυα ομότιμων και αυτόνομων κόμβων, χωρίς κάποιο κεντρικό συντονιστή, όπου όλοι οι συμμετέχοντες μοιράζονται ένα τμήμα των υλικών τους πόρων (υπολογιστική ισχύ, αποθηκευτικό χώρο, εκτυπωτές, bandwidth,...). Ο διαμοιρασμός των υλικών πόρων είναι απαραίτητος προκειμένου να είναι δυνατή η υποστήριξη υπηρεσιών περιεχομένου πάνω από το δίκτυο(π.χ. διαμοιρασμός αρχείων) [12]. Όπως αντιλαμβανόμαστε, σε ένα δίκτυο ομότιμων κόμβων κάθε συμμετέχων κόμβος ενεργεί και ως πελάτης και ως εξυπηρετητής. Ο κόμβος αυτός "αποπληρώνει" τη συμμετοχή του στο δίκτυο προσφέροντας πρόσβαση σε κάποιες από τις πηγές του και μέρος των υλικών του πόρων [9].

1.2: Ιστορική αναδρομή – οι νέες προκλήσεις που παρουσιάζονται

Τα P2P συστήματα έγιναν δημοφιλή τα τελευταία χρόνια μέσα από τις εφαρμογές του διαδικτύου. Αυτό που κάνει τα P2P συστήματα ανταγωνιστικά, είναι η ικανότητά τους να μοιράζονται πόρους, έτσι ώστε οι εξυπηρετητές μεγάλου κόστους να μπορούν να αντικατασταθούν από συστήματα υπολογιστικά ασθενέστερων άρα και φθηνότερων υπολογιστών. Η μεγαλύτερη πρόκληση στην κατασκευή ενός αποτελεσματικού P2P συστήματος είναι πώς όλα τα διάφορα αυτόνομα υπολογιστικά μέρη θα οργανωθούν σε ένα συνεκτικό και ταυτόχρονα αποτελεσματικό σύστημα.. Αυτό συνήθως γίνεται μέσω ενός ιδεατού δικτύου, που βρίσκεται πάνω από το πραγματικό δίκτυο που απαρτίζουν οι κόμβοι που συμμετέχουν στο σύστημα. Το εικονικό δίκτυο χρησιμοποιείται για να οργανώσει τα δεδομένα και τους πόρους που διαχειρίζονται οι υπολογιστές του συστήματος (ή αλλιώς σταθμοί), οι οποίοι στο εικονικό δίκτυο αντιμετωπίζονται ως κόμβοι .

Μέσα από την πορεία των P2P συστημάτων έχουν προταθεί διάφορες τοπολογίες για το εικονικό αυτό δίκτυο, όπως έχουν προκύψει επίσης και κάποια επιθυμητά

χαρακτηριστικά. Όλα αυτά θα παρουσιαστούν ένα-ένα και με τη σειρά με την οποία προέκυψαν στη μέχρι τώρα ιστορία των P2P συστημάτων [9] [13].

- *Διαμοιρασμός Πόρων (sharing-resources)*: Είναι ένα βασικό χαρακτηριστικό και η κινητήρια ιδέα των P2P δικτύων. Όλα τα P2P συστήματα περιλαμβάνουν μια πτυχή της κοινής εκμετάλλευσης πόρων, όπου οι πόροι μπορούν να είναι φυσικοί πόροι, όπως το εύρος ζώνης δικτύων ή μέρος του δίσκου, καθώς επίσης και λογικοί πόροι, όπως υπηρεσίες, δεδομένα ή διαφορετικοί τύποι γνώσης. Με τη διανομή των πόρων μπορούν να εκτελεστούν εφαρμογές οι οποίες δεν θα μπορούσαν να οργανωθούν από έναν ενιαίο κόμβο.
- *Δυναμική εισαγωγή και διαγραφή κόμβων από το δίκτυο*: Το μέγεθος του δικτύου θα πρέπει να αλλάζει δυναμικά. Οποιοσδήποτε κόμβος επιθυμεί να εισέλθει στο δίκτυο (ή να αποχωρήσει από αυτό) να μπορεί να το κάνει χωρίς να προκαλούνται προβλήματα συνέπειας στο σύστημα.

Έτσι ξεκινούν τα P2P συστήματα και ένας από τους πρωτοπόρους είναι το σύστημα Napster. Είναι ένα από τα πιο δημοφιλή συστήματα αυτού του είδους, το οποίο έγινε διάσημο ως σύστημα ανταλλαγής μουσικών κομματιών που πρόσφερε πρόσθετα και άλλες υπηρεσίες αρεστές στους χρήστες. Από τεχνικής άποψης το Napster είναι ένα αρκετά απλό σύστημα ομότιμων κόμβων. Περιέχει έναν εξυπηρετητή, τον Napster εξυπηρετητή, ο οποίος διατηρεί μια βάση δεδομένων όλων των μουσικών αρχείων που προσφέρονται από τους κόμβους που βρίσκονται την τρέχουσα στιγμή στο δίκτυο. Κάθε κόμβος συνδέεται και στέλνει στον εξυπηρετητή τον κατάλογο αρχείων που μπορεί να προσφέρει. Επίσης όταν ένας κόμβος κάνει αναζήτηση κάποιου κομματιού στέλνει το αίτημά του στον εξυπηρετητή του Napster και λαμβάνει πίσω τη λίστα των κόμβων που μπορούν να προσφέρουν το αρχείο αυτό. Έπειτα ο αιτών κόμβος διαλέγει ένα κόμβο από τη λίστα και ζητά να του μεταφορτώσει το αρχείο. Όπως παρατηρούμε το Napster δεν είναι γνήσιο P2P σύστημα αλλά ένας συνδυασμός P2P και τεχνοτροπίας πελάτη/εξυπηρετητή. Γι' αυτό περνάμε σε δύο άλλα χαρακτηριστικά αυτά της αποκέντρωσης και της αυτόνομης οργάνωσης.

- *Αποκέντρωση (decentralization)*: Κάποια μέρη του συστήματος ή και ολόκληρο το σύστημα δε διαχειρίζονται πλέον κεντρικά όπως γινόταν με το Napster. Η αποκέντρωση είναι ιδιαίτερα σημαντική, προκειμένου να αποφευχθούν αποτυχίες ή συμφορήσεις στο δίκτυο.
- *Αυτόνομη Οργάνωση (self-organization)*: Όταν ένα P2P σύστημα είναι πλήρως αποκεντρωμένο, δεν υφίσταται πλέον κόμβος που να μπορεί να συντονίσει κεντρικά τις διάφορες δραστηριότητες ή μια βάση δεδομένων που να αποθηκεύει τις καθολικές πληροφορίες για το σύστημα. Επομένως οι κόμβοι πρέπει να οργανωθούν ατομικά, μόνοι τους, βασισμένοι κυρίως σε όποιες τοπικές πληροφορίες είναι διαθέσιμες, αλληλεπιδρώντας ταυτόχρονα με τους γειτονικούς τους κόμβους. Η καθολική(global) συμπεριφορά και εικόνα σχηματίζεται σταδιακά, ως αποτέλεσμα των επιμέρους τοπικών πράξεων και συμπεριφορών.

Παραδείγματα πλήρως αποκεντρωμένων συστημάτων είναι το Gnutella και το Freenet. Από τεχνικής άποψης το σύστημα Gnutella είναι ένα αποκεντρωμένο σύστημα διαμοιραζόμενων αρχείων, του οποίου οι συμμετέχοντες κόμβοι διαμορφώνουν ένα ιδεατό δίκτυο επικοινωνώντας μέσω του πρωτοκόλλου Gnutella. Για να συμμετέχει στο Gnutella ένας κόμβος πρέπει να συνδεθεί με ένα γνωστό του κόμβο που συμμετέχει ήδη στο Gnutella. Το πρωτόκολλο Gnutella που χρησιμοποιείται για τη κατανεμημένη αναζήτηση αρχείων είναι αρκετά απλό. Ένας κόμβος που επιθυμεί κάποιο αρχείο στέλνει την αίτηση αναζήτησης σε όλους τους γειτονικούς του κόμβους και κάθε κόμβος που λαμβάνει μια αίτηση αναζήτησης αν δεν ικανοποιεί την ερώτηση την προωθεί κι αυτός με τη σειρά του στους γείτονές του. Όταν βρεθεί κάποιος κόμβος που ικανοποιεί την ερώτηση αναζήτησης, η απάντηση επιστρέφεται πίσω από το ίδιο μονοπάτι. Για τη διαδικασία αναζήτησης λοιπόν χρησιμοποιείται η μέθοδος της πλημμύρας (flooding). Η μέθοδος αυτή δεν είναι καθόλου αποδοτική καθώς κατασπαταλιούνται πόροι του δικτύου και η απόδοση πέφτει κατακόρυφα όσο το μέγεθος του δικτύου αυξάνεται. Παρόμοια τεχνική αναζήτησης ακολουθείται και στο FreeNet, μόνο που εδώ έχουμε αναζήτηση κατά βάθος και όχι κατά πλάτος όπως στο Gnutella. Προκειμένου να βελτιωθεί η αποδοτικότητα αναζήτησης, το FreeNet διατηρεί πίνακες δρομολόγησης που ενημερώνονται δυναμικά, καθώς συμβαίνουν αναζητήσεις και εισαγωγές νέων

στοιχείων. Οι πίνακες δρομολόγησης στο FreeNet αποθηκεύουν τις διευθύνσεις των γειτονικών κόμβων, τα κλειδιά των δεδομένων που αποθηκεύουν οι κόμβοι που είναι καταχωρημένοι, και τα αντίστοιχα δεδομένα. Όταν ένας πίνακας δρομολόγησης είναι πλήρης, η πολιτική αντικατάστασης που εφαρμόζεται είναι αυτή της λιγότερο πρόσφατης καταχώρησης που χρησιμοποιήθηκε. Μπορεί επίσης να αρχίσουν να διαγράφονται πρώτα κάποια δεδομένα για να αδειάσει γρήγορα αρκετός αποθηκευτικός χώρος ενώ οι αντίστοιχες πληροφορίες για τους γειτονικούς κόμβους και τα κλειδιά τους να διατηρούνται. Έτσι, ο γράφος του FreeNet εξελίσσεται δυναμικά με το πέρασμα του χρόνου, μέσα από τις ενημερώσεις των πινάκων δρομολόγησης. Για την παραπέρα βελτίωση της απόδοσης της αναζήτησης, το FreeNet χρησιμοποιεί δυναμική απόκριση σε αναζητήσεις των πιο δημοφιλών αρχείων έτσι ώστε τα αρχεία να μπορούν να μεταφέρονται σε κόμβους όπου είναι περισσότερο πιθανό να βρεθούν. Η απόδοση του Freenet όμως είναι δύσκολο να εκτιμηθεί, δεν παρέχει καμία εγγύηση για την καθυστέρηση της αναζήτησης και επίσης επιτρέπει σε προσβάσιμα δεδομένα να χαθούν, γεγονός που φέρνει το σύστημα σε μη συνεπή κατάσταση. Από όσα συζητήθηκαν για το Gnutella και το Freenet, προκύπτουν άλλα δύο επιθυμητά χαρακτηριστικά: η κλιμάκωση και η διαθεσιμότητα των δεδομένων.

- *Κλιμάκωση (scalability)*: Το σύστημα συνεχίζει να αποδίδει ικανοποιητικά με την προσθήκη επιπλέον κόμβων και δεδομένων.
- *Διαθεσιμότητα Δεδομένων(data availability)*: Διαθεσιμότητα δεδομένων αλλά και υπηρεσιών παρά τις βλάβες που μπορεί να συμβούν ανά πάσα στιγμή και σε οποιοδήποτε κόμβο.

Με όλα όσα αναφέραμε παραπάνω αλλά και με ένα επιπλέον χαρακτηριστικό αυτό της εξισορρόπησης φόρτου, περνάμε σε μια νέα γενιά P2P συστημάτων με αντιπροσώπους όπως το CAN, το Chord , το Pastry, το Tapestry και τοViceroy. Τα συστήματα αυτά χρησιμοποιούν έναν κατανεμημένο πίνακα κατακερματισμού (distributed hast table, DHT). Κάθε πόρος συσχετίζεται με ένα κλειδί μέσω μιας συνάρτησης κατακερματισμού και ορίζεται σε ποιόν κόμβο θα αποθηκευτούν τα εκάστοτε ζεύγη κλειδί-πόρος. Όλος ο φόρτος των δεδομένων κατανέμεται ομοιόμορφα στους κόμβους του συστήματος. Το

κύριο ζήτημα σ' αυτά τα συστήματα είναι η ανάκτηση του αναγνωριστικού του κόμβου που αποθηκεύει κάποιο συγκεκριμένο πόρο διακρίνοντας τον από όλους τους κόμβους του δικτύου. Ένας εικονικός γράφος σχηματίζεται, σύμφωνα με τον οποίο, η θέση των κόμβων και των πόρων καθορίζεται από τις κατακερματισμένες τιμές των αναγνωριστικών και των κλειδιών αντίστοιχα. Η τοποθέτηση των πόρων και η αναζήτηση μέσω του εικονικού γράφου γίνονται στα διάφορα συστήματα χρησιμοποιώντας διάφορους αλγορίθμους δρομολόγησης. Το Pastry και το Tapestry χρησιμοποιούν τον αλγόριθμο του Plaxton, ο οποίος βασίζεται στην υπέρ-κυβική δρομολόγηση, όπου η αίτηση προωθείται ντετερμινιστικά σε έναν γείτονα του οποίου το αναγνωριστικό είναι κατά ένα ψηφίο πιο κοντά στο αναγνωριστικό του κόμβου «στόχου». Το CAN χρησιμοποιεί έναν πολυδιάστατο χώρο, χωρισμένο σε ζώνες, σαν υποκείμενη αφαίρεση για τη δρομολόγηση. Τα κλειδιά αντιστοιχίζονται στις ζώνες και κάθε κόμβος ορίζεται υπεύθυνος για μια ζώνη (άρα και για κάποια κλειδιά) αποθηκεύοντας παράλληλα τα αναγνωριστικά των κόμβων που είναι υπεύθυνοι για τις γειτονικές ζώνες. Για τη δρομολόγηση χρησιμοποιείται άπληστος αλγόριθμος σύμφωνα με τον οποίο οι αιτήσεις διαβιβάζονται στις ζώνες που βρίσκονται πιο κοντά στη ζώνη «στόχο». Κατ' αυτό τον τρόπο ο αναμενόμενος αριθμός βημάτων για μια αναζήτηση είναι ο $(d \cdot N^{1/d})$, όπου το d είναι η διάσταση του χώρου (dimensionality).

- *Εξισορρόπηση Φόρτου (load balancing)*: Η εξισορρόπηση φόρτου είναι απαραίτητη έτσι ώστε τα δεδομένα να είναι ομοιόμορφα κατανεμημένα στους κόμβους. Είναι μια διαδικασία που πρέπει να επαναλαμβάνεται ανά τακτά χρονικά διαστήματα στο δίκτυο, έτσι ώστε το σύστημα να παραμένει ισορροπημένο. Επίσης, οι περισσότεροι αλγόριθμοι αναζήτησης που εφαρμόζονται σε P2P συστήματα, αποδίδουν πολύ καλύτερα όταν τα δεδομένα είναι ομοιόμορφα τοποθετημένα στο δίκτυο.
- *Ανοχή Βλαβών (fault tolerance)*: Ένα κατανεμημένο P2P σύστημα πρέπει να σχεδιαστεί και να υλοποιηθεί έτσι ώστε να ανέχεται βλάβες. Κατά το σχεδιασμό του θα πρέπει να προβλεφθεί πια θα είναι η απόδοση αλλά και η κατάσταση του συστήματος όσο ένας αριθμός κόμβων ή/και συνδέσεων καταστρέφεται. Οι κυριότερες μέθοδοι που εφαρμόζονται είναι η διατήρηση αντιγράφων των

δεδομένων, τα μόνιμα δεδομένα(η διατήρηση κάποιων δεδομένων σε μόνιμη αποθήκη) και τα εναλλακτικά μονοπάτια για την προσέγγιση κάθε κόμβου.

Πριν προχωρήσουμε στην αναλυτικότερη εξήγηση του σχήματος Chord [11] να αναφέρουμε κάποια επιπλέον επιθυμητά χαρακτηριστικά. Όταν γίνεται μια αναζήτηση, καλό θα ήταν, να λαμβάνεται υπόψη και η πραγματική γεωγραφική θέση των κόμβων κατά τη δρομολόγηση των αιτημάτων, έτσι ώστε να εξοικονομούνται πόροι και χρόνος. Μια άλλη ποιοτική απαίτηση που μπορεί να συμπεριληφθεί είναι η «φήμη» των κόμβων έτσι ώστε να μπορέσουν να αποφευχθούν κάποιοι κόμβοι που θεωρούνται αναξιόπιστοι. Αυτό θα βοηθούσε και γενικά τη φήμη των P2P συστημάτων, καθώς πολλοί υποστηρίζουν πως πολλοί κόμβοι που συμμετέχουν σε P2P δίκτυα συνήθως δεν είναι έμπιστοι.

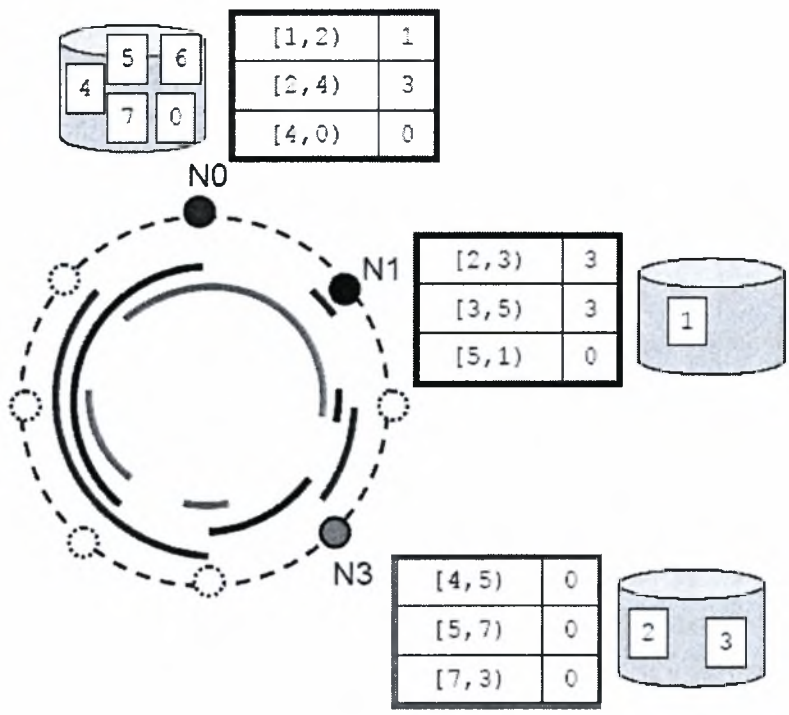
1.3: Ένα αναλυτικό παράδειγμα: Το Chord

Ένα από τα πιο αντιπροσωπευτικά παραδείγματα P2P συστημάτων που χρησιμοποιούν κατανεμημένο ευρετήριο (DHT ή κατανεμημένο πίνακα κατακερματισμού όπως αναφέραμε παραπάνω) είναι το σύστημα Chord. Θεωρούμε ότι όλοι οι κόμβοι του συστήματος σχηματίζουν έναν εικονικό δακτύλιο και καθένας τους έχει ένα μοναδικό αναγνωριστικό(π.χ. με βάση τη διεύθυνση του). Τα αναγνωριστικά είναι διατεταγμένα και σταθερού μεγέθους. Επίσης σε κάθε πληροφοριακό πόρο ανατίθεται και ένα αναγνωριστικό(κλειδί). Στον κόμβο με αναγνωριστικό n αποθηκεύονται όλες οι εγγραφές (διευθύνσεις) για τους πληροφοριακούς πόρους με κλειδί k, όπου $previousNode(n) < k \leq n$. Υποθέτουμε ότι τα αναγνωριστικά των πληροφοριακών πόρων καθώς και των κόμβων που υπάρχουν ανά πάσα χρονική στιγμή στο σύστημα είναι τέτοια, ώστε (για μεγάλα συστήματα) οι εγγραφές του ευρετηρίου να ισοκατανέμονται στους κόμβους του συστήματος [14].

Η δρομολόγηση στο Chord

Κάθε κόμβος n διατηρεί πίνακα δρομολόγησης(finger table) με m=O(logN) γραμμές, όπου κάθε γραμμή i $(0 \leq i < m)$ περιέχει το αναγνωριστικό του κόμβου που αντιστοιχεί στο διάστημα τιμών [n+2i, n+2i+1). Όταν ένας κόμβος αναζητεί την εγγραφή με τιμή k,

και δεν είναι ο ίδιος υπεύθυνος για αυτή, προωθεί την αίτηση στον κόμβο που αντιστοιχεί στο διάστημα που περιέχει το k, σύμφωνα με το πίνακα δρομολόγησης που διατηρεί. Στην εικόνα 1.1, φαίνεται ένα παράδειγμα του συστήματος Chord με 8



Πηγή: Σ.Λάλης, Υπηρεσίες Καταλόγου [14]

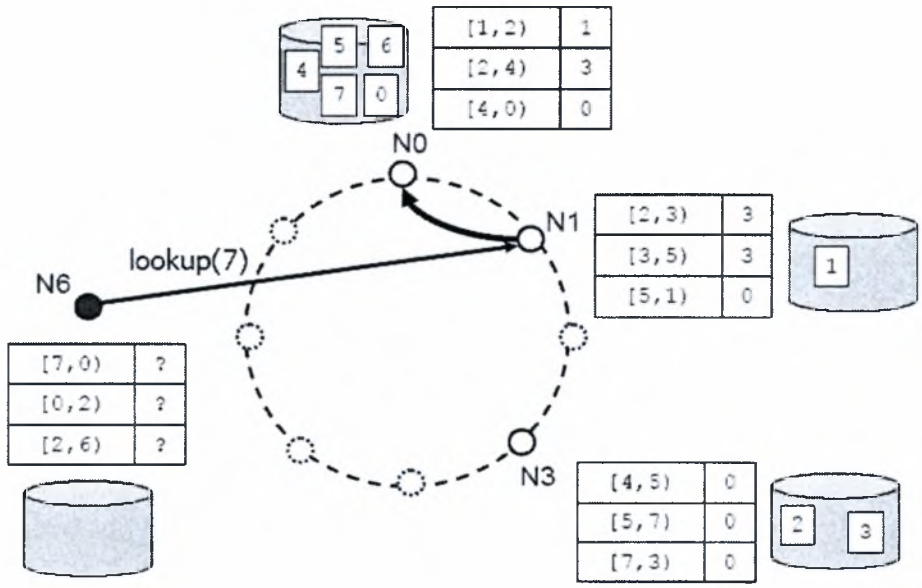
Εικόνα 1.1: Ο εικονικός δακτύλιος και οι finger tables του Chord

πληροφοριακούς πόρους και 3 κόμβους. Οι χρωματιστές γραμμές στον εικονικό δακτύλιο δηλώνουν τα διαστήματα τιμών σε κάθε γραμμή του αντίστοιχου finger table. Έστω ότι ζητείται από τον N_1 να βρει την εγγραφή με κλειδί 4. Ο N_1 δεν την έχει, οπότε σύμφωνα με τη δεύτερη γραμμή του finger table του, στέλνει αίτηση lookup(4) στον N_3. Ο N_3 λαμβάνει την αίτηση και αφού δεν έχει ούτε αυτός την εγγραφή 4, προωθεί την αίτηση στον N_0. Τελικά, ο N_0 στέλνει στον N_1 την εγγραφή του πόρου 4.

Προσθήκη νέου κόμβου

Όταν προστίθεται ένας νέος κόμβος, πρέπει να ανακατανεμηθούν οι εγγραφές του ευρετηρίου και να προσαρμοστούν κατάλληλα οι πίνακες δρομολόγησης. Ακολουθείται μια διαδικασία τεσσάρων βημάτων:

1. Αρχικοποίηση του πίνακα finger στον νέο κόμβο.
2. Αντιγραφή των καταχωρήσεων που αντιστοιχούν στον νέο κόμβο (μόνο από τον αμέσως «επόμενο» κόμβο).
3. Ενημέρωση των πινάκων finger των υπολοίπων.
4. Απομάκρυνση των καταχωρήσεων που μεταφέρθηκαν στον νέο κόμβο από τον αμέσως επόμενο κόμβο.

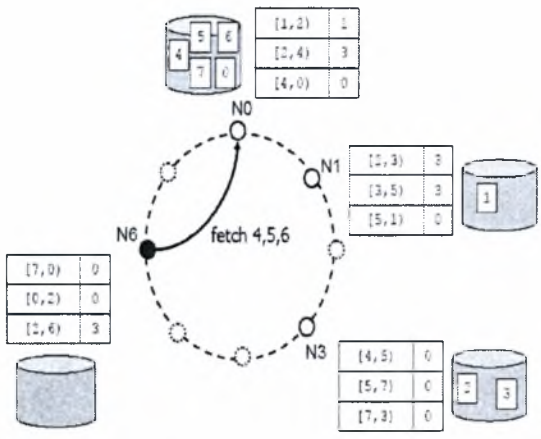


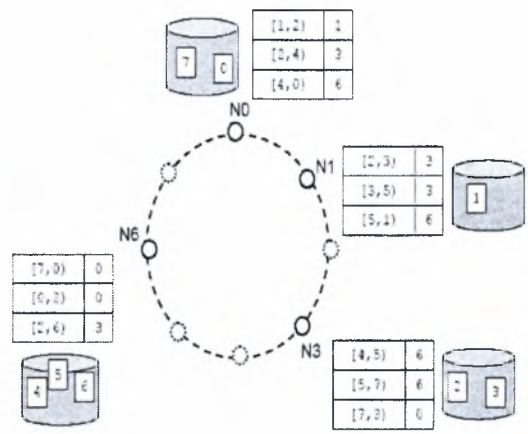
Πηγή: Σ.Λάλης, Υπηρεσίες Καταλόγου [14]

Εικόνα 1.2: 1° Βήμα – Αρχικοποίηση του finger table στο νέο κόμβο

Στην εικόνα 1.2, φαίνεται το 1° βήμα της διαδικασίας εισαγωγής του κόμβου N_6. Ο N_6 Θα στείλει τρεις αιτήσεις lookup (lookup(7), lookup(0), lookup(2)) σε έναν κόμβο, εδώ τον N_1, για να μπορέσει να κατασκευάσει τον πίνακα δρομολόγησης. Έπειτα, θα ζητήσει από τον N_0 που είναι ο επόμενος κόμβος στο δακτύλιο να αντιγράψει τις εγγραφές με κλειδιά 4, 5 και 6 (εικόνα 1.3). Στο επόμενο βήμα, θα ενημερώσει τους υπόλοιπους

κόμβους και για κάθε γραμμή σε κάθε finger table, όπου κάποιο από τα διαστήματα ξεκινά με 4, 5 ή 6, θα αλλάξει ο κόμβος της συγκεκριμένης γραμμής και θα γίνει N_6. Τέλος, διαγράφονται οι καταχωρήσεις 4, 5 και 6 από τον N_0 καθώς τώρα είναι υπεύθυνος για αυτές ο N_6. Η προσθήκη του κόμβου έχει ολοκληρωθεί και το σύστημα βρίσκεται στην κατάσταση που φαίνεται στην εικόνα 1.4.





Πηγή: Σ.Λάλης, Υπηρεσίες Καταλόγου [14]

Εικόνα 1.3: Αντιγραφή καταχωρήσεων από τον N_0

Εικόνα 1.4: Τελική κατάσταση

Αντίστοιχη διαδικασία ακολουθείται όταν ένας κόμβος επιθυμεί να αποχωρήσει από το σύστημα. Κάθε αναζήτηση απαιτεί O(logN) μηνύματα κατά μήκος μιας αλυσίδας ακμών μήκους O(logN). Η ενημέρωση των πινάκων δρομολόγησης όταν εκτελείται μια εισαγωγή ή μια διαγραφή ενός κόμβου στο δίκτυο κοστίζει $O(\log^2 N)$.

1.4: Η ανάγκη για ερωτήσεις με εύρος τιμών

Το Chord όπως και τα παραπάνω DHT P2P συστήματα που αναφέραμε χρησιμοποιούν έναν κατανεμημένο πίνακα κατακερματισμού. Ο κατακερματισμός, όπως είναι γνωστό, καταστρέφει τη διάταξη των κλειδιών οπότε δε μπορούμε να βγάλουμε κανένα συμπέρασμα για τη σχετικότητα των δεδομένων που φιλοξενούν οι κόμβοι. Ως

αποτέλεσμα, τα παραπάνω συστήματα δε μπορούν με τη τρέχουσα μορφή που έχουν να υποστηρίξουν αποτελεσματικά ερωτήσεις με εύρος τιμών. Έτσι, γεννιέται η ανάγκη για σχεδιασμό συστημάτων τα οποία να μπορούν να χειρίζονται αποτελεσματικά πιο σύνθετες ερωτήσεις , όπως ερωτήσεις που περιέχουν εύρος τιμών. Πριν περάσουμε στη συζήτηση των κατευθύνσεων που δημιουργήθηκαν για να λύσουν αυτό το ζήτημα να ορίσουμε καλύτερα τον όρο «ερώτηση με εύρος τιμών».

Θεωρούμε ένα σύνολο διατεταγμένων τιμών, έστω το σύνολο ακεραίων [0, N]. Κάθε ερώτηση που περιλαμβάνει διαστήματα της μορφής [ν1, ν2] ή (ν1, ν2), ή [ν1, ν1), ή (ν1, ν2] ή ακόμα και ενώσεις διαστημάτων όπως [ν1, ν2] ∪[ν3, ν4] ∪.. Ψνκ, νλ] , όπου στη θέση της αγκύλης], [μπορεί να μπει το σύμβολο της παρένθεσης), (και το αντίστροφο, καθώς και κάθε ερώτηση τύπου «∀x > ν1» με χρήση όλων των συμβόλων <, >, =, != , όπως και συνδυασμό τους με χρήση λογικών προτάσεων, είναι μια «ερώτηση με εύρος τιμών».

1.5: Οργάνωση της εργασίας

Οι λύσεις που προτείνονται για να μπορέσουν να υποστηριχθούν και οι ερωτήσεις εύρους αποτελεσματικά από τα P2P συστήματα εστιάζονται σε τρεις κατευθύνσεις, ανάλογα με την τοπολογία του εικονικού δικτύου που βρίσκεται πάνω από το πραγματικό δίκτυο των κόμβων. Στο 2^{ο} κεφάλαιο αναλύουμε τις δενδρικές δομές που έχουν παρουσιαστεί κι αυτές είναι ένα δυαδικό ισορροπημένο δέντρο το BATON, η επεκταμένη δομή BATON* όπου το δέντρο εδώ έχει fan-out m και όχι 2 όπως στο BATON και τέλος ένα δυαδικό δέντρο εικονικού ευρετηρίου που αναφέρεται σε πολυδιάστατες δενδρικές δομές. Στο 3^{ο} κεφάλαιο παρουσιάζουμε τους γράφους αναπηδήσεως (skip graphs) μια λύση που στηρίζεται στις γνωστές λίστες αναπηδήσεως. Στο 4^{ο} κεφάλαιο περνάμε σε μια τοπολογία δυαδικού, αλλά όχι απαραίτητα ισορροπημένου, δέντρου για το εικονικό δίκτυο, το P-Grid. Τέλος, στο 5^{ο} κεφάλαιο παραθέτουμε συμπεράσματα από την ανάλυση των παραπάνω δομών.

Κεφάλαιο 2: Δενδρικές Δομές

2.1: Η δομή BATON

Στα κεντρικοποιημένα συστήματα βάσεων δεδομένων είναι πολύ συχνή η χρήση δέντρων ως δομές οργάνωσης των δεδομένων, όπως για παράδειγμα το B-δέντρο. Στα κατανεμημένα συστήματα όμως αποφεύγεται η χρήση τους, γιατί σε τέτοιες δομές η ρίζα και οι κόμβοι που βρίσκονται κοντά της έχουν την τάση να εμφανίζουν μεγαλύτερο ρυθμό προσπέλασης σε σχέση με τους υπόλοιπους κόμβους, με αποτέλεσμα να αποτελούν σημεία συμφόρησης. Η δομή BATON (BAlanced Tree Overlay Network) έρχεται να αντιμετωπίσει αυτή τη πρόκληση και χρησιμοποιώντας ένα δυαδικό ισορροπημένο δέντρο καταφέρνει να μην υπερφορτώνει την περιοχή της ρίζας.

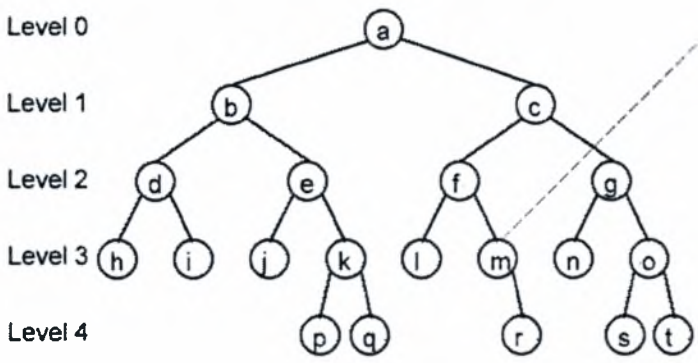
2.1.1: Βασικά Χαρακτηριστικά της δομής BATON

Το BATON [1] είναι ένα εικονικό δίκτυο πάνω από το πραγματικό δίκτυο σε τοπολογία δυαδικού ισορροπημένου δέντρου. Λόγω της δυαδικότητας κληρονομεί άμεσα τα πλεονεκτήματα της σχετικά εύκολης κλιμάκωσης και στιβαρότητας που εμφανίζουν τα δυαδικά δέντρα. Κάθε κόμβος στο δέντρο θεωρούμε ότι αντιστοιχεί ακριβώς σε έναν σταθμό στο P2P δίκτυο. Επίσης, κάθε κόμβος διατηρεί συνδέσεις προς το γονιό του, τα παιδιά του, τους in-order γείτονες του(θα εξηγηθεί παρακάτω λεπτομερέστερα), καθώς και δύο πίνακες δρομολόγησης όπου διατηρεί συνδέσεις προς κάποιους επιλεγμένους κόμβος του ίδιου επιπέδου με αυτόν. Ένα δέντρο BATON κατασκευάζεται με γνώμονα τον παρακάτω ορισμό.

Ορισμός 1*: Ένα δέντρο είναι ισορροπημένο αν και μόνο αν σε κάθε κόμβο, το ύψος των δύο υποδέντρων του διαφέρει το πολύ κατά 1.*

Έτσι ένα δέντρο με N κόμβους έχει ύψος το πολύ 1.44logN. Ένα παράδειγμα ενός δέντρου BATON φαίνεται στην εικόνα 2.1. Συσχετίζουμε κάθε κόμβο με τον αριθμό επιπέδου του στο δέντρο και έναν αριθμό. Η ρίζα βρίσκεται στο επίπεδο 0, τα παιδιά της στο επίπεδο 1 κ.ο.κ. Το επίπεδο κάθε κόμβου είναι κατά ένα μεγαλύτερο από το επίπεδο του γονιού του. Στο επίπεδο L υπάρχουν το πολύ 2^L κόμβοι. Σε κάθε επίπεδο αριθμούμε

αυτές τις 2^L θέσεις από αριστερά προς τα δεξιά από το 1 μέχρι το 2^L ανεξάρτητα από το αν υπάρχει πράγματι ή όχι ένας κόμβος σε μια θέση. Έτσι το επίπεδο και ο αριθμός ενός κόμβου οριοθετούν τη θέση του στο δυαδικό δέντρο.



Node m: level=3, number=6
parent=f, leftchild=null, rightchild=r
leftadjacent=f, rightadjacent=r
Left routing table:

	Node	Left child	Right child	Lower bound	Upper bound
0	l	null	null	l_{lower}	l_{upper}
1	k	p	q	k_{lower}	k_{upper}
2	i	null	null	i_{lower}	i_{upper}

Right routing table:

	Node	Left child	Right child	Lower bound	Upper bound
0	n	null	null	n_{lower}	n_{upper}
1	o	s	t	o_{lower}	o_{upper}

Πηγή: H.V.Jagadish [1]

Εικόνα 2.1: Ένα δέντρο BATON και οι δύο πίνακες δρομολόγησης του κόμβου m.
In-order διαπέραση: h d i b j e p k q a l f m r c n g s o t

Δεδομένου ενός κόμβου χ, ο αμέσως προηγούμενος κόμβος στην in-order διαπέραση είναι ο αριστερός γείτονας του χ (left adjacent node), ενώ ο αμέσως επόμενος στην in-order διαπέραση ο δεξιός γείτονάς του(right adjacent node). Παρατηρούμε ότι οι in-order γείτονες ενός κόμβου μπορεί να βρίσκονται σε διαφορετικά επίπεδα. Σε ένα πλήρες δυαδικό δέντρο, στην in-order διαπέραση τα φύλλα και οι εσωτερικοί κόμβοι εναλλάσσονται, δηλαδή και οι δύο γείτονες ενός εσωτερικού κόμβου είναι φύλλα. Ακόμα όμως κι αν το δέντρο δεν είναι πλήρες, αποδεικνύεται ότι κάθε εσωτερικός κόμβος έχει τουλάχιστον έναν in-order γείτονα που είναι είτε φύλλο είτε εσωτερικός κόμβος με μόνο ένα παιδί.

Όπως αναφέρεται και παραπάνω κάθε κόμβος χαρακτηρίζεται από το επίπεδό του και έναν αριθμό, το ζεύγος αυτό αποτελεί το λογικό αναγνωριστικό του κόμβου. Επίσης κάθε κόμβος έχει και ένα φυσικό αναγνωριστικό που είναι η IP διεύθυνσή του κόμβου ή κάποιο άλλο σχετικό αναγνωριστικό δικτύου που να προσδιορίζει τη διεύθυνση του στο πραγματικό δίκτυο.

Κάθε κόμβος διατηρεί τα φυσικά και λογικά αναγνωριστικά του γονιού του, των παιδιών του, του αριστερού και του δεξιού του γείτονα. Επίσης κάθε κόμβος διατηρεί πληροφορία και για επιλεγμένους κόμβους που βρίσκονται στο ίδιο επίπεδο με αυτόν σε μορφή δύο πινάκων δρομολόγησης: τον αριστερό πίνακα δρομολόγησης και τον δεξιό. Ο δεξιός πίνακας δρομολόγησης ενός κόμβου περιέχει «συνδέσεις» προς κόμβους που βρίσκονται στα δεξιά του και σε οποιασδήποτε απόσταση δύναμης του 2 από τον ίδιο. Αντίστοιχα, ο αριστερός πίνακας περιέχει κόμβους που βρίσκονται στα αριστερά του και απέχουν οποιαδήποτε απόσταση δύναμης του 2 από τον ίδιο δηλ. ένας κόμβος, έστω χ, στον αριστερό πίνακα δρομολόγησης περιέχει τον κόμβο που βρίσκεται $2^0 = 1$ θέση αριστερά του (δηλ. δίπλα του), $2^1 = 2$ θέσεις αριστερά του, $2^2 = 4$ θέσεις αριστερά του κ.ο.κ. Ακόμα κι αν δεν υπάρχει κάποιος κόμβος σε κάποια από τις θέσεις αυτές, δημιουργείται καταχώριση στον πίνακα, η οποία σημειώνεται ως null. Ένας πίνακας δρομολόγησης θεωρείται πλήρης αν δεν υπάρχει καμία null καταχώριση. Όπως φαίνεται και στην εικόνα 2.1, ο κόμβος m στον αριστερό πίνακα δρομολόγησης έχει τους κόμβους *l*, k και *i* ενώ ο δεξιός, τους κόμβους n και ο. Να σημειωθεί επίσης ότι και οι δύο πίνακες δρομολόγησης του m είναι πλήρεις καθώς καμία θέση σε απόσταση κάποιας δύναμης του 2 από τον m δεν είναι κενή. Άλλα παραδείγματα είναι ο κόμβος h που έχει μόνο δεξί πίνακα δρομολόγησης καθώς είναι ο αριστερότερος κόμβος του επιπέδου 3, και στο επίπεδο 4 κανένας κόμβος δεν έχει πλήρη πίνακα δρομολόγησης καθώς απουσιάζουν πολλοί κόμβοι. Τέλος, στους πίνακες δρομολόγησης κρατείται επιπλέον πληροφορία για το ποια παιδιά έχουν οι γειτονικοί κόμβοι ίδιου επιπέδου καθώς και τα άνω και κάτω όρια (upper and lower bounds) από τα εύρη τιμών των κόμβων που βρίσκονται καταχωρημένοι. Για τα εύρη τιμών θα γίνει εκτενέστερη αναφορά σε παρακάτω ενότητα (2.1.8 – 2.1.10)

2.1.2: Βασικά Θεωρήματα που ισχύουν στη δομή BATON

Ακολουθεί η περιγραφή δύο βασικών θεωρημάτων που ισχύουν στη δομή BATON τα οποία βοηθούν πολύ στις διαδικασίες που θα αναπτυχθούν σε παρακάτω ενότητες.

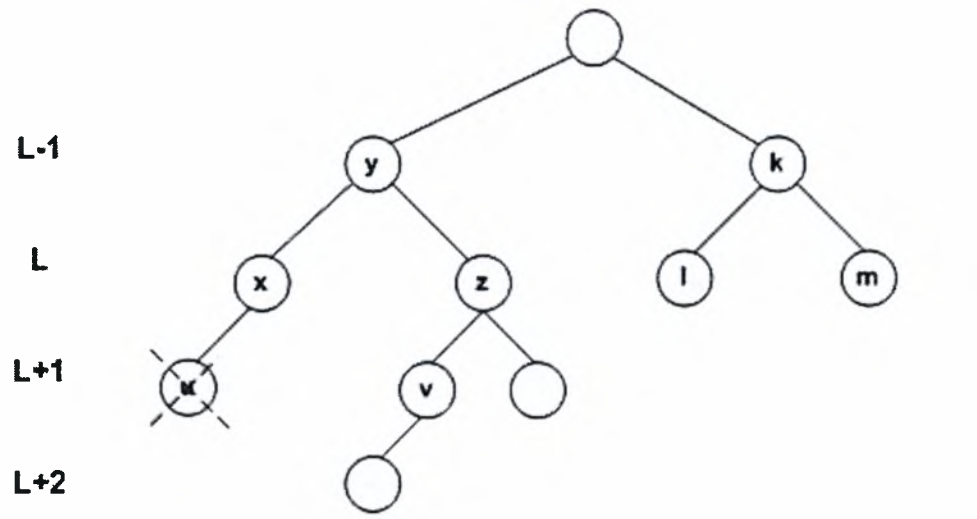
Θεώρημα 1: *Ένα δέντρο είναι ισορροπημένο αν κάθε κόμβος που έχει παιδί έχει επίσης και τους δύο πίνακες δρομολόγησης πλήρεις.*

Απόδειξη : Θεωρούμε την εισαγωγή ενός νέου κόμβου σε ένα ισορροπημένο δέντρο. Έστω ότι ο νέος αυτός κόμβος μπορεί να προστεθεί ως παιδί του χ. Έστω ότι ο κόμβος χ είναι στο επίπεδο L και ο νέος κόμβος στο επίπεδο L+1. Το νέο δέντρο θα μπορούσε να είναι μη ισορροπημένο αν σε κάποιο πρόγονο του χ, το βάθος του αριστερού και δεξιού υποδέντρου διαφέρουν περισσότερο από 1 εξαιτίας αυτής της νέας εισαγωγής. Έστω ο πρόγονος του χ, ο y ο οποίος βρίσκεται στο επίπεδο *i*. Χωρίς παραβίαση της γενικότητας υποθέτουμε ότι ο χ, είναι στο αριστερό υποδέντρο του y. Το βάθος του αριστερού υποδέντρου μπορεί να έχει αλλάξει από L σε L+1 εξαιτίας της νέας εισαγωγής(εάν το βάθος ήταν ήδη L+1 ή παραπάνω, τότε τίποτα δεν αλλάζει και το δέντρο εξακολουθεί να είναι ισορροπημένο). Αλλά από τη στιγμή που ο δεξιός πίνακας δρομολόγησης του χ είναι πλήρης, πρέπει να υπάρχει μια καταχώριση στον πίνακα από έναν κόμβο στο δεξί υποδέντρο του y και επιπλέον ο κόμβος αυτός να βρίσκεται στο επίπεδο L. Έτσι το δεξί υποδέντρο του y έχει βάθος τουλάχιστον L. Έτσι, παρατηρούμε ότι μια αλλαγή στο βάθος στο αριστερό υποδέντρο από L σε L+1 δεν παραβιάζει την ισορροπία. Εφαρμόζοντας το ίδιο επιχείρημα σε κάθε πρόγονο του χ στο δέντρο, μπορούμε να εγγυηθούμε ότι το δέντρο παραμένει ισορροπημένο έπειτα από οποιαδήποτε εισαγωγή κόμβου

Τώρα θεωρούμε διαγραφή ενός κόμβου, έστω u, που είναι παιδί του κόμβου χ στο επίπεδο L. Η διαγραφή αυτή μπορεί να προκαλέσει ανισορροπία σε κάποιο πρόγονο, έστω y, του χ, αν το βάθος του χ υποδέντρου αλλάξει από L+1 σε L, ενώ το άλλο υποδέντρο έχει βάθος L+2. Χωρίς παραβίαση της γενικότητας, υποθέτουμε ότι ο χ βρίσκεται στο αριστερό υποδέντρο του y. Πρέπει να υπάρχει κάποιος κόμβος, έστω z, στο δεξί πίνακα δρομολόγησης του χ και βρίσκεται στο δεξί υποδέντρο του y. Ο κόμβος z βρίσκεται επίσης στο επίπεδο L. Υποθέτοντας ότι το βάθος του δεξιού υποδέντρου του y είναι L+1 ή μικρότερο δεν δημιουργείται κανένα πρόβλημα. Αν όμως το βάθος είναι L+2 ή μεγαλύτερο, προκύπτουν δύο περιπτώσεις:

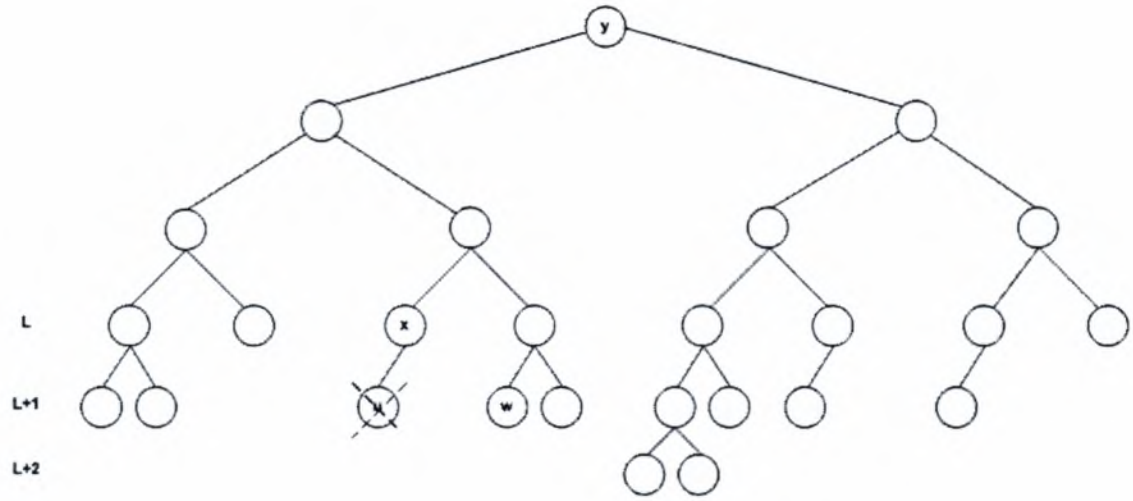
1. Υπάρχει κόμβος ν, παιδί του z, ο οποίος βρίσκεται στον δεξιό πίνακα δρομολόγησης του u και έχει και παιδί. Ο ν έχει παιδί αλλά δε μπορεί να έχει τους πίνακες δρομολόγησής του πλήρεις, γιατί τότε θα είχε κι άλλο κόμβο καταχωρημένο στον αριστερό πίνακα δρομολόγησης, ο οποίος θα βρισκόταν

στο αριστερό υποδέντρο του y και στο επίπεδο L+1, διαφορετικό από τον u, άρα με τη διαγραφή του u δε θα είχαμε πρόβλημα ανισορροπίας. Έτσι αφού ο ν δε μπορεί να έχει τους πίνακες δρομολόγησής του πλήρεις, δε μπορεί να έχει παιδί με βάση το θεώρημα. Έτσι δε δημιουργείται καμία ανισορροπία. Η περίπτωση αυτή παρουσιάζεται στην εικόνα 2.2



Εικόνα 2.2: 1η περίπτωση στο σενάριο διαγραφής του u.

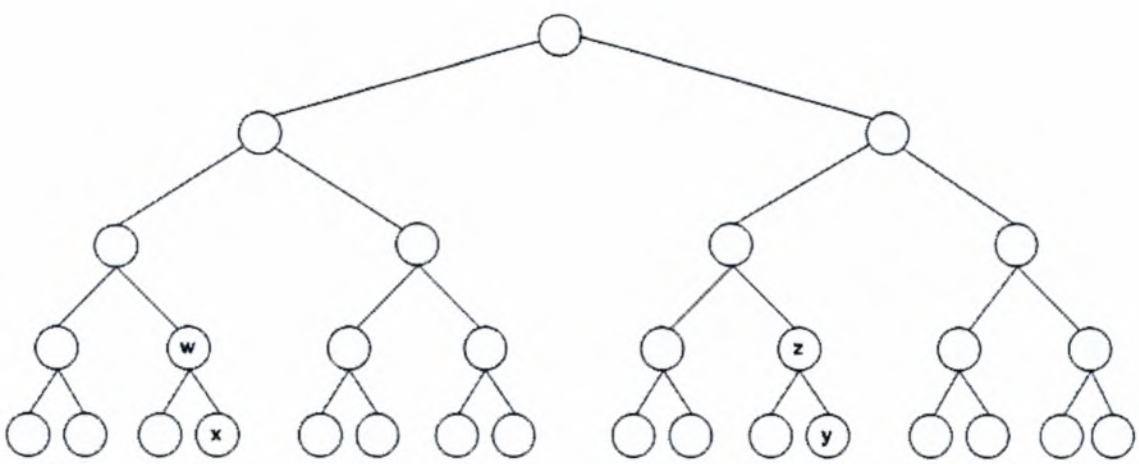
2. Δεν υπάρχει τέτοιος κόμβος ν. Αυτό σημαίνει πως κάθε κόμβος στο επίπεδο L+2 στο δεξί υποδέντρο του y έχει έναν γονιό στο επίπεδο L+1, ο οποίος έχει μια καταχώρηση στον αριστερό του πίνακα δρομολόγησης ενός κόμβου w, ο οποίος βρίσκεται στο αριστερό υποδέντρο του y και είναι όμως διαφορετικός από τον u. Ο κόμβος w είναι στο επίπεδο L+1, οπότε η αναχώρηση του κόμβου u δεν αλλάζει το βάθος του αριστερού υποδέντρου του y. Έτσι, δεν προκαλείται καμία ανισορροπία. Η περίπτωση αυτή παρουσιάζεται στην εικόνα 2.3



Εικόνα 2.3: $2^{η}$ περίπτωση στο σενάριο διαγραφής του u..

Χρησιμοποιώντας το ίδιο επιχείρημα αναδρομικά και σε κάθε πρόγονο του χ, μπορούμε να δείξουμε ότι δεν προκαλείται ανισορροπία στο δέντρο αν ισχύει η συνθήκη του θεωρήματος 1.

Θεώρημα 2: *Αν ένας κόμβος ,έστω χ, περιέχει μια σύνδεση σε έναν άλλο κόμβο έστω y ο οποίος είναι καταχωρημένος στον δεξιό ή αριστερό πίνακα δρομολόγησης, ο γονιός του χ πρέπει επίσης να περιέχει μια σύνδεση στο γονιό του y εκτός κι αν ο ίδιος κόμβος είναι γονιός και του χ και του y (δηλαδή τα χ και y είναι αδέρφια).*



Εικόνα 2.4: Παράδειγμα για το θεώρημα 2.

Απόδειξη: Έστω N_x ο αριθμός του κόμβου χ και w ο γονιός του χ. Χωρίς παραβίαση της γενικότητας, έστω ότι ο χ είναι το δεξί παιδί του w. Τότε ο N_x είναι άρτιος. Πρέπει επίσης να ισχύει $N_w = \frac{N_x}{2}$. Παρομοίως, έστω z ο γονιός του y, πρέπει $N_z = \frac{N_y}{2}$ αν ο N_y είναι άρτιος ή $N_z = \frac{N_y + 1}{2}$ αν ο N_y περιττός.

Περίπτωση 1: Έστω ότι ο y βρίσκεται σε απόσταση τουλάχιστον δύο θέσεων από τον χ. Τότε ισχύει $N_y = N_x \pm 2^k$ με $k \geq 1$(αφού ο y βρίσκεται σε έναν από τους πίνακες δρομολόγησης πρέπει να βρίσκεται 2^k θέσεις μακριά από τον χ). Οπότε ο N_y είναι άρτιος αν και ο N_x είναι άρτιος. Οπότε έχουμε $N_z = N_w \pm 2^{k-1}$ και άρα υπάρχει κάποια σύνδεση από τον w στον z(ο z βρίσκεται σε έναν από τους πίνακες δρομολόγησης του w).

Περίπτωση 2: Έστω ότι ο y βρίσκεται σε απόσταση 1 από τον χ και είναι αδερφός του, άρα έχουν τον ίδιο πατέρα.

Περίπτωση 3: Έστω ότι ο y βρίσκεται σε απόσταση 1 από τον χ χωρίς να είναι αδερφός του. Αφού ο χ είναι το δεξί παιδί του w, τότε ο y είναι ο δεξιός γείτονας (όχι in-order) του χ, οπότε $N_y = N_x + 1$ και N_y περιττός. Οπότε έχουμε $N_z = \frac{N_y + 1}{2} = \frac{N_x + 2}{2} = \frac{N_x}{2} + 1 = N_w + 1 \Rightarrow N_z = N_w + 1$, οπότε και ο N_z είναι ο δεξιός γείτονας του N_w, άρα υπάρχει «σύνδεση» από τον w στον z.

Στο επίπεδο L υπάρχουν το πολύ L καταχωρήσεις σε έναν αριστερό(ή δεξιό) πίνακα δρομολόγησης. Έτσι ο συνολικός αριθμός των καταχωρήσεων είναι O(logN) το ίδιο ασυμπτωτικό όριο με το Chord, παρόλο που στη χειρότερη περίπτωση ο αριθμός των καταχωρήσεων μπορεί να είναι διπλάσιος από αυτόν του BATON και κάθε καταχώρηση είναι επίσης μεγαλύτερη.

2.1.3: Εισαγωγή νέου κόμβου στο δίκτυο

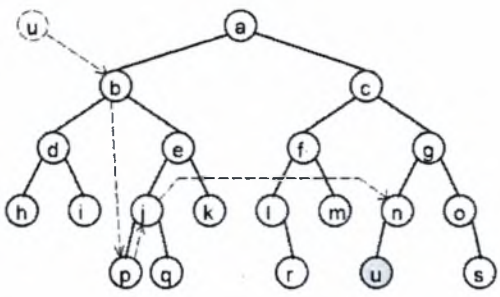
Όταν ένας κόμβος θέλει να εισέλθει στο δίκτυο πρέπει να γνωρίζει τουλάχιστον ένα κόμβο που είναι ήδη συνδεδεμένος στο δίκτυο και να του στείλει αίτηση εισαγωγής, την οποία θα αποκαλούμε αίτηση JOIN.

Υπάρχουν δύο φάσεις κατά την εισαγωγή ενός νέου κόμβου στο δίκτυο.

1. Προσδιορισμός της θέσης του δέντρου όπου θα εισέλθει ο νέος κόμβος.
2. Η εισαγωγή του στην θέση που αποφασίστηκε στη φάση 1.

Όταν ένας κόμβος λάβει αίτηση JOIN, αν έχει και τους δύο πίνακες δρομολόγησης πλήρεις και επιπλέον έχει λιγότερα από δύο παιδιά μπορεί να δεχτεί το νέο κόμβο ως παιδί του. Αλλιώς, θα πρέπει να προωθήσει την αίτηση σε άλλους κόμβους όπως θα δούμε παρακάτω. Ακολουθούν ο αλγόριθμος εισαγωγής σε ψευδοκώδικα και ένα παράδειγμα για την διαδικασία εισαγωγής στην εικόνα 2.5.

```
Algorithm 2.1.1: Join (node n){
if ( Full(LeftRoutingTable(n)) AND Full(RightRoutingTable(n)) AND
    (LeftChild(n) = = null OR RightChild(n) = = null)) )
  { Accept new node as child }
else
 {
   if ( !Full(LeftRoutingTable(n) OR  !Full(RightRoutingTable(n) )
     { Forward the JOIN request to parent(n) }
   else
     {
        m=SomeNodesNotHavingEnoughChildren
         ( LeftRoutingTable(n), RightRoutingTable(n) )
       if( there exists such an m )
          { Forward the JOIN request to m }
       else
        { Forward the JOIN request to one of its adjacent nodes }
     }
 }
}
```



Πηγή: H.V.Jagadish [1]

Εικόνα 2.5: Παράδειγμα Εισαγωγής Κόμβου
In-order διαπέραση:
h d i b p j q e k a l r f m c u n g o s

Έστω ότι ο κόμβος u επιθυμεί να εισέλθει στο δίκτυο και στέλνει αίτηση JOIN στον κόμβο b. Ο b(αφού έχει 2 παιδιά) προωθεί την αίτηση στον p που είναι ο επόμενος κόμβος στην in-order διαπέραση. Οι πίνακες δρομολόγησης του p δεν είναι πλήρεις οπότε στέλνει την αίτηση στον γονιό του j. Ο j, επειδή έχει δύο παιδιά, ελέγχει τους πίνακες δρομολόγησης του και στέλνει την αίτηση στον n που δεν έχει παιδιά. Ο n δέχεται τον u ως παιδί του.

Αναλύοντας τον αλγόριθμο, έστω ότι μέσω γειτονικής σύνδεσης φτάνουμε σε φύλλο, έστω w. Είτε ο w μπορεί να δεχτεί το νέο κόμβο ως παιδί του, είτε οι πίνακες δρομολόγησής του δεν είναι πλήρεις. Στην δεύτερη περίπτωση ο w προωθεί την αίτηση στον πατέρα του, ο οποίος εντοπίζει στον πίνακα δρομολόγησής του έναν κόμβο v, ο οποίος είναι πατέρας από κάποιον γειτονικό κόμβο του w, που όμως λείπει(δεν υπάρχει στο δέντρο). Ο v μπορεί να δεχτεί το νέο κόμβο ως παιδί του εκτός και εάν οι πίνακες δρομολόγησής του δεν είναι πλήρεις, οπότε τότε προωθεί την αίτηση στον πατέρα του. Καθώς το ύψος του δέντρου είναι O(logN), η αίτηση δε μπορεί να προωθηθεί προς τα πάνω περισσότερο από O(logN) φορές. Όλες οι άλλες κατευθύνσεις προώθησης προσθέτουν μόνο σταθερούς όρους. Έτσι έχουμε το O(logN) ως όριο στον αριθμό μηνυμάτων που στέλνονται για να εντοπιστεί η θέση εισαγωγής ενός νέου κόμβου. Ο αλγόριθμος ψάχνει στην ουσία για φύλλα και γονείς κόμβων με μη πλήρεις πίνακες δρομολόγησης, οι οποίοι στην ουσία θα πρέπει να είναι φύλλα λόγω του θεωρήματος 1. Έτσι οι προγονικοί κόμβοι δε χρειάζονται ποτέ και δεν υπάρχει άλλη εμπλοκή της ρίζας στον αλγόριθμο εκτός από αυτήν ενός συνηθισμένου κόμβου. Οπότε αναμένεται ότι η ρίζα δεν επιβαρύνεται δυσανάλογα.

Όταν ένας κόμβος χ δέχεται το νέο κόμβο y ως παιδί του, του μεταβιβάζει τα μισά από τα περιεχόμενα του. Με άλλα λόγια, το εύρος(διάστημα) που σχετίζεται με τον χ μοιράζεται μεταξύ αυτού και του παιδιού του. Αναλυτικά, αν ο y τελικά γίνεται το αριστερό παιδί

του χ, ο χ στέλνει το λογικό και φυσικό αναγνωριστικό του αριστερού in-order γείτονά του, έστω z, στον y και ο ορίζει εκ νέου τον y ως αριστερό του in-order γείτονα. Ο y με τη σειρά του, δημιουργεί μια σύνδεση προ τον χ τον οποίο ορίζει ως τον δεξιό in-order γείτονά του και άλλη μια προς τον z τον οποίο ορίζει ως αριστερό in-order γείτονα. Επίσης, ειδοποιεί τον z να αλλάξει τον δεξί in-order γείτονά του σε y από χ που ήταν πριν. Παρομοίως, αν ο y γίνεται τελικά το δεξί παιδί του χ, τότε ο χ μεταβιβάζει στον y τα στοιχεία του δεξιού in-order γείτονά του. Τελικά, ο χ επικοινωνεί με όλους τους κόμβους που βρίσκονται στους πίνακες δρομολόγησης του και τους ζητά να ενημερώσουν τα αντίστοιχα παιδιά τους σχετικά με τον y και παίρνει ως απάντηση πληροφορία που αφορά τα αντίστοιχα παιδιά των κόμβων που ενημέρωσε, την οποία και θα χρειαστεί ο y για να μπορέσει να κατασκευάσει τους πίνακες δρομολόγησης.

Η όλη διαδικασία απαιτεί O(logN) μηνύματα και O(logN) απαντήσεις. Ειδικότερα, ο χ πρέπει να στείλει το πολύ $2L_1$ μηνύματα στους κόμβους των πινάκων δρομολόγησης, όπου L_1 το επίπεδο του χ. Οι κόμβοι αυτοί πρέπει να στείλουν το πολύ $2L_2$ μηνύματα στα παιδιά τους να τα ενημερώσουν και αυτά θα στείλουν το πολύ $2L_2$ μηνύματα ως απάντηση στο νέο κόμβο, όπου L_2 το επίπεδο του νέου κόμβου. Ο νέος κόμβος χ πρέπει να στείλει μόνο ένα μήνυμα σε έναν από τους δύο in-order γείτονές του. Έτσι, ο μέγιστος αριθμός μηνυμάτων που απαιτούνται για την ανανέωση των πινάκων δρομολόγησης είναι $2L_1 + 2L_2 + 2L_2 + 1 < 6\log N$.

2.1.4: Αποχώρηση κόμβου από το δίκτυο

Μόνο κόμβοι φύλλα μπορούν εθελοντικά να φύγουν από το δίκτυο και μόνο αν η αποχώρησή τους δεν διαταράξει την ισορροπία του δέντρου. Σε άλλες περιπτώσεις, ένας κόμβος που επιθυμεί να αφήσει το δίκτυο πρέπει να βρει έναν αντικαταστάτη, ο οποίος θα είναι ένας κόμβος φύλλο του οποίου η απουσία δε θα επηρεάσει την ισορροπία του δέντρου. Υπάρχουν οι εξής περιπτώσεις:

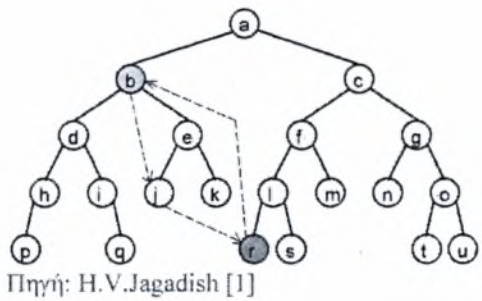
Εάν ένας κόμβος φύλλο, έστω χ, επιθυμεί να φύγει από το δίκτυο και δεν υπάρχει κανένας κόμβος στους πίνακες δρομολόγησης που να έχει παιδιά, μπορεί να φύγει από το δίκτυο χωρίς να επηρεάζει την ισορροπία του δέντρου, καθώς η απαίτηση του

θεωρήματος 1 ακόμα ικανοποιείται. Σε αυτή την περίπτωση, ο χ πρέπει να μεταφέρει όλο του το περιεχόμενο και το διάστημα τιμών για τις οποίες είναι υπεύθυνος, στον πατέρα του και στον αριστερό γειτονικό κόμβο, αν είναι αριστερό παιδί ή τον δεξί αν είναι δεξί παιδί. Επίσης πρέπει να στείλει μηνύματα αποχώρησης στους κόμβους που υπάρχουν στους πίνακες δρομολόγησης του για να ενημερώσουν κι εκείνοι τους δικούς τους πίνακες δρομολόγησης. Ο γονιός του χ, αφού λάβει το περιεχόμενο του χ, πρέπει και να στείλει μηνύματα στους κόμβους γείτονες που βρίσκονται στο ίδιο επίπεδο με τον ίδιο για να τους ενημερώσει για το νέο περιεχόμενο και την αποχώρηση του παιδιού του. Επίσης, ειδοποιεί τον άλλο in-order γείτονα του παιδιού του, έτσι ώστε στη θέση του παιδιού του να ορίσει ως in-order γείτονα τον ίδιο. Άρα προκύπτει ότι ο απαιτούμενος αριθμός μηνυμάτων που πρέπει να σταλούν είναι $2L_1 + 2L_2 + 1 < 4\log N$ όπου L_1 και L_2 είναι τα επίπεδα του γονιού του χ και του χ αντίστοιχα.

```
Algorithm 2.1.2: FindReplacement (node n){
if ( LeftChild(n) != null)
   { Forward the request to RightChild(n) }
else if ( RightChild(n) != null )
   { Forward the request to RightChild(n) }
else
   {
     m=SomeNodesHavingChildrenIn
          ( LeftRoutingTable(n), RightRoutingTable(n) )
       if ( there exists such an m )
          { Forward the request to a child of m }
        else
         { Come to replace the leaving node }
   }
}
```

Αν ένα φύλλο επιθυμεί να αποχωρήσει από το δίκτυο και υπάρχουν γειτονικοί κόμβοι στους πίνακες δρομολόγησης με παιδιά, πρέπει να βρει έναν κόμβο να αφήσει στη θέση του, οπότε στέλνει μια αίτηση εύρεσης αντικαταστάτη σε ένα από τα παιδιά των κόμβων αυτών. Αν ένας εσωτερικός κόμβος επιθυμεί να φύγει, πρέπει κι αυτός να βρει

αντικαταστάτη, οπότε στέλνει αίτηση εύρεσης αντικαταστάτη σε έναν από τους in-order γείτονές του, ο οποίος όμως πρέπει να είναι φύλλο, ή αλλιώς στον βαθύτερο από τους δύο γείτονες. Ο αλγόριθμος εύρεσης αντικαταστάτη ακολουθεί αμέσως μετά.



Πηγή: H.V.Jagadish [1]

Εικόνα 2.6: Ο κόμβος b αποχωρεί από το δίκτυο

Καθώς η διαδικασία εύρεσης αντικαταστάτη πηγαίνει πάντα προς τα κάτω, χρειάζονται το πολύ O(logN) βήματα, τόσα δηλαδή όσο και το ύψος του δέντρου. Π.χ. αν ο κόμβος b στην εικόνα 2.6 θέλει να φύγει από το δίκτυο θα πρέπει να βρει έναν κόμβο αντικαταστάτη. Ο b δημιουργεί μια αίτηση εύρεσης αντικαταστάτη και τη στέλνει στον in-order γείτονα j. Ο j ελέγχει τους πίνακες δρομολόγησης του και παρατηρεί ότι υπάρχουν κάποιοι γειτονικοί κόμβοι με παιδιά, γι' αυτό ο j προωθεί την αίτηση στον r, ο οποίος είναι παιδί ενός γειτονικού του κόμβου. Ο r από τη στιγμή που δεν έχει παιδιά και επιπλέον δεν υπάρχει κάποιος γειτονικός κόμβος στους πίνακες δρομολόγησης που να έχει παιδιά, ο r μπορεί να γίνει αντικαταστάτης του b. Απ' ότι φαίνεται το BATON προσαρμόζεται στην αποχώρηση κόμβων και συνεχίζει να διατηρεί την ισορροπημένη δομή του.

Πριν ένας κόμβος y αντικαταστήσει έναν κόμβο χ, ο κόμβος χ πρέπει να ειδοποιήσει τους γειτονικούς του κόμβους του ίδιου επιπέδου, καθώς και τον γονιό του, για την αναχώρηση του, όπως και προηγουμένως. Η διαδικασία ενημέρωσης διαρκεί 4logN βήματα. Επιπλέον, όλοι οι κόμβοι που έχουν συνδέσεις προς τον χ θα πρέπει να ενημερωθούν για να αλλάξουν τη φυσική διεύθυνση της σύνδεσης, η οποία έδειχνε στον χ και τώρα θα πρέπει να δείχνει στον y. Η διαδικασία αυτή είναι εύκολο να γίνει, χρησιμοποιώντας πληροφορία η οποία λαμβάνεται από τον χ. Ειδικότερα, ο πραγματικός γονιός του χ (τώρα πλέον του y) πρέπει να στείλει $2L^2$ μηνύματα στους γειτονικούς του κόμβους του ίδιου επιπέδου, για να τους ειδοποιήσει για την αντικατάσταση του παιδιού του στο επίπεδο L_1.Ο y πρέπει να στείλει $2L_2$ μηνύματα στους νέους γειτονικούς του

κόμβους, όπου L_2 είναι το νέο του επίπεδο. Έτσι, προκύπτει ότι ο μέγιστος αριθμός μηνυμάτων που απαιτούνται για την ενημέρωση των πινάκων δρομολόγησης είναι 8logN.

2.1.5: Αστοχία κόμβου

Μερικές φορές συμβαίνει ένας κόμβος να αστοχεί(αποτυγχάνει) ή να φεύγει ξαφνικά από το δίκτυο. Έτσι υπάρχουν περιπτώσεις, όπου μερικοί κόμβοι επιθυμούν να αποκτήσουν πρόσβαση σε έναν κόμβο χ, ο οποίος όμως έχει «αποτύχει» κι έτσι ανακαλύπτουν ότι η διεύθυνση είναι απροσπέλαστη. Οι κόμβοι που ανακάλυψαν κάτι τέτοιο, θα πρέπει να αναφέρουν την αστοχία αυτή στον γονιό του χ, τον y, ο οποίος είναι τώρα υπεύθυνος για τη διαχείριση της αναχώρησης του χ. Ο y χρησιμοποιεί τις συνδέσεις που υπάρχουν αποθηκευμένες στους πίνακες δρομολόγησής του, επικοινωνεί με παιδιά κόμβων που βρίσκονται εκεί(στους πίνακες δρομολόγησης) και άμεσα «αναγεννά» τον δεξί και αριστερό πίνακα δρομολόγησης του χ. Τα παιδιά αυτά μπορούν επίσης να βοηθήσουν στο να εντοπιστούν τα παιδιά του χ, αν υπάρχουν βέβαια. Ο y τώρα μπορεί να εκτελέσει τη διαδικασία αποχώρησης του παιδιού του όπως αυτή περιγράφηκε παραπάνω. Από τη στιγμή που όλη η πληροφορία που είχε αποθηκευμένη ο χ, αναγεννήθηκε από τον y, ο αλγόριθμος λειτουργεί κανονικά με ελάχιστες τροποποιήσεις.

2.1.6: Ανοχή σφαλμάτων

Παραπάνω έγινε η περιγραφή του χειρισμού μιας αποτυχίας ενός κόμβου ή της απρόσμενης αποχώρησής του από το δίκτυο. Η διαδικασία της «ανάρρωσης» είναι ίδια με την διαδικασία αποχώρησης ενός κόμβου και απαιτεί μόνο O(logN) μηνύματα όχι όμως και αντίστοιχο χρόνο. Εδώ, περιγράφεται ο τρόπος που στο δίκτυο συνεχίζονται κανονικά οι υπόλοιπες λειτουργίες, παράλληλα με τη δρομολόγηση γύρω από τον «χαμένο» κόμβο.

Υπάρχουν δύο άξονες στους οποίους τα μηνύματα δρομολογούνται στο BATON: ο οριζόντιος(πλάγιος) άξονας, μέσω του αριστερού και του δεξιού πίνακα δρομολόγησης και ο κατακόρυφος άξονας μέσω του γονιού, των παιδιών και των in-order γειτόνων. Ο πρώτος άξονας είναι εκ φύσεως ανεχτός σε σφάλματα, αφού υπάρχει μια λογαριθμική

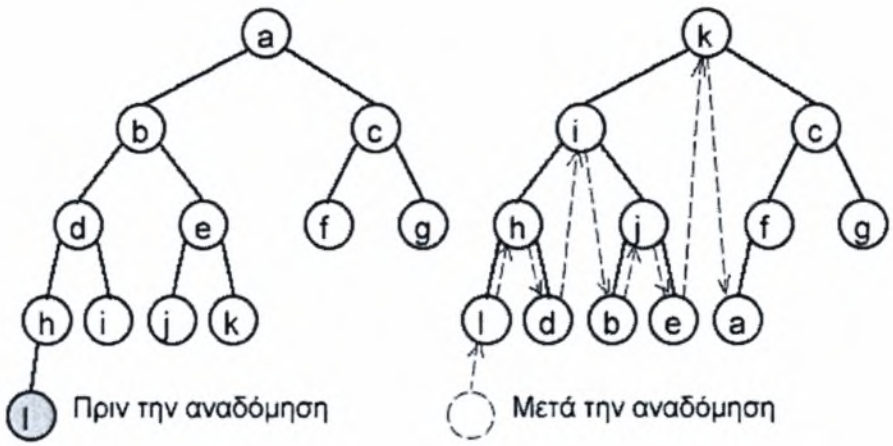
σαν του Chord επέκταση των συνδέσμων και γι' αυτό προκύπτει κι ένας μεγάλος αριθμός εναλλακτικών μονοπατιών ανάμεσα σε ζεύγη κόμβων. Ο δεύτερος άξονας είναι κι αυτός ανεχτός σε σφάλματα καθώς ένας κόμβος μπορεί να πάει σε έναν γειτονικό κόμβο του πατέρα του, να βρει ένα παιδί του κόμβου αυτού και έπειτα να συνδεθεί με το παιδί αυτό, επαναφέροντας έτσι μια «χαμένη» σύνδεση γονιού – παιδιού. Μέχρι εδώ έχουμε θεωρήσει αποτυχία ενός μόνο κόμβου. Αν δύο ή περισσότεροι κόμβοι αποτύχουν, μπορούμε να θεωρήσουμε δύο δυνατότητες. Αν οι αποτυχημένοι κόμβοι έχουν σχέση γονιού – παιδιού, τότε μπορούμε ακόμα να εφαρμόσουμε την τεχνική που περιγράψαμε παραπάνω – ταξιδεύοντας μέσω κόμβων γειτόνων. Αν οι αποτυχημένοι κόμβοι δεν έχουν σχέση γονιού – παιδιού, τότε οι αποτυχίες τους μπορούν να διορθωθούν ανεξάρτητα και δεν προκύπτει επιπλέον πολυπλοκότητα λόγω της προσωρινής παραλληλότητας των αποτυχιών τους. Αν πάρουμε την ειδική περίπτωση, όλοι οι κόμβοι ενός επιπέδου να αποτύχουν, το δέντρο δεν σπάει σε κομμάτια, καθώς οι συνδέσεις προς τους in-order γείτονες μπορούν να χρησιμοποιηθούν για τη δρομολόγηση κατά μήκος του κενού.

2.1.7: Αναδόμηση Δικτύου

Όπως περιγράψαμε παραπάνω, οι κόμβοι που εισέρχονται στο δίκτυο αναγκάζονται να μετακινούνται σε συγκεκριμένα μέρη του δέντρου ενώ οι κόμβοι που φεύγουν από το δίκτυο πρέπει να βρουν κόμβους αντικαταστάτες σε περίπτωση που η αποχώρηση τους προκαλεί κάποια ανισορροπία στο δέντρο. Υπάρχουν περιπτώσεις όπου, η εισαγωγή ενός κόμβου στο δίκτυο ή η αποχώρηση του από αυτό, μπορεί να προκαλέσει παραβίαση του θεωρήματος 1 ή να γίνεται εσκεμμένα, όπως θα δούμε παρακάτω για λόγους εξισορρόπησης φόρτου, με αποτέλεσμα να υπάρχουν πιθανότητες, το σύστημα να έρθει σε μη ισορροπημένη κατάσταση. Μια εναλλακτική λύση για να πετύχουμε ισορροπία είναι η αναδόμηση του συστήματος. Η αναδόμηση είναι παρόμοια με την περιστροφή σε ένα AVL δέντρο.

Όταν ένας κόμβος χ δέχεται ένα νεοεισερχόμενο κόμβο y ως παιδί του και ανακαλύπτει πως το θεώρημα 1 παραβιάζεται, ξεκινά τη διαδικασία της αναδόμησης. Χωρίς παραβίαση της γενικότητας, θα θεωρήσουμε ένα παράδειγμα όπου η διαδικασία της αναδόμησης γίνεται προς τα δεξιά. Έστω ότι ο κόμβος y εισάγεται στο δέντρο ως

αριστερό παιδί του χ. Για να επιφέρουμε ξανά ισορροπία στο σύστημα, ο χ ειδοποιεί τον y να πάρει τη θέση του χ, επιπλέον ο χ ειδοποιεί τον δεξί in-order γείτονά του z ότι ο χ θα αντικαταστήσει τον z (εάν ο y είχε εισαχθεί ως δεξί παιδί του χ, τότε ο χ παραμένει ανέπαφος και ο y αντικαθιστά απευθείας τον z). Οπότε ο z ελέγχει κι αυτός με τη σειρά του τον δεξιό in-order γείτονά του, τον t, για να δει αν το αριστερό του παιδί λείπει. Εάν λείπει, και η προσθήκη ενός παιδιού στον κόμβο t δεν επηρεάζει την ισορροπία του δέντρου, ο z παίρνει τη θέση του αριστερού παιδιού του t και η διαδικασία της αναδόμησης σταματά εδώ. Αν ο t έχει αριστερό παιδί ή δε μπορεί να δεχτεί τον z ως αριστερό παιδί του, χωρίς να παραβιάζει τη συνθήκη της ισορροπίας, ο z παίρνει τη θέση του t, ενώ ο t θα πρέπει να βρει μια νέα θέση για τον εαυτό του συνεχίζοντας με τον δεξί in-order γείτονά του.

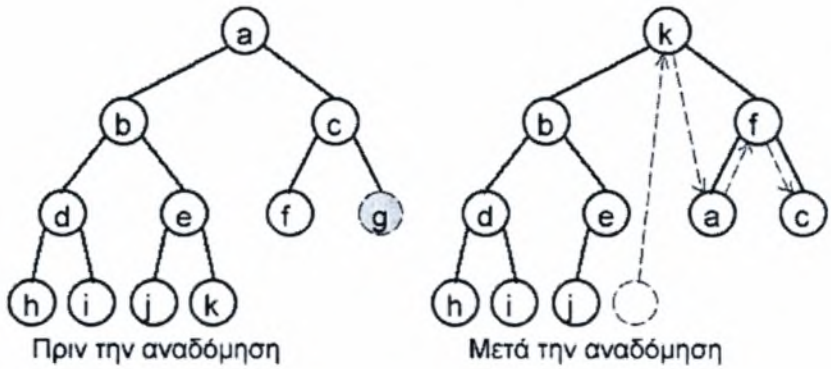


Πηγή: H.V.Jagadish [1]

Εικόνα 2.7: Διαδικασία αναδόμησης λόγω της παραβίασης του θεωρήματος 1 μετά την εισαγωγή του κόμβου *l*

Για καλύτερη κατανόηση ας δούμε το παράδειγμα της εικόνας 2.7. Υποθέτουμε ότι ο *l* εισέρχεται στο δίκτυο ως αριστερό παιδί του h και η εισαγωγή του παραβιάζει τη συνθήκη της ισορροπίας. Η διαδικασία της αναδόμησης ξεκινά με τον h, όπου ο l αντικαθιστά τον h , ο h αντικαθιστά τον d, ο d αντικαθιστά τον i, ο i αντικαθιστά τον b, ο b αντικαθιστά τον j, ο j αντικαθιστά τον e, ο e αντικαθιστά τον k, ο k αντικαθιστά τον a, και τελικά ο a γίνεται το αριστερό παιδί του f γιατί ο f μπορεί να δεχτεί ένα παιδί χωρίς να προκαλεί ανισορροπία στο δέντρο. Το δέντρο είναι τώρα ξανά ισορροπημένο.

Όταν ένας κόμβος φύλλο έστω χ επιθυμεί να φύγει από το δίκτυο και προκαλεί ανισορροπία στο δέντρο, ο γονιός του y ξεκινά την διαδικασία της αναδόμησης(κάθε κόμβος που δεν είναι φύλλο θα πρέπει κι εδώ να βρει κόμβο αντικαταστάτη). Χωρίς παραβίαση της γενικότητας, θεωρούμε αριστερή αναδόμηση. Έστω ότι, ο χ είναι το δεξί παιδί του y. Για να επαναφέρουμε ισορροπία στο δέντρο, ο y θα πρέπει να αντικαταστήσει τον χ και ο αριστερός in-order γείτονας του χ, ο z, θα πρέπει να πάρει τη θέση του y.(Αν ο χ είναι το αριστερό παιδί του y, τότε ο z μπορεί απευθείας να αντικαταστήσει τον χ και ο y να παραμείνει ανέπαφος.) Εάν η κίνηση του z, να αντικαταστήσει τον χ, δε διαταράσσει την ισορροπία του δέντρου, η διαδικασία της αναδόμησης σταματά. Εάν όμως η κίνηση του z παραβιάζει την ισορροπία, τότε ο z παίρνει τη θέση του αριστερού του in-order γείτονα t και έτσι αναδρομικά ο t βρίσκει έναν κόμβο να αντικαταστήσει. Για παράδειγμα στην εικόνα 2.8, ο g φεύγει από το δίκτυο και αφήνει το σύστημα σε κατάσταση ανισορροπίας. Η διαδικασία αναδόμησης ξεκινά στον κόμβο c, όπου ο c αντικαθιστά τον g, ο f αντικαθιστά τον c, ο a αντικαθιστά τον f και τελικά ο k αντικαθιστά τον a. Η διαδικασία σταματά στον a γιατί η κίνηση του a δεν επέφερε ανισορροπία.



Πηγή:H.V.Jagadish [1]

Εικόνα 2.8: Διαδικασία αναδόμησης λόγω της παραβίασης του θεωρήματος 1 μετά την αποχώρηση του κόμβου g

Καμία μεταφορά δεδομένων δε χρειάζεται κατά τη διάρκεια της αναδόμησης του δικτύου. Ωστόσο, πολλοί κόμβοι αλλάζουν θέσεις στο δέντρο οπότε επηρεάζεται ο αριθμός τους, ο αριθμός επιπέδου τους αλλά και οι πίνακες δρομολόγησής τους. Για κάθε

κόμβο που μετακινείται και χρειάζεται να αναπροσαρμόσει τον πίνακα δρομολόγησής του απαιτείται προσπάθεια O(logN) φόρτου. Γι' αυτό, όσο περισσότεροι κόμβοι συμμετέχουν στη διαδικασία της αναδόμησης, τόσο μεγαλύτερο κόστος απαιτείται για την ανανέωση των πινάκων δρομολόγησης.

2.1.8: Κατασκευή ευρετηρίου

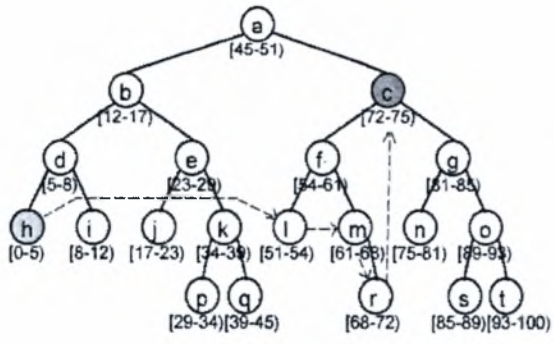
Μέχρι τώρα το δίκτυο περιγράφεται ως ένα δυαδικό ισορροπημένο δέντρο. Στο τμήμα αυτό όμως, δείχνουμε πως μπορεί κανείς να χρησιμοποιήσει ένα επικαλύπτων δίκτυο, για να κατασκευάσει μια κατανεμημένη δομή ευρετηρίου, πολύ κοντά στην έννοια του AVL δέντρου.

Αναθέτουμε σε κάθε κόμβο, εσωτερικό ή όχι, ένα εύρος τιμών. Κάθε κόμβος καταγράφει για όλους τους κόμβους, προς τους οποίους διατηρεί συνδέσεις, το εύρος τιμών για το οποίο είναι υπεύθυνοι. Οποτεδήποτε το εύρος τιμών σε κάποιον από τους κόμβους αυτούς αλλάζει, η αλλαγή θα πρέπει να καταγράφεται. Το εύρος τιμών το οποίο διαχειρίζεται απευθείας ένας κόμβος, θα πρέπει να περιέχει μεγαλύτερες τιμές από αυτές του εύρους του κόμβου που διαχειρίζεται το αριστερό υποδέντρο του, και μικρότερες από αυτές του εύρος του κόμβου που διαχειρίζεται το δεξί του υποδέντρο. Αντίθετα με ότι συμβαίνει στα B^+ δέντρα, οι εσωτερικοί κόμβοι στο δέντρο διαχειρίζονται και οι ίδιοι ένα εύρος τιμών δεδομένων απευθείας. Έτσι, είναι εύκολο να δει κανείς, πως η δομή BATON συμπεριφέρεται όπως και μια δενδρική δομή λεξικού. Το ευρετήριο είναι δομικά παρόμοιο με το ευρετήριο κύριας μνήμης το λεγόμενο T-δέντρο, το οποίο σχεδιάστηκε για να μειώσει τον αριθμό των δεικτών και το «εσωτερικό» κυνήγι δεικτών.

2.1.9: Ερωτήσεις σημείου ή ακρίβειας

Για μια ερώτηση σημείου που τίθεται σε έναν κόμβο χ, ο κόμβος θα ελέγξει πρώτα το δικό του εύρος (τη δική του εμβέλεια). Εάν η ερώτηση βρίσκεται μέσα στο τρέχων εύρος, γίνεται αναζήτηση στο τοπικό ευρετήριο για την τιμή και η αναζήτηση σταματά εδώ. Αλλιώς, ο χ προωθεί την ερώτηση στον κόμβο «προορισμό» όπως περιγράφεται παρακάτω στον αλγόριθμο ακριβούς αναζήτησης.

```
Algorithm 2.1.3: SearchExact(node n, query r, value v){
if ( (LowerBound(n) <= v AND (v <= UpperBound(n)) )
   { q is executed at n }
else
  {
    if ( UpperBound(n) < v )
      {
         m=TheFarthestNodeSatisfyingCondition(LowerBound(m) <= v)
        if( there exists such an m )
          { Forward q to m }
        else
        {
          if ( RightChild(n) != null )
            { Forward q to RightChild(n) }
          else
            { Forward q to RightAdjacentNode(n) }
        }
      }
    else
        { // A similar Process is followed towards the left }
  }
}
```



Πηγή: H.V.Jagadish[1]

Εικόνα 2.9 : Αναζήτηση σημείου

Τώρα ας δούμε ένα παράδειγμα ακριβούς αναζήτησης όπως αυτό φαίνεται στην εικόνα 2.9. Έστω ότι ο κόμβος h θέλει να ψάξει για δεδομένα που βρίσκονται αποθηκευμένα στον κόμβο c. Από τη στιγμή που η προς αναζήτηση τιμή είναι μεγαλύτερη από το άνω όριο του εύρους του κόμβου h, ο h ελέγχει τον δεξιό πίνακα δρομολόγησης και προωθεί την ερώτηση στον *l*, ο οποίος είναι ο δεξιότερος κόμβος που έχει το κατώτερο όριο εύρους, το οποίο είναι ταυτόχρονα και μικρότερο από την τιμή που αναζητείται. Ο *l* στην συνέχεια ελέγχει τον

δεξιό πίνακα δρομολόγησης και προωθεί την ερώτηση στον m. Ο κόμβος m, αφού δε μπορεί να βρει κανέναν γειτονικό κόμβο να προωθήσει την ερώτηση, την προωθεί στο δεξί παιδί του r. Τελικά, ο r προωθεί την ερώτηση στον c, ο οποίος είναι και ο κόμβος «προορισμού».

Όταν ένας κόμβος χ εκτελεί αναζήτηση για μια συγκεκριμένη τιμή, εάν ο χ είναι η ρίζα, τότε η αίτηση της αναζήτησης προωθείται πάντα προς τα κάτω στον κόμβο «προορισμού», του οποίου το εύρος τιμών περιέχει την τιμή προς αναζήτηση . Έτσι, ο μέγιστος αριθμός βημάτων της επεξεργασίας είναι το ύψος του δέντρου δηλ. logN. Αν ο χ δεν είναι ρίζα, χωρίς παραβίαση της γενικότητας, υποθέτουμε ότι ο κόμβος που κάνει την αίτηση βρίσκεται στο αριστερό μέρος του δέντρου. Έτσι, θεωρούμε δύο περιπτώσεις του κόμβου «προορισμού». Στην πρώτη περίπτωση, αν ο κόμβος προορισμού είναι η ρίζα, ακολουθώντας τον αλγόριθμο, η αίτηση αναζήτησης πάντα προωθείται στον δεξιότερο κόμβο, έστω r, του αριστερού υποδέντρου και από εκεί προωθείται στη ρίζα, η οποία αποτελεί τον δεξί in-order γείτονα του r. Το κόστος προώθησης της αίτησης στον r είναι logN-1, το οποίο είναι το ύψος του αριστερού υποδέντρου, καθώς για κάθε προώθηση, είτε η προώθηση γίνεται προς γειτονικό κόμβο του πίνακα δρομολόγησης, είτε προς το δεξί παιδί, είτε προς τον δεξιό in-order γείτονα, ο χώρος αναζήτησης πάντα μειώνεται στο μισό. Γι' αυτό, ο μέγιστος αριθμός βημάτων είναι επίσης logN. Στη δεύτερη περίπτωση, αν ο κόμβος «προορισμού» βρίσκεται στο δεξί κομμάτι του δέντρου, κατά τη διάρκεια της αναζήτησης, χρειάζεται ένα βήμα για να προωθηθεί η αίτηση από έναν κόμβο στο αριστερό υποδέντρο, σε έναν κόμβο στο δεξί υποδέντρο μέσω του πίνακα δρομολόγησής του. Ανάλογα με την τιμή αναζήτησης, το βήμα αυτό μπορεί να γίνει νωρίτερα ή αργότερα στη διαδικασία της αναζήτησης. Παρόλα αυτά, αν το βήμα αυτό συμβεί αργότερα, τα προηγούμενα βήματα αναζήτησης μπορούν ακόμη να βοηθήσουν, ώστε να μειωθεί ο χώρος αναζήτησης του δεξιού υποδέντρου στο μισό. Γι' αυτό, τα τελικά βήματα είναι επίσης 1 + (logN - 1) = logN βήματα, όπου το logN – 1 είναι το κόστος αναζήτησης στο δεξί υποδέντρο. Ο αλγόριθμος αυτός δείχνει ότι η αίτηση αναζήτησης πάντα προωθείται μέσω κόμβων γειτόνων ή κόμβους παιδιά. Η αίτηση χρειάζεται μόνο να προωθηθεί σε υψηλότερο επίπεδο σε δύο περιπτώσεις: ο κόμβος υψηλότερου επιπέδου περιέχει την τιμή αναζήτησης ή ο κόμβος που

επεξεργάζεται την αίτηση δεν έχει δύο παιδιά (είναι κόμβος φύλλο ή κόμβος δίπλα σε φύλλο). Η ιδιότητα αυτή φαίνεται καθαρά ότι βοηθά ώστε να μη συγκεντρώνονται στη ρίζα περισσότερες αιτήσεις από ότι σε άλλους κόμβους.

2.1.10: Ερωτήσεις με εύρος τιμών

Για μια ερώτηση εύρους ακολουθείται ή ίδια διαδικασία όπως και σε μια ερώτηση ακρίβειας. Από τη στιγμή που εντοπίζεται τομή του εύρους της ερώτησης με το εύρος κάποιου κόμβου, προκύπτουν απευθείας κάποιες τμηματικές(μερικώς απαντημένες) απαντήσεις για την ερώτηση εύρους. Έπειτα η διαδικασία συνεχίζεται προς τα αριστερά ή/και προς τα δεξιά για να καλυφθεί και το υπόλοιπο του αναζητούμενου εύρους.

Όπως και στην περίπτωση μια ερώτησης ακριβείας, απαιτούνται O(logN) βήματα για να βρεθεί η πρώτη τομή. Από εκεί και μετά, προστίθεται κόστος της τάξης του O(1) για κάθε επιπλέον κόμβο που επισκέπτεται. Έτσι, για να απαντηθεί μια ερώτηση εύρους όπου το εύρος καλύπτεται από X κόμβους, απαιτούνται O(logN +X) βήματα.

2.1.11: Εισαγωγή Δεδομένων

Όταν πρόκειται να εισαχθούν δεδομένα, πρώτα ακολουθείται η διαδικασία αναζήτησης που ακολουθείται για ερωτήσεις ακριβείας, έτσι ώστε να βρεθεί ο κόμβος στον οποίο θα εισαχθούν τα δεδομένα και μετά πραγματοποιείται η εισαγωγή. Παρόλα αυτά, για τον αριστερότερο και δεξιότερο κόμβο, ίσως χρειαστεί το εύρος τους να προσαρμοστεί, αν η τιμή των εισαγόμενων δεδομένων είναι εκτός του τρέχοντος εύρους. Αν ο αριστερότερος κόμβος λάβει αίτηση εισαγωγής και η τιμή που πρόκειται να εισαχθεί είναι μικρότερη από το τρέχον εύρος τιμών, τότε επεκτείνεται το εύρος τιμών προς τα αριστερά, έτσι ώστε ο κόμβος να μπορέσει να καλύψει τη νεοεισερχόμενη τιμή(ένας κόμβος ξέρει ότι είναι ο αριστερότερος εάν ο αριθμός του είναι 1 και δεν έχει αριστερό παιδί). Παρομοίως, αν ο δεξιότερος κόμβος λάβει μια αίτηση εισαγωγής και η τιμή εισαγωγής είναι μεγαλύτερη από ότι η μεγαλύτερη τιμή του εύρους που καλύπτει ο κόμβος, τότε επεκτείνει το εύρος του προς τα δεξιά και δέχεται τη νεοεισερχόμενη τιμή. Σε αυτές τις ειδικές περιπτώσεις, απαιτούνται επιπλέον logN βήματα για την ενημέρωση των πινάκων

δρομολόγησης. Το κόστος εντοπισμού του κόμβου για την εισαγωγή των νέων δεδομένων είναι O(logN) όπως και στην διαδικασία αναζήτησης ερωτήσεων ακριβείας.

2.1.12: Διαγραφή Δεδομένων

Για να διαγράψουμε κάποια υπάρχοντα δεδομένα, εντοπίζουμε τον κόμβο που διαχειρίζεται την τιμή των δεδομένων που επιθυμούμε να διαγράψουμε και έπειτα διαγράφουμε τα δεδομένα. Το κόστος διαγραφής δεδομένων είναι ακριβώς το ίδιο με αυτό της αναζήτησης, δηλαδή O(logN).

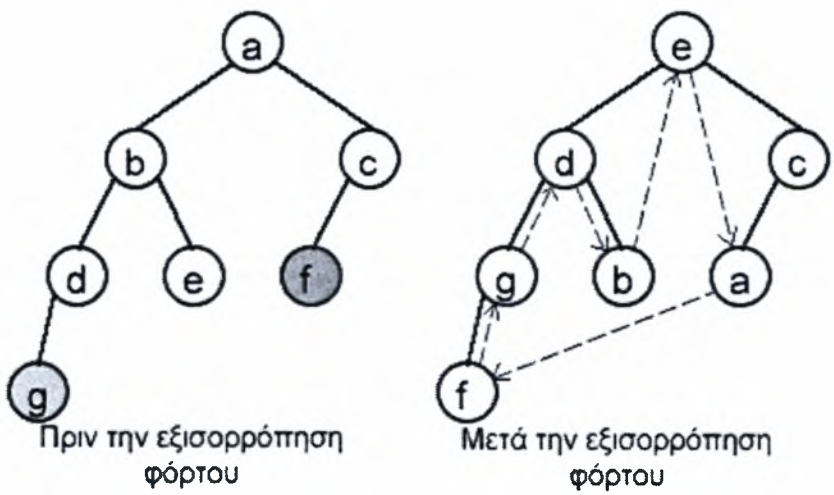
2.1.13: Εξισορρόπηση Φόρτου

Θα ήταν καλύτερα για το δίκτυο, αν ο υπολογιστικός φόρτος ήταν εξ' ίσου κατανεμημένος σε όλους τους κόμβους του δικτύου, και όταν η αναλογία φόρτου ανά κόμβο υπερβαίνει κάποιο όριο, να καλείται κάποια διαδικασία, η οποία να επιφέρει εκ νέου ισορροπία στον υπολογιστικό φόρτο. Ο φόρτος αυτός μπορεί να εκτιμηθεί είτε μετρώντας αριθμό μηνυμάτων είτε αριθμό ερωτήσεων. Τυπικά, όσο μεγαλύτερο εύρος καλύπτει ένας κόμβος τόσο μεγαλύτερη ποσότητα δεδομένων διαχειρίζεται και άρα περισσότερο φόρτο. Η διαδικασία εξισορρόπησης φόρτου επιτρέπει σε έναν κόμβο να μοιράσει μέρος του εύρους του σε άλλους κόμβους ή να ζητήσει επιπλέον εύρος από άλλους. Στόχος είναι η προσαρμογή του εύρους των δεδομένων για την εξισορρόπηση του φόρτου. Βέβαια, αυτό δε σημαίνει ότι όλοι οι κόμβοι έχουν ισομερή διαστήματα τιμών.

Η εξισορρόπηση φόρτου που βασίζεται σε απλή μετακίνηση δεδομένων μεταξύ δύο γειτονικών in-order κόμβων ίσως να μην επαρκεί για την αποτελεσματική διαχείριση δεδομένων που δεν είναι ομοιόμορφα κατανεμημένα. Επιπρόσθετα, το πρόβλημα που δημιουργείται είναι, πως μπορεί να προκληθεί μια αλυσιδωτή μετακίνηση δεδομένων κατά μήκος του δικτύου, γεγονός που θα επιφέρει υψηλό συνολικό κόστος. Έτσι, αντί για εξισορρόπηση φόρτου, όπου εμπλέκονται μόνο in-order γείτονες, προτείνουμε, ένας κόμβος να προβεί σε εξισορρόπηση φόρτου με τους in-order γείτονές του μόνο αν δεν είναι κόμβος φύλλο. Εάν είναι κόμβος φύλλο, μπορεί είτε να μοιράσει τον φόρτο του με τους in-order γείτονές του είτε να βρει έναν άλλο κόμβο φύλλο με λιγότερο φόρτο και να

του δώσει κάποιο από το φορτίο του. Πιο συγκεκριμένα, όταν ένας κόμβος φύλλο υπερφορτώνεται, πρώτα προσπαθεί να εξισορροπήσει το φορτίο του με τους in-order γείτονές του. Αν οι in-order γείτονές του έχουν κι αυτοί αρκετό φορτίο, τότε βρίσκει έναν ελαφρύτερα φορτωμένο κόμβο φύλλο να μοιραστεί το φορτίο του. Χωρίς παραβίαση της γενικότητας, έστω ότι αυτός ο λιγότερο φορτωμένος κόμβος βρίσκεται στα δεξιά του υπερφορτωμένου κόμβου. Ο «ελαφρύτερος» κόμβος δίνει το φορτίο του στον δεξιό του in-order γείτονα. Έπειτα, αφήνει την τρέχουσα θέση του στο δίκτυο και επανέρχεται ως παιδί του υπερφορτωμένου κόμβου προκαλώντας αναγκαστική αναδόμηση του δικτύου(προς τα αριστερά για την θέση που αφήνει και προς τα δεξιά για τη θέση που καταλαμβάνει) αν χρειάζεται.

Για παράδειγμα, ας θεωρήσουμε τον κόμβο g της εικόνας 2.10. Ο g είναι υπερφορτωμένος και ανακαλύπτει πως ο f είναι «ελαφρύτερος» από αυτόν. Έτσι ο f δίνει το φορτίο του (το εύρος του) στον κόμβο c και επανέρχεται στο δίκτυο ως παιδί του g. Η μετακίνηση του κόμβου προκαλεί ανισορροπία στο δίκτυο και γι' αυτό καλείται η διαδικασία αναδόμησης. Οπότε ο f αντικαθιστά τον g, ο g με τη σειρά του αντικαθιστά τον d, ο d αντικαθιστά τον b, ο b αντικαθιστά τον e, ο e αντικαθιστά τον a και ο a τελικά καταλαμβάνει την αρχική θέση του κόμβου f.



Πηγή:H.V.Jagadish [1]

Εικόνα 2.10: Ο υπερφορτωμένος κόμβος g καλεί τη διαδικασία εξισορρόπησης φόρτου για να μοιραστεί το φορτίο του με τον «ελαφρύτερο» κόμβο f

Αξίζει να παρατηρηθεί ότι η αναγκαστική αναδόμηση του δικτύου, στη χειρότερη περίπτωση, περιλαμβάνει μια πλήρη ολίσθηση από την υπερφορτωμένη θέση στην λιγότερο φορτωμένη θέση που ανιχνεύτηκε. Γενικότερα, απαιτούνται μικρότερες ολισθήσεις, οι οποίες επηρεάζουν μόνο μερικούς κόμβους, μέχρι να βρεθούν κατάλληλα σημεία στο δίκτυο, για τη στέγαση της αναχώρησης και άφιξης του κόμβου. Στην πραγματικότητα, η πιθανότητα ολίσθησης κ κόμβων μειώνεται εκθετικά με την τιμή του κ. Αναλύοντας το λίγο περισσότερο, μπορεί κανείς να δείξει ότι το επιπλέον κόστος της εξισορρόπησης φόρτου ανά εισαγωγή ή διαγραφή είναι ακριβώς O(logN).

2.2: Η δομή BATON*

Τα P2P συστήματα έχουν γίνει πλέον το πιο πρόσφατο μέσο ανταλλαγής δεδομένων. Η αποτελεσματική αναζήτηση είναι μια κρίσιμη απαίτηση σε τέτοιου είδους συστήματα και γι' αυτό έχουν προταθεί κατανεμημένες δομές αναζήτησης. Οι περισσότερες από αυτές τις δομές εγγυώνται λογαριθμικό χρόνο αναζήτησης, όπου η βάση του λογαρίθμου είναι 2. Αυτό σημαίνει πως σε ένα σύστημα με N κόμβους, το κόστος αναζήτησης είναι $O(\log_2 N)$. Στα συστήματα βάσεων δεδομένων, έχει αναγνωριστεί η σημασία των δομών με μεγάλο αριθμό παιδιών(δηλ. με μεγάλο fan-out). Το ίδιο συμβαίνει και στα P2P συστήματα όπου το κόστος της αναζήτησης θα μπορούσε να μειωθεί σε αξιοσημείωτο βαθμό αν ο λογάριθμος είχε μεγαλύτερη βάση. Σε αυτήν την ενότητα παρουσιάζεται ένα multi-way δέντρο αναζήτησης, το οποίο μειώνει το κόστος της αναζήτησης σε $O(\log_m N)$ όπου m είναι το fan-out. Μειώνοντας το κόστος της αναζήτησης όμως, προκύπτει μια αύξηση στο κόστος της ενημέρωσης και άρα παρατηρούμε ένα trade-off. Θα αποδειχθεί, παρόλα αυτά, πως η αύξηση στο κόστος ενημέρωσης μπορεί να διατηρηθεί στη χειρότερη περίπτωση γραμμική ως προς m. Το trade-off μεταξύ κόστους αναζήτησης και ενημέρωσης, αναλύεται ως συνάρτηση του m και προτείνεται πώς να βρεθεί ένα καλό σημείο trade-off.

Η multi-way δομή δέντρου που προτείνεται, και στο εξής θα ονομάζεται BATON* [2], προέρχεται από τη δομή BATON της προηγούμενης ενότητας. Επιπλέον, η δομή BATON* μπορεί να υποστηρίξει αποτελεσματικά εκτός από ερωτήσεις εύρους και ερωτήσεις με πολλαπλά γνωρίσματα (multi-attribute queries), σε αντίθεση με τη δομή BATON, η οποία δε μπορεί να υποστηρίξει τέτοιου είδους ερωτήσεις αποτελεσματικά.

2.2.1: Προβλήματα που πρέπει να αντιμετωπιστούν κατά την επέκταση της δομής BATON σε BATON*

Η δομή BATON δημιουργεί μια δυαδική δενδρική δομή αναζήτησης. Καθώς προσπαθούμε να αυξήσουμε το fan-out, έρχονται στην επιφάνεια ζητήματα τα οποία πρέπει να αντιμετωπιστούν.

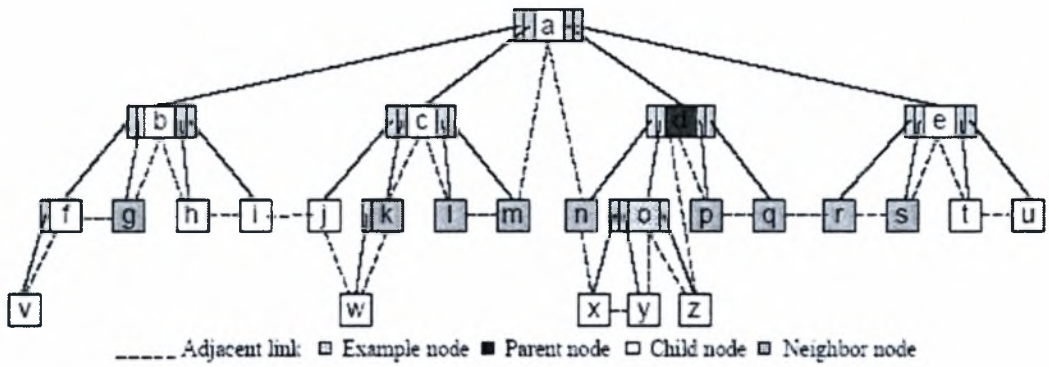
- Στη δομή BATON, όλοι οι κόμβοι είτε φύλλα είτε εσωτερικοί είναι υπεύθυνοι για κάποιο εύρος τιμών. Αφού εφαρμοστεί in-order διαπέραση στο δέντρο τα εύρη

κατανέμονται στους κόμβους, σε αύξουσα σειρά, από αριστερά προς τα δεξιά(θεωρώντας ως γραμμή την in-order διαπέραση). Αν ένας κόμβος έχει παραπάνω από δύο παιδιά είναι αδύνατο να εφαρμοστεί in-order διαπέραση. Έτσι δε μπορεί να προκύψει ένας ξεκάθαρος ορισμός της γειτονίας(adjacency), για τα εύρη τιμών στους εσωτερικούς κόμβους και δεν είναι ξεκάθαρο, ποιες γειτονικές συνδέσεις θα πρέπει να διατηρηθούν. Ακόμα κι αν επιλέξουμε να είμαστε γενναιόδωροι και αποθηκεύουμε πολλαπλές γειτονίες, μια θα πρέπει να επιλεχθεί για διαπέραση αναζήτησης ή για εξισορρόπηση φόρτου. Η επιλογή αυτή δεν είναι εύκολο να γίνει όταν υπάρχουν πολλοί κόμβοι «γείτονες» (adjacent nodes). Για παράδειγμα, αν ένας εσωτερικός κόμβος είναι υπερφορτωμένος απλά από μόνο ένα εύρος τιμών, μπορεί να επιχειρήσει εξισορρόπηση φόρτου μόνο με τους δύο κόμβους «γείτονες» (και όχι με όλους τους 2(m-1) κόμβους «γείτονες»). Όλα τα παραπάνω φέρνουν ξανά στο προσκήνιο, το πρόβλημα της προώθησης των πράξεων της διαδικασίας εξισορρόπησης φόρτου όταν ένας κόμβος υπερφορτώνεται, πράγμα το οποίο αποφεύγεται στη δομή BATON.

- Αρχικά, φαίνεται σωστή η επέκταση των γειτονικών πινάκων δρομολόγησης για να διατηρούνται γειτονικοί κόμβοι, που βρίσκονται σε αποστάσεις που είναι δυνάμεις του m, από ότι αυτές που είναι δυνάμεις του 2. Αλλά αυτό τελικά αποδεικνύεται να είναι η λάθος απάντηση, καθώς αφήνει ο δίκτυο αναζήτησης υποσυνδεδεμένο(δηλαδή υπάρχουν λιγότεροι γειτονικοί κόμβοι ίδιου επιπέδου αφού διαλέγουμε μεγαλύτερες αποστάσεις)

2.2.2: Ορισμός της δομής BATON*

Στην υποενότητα αυτή ορίζεται η νέα δομή BATON*, η οποία διευθετεί τα ζητήματα που τέθηκαν παραπάνω κατά την επέκταση της δομή BATON σε μια multi-way δομή. Η νέα δομή φαίνεται στην εικόνα 2.11.



Πηγή: H.V.Jagadish [2]

Εικόνα 2.11: Η δομή BATON*

Υπάρχουν τρεις σημαντικές διαφορές μεταξύ του BATON* και του δυαδικού BATON.

- Κάθε κόμβος στο BATON* μπορεί να έχει μέχρι m παιδιά αντί για δύο όπως είδαμε στην πρωταρχική δομή. Επίσης, κάθε κόμβος διατηρεί συνδέσεις προς τα παιδιά του και επιπλέον αποθηκεύει και τα εύρη τιμών που διαχειρίζονται τα παιδιά του. Κάθε κόμβος διατηρεί και συνδέσεις προς δύο κόμβους «γείτονες», (adjacent nodes) οι οποίοι ορίζονται από τη γραμμική διαπέραση που σημειώνεται με κόκκινη διακεκομμένη γραμμή στην εικόνα 2.11.
- Στους πίνακες δρομολόγησης που διατηρεί ένας κόμβος, αποθηκεύονται συνδέσεις προς γειτονικούς κόμβους του ίδιου επιπέδου, οι οποίοι απέχουν απόσταση ίση με $d*m^i$, όπου d = 1...m-1 και $i \geq 0$, από τον εν λόγω κόμβο. Όπως φαίνεται και στην εικόνα 2.10, ο αριστερός πίνακας δρομολόγησης του κόμβου ο διατηρεί συνδέσεις στους κόμβους n, m, l, k, g, οι οποίοι απέχουν απόσταση από τον ο ίση με $1 \cdot 4^0, 2 \cdot 4^0, 3 \cdot 4^0, 1 \cdot 4^1$ και $2 \cdot 4^1$. Παρομοίως, ο δεξιός πίνακας δρομολόγησης του ο διατηρεί συνδέσεις προς τους κόμβους p, q, r, s. Έτσι προκύπτει ότι, ο μέγιστος αριθμός συνδέσεων, που μπορούν να έχουν οι πίνακες δρομολόγησης, ενός κόμβου στο επίπεδο L είναι $(m-1) \cdot \log_m \cdot$(αριθμός των κόμβων στο επίπεδο L) $= (m-1) \cdot \log_m m^L = (m-1) \cdot L$. Να σημειωθεί ότι, αν αυξηθεί το fan-out ενός κόμβου, για να μειωθεί το κόστος αναζήτησης, θα πρέπει να αυξηθεί και το μέγεθος των πινάκων δρομολόγησης και άρα αυξάνεται το κόστος ενημέρωσης των πινάκων. Συνεπώς, ανάλογα με τις απαιτήσεις της κάθε

εφαρμογής, θα μπορούσε κάθε φορά να επιλέγεται μια κατάλληλη τιμή του παράγοντα fan-out m. Θα δείξουμε παρακάτω τον τρόπο επιλογής του παράγοντα m.

- Για ένα σύστημα της τάξεως m, το εύρος των τιμών που θα διαχειρίζεται ένας κόμβος θα είναι μεγαλύτερο από τα εύρη τιμών που θα διαχειρίζονται τα πρώτα $\lceil m/2 \rceil$ παιδιά ενώ μικρότερο από το εύρος τιμών που θα διαχειρίζονται τα τελευταία $\lfloor m/2 \rfloor$ παιδιά. Για παράδειγμα, στην εικόνα 2.11, το εύρος τιμών που διαχειρίζεται ο κόμβος ο είναι μεγαλύτερο από αυτά που διαχειρίζονται οι κόμβοι y, x αλλά μικρότερο από αυτά που διαχειρίζεται ο κόμβος z.

Να σημειωθεί ότι, στο BATON*, ένας κόμβος διαχειρίζεται μόνο ένα εύρος τιμών και m συνδέσεις, αντί για (m-1) εύρη τιμών και m συνδέσεις, όπως στην αρχική multi-way δομή. Επίσης δε μοιάζει με B-δέντρο, όπου κάθε εσωτερικός κόμβος πρέπει να έχει αριθμό παιδιών μεταξύ m/2 και m. Αντίθετα, στο BATON*, όλοι οι εσωτερικοί κόμβοι, εκτός από τους γονείς των κόμβων φύλλων, πρέπει αναγκαστικά να έχουν ακριβώς m παιδιά. Οι γονείς των κόμβων φύλλων μπορούν να έχουν λιγότερα παιδιά.

2.2.3: Χρήσιμοι ορισμοί και θεωρήματα

Για ένα δέντρο με fan-out m όπου m>2, ο ορισμός ενός δυαδικού ισορροπημένου δέντρου επεκτείνεται στη μορφή:

Ορισμός 1: *Ένα δέντρο είναι ισορροπημένο αν και μόνο αν σε κάθε κόμβο στο δέντρο, το ύψος δύο οποιωνδήποτε υποδέντρων των παιδιών του διαφέρουν το πολύ κατά ένα.*

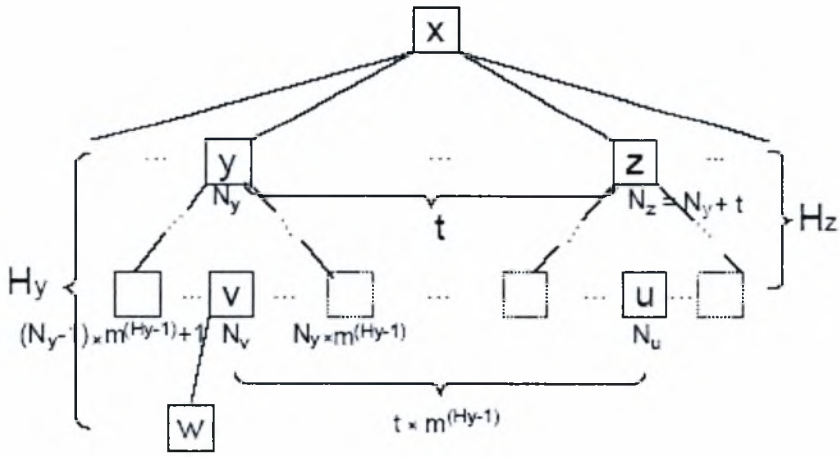
Συνεχίζουμε με τα δύο θεωρήματα που αποδείξαμε και στη δομή BATON.

Θεώρημα 1: *Ένα δέντρο είναι ισορροπημένο αν κάθε κόμβος του ο οποίος έχει παιδί έχει επίσης και τους δυο, και τον δεξί και τον αριστερό, πίνακες δρομολόγησης πλήρεις.*

Απόδειξη: Υποθέτουμε ότι ένα υποδέντρο με ρίζα τον κόμβο, έστω χ, είναι εκτός ισορροπίας. Τότε, πρέπει να υπάρχουν δύο παιδιά του κόμβου χ, έστω ο y και ο z, των οποίων τα υποδέντρα διαφέρουν παραπάνω από ένα στο ύψος. Έστω ότι, N_y και N_z οι αριθμοί των κόμβων y και z, H_y και H_z τα ύψη των αντίστοιχων υποδέντρων. Χωρίς παραβίαση της γενικότητας, υποθέτουμε ότι ο z είναι ο δεξιός αδερφός του y και ότι $H_y - 1 > H_z$. Επίσης έστω w το πιο μακρινό φύλλο (βρίσκεται στο μεγαλύτερο βάθος) του υποδέντρου με ρίζα τον y, v ο πατέρας του και N_v ο αριθμός του v. Από τη στιγμή που ο z είναι ο δεξιός αδερφός του y, έχουμε $N_z = N_y + t$, όπου $1 \le t < m$ (1). Από τον τρόπο που αριθμούνται οι κόμβοι έχουμε $(N_y - 1) \cdot m^{H_y - 1} + 1 \le N_v \le N_y \cdot m^{H_y - 1}$ (2). Από τη στιγμή που ο v έχει ένα παιδί τον w, θα πρέπει να έχει και πλήρεις τους πίνακες δρομολόγησης του. Στον δεξιό πίνακα δρομολόγησης του v θα πρέπει να υπάρχει ένας κόμβος u με αριθμό $N_u = N_v + d \cdot m^i$, όπου $1 \le d \le m$ και $i \ge 0$. Επιλέγοντας d = t, i = H_y -1 έχουμε $N_u = N_v + t \cdot m^{H_y - 1} \Rightarrow N_v = N_u - t \cdot m^{H_y - 1}$ (3). Από τις σχέσεις (2) και (3), έχουμε $(N_y - 1) \cdot m^{H_y - 1} + 1 \le N_u - t \cdot m^{H_y - 1} \le N_y \cdot m^{H_y - 1} \Rightarrow$

$\Rightarrow (N_y + t - 1) \cdot m^{H_y - 1} + 1 \le N_u \le (N_y + t) \cdot m^{H_y - 1}$ (4).

Από την (1) και την (4), έχουμε $(N_z - 1) \cdot m^{H_y - 1} + 1 \le N_u \le N_z \cdot m^{H_y - 1}$.



Πηγή:H.V.Jagadish [2]

Εικόνα 2.12: Απεικόνιση της απόδειξης του θεωρήματος 1

Ως αποτέλεσμα, ο N_u πρέπει να είναι αριθμός ενός προγονικού κόμβου που βρίσκεται σε ύψος H_y-1 του υποδέντρου με ρίζα τον z. Αυτό σημαίνει ότι, το ύψος του υποδέντρου με ρίζα τον z, είναι τουλάχιστον H_y-1. Καθώς το αποτέλεσμα αυτό έρχεται σε αντιπαράθεση με την υπόθεση, δεν υπάρχει κάποιο υποδέντρο το οποίο βρίσκεται εκτός ισορροπίας. Η απόδειξη για καλύτερη κατανόηση απεικονίζεται και στην εικόνα 2.12.

Θεώρημα 2: *Αν ένα κόμβος χ, περιέχει μια σύνδεση προς έναν άλλο κόμβο y στον αριστερό ή στον δεξιό πίνακα δρομολόγησης, ο γονιός του χ πρέπει επίσης να περιέχει μια σύνδεση προς τον γονιό του y εκτός κι αν ο χ και ο y έχουν τον ίδιο γονιό είναι δηλαδή αδέρφια.*

Απόδειξη: Έστω y ένας γειτονικός κόμβος στον πίνακα δρομολόγησης του χ, w ο γονιός του χ, v γονιός του y και N_x, N_y, N_w, N_v οι αριθμοί των χ, y, w, v αντίστοιχα. Από τον τρόπο που αριθμούνται οι κόμβοι προκύπτει ότι $N_w = N_x$ div m και $N_v = N_y$ div m. Από τον τρόπο που δημιουργούνται οι γειτονικές συνδέσεις, έτσι ώστε οι όλοι κόμβοι να διαπερνώνται κατά μήκος μιας γραμμής, έχουμε $N_y = N_x \pm d \cdot m^l$. Θα ερευνήσουμε δύο περιπτώσεις.

Περίπτωση 1: Αν i = 0, τότε $N_y = N_x \pm d$. Ως αποτέλεσμα, $N_v = (N_x \pm d)$ div m. Από τη στιγμή που είναι $1 \le d \le m-1$, τότε ισχύει $N_w - 1 \le N_v \le N_w + 1$ ή $N_w - 1 \cdot m^0 \le N_v \le N_w + 1 \cdot m^0$. Συνεπώς, ο v θα πρέπει να είναι γειτονικός κόμβος του w.$1 \le d \le$ m-1,

Περίπτωση 2: Αν $i \ge 1$, τότε $N_v = (N_x \pm d \cdot m^i)$ div m $= N_w \pm d \cdot m^j$, όπου j = i - 1. Ως αποτέλεσμα, ο v θα πρέπει να είναι γειτονικός κόμβος του w.

Θεώρημα 3: *Ο συνολικός αριθμός κόμβων στο επίπεδο l είναι (m-1) φορές μεγαλύτερος από τον συνολικό αριθμό των κόμβων σε όλα τα προηγούμενα επίπεδα από το 0 έως το l-1.*

Απόδειξη: Έστω χ, ο συνολικός αριθμός κόμβων στο επίπεδο *l*, και y ο συνολικός αριθμός κόμβων σε όλα τα προηγούμενα επίπεδα, τότε $\chi = m^l$ και $y = 1 + m^1 + m^2 + ... + m^{l-1} = (m^l - 1)/(m-1)$. Έτσι προκύπτει ότι, $y \cdot (m-1) = m^l - 1 < x$.

Το θεώρημα αυτό δηλώνει ότι η πλειοψηφία των κόμβων στο σύστημα έχουν υψηλό αριθμό επιπέδου και πιθανότατα είναι φύλλα. Η κυριαρχία, αυτή, των κόμβων φύλλων αποβαίνει κρίσιμη, για να διατηρηθεί χαμηλό το κόστος εξισορρόπησης φόρτου όπως θα δούμε στη συνέχεια.

Θεώρημα 4: *Το μέγιστο μέγεθος του πίνακα δρομολόγησης ενός κόμβου στο επίπεδο l είναι* $m \cdot l$.

Απόδειξη: Το θεώρημα αυτό προκύπτει από τον τρόπο κατασκευής των πινάκων δρομολόγησης. Από τη στιγμή που υπάρχουν συνολικά m^l κόμβοι στο επίπεδο l και ένας κόμβος διατηρεί στους πίνακες δρομολόγησης του κόμβους του ίδιου επιπέδου με αυτόν που βρίσκονται σε απόσταση $d \cdot m^i$, όπου d = 1..m-1, το μέγιστο μέγεθος ενός πίνακα δρομολόγησης είναι $d \cdot \log_m m^l = d \cdot l$.

Πόρισμα 4.1: *Το μέγιστο μέγεθος του πίνακα δρομολόγησης ενός κόμβου στο δίκτυο είναι* $m \cdot \log_m N$.

Απόδειξη: Η απόδειξη είναι τετριμμένη καθώς το μέγιστο ύψος του δέντρου είναι $\log_m N$.

Το πόρισμα αυτό μας παρέχει τα κατάλληλα εργαλεία για αποτελεσματική ενημέρωση του δικτύου. Αν μπορέσουμε να περιορίσουμε το κόστος μιας ενημέρωσης, να είναι όσο η κατασκευή ενός μόνο από τους δύο πίνακες δρομολόγησης, τότε πετυχαίνουμε την απαραίτητη υπο-τετραγωνική εξάρτηση.
Αλγόριθμοι για διάφορες πράξεις έχουν υιοθετηθεί από τη δομή BATON και χρησιμοποιούνται στο BATON*. Οι αλγόριθμοι αυτοί περιγράφονται παρακάτω.

2.2.4: Εισαγωγή και αποχώρηση κόμβου από το δίκτυο

Ο αλγόριθμος εισαγωγής ενός νέου κόμβου στη δομή BATON*, μοιάζει αρκετά με τον αντίστοιχο αλγόριθμο της δομής BATON και δίδεται ακριβώς από κάτω.

```
Algorithm 2.2.1: Join (node n, node newNode){
if ( Full(LeftRoutingTable(n)) AND Full(RightRoutingTable(n)) AND
    NOT Full(Children(n)) )
  {
    n.AcceptChild(newNode)
    if ( Adjacent(newNode) = n )
      { SplitData(n, newNode) }
    else
      { SplitData( AdjacentSibling(newNode), newNode ) }
  }
else
 {
   if ( NOT Full(LeftRoutingTable(n) OR  NOT Full(RightRoutingTable(n) )
     { Join(Parent(n), newNode) }
   else
     {
        m=SomeNodesNotHavingEnoughChildrenIn
         ( LeftRoutingTable(n), RightRoutingTable(n) )
       if( there exists such an m )
          { Join(m, newNode) }
        else
         {
          a = one of its adjacent nodes
           Join(a, newNode)
         }
     }
 }
}
```

Ένας κόμβος μπορεί να δεχτεί έναν νεοεισερχόμενο κόμβο ως παιδί του, μόνο αν έχει πλήρεις και τους δύο πίνακες δρομολόγησής του και λιγότερα από m παιδιά. Διαφορετικά θα πρέπει να προωθήσει την αίτηση εισαγωγής είτε στον πατέρα του είτε στον «in-order» γείτονά του, που βρίσκεται στο κατώτερο επίπεδο, είτε σε έναν «γειτονικό» κόμβο του ίδιου επιπέδου, ο οποίος όμως έχει λιγότερα από m παιδιά. Στο BATON, το εύρος τιμών, που θα διαχειρίζεται ένας νεοεισερχόμενος κόμβος, είναι ένα κομμάτι από το εύρος που διαχειρίζεται ο πατέρας του. Στο BATON*, με την ανάθεση

ενός κομματιού εύρους στον νέο κόμβο, το οποίο προέρχεται αποκλειστικά και μόνο από τον πατέρα του, θα ανάγκαζε τους νεοεισελθόντες κόμβους να είναι «in-order» γείτονες με τον πατέρα τους. Αντί γι' αυτό, στους νεοεισερχόμενους κόμβους επιτρέπεται να εμφανίζονται οπουδήποτε στην in-order γειτονική σειρά (θα μπορούσαμε να επιλέξουμε συγκεκριμένα κάποιον κόμβο από τη σειρά αυτή έτσι ώστε να πετύχουμε τη καλύτερη τοπική κατανομή φόρτου). Το εύρος τιμών που διαχειρίζεται ένας νέος κόμβος, προέρχεται από έναν από τους in-order γείτονές του ή από τον πατέρα του ή από έναν αδερφό του.

Ένας κόμβος μπορεί να αποχωρήσει από το δίκτυο μόνο αν δεν προκαλεί ανισορροπία, με το να αφήσει κενή τη θέση του στο δέντρο. Διαφορετικά θα πρέπει να βρει έναν κόμβο αντικαταστάτη, στέλνοντας μια αίτηση εύρεσης αντικαταστάτη στον in-order γείτονά του στο κατώτερο επίπεδο.

Στη δομή BATON*, η αναζήτηση μια θέσης για έναν καινούργιο κόμβο, ή η αναζήτηση αντικαταστάτη κοστίζει $O(\log_m N)$ αφού το ύψος του δέντρου είναι $O(\log_m N)$. Το κόστος ενημέρωσης ενός πίνακα δρομολόγησης είναι $O(m \cdot \log_m N)$ για τους γειτονικούς πίνακες δρομολόγησης, αφού ο μέγιστος αριθμός γειτόνων του ιδίου επιπέδου που μπορεί να έχει ένας κόμβος είναι $O(m \cdot \log_m N)$, και κάθε ένας από αυτούς θα πρέπει να προσθέσει μια καινούργια εγγραφή ή να αφαιρέσει μια εγγραφή από τους πίνακες δρομολόγησής του. Επίσης, ένας νεοεισερχόμενος κόμβος πρέπει να κατασκευάσει και τους δικούς του πίνακες δρομολόγησης, με το πολύ $O(m \cdot \log_m N)$ εγγραφές, κάθε μια από τις οποίες μπορεί να ανακτηθεί σε σταθερό χρόνο από τον πατέρα του νέου κόμβου. Επιπρόσθετα, θα πρέπει να δημιουργηθούν/διαγραφούν μια σύνδεση προς τον πατέρα και άλλες δύο συνδέσεις προς τους in-order γείτονες. Δεν υπάρχουν συνδέσεις προς παιδιά, για έναν κόμβο που εισήχθη μόλις στο δίκτυο ή που είναι να αποχωρήσει. Προσθέτοντας όλα αυτά, το συνολικό κόστος εισαγωγής ή διαγραφής ενός κόμβου είναι $O(m \cdot \log_m N)$.

2.2.5: Επεξεργασία ερωτήσεων, Εισαγωγή και Διαγραφή Δεδομένων

Οι αλγόριθμοι για επεξεργασία ερωτήσεων όπως και για εισαγωγή και διαγραφή δεδομένων είναι επίσης λίγο διαφορετικοί στο BATON* σε σχέση με το BATON. Η βασικότερη πράξη όλων αυτών είναι η απλή αναζήτηση μιας ακριβούς τιμής και περιγράφεται σε μορφή ψευδοκώδικα στον αλγόριθμο 2.2.2.

```
Algorithm 2.2.2: Search-Exact(node n, query q){
if ( (n.LowerBound <= q.Value) AND (q.Value <= n.UpperBound) )
   { LocalSearch(n, q) }
else
  {
    if ( n.UpperBound < q.Value )
      {
         m=TheFarthestNodeSatisfyingCondition(m.LowerBound <= q.Value)
        if( there exists such an m )
          { Search-Exact(m, q) }
        else
         {
          l=TheFarthestNodeSatisfyingCondition(l.LowerBound <= q.Value)
          if (there exists such an l )
            { Search-Exact(l, q) }
           else
            { Search-Exact(RightAdjacentNode(n), q) }
         }
      }
    else // ( n.LowerBound > q.Value )
      { // A similar Process is followed towards the left }
  }
}
```

Ένας κόμβος u λαμβάνει μια αίτηση αναζήτησης, οπότε ελέγχει αν υπάρχει ένας γειτονικός κόμβος στους πίνακες δρομολόγησης, ο οποίος να ξέρει ποιος είναι ο καταλληλότερος κόμβος για να χειριστεί την αίτηση. Αν η τιμή που αναζητείται είναι μεγαλύτερη από το άνω όριο του εύρους του u, ενώ δεν υπάρχει δεξιός γειτονικός κόμβος, του οποίου το κάτω όριο είναι μικρότερο από την τιμή που αναζητείται, τότε η

αίτηση θα πρέπει να προωθηθεί στο κατάλληλο παιδί. Στο BATON, το παιδί αυτό θα ήταν απλά το δεξί παιδί. Στο BATON* όμως η διαθέσιμη πληροφορία που αφορά τα όρια των ευρών, υπάρχει αποθηκευμένη σε διάφορα παιδιά, οπότε θα πρέπει να βρεθεί το δεξιότερο παιδί, του οποίου το κάτω όριο να είναι μικρότερο από την τιμή που αναζητείται και έπειτα η αίτηση προωθείται σ' αυτό. Παρομοίως, αν η τιμή προς αναζήτηση είναι μικρότερη από το κάτω όριο του κόμβου, ενώ δεν υπάρχει αριστερός γειτονικός κόμβος του οποίου το άνω όριο να είναι μεγαλύτερο από την τιμή που αναζητείται, τότε ο κόμβος προσπαθεί να βρει το αριστερότερο παιδί του οποίου το άνω όριο είναι μεγαλύτερο από την τιμή προς αναζήτηση, για να του προωθήσει την αίτηση.

Οι αλγόριθμοι για ερωτήσεις αναζήτησης, με εύρος τιμών, για εισαγωγή και διαγραφή δεδομένων τροποποιούνται ανάλογα.

2.2.6: Αστοχία Κόμβου και Ανοχή Σφαλμάτων

Η αστοχία κόμβου αντιμετωπίζεται μέσω του πατέρα του κόμβου. Εδώ, εστιάζεται η προσοχή μας στην ανοχή σφαλμάτων, η οποία έχει βελτιωθεί αρκετά λόγω του μεγαλύτερου fan-out και την εξισορρόπηση φόρτου, μια διαδικασία που πλέον με το νέο σχεδιασμό γίνεται ευκολότερα.

Έστω μια περίπτωση όπου ένας κόμβος είναι απομονωμένος από το δίκτυο. Αν ένας κόμβος έχει πλήρεις τους πίνακες δρομολόγησής του, είναι εύκολο να διαπιστώσει πως, όσο μεγαλύτερος είναι ο αριθμός επιπέδου ενός εσωτερικού κόμβου, τόσο μεγαλύτερο αριθμό συνδέσεων έχει, καθώς οι κόμβοι μεγάλου αριθμού επιπέδου, πάντα έχουν περισσότερες γειτονικές συνδέσεις απ' ότι οι κόμβοι με μικρό αριθμό επιπέδου(δηλ. αυτοί που βρίσκονται πιο κοντά στη ρίζα), ενώ ο αριθμός των συνδέσεων προς τους γονείς, τους in-order γείτονες και τα παιδιά δε διαφέρει ανάμεσα στους κόμβους. Ως αποτέλεσμα, οι κόμβοι με μικρό αριθμό επιπέδου, μπορούν πιο εύκολα να χωριστούν από το δίκτυο, απ' ότι κόμβοι με μεγάλο αριθμό επιπέδου. Όπως και στο BATON, υπάρχουν δύο περιπτώσεις. Στην πρώτη περίπτωση, η ρίζα είναι ένας κόμβος ο οποίος εύκολα απομονώνεται, επειδή δεν έχει καμία γειτονική σύνδεση. Σε αυτή τη περίπτωση, αν και τα m παιδιά του και οι δύο in-order συνδέσεις του είναι κατεστραμμένες, τότε

απομονώνεται από το υπόλοιπο σύστημα. Στη δεύτερη περίπτωση, αν οι πίνακες δρομολόγησης ενός κόμβου δεν είναι πλήρεις, τότε επίσης απομονώνεται εύκολα από το δίκτυο. Ωστόσο, εκείνος ο κόμβος, πιθανότατα είναι ένας νεοεισερχόμενος κόμβος(αφού οι πίνακες του δεν είναι πλήρεις) και το κόστος επανασύνδεσης στο δίκτυο είναι χαμηλό. Να σημειωθεί ότι στο BATON*, όσο περισσότερο διάστημα παραμένει ένας κόμβος στο δίκτυο, φαίνεται να ανεβαίνει σταδιακά προς τη ρίζα, με την έννοια ότι οι νέοι κόμβοι πάντα εισάγονται στο δέντρο, ως παιδιά «παλιών» κόμβων, και άρα ως φύλλα. Οπότε, οι παλιοί κόμβοι με την πάροδο του χρόνου φαίνονται να είναι πιο κοντά στη ρίζα παρά στα φύλλα.

Έστω η περίπτωση, όπου μια ομάδα συνδεδεμένων κόμβων αποκόπτεται από το δίκτυο. Οι κόμβοι είναι απομονωμένοι, αν όλες οι συνδέσεις τους προς κόμβους έξω από την ομάδα είναι κατεστραμμένες. Με άλλα λόγια, ο ελάχιστος αριθμός κατεστραμμένων συνδέσεων που απαιτείται, έτσι ώστε μια ομάδα κόμβων να αποκοπεί από το δίκτυο είναι $S = \sum$ *συνδέσεων όλων των κόμβων που ανήκουν στην ομάδα* - $\sum$ *εσω-ομαδικών συνδέσεων ανάμεσα σε κόμβους που ανήκουν στην ομάδα*. Από τη στιγμή που οι κόμβοι που βρίσκονται ψηλά στο δέντρο(δηλαδή κοντά στη ρίζα), πάντα θα περιέχουν λιγότερες συνδέσεις από τους κόμβους που βρίσκονται χαμηλότερα, θα αναλύσουμε μόνο την περίπτωση όπου το δίκτυο διαχωρίζεται σε δύο ομάδες, αυτή με κόμβους χαμηλότερου επιπέδου(δηλ. με μεγάλο αριθμό επιπέδου) και άλλη μια με κόμβους υψηλότερου επιπέδου(σε άλλες περιπτώσεις, το S έχει πάντα υψηλότερες τιμές). Με άλλα λόγια, το δίκτυο χωρίζεται σε δύο κομμάτια, την κεφαλή(κόμβους γύρω από τη ρίζα) του δέντρου και τη βάση(κόμβοι που είναι πιο κοντά στα φύλλα). Από τη στιγμή που ο αριθμός των κόμβων χαμηλού επιπέδου κυριαρχεί στο σύστημα (αφού όσο μεγαλώνει ο αριθμός επιπέδου αυξάνονται και οι κόμβοι ανά επίπεδο), ο αριθμός των κόμβων υψηλού επιπέδου που θα αποκοπούν από το δίκτυο δεν είναι αξιοσημείωτος. Να σημειωθεί ότι η απομάκρυνση κόμβων, που βρίσκονται ψηλά στο δέντρο, δε δημιουργεί επιπλέον διάσπαση, απ' ότι η απομάκρυνση κόμβων που βρίσκονται κοντά στα φύλλα – δεν είναι πλέον σημαντικοί για σκοπούς ενημέρωσης και αναζήτησης. Στην πραγματικότητα, η ρίζα του δέντρου μπορεί να απομακρυνθεί και να μη δημιουργήσει κατάρρευση του δέντρου και διάσπασή του σε ένα εξαρθρωμένο δάσος.

Για περαιτέρω ανάλυση, ας προσπαθήσουμε να μικρύνουμε όσο το δυνατό περισσότερο την ποσότητα S. Η ποσότητα S ελαχιστοποιείται όταν όλοι οι κόμβοι στο πρώτο κομμάτι(κόμβοι κοντά στη ρίζα), είναι κανονισμένοι έτσι ώστε όλα τα επίπεδα να ανήκουν στην ομάδα, από τη στιγμή που όλες οι γειτονικές συνδέσεις ,οι οποίες συνεισφέρουν τον μεγαλύτερο αριθμό ανάμεσα σε όλα τα είδη των συνδέσεων, είναι εσω-ομαδικές συνδέσεις. Σε αυτή την περίπτωση, οι εναπομένουσες συνδέσεις οι οποίες πρέπει να είναι κατεστραμμένες, είναι in-order συνδέσεις κόμβων προς κόμβους φύλλα και συνδέσεις από κόμβους που βρίσκονται στο τελευταίο επίπεδο προς τα παιδιά τους. Έστω k, ο αριθμός των κόμβων στο πρώτο κομμάτι και k' ο συνολικός αριθμός κόμβων στο τελευταίο επίπεδο του πρώτου κομματιού, τότε $S = 2 \cdot k + m \cdot k'$. Ακολουθώντας το θεώρημα 3, έχουμε $k' > (m-1) \cdot (k-k')$ ή $k' > k \cdot (m-1)/m$. Έτσι προκύπτει ότι $S < 2 \cdot k + m \cdot k \cdot (m-1) / m = 2 \cdot k + (m-1) \cdot k$ **(1)**. Στη χειρότερη περίπτωση, αν το τελευταίο επίπεδο του πρώτου κομματιού είναι ακριβώς ένα επίπεδο πάνω από το τελευταίο επίπεδο του δέντρου, οι in-order συνδέσεις μπορεί να δείχνουν στον ίδιο κόμβο και γι' αυτό $2 \cdot k$ κόμβοι ίσως να υπολογιστούν διπλά με τον προηγούμενο τρόπο. Έτσι προκύπτει ότι, το S είναι μόνο μικρότερο από $(m-1) \cdot k$ **(2).**

(1) και **(2)** $\Rightarrow S < [(m-1) \cdot k, (m+1) \cdot k]$. Με άλλα λόγια, έστω f ο αριθμός των «πεσμένων» κόμβων στο σύστημα, τότε ο μεγαλύτερος αριθμός κόμβων οι οποίοι μπορούν να αποκοπούν από το δίκτυο, ως αποτέλεσμα της αστοχίας, είναι από $f/(m+1)$ έως $f/(m-1)$. Μπορεί εύκολα να διαπιστώσει κανείς, ότι όταν το m είναι αρκετά μεγάλο, ο ρυθμός αυτός είναι πολύ μικρός. Ως αποτέλεσμα, το σύστημα αυτό παρέχει υψηλή ανοχή σφαλμάτων.

2.2.7: Εξισορρόπηση Φόρτου και Αναδόμηση Δικτύου

Η τοπική εξισορρόπηση φόρτου με τη βοήθεια in-order κόμβων είναι εύκολο να γίνει και επίσης είναι ο φθηνότερος τρόπος εξισορρόπησης φόρτου, εκεί όπου η μέθοδος αυτή επαρκεί σχετικά με τις απαιτήσεις. Ωστόσο, μια τοπική λύση δε μπορεί να τα καταφέρει όταν προκαλείται καθολική ανισορροπία. Για παράδειγμα, όταν η κατανομή των δεδομένων είναι ασύμμετρη, η χρήση μόνο τοπικών μεθόδων εξισορρόπησης φόρτου μπορεί να επιφέρει διαδοχική μετακίνηση δεδομένων, κάτι το οποίο κοστίζει πολύ. Μια

λύση γι' αυτό το πρόβλημα είναι ότι θα πρέπει να αφαιρούμε κόμβους από λιγότερο φορτωμένες περιοχές και να τους προσθέτουμε σε υπερφορτωμένες περιοχές. Από τη στιγμή που το σύστημά μας είναι μια δενδρική δομή, οι εσωτερικοί κόμβοι δεν είναι εύκολο να μετακινηθούν, έτσι η λύση αυτή είναι δυνατή μόνο για κόμβους φύλλα. Ειδικότερα, ένα φορτίο μπορεί να μοιραστεί μεταξύ ενός βαριά φορτωμένου φύλλου κι ενός ελαφρώς φορτωμένου φύλλου. Καθώς αυξάνεται το fan-out, οδηγούμαστε σε έναν αυξανόμενο αριθμό φύλλων. Όταν η ποσότητα m είναι μεγάλη, το να βρεθεί ένα φύλλο που να ικανοποιεί τη συνθήκη για να γίνει εξισορρόπηση φόρτου είναι εύκολο. Γενικά, αν ένας κόμβος είναι υπερφορτωμένος, προσπαθεί να μετριάσει το φορτίο του πρώτα με τους in-order γείτονές του. Αν δεν υπάρχει κανένας ελαφρά φορτωμένος in-order γείτονας, τότε προσπαθεί να βρει ένα ελαφρά φορτωμένο φύλλο για να μοιραστεί μαζί του το φορτίο. Από τη στιγμή που θα βρεθεί ένα τέτοιο φύλλο, θα πρέπει να αφήσει την τρέχουσα θέση του στο δέντρο και να επανεισαχθεί στη νέα θέση για να μοιραστεί το φορτίο του υπερφορτωμένου κόμβου. Η διαδικασία αυτή μπορεί να προκαλέσει ανισορροπία στο δέντρο γι' αυτό, αν είναι απαραίτητο, ενεργοποιείται αμέσως η διαδικασία αναδόμησης (ίδια με την αντίστοιχη διαδικασία της δομής BATON). Το κόστος αναδόμησης του δικτύου δεν σχετίζεται με το fan-out αλλά εξαρτάται μόνο από τον αριθμό των κόμβων που μεσολαβούν ανάμεσα στην παλιά και στη νέα θέση του κόμβου που μετακινήθηκε.

2.2.8: Ρύθμιση του παράγοντα m (fan-out)

Το κόστος αναζήτησης μειώνεται λογαριθμικά όσο αυξάνεται το fan-out. Το κόστος όμως για την εισαγωγή ή τη διαγραφή ενός κόμβου αυξάνεται. Από τη στιγμή που η αναζήτηση είναι μια διαδικασία που, στα περισσότερα συστήματα, καλείται πιο συχνά απ' ότι η εισαγωγή και η διαγραφή ενός κόμβου. Η επιλογή του κατάλληλου μεγέθους του fan-out εξαρτάται από τον λόγο του αριθμού των ερωτήσεων προς τον αριθμό των κόμβων που ενημερώνονται. Στο τμήμα αυτό, παρέχεται ένα απλό μοντέλο υπολογισμού της βέλτιστης τιμής του m.

Από τη στιγμή που η αναζήτηση και η ενημέρωση είναι οι κυρίαρχες πράξεις στο σύστημα, θα πρέπει να κατασκευάσουμε ένα απλό μοντέλο κόστους (C) που να

βασίζεται σε αυτές. Έστω α, η πιθανότητα η πράξη που θα συμβεί στο σύστημα να είναι αναζήτηση και 1-α να είναι ενημέρωση. Τότε C = α·S + (1-α)·U, $0 \le \alpha \le 1$, όπου με S και U σημειώνεται το κόστος αναζήτησης και ενημέρωσης αντίστοιχα. Από τη στιγμή που τα προσεγγιστικά κόστη των πράξεων αναζήτησης και ενημέρωσης είναι αντίστοιχα $O(\log_m N)$ και $O(m \cdot \log_m N)$ (εκφρασμένο σε μηνύματα δρομολόγησης), έχουμε $C(m) = \alpha \cdot \log_m N + (1-\alpha) \cdot m \cdot \log_m N$. Εδώ το α μπορεί να θεωρηθεί ως ο ρυθμιστής. Αν όλες οι πράξεις στο δίκτυο εμπλέκουν πράξεις αναζήτησης, το α πρέπει να είναι μεγάλο και αντίστροφα.

Παραγωγίζοντας το C(m), έχουμε $C'(m) = \frac{\ln N}{m \cdot \ln^2 m} \cdot [(1-a) \cdot m \cdot (\ln m - 1) - a]$. Έστω $F(m) = (1-a) \cdot m \cdot (\ln m - 1) - a$. Από τη στιγμή που ισχύει $F'(m) = (1-a) \cdot \ln m \ge 0$, η $F(m)$ είναι μια γνησίως αύξουσα συνάρτηση. Γι' αυτό η εξίσωση $F(m) = 0$ έχει μοναδική λύση την m_0. Έτσι έχουμε $C'(m) = 0$ όταν $m = m_0$, $C'(m) < 0$ όταν $m < m_0$ και $C'(m) > 0$ όταν $m > m_0$. Έτσι προκύπτει ότι το $C(m)$ έχει ελάχιστο στο m_0.

Παρόλο που δεν είναι εύκολο να βρούμε μια λύση κλειστής μορφής για την εξίσωση $F(m) = 0$, δεν είναι δύσκολο μια αριθμητική λύση χρησιμοποιώντας αριθμητικές μεθόδους όπως η μέθοδος Newton. Για να βρούμε το m_0, υπολογίζουμε το $C(m_1)$ με $m_1 = \lfloor m_0 \rfloor$ και το $C(m_2)$ με $m_2 = \lceil m_0 \rceil$ και επιλέγουμε την τιμή που ελαχιστοποιεί το κόστος. Με αυτόν τον τρόπο βρίσκουμε μια καλή τιμή για το fan-out. Για παράδειγμα, σε ένα σύστημα με 10,000 κόμβους, όπου α = 0.6, χρησιμοποιώντας τη μέθοδο του Newton με αρχική τιμή 2 και 1,000 επαναλήψεις, έχουμε $m_0 = 3.9673$. Ελέγχοντας τα $C(m_1 = 3)$ και $C(m_2 = 4)$, η τιμή που παίρνει το fan-out είναι 4 γιατί $C(3) = 15.0905 > C(4) = 14.6165$.

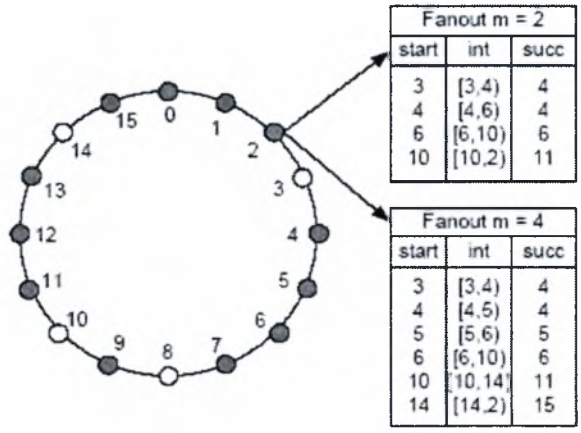
2.2.9: Μια σημείωση για τις multi-way δομές

Είναι πολύ λογική η επέκταση πολλών ήδη γνωστών κατανεμημένων δομών αναζήτησης, έτσι ώστε να προκύψει ένα fan-out μεγέθους m αντί 2 που είναι το σύνηθες, χρησιμοποιώντας τεχνικές παρόμοιες με αυτή του BATON*. Ωστόσο το κόστος της

ενημέρωσης, τυπικά, θα είναι τετραγωνικό ως προς το fan-out, το ίδιο όμως δε συμβαίνει στο BATON*. Εδώ παρουσιάζουμε μια περιγραφή ενός αλγορίθμου και αναλύουμε το Chord. Τα επιχειρήματα για τις υπόλοιπες κατανεμημένες δομές αναζήτησης είναι παρόμοια.

Ας ονομάσουμε, για λόγους ευκολίας, την επεκταθείσα δομή του Chord με μεγαλύτερο fan-out, Chord*. Παρόμοια με το BATON*, θα πρέπει να τροποποιήσουμε τον τρόπο οργάνωσης των πινάκων δρομολόγησης. Αντί να διατηρούμε συνδέσεις προς κόμβους σε απόσταση 2^i, θα διατηρούμε συνδέσεις προς κόμβους σε απόσταση $d \cdot m^i$, όπου $d = 1..m-1$. Πρακτικά , ο τρόπος υπολογισμού του $finger[k].start$ τροποποιείται και γίνεται: $finger[k].start = (n + d \cdot m^{k-1}) \bmod N$.

Η εικόνα 2.13 δείχνει δύο διαφορετικούς πίνακες δρομολόγησης, που αντιστοιχούν σε δύο διαφορετικά fan-outs: $m = 2$ και $m = 4$. Εκτός από τις αλλαγές στους πίνακες δρομολόγησης, όλες οι πράξεις του Chord παραμένουν ίδιες και για το Chord*.



Fanout m = 2

start	int	succ
3	[3,4)	4
4	[4,6)	4
6	[6,10)	6
10	[10,2)	11

Fanout m = 4

start	int	succ
3	[3,4)	4
4	[4,5)	4
5	[5,6)	5
6	[6,10)	6
10	[10,14)	11
14	[14,2)	15

Πηγή: H.V.Jagadish [2]

Εικόνα 2.13: Η δομή Chord* με fan-out m=2 και fan-out m=4

Το πρόβλημα με την επέκταση του Chord είναι το κόστος ενημέρωσης των πινάκων δρομολόγησης. Στο Chord* το μέγεθος τω πινάκων δρομολόγησης είναι $(m-1) \cdot \log_m N$. Έτσι, το κόστος ενημέρωσης των πινάκων δρομολόγησης για έναν καινούργιο κόμβο ή για έναν κόμβο που αποχωρεί από το δίκτυο ανεβαίνει στο $O(((m-1) \cdot \log_m N)^2)$. Σε

σύγκριση με το προηγούμενο κόστος του Chord $O((\log_2 N)^2)$, ο παράγοντας που αυξάνει το κόστος είναι $((m-1)/\log_2 m)^2 \cong (m/\log_2 m)^2$. Το παραπάνω κόστος είναι πολύ μεγάλο σε σχέση με αυτό του BATON*, αφού στο BATON* το κόστος αυξάνεται μόνο κατά έναν παράγοντα $m \cdot \log_m N / \log_2 N = m / \log_2 m$ (γραμμικό κόστος ως προς m). Επιπλέον, μια overlay δομή όπως το Chord που στηρίζεται στον κατακερματισμό δε μπορεί να υποστηρίξει ερωτήσεις με εύρος αποτελεσματικά.

2.2.10: Μια ευέλικτη μέθοδος για υποστήριξη πολύ-παραμετρικών ερωτήσεων

Ακολουθεί μια συνοπτική αναφορά για την υποστήριξη πολύ-παραμετρικών ερωτήσεων από τη δομή BATON*.

Σε εφαρμογές που εμπλέκουν πολλά γνωρίσματα, δεν είναι καθόλου ασυνήθιστο σε ερωτήσεις, να υπάρχει ένας μικρός αριθμός από αυτά τα γνωρίσματα(αντί όλων των γνωρισμάτων). Οπότε, αν θεωρήσουμε τις πολύ-παραμετρικές ερωτήσεις, ως πολυδιάστατες ερωτήσεις, και χρησιμοποιήσουμε συστήματα όπως το CAN ή toVBI-tree για να τις υποστηρίξουμε, κάτι τέτοιο δε θα ήταν αποτελεσματικό γιατί τις περισσότερες φορές οι ερωτήσεις εμπλέκουν μια μεγάλη περιοχή αναζήτησης. Μια εναλλακτική μέθοδος που προτείνεται από το MAAN και το Mercury είναι η ευρετηριοποίηση της κάθε τιμής γνωρίσματος χωριστά. Γι' αυτό το πρόβλημα των πολύ-παραμετρικών ερωτήσεων λύνεται με το να επιλέγεται ένα χαρακτηριστικό, με βάση το οποίο, γίνεται η αναζήτηση ενώ τα υπόλοιπα χρησιμοποιούνται ως μετά-φίλτρα. Παρόλο που με αυτή τη μέθοδο ξεπερνιέται το πρόβλημα, ακόμα παύει να είναι αποτελεσματική όταν ο αριθμός των γνωρισμάτων μεγαλώνει.

Για να υποστηρίξει το BATON* πολύ-παραμετρικές ερωτήσεις, ακολουθείται μια μέθοδος όπου όλο το διάστημα των γνωρισμάτων χωρίζεται σε τομείς: κάθε τομέας χρησιμοποιείται για την ευρετηριοποίηση ενός χαρακτηριστικού (αν αυτό εμφανίζεται συχνά στις ερωτήσεις) ή μιας ομάδας χαρακτηριστικών (αν τα χαρακτηριστικά αυτά εμφανίζονται σπάνια σε ερωτήσεις). Από τη στιγμή που το BATON* μπορεί μόνο να υποστηρίζει ερωτήσεις πάνω σε μια διάσταση δεδομένων, αν ευρετηριοποιήσουμε μια ομάδα γνωρισμάτων, θα πρέπει να μετατρέψουμε τις τιμές τους σε μονοδιάστατες τιμές:

στο σύστημα αυτό επιλέγεται η καμπύλη Hilbert Space Filling. Για παράδειγμα, αν έχουμε ένα σύστημα με 10 γνωρίσματα: $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{10}$ από τα οποία μόνο τα 4 από τα 10 εμφανίζονται συχνά(π.χ. στο 90% των ερωτήσεων), θα πρέπει να χτίσουμε 4 διαφορετικά ευρετήρια. Τα υπόλοιπα γνωρίσματα χωρίζονται ισομερώς σε δύο ομάδες για ευρετηριοποίηση, με 3 γνωρίσματα η κάθε ομάδα. Με αυτόν τον τρόπο μπορούμε να μειώσουμε σημαντικά τον αριθμό των αντιγραφών από 10 σε 6.

Επεξεργασία πολύ-παραμετρικών ερωτήσεων

Γενικά μια πολύ-παραμετρική ερώτηση Q μπορεί να οριστεί ως ένα σύνολο από q_i υποερωτήσεις, όπου κάθε υποερώτηση εμπλέκει κι ένα γνώρισμα. Το αποτέλεσμα της ερώτησης Q είναι μια τομή όλων τω αποτελεσμάτων που επιστρέφονται από τις υποερωτήσεις. Έτσι, μπορούμε να βρούμε το τελικό αποτέλεσμα της ερώτησης Q, απλά βρίσκοντας αποτελέσματα που να ικανοποιούν μια αυθαίρετη υποερώτηση και χρησιμοποιώντας έπειτα τις υπόλοιπες ως φίλτρα. Η επιλεχθείσα ερώτηση ονομάζεται επικρατούσα ερώτηση. Αφού επιλέξουμε αυθαίρετα μια υποερώτηση και τη θεωρήσουμε επικρατούσα, έπειτα θα πρέπει να τη μετατρέψουμε σε ερωτήσεις οι οποίες θα μπορούν να εκτελεστούν στο δίκτυο που σχηματίστηκε με βάση τη δομή BATON*, αφού το πεδίο ορισμού των τιμών μιας υποερώτησης είναι απλά ένας τομέας στο δίκτυο. Για να προχωρήσουμε με τη μετατροπή θα πρέπει να εξετάσουμε δύο περιπτώσεις. Αν η επικρατούσα υποερώτηση εμπλέκει ένα γνώρισμα, το οποίο έχει ευρετηριοποιηθεί ξεχωριστά από τα άλλα, τότε μετατρέπεται σε μια BATON* ερώτηση με άμεσο τρόπο – χρησιμοποιώντας μια συνάρτηση ταιριάσματος. Ωστόσο, αν η επικρατούσα υποερώτηση εμπλέκει ένα γνώρισμα, το οποίο έχει ευρετηριοποιηθεί μαζί με άλλα σε ομάδα(πολυδιάστατο ευρετήριο), θα πρέπει πρώτα να διασπαστεί σε μικρότερες υποερωτήσεις χρησιμοποιώντας την καμπύλη Hilbert Space Filling. Έπειτα, οι υποερωτήσεις αυτές μετατρέπονται σε BATON* ερωτήσεις. Τελικά, οι ερωτήσεις προωθούνται παράλληλα στο σύστημα.

Εισαγωγή και Διαγραφή Δεδομένων

Όταν εισάγονται δεδομένα πρέπει να ευρετηριοποιηθούν (εισαχθούν) σε όλα τα ευρετήρια. Από τη στιγμή που τα δεδομένα που είναι να εισαχθούν μπορούν να θεωρηθούν ως μια πολύ-παραμετρική ερώτηση, στην οποία οι υποερωτήσεις αποτελούν ερωτήσεις ακρίβειας με συγκεκριμένες τιμές γνωρισμάτων, ο αλγόριθμος εισαγωγής δεδομένων είναι παρόμοιος με τον αλγόριθμο επεξεργασίας ερωτήσεων, που περιγράφηκε παραπάνω, μόνο που όλες οι υποερωτήσεις θα πρέπει να εφαρμοστούν σε όλα τα ευρετήρια που έχουν δημιουργηθεί.

Για τη διαγραφή δεδομένων, θα πρέπει να διαγραφούν όλα τα αντίγραφα που έχουν δημιουργηθεί. Όπως και στον αλγόριθμο εισαγωγής , οι κόμβοι που διατηρούν τιμές ευρετηρίου των δεδομένων προς διαγραφή, εντοπίζονται από μια αντίστοιχη πολύ-παραμετρική ερώτηση. Έπειτα, τα δεδομένα ευρετηρίου διαγράφονται από αυτούς τους κόμβους.

2.3: Το δέντρο VBI (Virtual Binary Index)

Το δέντρο εικονικού δυαδικού ευρετηρίου είναι μια αφηρημένη δομή η οποία χτίζεται πάνω από ένα δίκτυο. Οι αφηρημένες μέθοδοι, οι οποίες ορίζονται παρακάτω, μπορούν να υποστηρίξουν κάθε είδους ιεραρχική δενδρική δομή ευρετηρίου, όπου το εύρος τιμών το οποίο διαχειρίζεται ένας κόμβος, καλύπτει τα εύρη τιμών(είναι υπερσύνολο των διαστημάτων για τα οποία είναι υπεύθυνα τα παιδιά του) που διαχειρίζονται τα παιδιά του. Οι δομές που μπορεί να υποστηρίξει είναι δημοφιλείς πολυδιάστατες δενδρικές δομές ευρετηρίου όπως το R-δένδρο, το X-δένδρο, το SS-δένδρο, το M-δέντρο και άλλες παρόμοιες δομές.

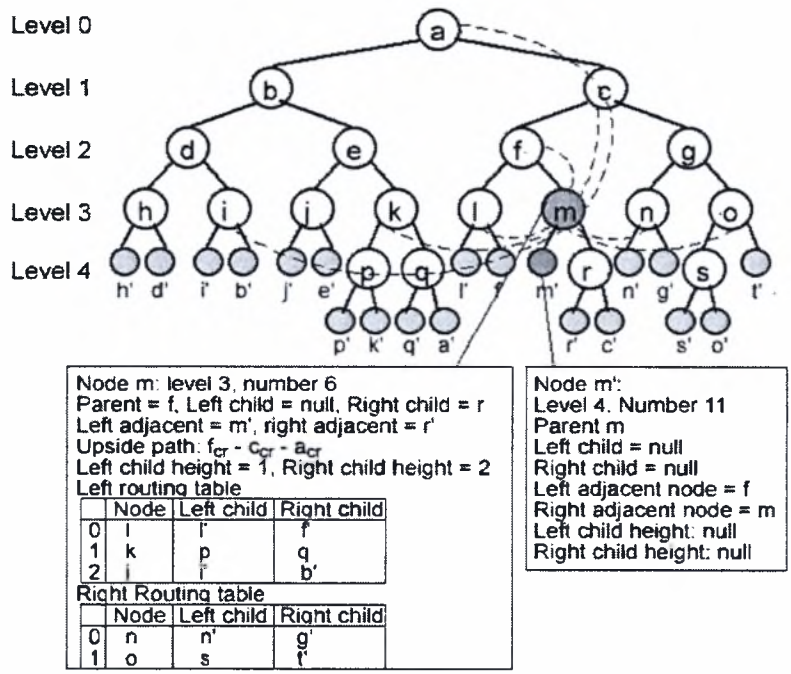
2.3.1: Βασικά χαρακτηριστικά και θεωρήματα του VBI-δέντρου

Όπως και στις περισσότερες κεντρικοποιημένες δομές ευρετηρίου, έτσι και στο VBI-δέντρο [3], οι κόμβοι διακρίνονται σε δύο είδη: τους κόμβους δεδομένων (ή τα φύλλα) και τους κόμβους δρομολόγησης (ή τους εσωτερικούς κόμβους). Όπως καταλαβαίνουμε και από τα ονόματα τους, οι κόμβοι δεδομένων στην ουσία αποθηκεύουν δεδομένα ενώ οι κόμβοι δρομολόγησης κρατάνε μόνο πληροφορίες που αφορούν την δρομολόγηση των μηνυμάτων. Όπως και στο BATON, κάθε κόμβος δρομολόγησης στο VBI-δέντρο διατηρεί συνδέσεις προς τον πατέρα του, τα παιδιά του, τους in-order γείτονές του και τους πλευρικούς πίνακες δρομολόγησης. Ωστόσο, οι εγγραφές στους πίνακες δρομολόγησης δε χρειάζεται να κρατούν πληροφορία για τις περιοχές που καλύπτουν οι γειτονικοί κόμβοι. Αντίθετα, κάθε κόμβος δρομολόγησης διατηρεί έναν «άνω-πλευρικό» πίνακα με πληροφορίες για τις περιοχές που καλύπτει κάθε πρόγονός του. Επιπρόσθετα, κάθε κόμβος χρειάζεται να κρατά κάπου τα ύψη των υποδέντρων του. (Η πληροφορία αυτή χρησιμοποιείται στη διαδικασία αναδόμησης του δικτύου). Επίσης, στο VBI-δέντρο κάθε εσωτερικός κόμβος πρέπει να έχει ακριβώς δύο παιδιά. Σύμφωνα με την αρχή αυτή ο συνολικός αριθμός κόμβων που προκύπτει είναι περιττός αριθμός και ακριβώς οι μισοί από αυτούς είναι κόμβοι δεδομένων. Έτσι, προκύπτει το παρακάτω θεώρημα:

Θεώρημα 3: *Στην εσωτερική (in-order) διαπέραση ενός* VBI-δέντρου οι κόμβοι δεδομένων εναλλάσσονται με τους κόμβους δρομολόγησης.

Το θεώρημα 3 προκύπτει ως συμπέρασμα από τον ορισμό ενός δυαδικού ισορροπημένου δέντρου – από τη στιγμή που κάθε κόμβος έχει ένα παιδί, πρέπει να έχει γεμάτους τους πλευρικούς πίνακες δρομολόγησης(στην καλύτερη περίπτωση, στο επίπεδο 1, μόνο την εγγραφή για τον αδερφό του).

Κάθε σταθμός στο δίκτυο αντιστοιχεί σε ένα ζευγάρι κόμβων στο VBI-δέντρο: ένα κόμβο δεδομένων και έναν δρομολόγησης, όπου ο κόμβος δεδομένων είναι ο αριστερός in-order κόμβος του κόμβου δρομολόγησης. Μοναδική εξαίρεση αποτελεί ο σταθμός, ο οποίος αντιστοιχεί στο δεξιότερο κόμβο δεδομένων στο δέντρο, ο οποίος δεν έχει κόμβο δρομολόγησης. Από τη στιγμή που κάθε σταθμός αντιστοιχεί σε έναν κόμβο δρομολόγησης και έναν δεδομένων και οι ερωτήσεις μπορούν να προωθηθούν μέσω συνδέσεων του κόμβου δρομολόγησης, για εξοικονόμηση χώρου, οι κόμβοι δεδομένων δε χρειάζεται να κρατούν πλευρικούς πίνακες δρομολόγησης αλλά ούτε και «άνω-πλευρικό» πίνακα.



Πηγή: H.V.Jagadish [3]

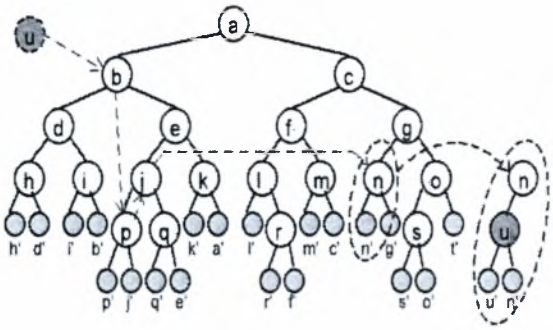
Εικόνα 2.14: Το VBI-δέντρο και οι πληροφορίες που κρατούνται για έναν κόμβο δρομολόγησης και έναν κόμβο δεδομένων.

Ο ειδικός σταθμός που αντιστοιχεί στο δεξιότερο κόμβο δεδομένων, ο οποίος δεν έχει κόμβο δρομολόγησης, πάντα προωθεί τις αιτήσεις στον πατέρα του. Η δομή του VBI-δέντρου φαίνεται στην εικόνα 2.14, όπου κόμβοι με το ίδιο όνομα αντιστοιχούν στον ίδιο σταθμό.

2.3.2: Εισαγωγή νέου κόμβου στο δίκτυο

Για να εισέλθει ένας νέος κόμβος στο δίκτυο, πρέπει να γνωρίζει τουλάχιστον έναν από τους κόμβους που απαρτίζουν, αυτή τη στιγμή, το δίκτυο και να του στείλει μια αίτηση JOIN. Η διαδικασία εισαγωγής διακρίνεται σε δύο φάσεις. Η πρώτη φάση είναι ο προσδιορισμός της θέσης που πρέπει να εισαχθεί ο καινούργιος κόμβος. Η διαδικασία αυτή είναι ίδια με αυτή του BATON, με τη διαφορά ότι μόνο οι κόμβοι δρομολόγησης λαμβάνονται υπ' όψη κατά τη διάρκεια της διαδικασίας. Το κόστος της φάσης αυτής είναι O(logN) όσο και το ύψος του δέντρου. Ο αλγόριθμος εισαγωγής ενός κόμβου είναι ακριβώς ίδιος με τον αλγόριθμο εισαγωγής του BATON(αλγόριθμος 2.1.1).

Αφού προσδιοριστεί η θέση για το νέο κόμβο, ξεκινά η δεύτερη φάση. Κατ' αρχήν, ο κόμβος δεδομένων στη θέση του νέου κόμβου δρομολόγησης χωρίζει την περιοχή που καλύπτει σε δύο υπό-περιοχές χρησιμοποιώντας των αλγόριθμο διαχωρισμού κόμβων. Έπειτα, ένας κόμβος δρομολόγησης δημιουργείται και αντικαθιστά τον κόμβο δεδομένων. Επίσης, δύο νέοι κόμβοι δεδομένων δημιουργούνται και συνδέονται στο δέντρο ως παιδιά του νέου κόμβου δρομολόγησης. Οι υπό-περιοχές και τα δεδομένα που καλύπτονται από τις περιοχές αυτές αναθέτονται στους δύο νέους κόμβους δεδομένων. Τελικά, ο νέος κόμβος δρομολόγησης και το αριστερό του παιδί αντιστοιχούν στο νέο σταθμό του δικτύου. Ο άλλος κόμβος δεδομένων δίνεται στην ευθύνη του δεξιού in-order κόμβου δρομολόγησης. Στην περίπτωση που ο νέος κόμβος είναι το πρώτο παιδί, ο γονιός αυξάνει το ύψος κατά ένα και ενημερώνει το δικό του γονιό. Ο γονιός εκείνος, με τη σειρά του, ελέγχει αν το ύψος του έχει αυξηθεί και αν ναι, ενημερώνει το δικό του γονιό κ.ο.κ.

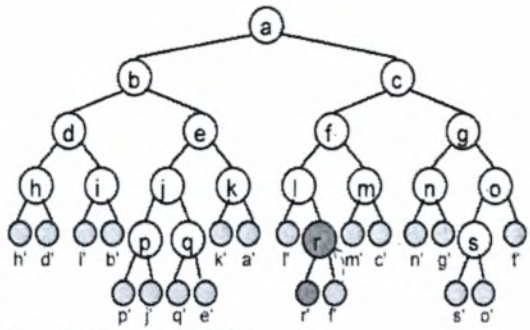


Πηγή: H.V.Jagadish [3]
Εικόνα 2.15: Παράδειγμα Εισαγωγής Κόμβου

Για παράδειγμα υποθέτουμε ότι ο κόμβος u θέλει να εισέλθει στο δίκτυο και στέλνει αίτηση JOIN στον κόμβο b όπως φαίνεται και στην εικόνα 2.15. Ο b έχει τους πλευρικούς του πίνακες δρομολόγησης πλήρεις, έχει ήδη δύο παιδιά (δύο κόμβους δρομολόγησης) και όλοι οι κόμβοι που βρίσκονται καταχωρημένοι στους πίνακες δρομολόγησής του(στη συγκεκριμένη περίπτωση μόνο ο c)έχουν επίσης ως παιδιά δύο κόμβους δρομολόγησης. Γι' αυτό ο b προωθεί την αίτηση σε έναν από τους in-order γείτονές του, τον p. (Να παρατηρηθεί ότι κόμβοι δεδομένων και δρομολόγησης εναλλάσσονται, έτσι ώστε ο ένας in-order γείτονας να είναι κόμβος δεδομένων, και εδώ είναι ο p', ο οποίος αντιστοιχεί στον ίδιο σταθμό όπως και ο in-order γείτονας p). Επειδή, οι πίνακες δρομολόγησης του p δεν είναι πλήρεις, προωθεί την αίτηση στον πατέρα του j, ο οποίος έπειτα προωθεί την αίτηση στον n. Οι πίνακες δρομολόγησης του n είναι πλήρεις και δεν έχει ως παιδιά δύο κόμβους δρομολόγησης(οι κόμβοι δεδομένων δε λαμβάνονται υπ' όψη ως παιδιά σε αυτόν τον αλγόριθμο) και έτσι δέχεται τον νέο κόμβο u ως αριστερό παιδί του. Οπότε, ένας νέος κόμβος δρομολόγησης δημιουργείται και τα δεδομένα που μέχρι τώρα καλύπτονταν από τον κόμβο δεδομένων n', τώρα χωρίζονται σε δύο υποπεριοχές και η μία από αυτές ανατίθεται στον κόμβο u' ενώ η άλλη παραμένει στον κόμβο n'.

Το κόστος ενημέρωσης των πινάκων δρομολόγησης για την εισαγωγή του νέου σταθμού στο δίκτυο είναι 6logN, από το οποίο 2logN είναι για την ενημέρωση των πινάκων δρομολόγησης των γειτόνων του πατέρα, 2logN για την ενημέρωση των πινάκων δρομολόγησης των γειτόνων του νέου κόμβου και τέλος 2logN για την κατασκευή των δύο πλευρικών πινάκων δρομολόγησης του νέου κόμβου. Για την ενημέρωση των τιμών ύψους των υποδέντρων αν αυτές έχουν αλλάξει, απαιτείται μέγιστο κόστος logN, αν στη χειρότερη περίπτωση, η διαδικασία ενημέρωσης διαδίδεται από ένα φύλλο στη ρίζα

2.3.3: Αποχώρηση κόμβου από το δίκτυο


Πηγή: H.V.Jagadish [3]
Εικόνα 2.16: Παράδειγμα Διαγραφής Κόμβου

Όπως και στην εισαγωγή κόμβου έτσι κι εδώ, μόνο οι κόμβοι δρομολόγησης λαμβάνονται υπ' όψη. Αν ένα φύλλο, δηλ. ένας κόμβος δρομολόγησης ο οποίος δεν έχει ως παιδιά κανένα κόμβο δρομολόγησης, επιθυμεί να αποχωρήσει από το δίκτυο και δεν υπάρχει κανένας κόμβος δρομολόγησης στους πλευρικούς πίνακές του, ο οποίος να έχει ως παιδιά κάποιους κόμβους δρομολόγησης, μπορεί να αφήσει το δίκτυο χωρίς να επηρεάζει την ισορροπία του δέντρου. Σε αυτή την περίπτωση, ο κόμβος δεδομένων που αντιστοιχούσε μέχρι τώρα στο σταθμό που αποχωρεί, θα συγχωνευτεί με τον κόμβο δεδομένων του αδερφού του. Ο νέος κόμβος δεδομένων που προκύπτει, θα πάρει τη θέση του κόμβου δρομολόγησης που φεύγει. Για παράδειγμα, θεωρούμε έναν σταθμό που αντιστοιχεί στον κόμβο δρομολόγησης r και θέλει να αποχωρήσει όπως φαίνεται και στην εικόνα 2.16. Είναι φανερό πως ο κόμβος r μπορεί να φύγει, χωρίς να επηρεάσει την ισορροπία του δέντρου(επαναλαμβάνουμε μόνο οι κόμβοι δρομολόγησης λαμβάνονται υπ' όψη). Γι' αυτό, ο αντίστοιχος κόμβος δεδομένων r' συγχωνεύει τα δεδομένα του με τον αδερφό του f', ο οποίος αργότερα ανεβαίνει για να πάρει τη θέση του r που φεύγει. Το κόστος της διαδικασίας αποχώρησης ενός κόμβου είναι 4logN, όπου 2logN χρειάζονται για την ενημέρωση των πινάκων δρομολόγησης των γειτόνων του κόμβου που φεύγει, και 2logN για την ενημέρωση των πινάκων δρομολόγησης των γειτόνων του πατέρα του αποχωρούντος κόμβου. Επιπρόσθετα,. όπως και στην περίπτωση της εισαγωγής ενός κόμβου, πρέπει να υπολογιστεί και ένα μέγιστο κόστος logN για την ενημέρωση του ύψους των υποδέντρων που ενδεχομένως τώρα να έχει αλλάξει.

Αν ένας κόμβος δρομολόγησης θέλει να αποχωρήσει από το δίκτυο και είναι εσωτερικός κόμβος ή φύλλο, του οποίου οι γείτονες όμως έχουν παιδιά, θα πρέπει να βρει έναν αντικαταστάτη που να είναι και φύλλο(δηλ. κόμβος δρομολόγησης χωρίς να έχει κανένα

παιδί που να είναι κόμβος δρομολόγησης). Ο αλγόριθμος εύρεσης αντικαταστάτη είναι ίδιος με τον αντίστοιχο αλγόριθμό στη δομή BATON (αλγόριθμος 2.1.2) και κοστίζει O(logN) μηνύματα. Το κόστος ενημέρωσης των πινάκων δρομολόγησης είναι 8logN, από το οποίο, 4logN είναι για την ενημέρωση των πινάκων δρομολόγησης των γειτόνων του κόμβου αντικαταστάτη και του πατέρα του και 4logN για την αντίστοιχη ενημέρωση των γειτόνων και του πατέρα του κόμβου που αποχωρεί. Να σημειωθεί ότι, τα «άνω-πλευρικά» μονοπάτια δε χρειάζονται ενημέρωση, καθώς διατηρούν μόνο πληροφορία για τις περιοχές που καλύπτουν οι πρόγονοι και όχι φυσικές αναφορές προς αυτούς.

2.3.4: Αστοχία κόμβου και ανοχή σφαλμάτων

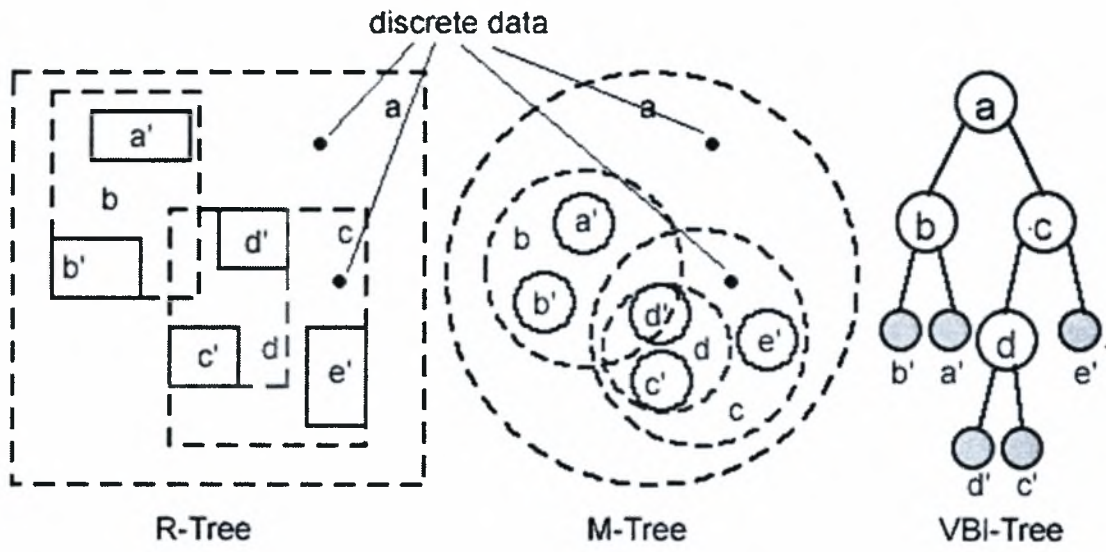
Η ανάκαμψη από σφάλματα σε ένα VBI-δέντρο αντιμετωπίζεται ακριβώς όπως και στη δομή BATON. Όταν ανακαλύπτεται, ότι ένας κόμβος δεν είναι πλέον προσβάσιμος, στέλνεται μια αναφορά στον πατέρα του, κι έπειτα ο πατέρας του είναι υπεύθυνος για τη διαδικασία ανάκαμψης. Ο γονιός του κόμβου που αστόχησε, αν χρειαστεί, βρίσκει έναν κόμβο αντικαταστάτη και μπορεί να επαναφέρει τις συνδέσεις από την πλεονάζουσα πληροφορία που διατηρεί για τους κόμβους που υπάρχουν καταχωρημένοι στους πίνακες δρομολόγησής του καθώς και για τα παιδιά τους. Οι υπόλοιπες λειτουργίες που τυχόν εκτελούνται κατά την διαδικασία της ανάκαμψης, μπορούν να συνεχιστούν κανονικά και οι κόμβοι που «πέσανε» να παρακαμφτούν, είτε μέσω πλευρικών μονοπατιών από συνδέσεις που υπάρχουν στους πίνακες δρομολόγησης, είτε μέσω των συνδέσεων γονιού-παιδιού ή των in-order γειτόνων.

2.3.5: Κατασκευή Ευρετηρίου

Στο παραπάνω τμήμα, έγινε η περιγραφή του δικτύου με βάση τη δομή του VBI-δέντρου, τώρα θα εξηγήσουμε πως θα χτιστεί με βάση τη δομή αυτή, ένα γενικό πολυδιάστατο ευρετήριο. Η βασική ιδέα είναι η ανάθεση μιας περιοχής του διαστήματος του γνωρίσματος σε κάθε κόμβο δεδομένων. Κάθε εσωτερικός κόμβος σχετίζεται με μια περιοχή, η οποία καλύπτει το σύνολο των περιοχών που καλύπτουν τα παιδιά του. Πολλές γνωστές δομές ευρετηρίου ακολουθούν αυτού του είδους την ανάθεση. Η διαφορά είναι στον τρόπο που αυτές οι περιοχές επιλέγονται και χωρίζονται. Εδώ, επιτρέπουμε το ίδιο σύνολο επιλογών, προσαρμοσμένο, στον τρόπο επιλογής και

διαχωρισμού των περιοχών της εκάστοτε δομής. Με αυτό τον τρόπο, μπορούμε να δημιουργήσουμε ένα VBI-δέντρο, ένα VBI-M-δέντρο, ένα VBI-SS-δέντρο, κ.ο.κ. Αρχικά, η ρίζα είναι ο μόνος κόμβος και φυσικά καλύπτει όλο το πεδίο. Όταν νέοι κόμβοι εισέρχονται, το πεδίο χωρίζεται σε μικρότερες υποπεριοχές ενώ όταν σταθμοί φεύγουν από το δίκτυο, οι περιοχές τους συγχωνεύονται όπως αναφέρθηκε.

Η εισαγωγή (ή διαγραφή) δεδομένων γίνεται όπως ακριβώς και στα κεντρικοποιημένα σχήματα. Ωστόσο, προκύπτει ένα θέμα: όταν ένα νεοεισερχόμενο αντικείμενο σε ένα κόμβο, έχει μια τιμή, η οποία δε βρίσκεται μέσα στο διάστημα που καλύπτεται από τα παιδιά ενός κόμβου, ένα από τα δύο παιδιά θα πρέπει να επεκτείνει την περιοχή που καλύπτει. Σε ένα κατανεμημένο σύστημα, μια τέτοια επέκταση μπορεί να επιφέρει μεγάλο κόστος, καθώς περιλαμβάνει ενημέρωση όλων των «άνω-πλευρικών» μονοπατιών των απογόνων του κόμβου αυτού. Για να αποφευχθεί ένα τέτοιο πρόβλημα, προτείνουμε τη χρήση «ξεχωριστών» (discrete) δεδομένων. Τα «ξεχωριστά» δεδομένα ορίζονται ως δεδομένα, τα οποία δεν ανήκουν σε κανένα διάστημα από αυτά που καλύπτουν τα παιδιά ενός κόμβου και γι' αυτό μπορούν να αποθηκευτούν σε ένα κόμβο, που δεν είναι φύλλο (αυτό συνήθως δεν είναι καλή ιδέα σε μια κεντρικοποιημένη βάση αφού οι εσωτερικοί κόμβοι συνήθως διατηρούνται στη μνήμη κι έτσι και το μέγεθος μας περιορίζει αλλά και το υψηλό fan-out είναι πιο σημαντικός απαγορευτικός παράγοντας). Ένας κόμβος δρομολόγησης μπορεί να αποθηκεύσει αυτά τα «ξεχωριστά» δεδομένα, και μόνο όταν ο αριθμός των «ξεχωριστών» αντικειμένων ξεπεράσει ένα κατώφλι, τότε πραγματοποιείται επέκταση των περιοχών από τα παιδιά. Επιπρόσθετα, εφαρμόζεται τεχνική «χαλαρής» ενημέρωσης. Όταν μια περιοχή, την οποία κάλυπτε ένας κόμβος επεκτείνεται, τα μηνύματα ενημέρωσης δεν στέλνονται άμεσα αν το δίκτυο είναι απασχολημένο: δηλ., στέλνονται σε απογόνους του κόμβου αργότερα όταν το δίκτυο είναι πλέον ελεύθερο. Με τη «χαλαρή» πολιτική ενημέρωσης, τελικά δεν γλυτώνουμε κάποιο αριθμό μηνυμάτων, απλά επιτρέπουμε γρηγορότερη ανταπόκριση κατά τη διαδικασία εισαγωγής ενός κόμβου, η οποία προκαλεί τα «ξεχωριστά» δεδομένα να ξεπεράσουν το κατώφλι και επιπλέον αποτρέπουμε τη συμφόρηση του δικτύου σε διαστήματα φόρτου.



Πηγή: H.V.Jagadish [3]

Εικόνα 2.17: Δημιουργία ενός VBI-δέντρου με βάση τον τρόπο επιλογής και διαχωρισμού των περιοχών σε ένα R- και ένα M-δέντρο

Το γενικό σχήμα του ευρετηρίου απεικονίζεται στην εικόνα 2.17. Στην εικόνα αυτή βλέπουμε πως μπορούμε να αντιστοιχίσουμε ένα R-δέντρο και ένα M-δέντρο στο πλαίσιο που περιγράφεται εδώ. Τα δεδομένα έχουν επιλεχθεί έτσι ώστε να προκύπτει το ίδιο VBI-δέντρο και στις δύο περιπτώσεις. Υπάρχουν δύο «ξεχωριστά» αντικείμενα στην εικόνα: το ένα είναι αποθηκευμένο στον κόμβο δρομολόγησης a και το άλλο στον κόμβο δρομολόγησης c.

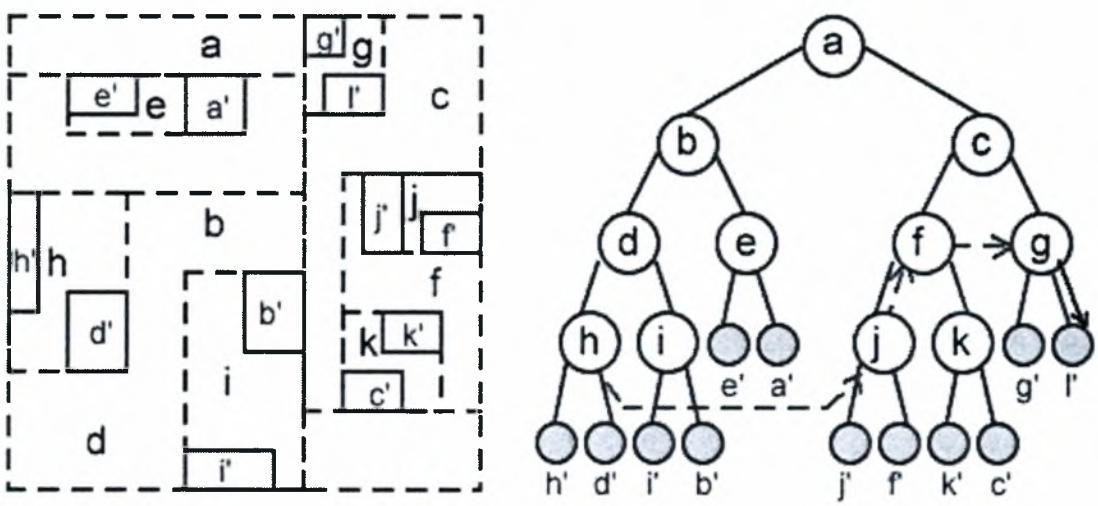
2.3.6: Ερωτήσεις ακριβείας

Για λόγους απλότητας, πρώτα θεωρούμε την περίπτωση, οι περιοχές που καλύπτουν δύο κόμβοι αδέρφια να μην επικαλύπτονται. Η διαδικασίας αναζήτησης με ερώτηση ακριβείας περιγράφεται με λεπτομέρεια στον αλγόριθμο 2.3.3.

```
Algorithm 2.3.3: PointQuery(node n, point p, nearest-checked-ancestor a)
if ( Region(n) covers p )
   if ( Region(LeftChild(n)) covers p )
      { PointQuery(LeftChild(n), p, LeftChild(n)) }
   else if ( Region(RightChild(n)) covers p )
      { PointQuery(RightChild(n), p, RightChild(n)) }
   else
      { LocalSearch(n, p) }
else
  {
     for i=0 to Level(n) – Level(a) -1
       {
          x = UpsidePath(n).get(i)
         if ( Region(x) covers p )
          {
            if (x == a )
              { LocalSearch(x, p) } //finding discrete data
            else
              {
                y = get a neighbor node in other side of the tree rooted at x }
                if ( y != null )
                   { PointQuery(y, p, x) }
                else
                   { PointQuery(Parent(n), p, x) }
              }
            break }
       }
}
```

Για μια ερώτηση ακριβείας ή οποία τίθεται ή φτάνει σε έναν κόμβο έστω n, αν η περιοχή που καλύπτει ο n περιλαμβάνει την ερώτηση, το σημείο αναζήτησης στο δέντρο είναι ο κόμβος n και γι' αυτό η αίτηση είτε επεξεργάζεται από τον ίδιο τον n είτε προωθείται σε ένα από τα παιδιά του. Αν η περιοχή που αντιστοιχεί στον n δεν περιλαμβάνει την ερώτηση, τότε ο n θα πρέπει να βρει τον κοντινότερο πρόγονό του, έστω χ, ο οποίος καλύπτει την περιοχή που βρίσκεται η ερώτηση. Ο n βρίσκει τον χ, χρησιμοποιώντας τον «άνω-πλευρικό» πίνακα. Έπειτα, προωθεί την αίτηση σε έναν γείτονά του, έστω y, τον

οποίο εντόπισε μέσω των πλευρικών πινάκων δρομολόγησης και ο οποίος βρίσκεται στην άλλη πλευρά του δέντρου και είναι απόγονος του χ. Τώρα, ο y χρησιμοποιεί ακριβώς τον ίδιο αλγόριθμο με τον n και συνεχίζει την αναζήτηση. Για να αποφύγει να λάβει πίσω την αίτηση αναζήτησης από τον y, όταν τα δεδομένα είναι «ξεχωριστά» και είναι αποθηκευμένα στον χ, ο οποίος είναι κοινός πρόγονος των n και y, χρησιμοποιεί την παράμετρο του κοντινότερου-ελεγμένου-πρόγονου. Η παράμετρος αυτή δηλώνει τον πατέρα της ρίζας του δέντρου στον οποίο η αναζήτηση πρέπει να περιοριστεί. Αρχικά, η παράμετρος αυτή έχει τιμή null. Η διαδικασία αναζήτησης απεικονίζεται στην εικόνα 2.18.



Πηγή: H.V.Jagadish [3]

Εικόνα 2.18: Διαδικασία αναζήτησης με ερώτηση ακριβείας στον κόμβο h.

Υποθέτουμε ότι ο κόμβος h θέλει να ψάξει για ένα σημείο, που είναι αποθηκευμένο στην περιοχή που καλύπτει ο κόμβος g, σε ένα ευρετηριακό σχήμα R-δέντρου για δεδομένα δύο διαστάσεων. Από τη στιγμή που το σημείο αναζήτησης δεν καλύπτεται από τη περιοχή που διαχειρίζεται ο h, ελέγχει το «άνω-πλευρικό» του μονοπάτι και ανακαλύπτει ότι η μόνη περιοχή που καλύπτει το σημείο αναζήτησης, αντιστοιχεί στον κόμβο a. Ως αποτέλεσμα, ο h προωθεί την ερώτηση στον j, ο οποίος είναι γειτονικός κόμβος του h, που βρίσκεται στο υποδέντρο με ρίζα τον a στην άλλη πλευρά του δέντρου. Τώρα ο j ελέγχει το δικό του «άνω-πλευρικό» μονοπάτι και διαπιστώνει ότι ο κοντινότερος

πρόγονος, του οποίου η περιοχή καλύπτει το σημείο αναζήτησης, είναι ο c. Ωστόσο, δε μπορεί να βρει ένα γειτονικό κόμβο δρομολόγησης στην άλλη πλευρά του δέντρου, ο οποίος να έχει ρίζα τον κόμβο c. Γι' αυτό ο j προωθεί την ερώτηση στον πατέρα του f. Τελικά, ο f προωθεί την ερώτηση στον g, ο οποίος είναι και ο κόμβος προορισμού και έτσι ο g προωθεί την αίτηση στον κόμβο δεδομένων l'.

Με τον παραπάνω αλγόριθμο, όταν ένας κόμβος n θέλει να ψάξει για ένα σημείο, αν το σημείο αυτό καλύπτεται από την περιοχή του n το κόστος αναζήτησης της διαδικασίας αναζήτησης είναι $h \leq \log N$, όπου h είναι το ύψος του υποδέντρου που έχει ρίζα τον n. Αν το σημείο αναζήτησης δεν καλύπτεται από την περιοχή του n, ούτε από την περιοχή της ρίζας, αφού έχουν ελεγχθεί το «άνω-πλευρικό» μονοπάτι και οι πίνακες δρομολόγησης, τότε χρειάζεται μόνο ένα βήμα, να προωθηθεί η αίτηση πλευρικά σε έναν κόμβο που έχει υπό τον έλεγχό του ένα υποδέντρο, που το ύψος του είναι μικρότερο τουλάχιστον κατά ένα από το ύψος του τρέχοντος δέντρου αναζήτησης(περιορίζεται από την παράμετρο του κοντινότερου-ελεγμένου-προγόνου). Στη δεύτερη περίπτωση, αν το σημείο αναζήτησης καλύπτεται από την περιοχή της ρίζας, χρειάζεται μόνο ένα βήμα για να προωθηθεί η αίτηση σε γειτονικό κόμβο του n. Μετά από αυτό, η αίτηση προωθείται προς τα πάνω στη ρίζα, διαδικασία που κοστίζει logN βήματα. Ως αποτέλεσμα, το συνολικό κόστος της διαδικασίας αναζήτησης είναι O(logN). Να σημειωθεί ότι, η προώθηση της αίτησης αναζήτησης από έναν κόμβο σε έναν πρόγονό του, μπορεί να μειωθεί, με το να χρησιμοποιούνται οι in-order συνδέσεις μεταξύ των κόμβων, αντί να χρησιμοποιούνται μόνο οι συνδέσεις γονιού-παιδιού.

Η μόνη ανησυχία με τη δενδρική δομή δικτύου είναι, ότι ένας μικρός αριθμός κόμβων κοντά στη ρίζα θα πρέπει να κάνει δυσανάλογα περισσότερη δουλειά. Ωστόσο, τέτοια περίπτωση δεν υπάρχει σε αυτόν τον αλγόριθμο, καθώς οι αιτήσεις αναζήτησης προωθούνται προς τα πάνω μόνο σε δύο περιπτώσεις. Στην πρώτη περίπτωση, ο κόμβος παραλήπτης έχει αποθηκευμένα, «ξεχωριστά» δεδομένα, τα οποία σχετίζονται με την ερώτηση και άρα θα πρέπει να ληφθούν υπ' όψη. Στη δεύτερη περίπτωση, ο τρέχων κόμβος δεν βρίσκει κάποιο κατάλληλο γειτονικό κόμβο να προωθήσει την αίτηση, έτσι ώστε αυτό να ήταν μια πλευρική προώθηση, και τώρα θα πρέπει η διαδικασία να ανέβει

ένα επίπεδο. Παρόλα αυτά στη δεύτερη περίπτωση, ο κόμβος θα πρέπει να είναι φύλλο και άρα είναι αρκετά μακριά από τη ρίζα.

Μέχρι εδώ, έχει συζητηθεί η αναζήτηση ερωτήσεων που αφορούν σημεία και επιπλέον χωρίς αλληλεπικαλυπτόμενες περιοχές. Ωστόσο, τα περισσότερα πολυδιάστατα σχήματα ευρετηρίου επιτρέπουν περιοχές που σχετίζονται με κόμβους ευρετηρίου να επικαλύπτονται. Εξαιτίας αυτού, οι ερωτήσεις θα πρέπει να προωθούνται σε περισσότερους από έναν κόμβους. Και σε αυτή τη περίπτωση χρησιμοποιείται ο ίδιος μηχανισμός. Επειδή η αναζήτηση ενός σημείου μπορεί να θεωρηθεί ειδική περίπτωση μιας αναζήτησης εύρους όπου η ακτίνα εύρους της ερώτησης είναι 0, προχωρούμε κατευθείαν στις ερωτήσεις εύρους.

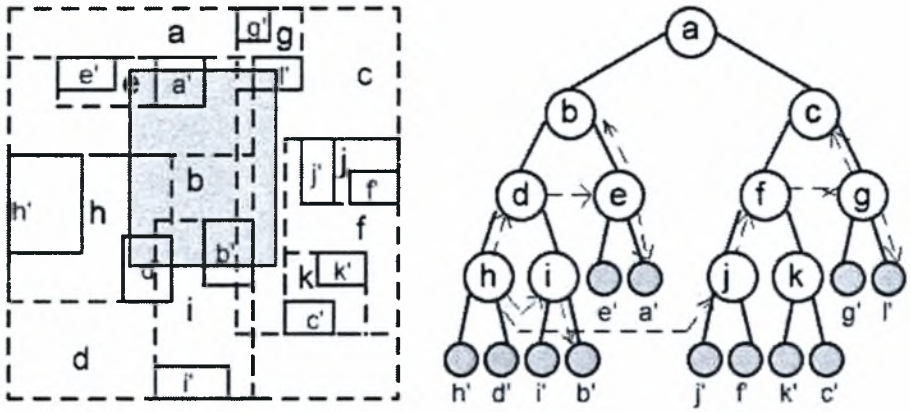
2.3.7: Ερωτήσεις με εύρος τιμών

Στο τμήμα αυτό περιγράφεται ένας γενικός αλγόριθμος για την επεξεργασία ερωτήσεων εύρους. Ακολουθεί λοιπόν ο αλγόριθμος 2.3.4. Προφανώς, τώρα κανείς ψάχνει κόμβους με περιοχές που συμπληρώνουν το διάστημα της ερώτησης, παρά μόνο για έναν, που να περιέχει το σημείο της ερώτησης, όπως γινόταν πριν. Καθώς, πολλοί κόμβοι μαζί μπορεί να συνεισφέρουν ώστε να καλυφθεί το διάστημα της ερώτησης, η ερώτηση μπορεί να χρειαστεί να προωθηθεί σε πολλούς κόμβους.

```
Algorithm 2.3.4: RangeQuery(node n, search-region r, nearest-checked-ancestor a)
if ( Region(n) ∩r != ∅ )
  {
    LocalSearch(n, r)
    if ( Region(LeftChild(n)) ∩r != ∅
       AND the search request is not sent from LeftChild(n) )
      { RangeQuery(LeftChild(n), r, LeftChild(n)) }
    if ( Region(RightChild(n)) ∩r != ∅
       AND the search request is not sent from RightChild(n) )
      { RangeQuery(RightChild(n), r, RightChild(n)) }
  }
else
    for i=0 to Level(n) – Level(a) -1
      {
        x = UpsidePath(n).get(i)
       if ( Region(x) ∩r != ∅)
        {
          if (x = = a )
            if ( No Region(Children(x) covers the whole r) )
            { LocalSearch(x, p) } //finding discrete data
          else
            {
              y = get a neighbor node in other side of the tree rooted at x }
              if ( y != null )
                { RangeQuery(y, r, x) }
              else
                { RangeQuery(Parent(n), r, x) }
            }
        }
```

Για παράδειγμα, θεωρούμε ότι σε ένα R-δέντρο για δεδομένα δύο διαστάσεων, ο κόμβος h θέλει να ψάξει για ένα εύρος τιμών όπως φαίνεται στην εικόνα 2.19. Καταρχήν, ο h εκτελεί την ερώτηση τοπικά καθώς η περιοχή η οποία καλύπτει τέμνεται με την περιοχή της ερώτησης. Έπειτα, προσπαθεί να προωθήσει την ερώτηση σε άλλους κόμβους. Από

τη στιγμή που ο d' είναι παιδί του h και η περιοχή του τέμνεται με την περιοχή αναζήτησης, ο h του προωθεί την ερώτηση.



Πηγή: H.V.Jagadish [3]

Εικόνα 2.19: Διαδικασία αναζήτησης με ερώτηση εύρους στον κόμβο h.

Επειδή όλοι οι πρόγονοι του h τέμνουν την περιοχή αναζήτησης, ο h προωθεί την ερώτηση στους i και j, οι οποίοι είναι γειτονικοί κόμβοι του h και βρίσκονται στην άλλη πλευρά του υποδέντρου με ρίζες αντίστοιχα τον κόμβο d και a. Ο h προωθεί την αίτηση και στον πατέρα του d επειδή δε μπορεί να βρει ένα γειτονικό κόμβο, ο οποίος να βρίσκεται στην άλλη πλευρά του υποδέντρου και να έχει ρίζα τον b. Από εκεί, η αίτηση προωθείται από τον d στον e, από τον j στον g μέσω του f. Τελικά από τους κόμβους προορισμού i, e και j, η ερώτηση προωθείται στους κόμβους δεδομένων b', r' και l' καθώς και στους κόμβους δρομολόγησης b και c για «ξεχωριστά» δεδομένα.

Ας θεωρήσουμε έναν κόμβο χ, στον οποίο ανατίθεται η αναζήτηση μιας περιοχής με εύρος r και έναν κόμβο, έστω y, του οποίου η περιοχή τέμνεται με την περιοχή αναζήτησης. Αν ο y είναι απόγονος του χ, η αίτηση αναζήτησης προωθείται πάντα προς τα κάτω μέχρι να φτάσει στον y. Αν ο y είναι πρόγονος του χ, η αίτηση προωθείται πρώτα σε έναν γειτονικό κόμβο του χ τον z, ο οποίος βρίσκεται στην άλλη πλευρά του δέντρου με ρίζα τον y. Έπειτα, ο z ελέγχει και διαπιστώνει ότι κανένα παιδί του y δεν καλύπτει ολόκληρη την περιοχή r. Γι' αυτό ο z προωθεί την αίτηση στον y μέσω των συνδέσεων πατέρα-παιδιού. Στην τελευταία περίπτωση, αν ο y δεν είναι απόγονος ούτε πρόγονος του χ, ο y θα πρέπει να πάει σε ένα υποδέντρο με ρίζα τον z, ο οποίος είναι πρόγονος του χ και η περιοχή του θα πρέπει να τέμνει την περιοχή r. Ως αποτέλεσμα, η

αίτηση προωθείται σε έναν γειτονικό κόμβο t, ο οποίος βρίσκεται στην άλλη πλευρά του δέντρου με ρίζα τον κόμβο z. Η διαδικασία αυτή συνεχίζεται μέχρι ο y να είναι είτε πρόγονος είτε απόγονος του κόμβου που λαμβάνει την αίτηση αναζήτησης. Ως συνέπεια, ο αλγόριθμος εγγυάται να προωθήσει την αίτηση αναζήτησης σε όλους τους κόμβους που τέμνουν την περιοχή αναζήτησης. Παρόλο που ο αλγόριθμος αυτός είναι λίγο διαφορετικός από τον αλγόριθμο για τις ερωτήσεις ακριβείας χωρίς επικαλυπτόμενες περιοχές, γιατί σε κάθε βήμα, μια αίτηση αναζήτησης μπορεί να χρειαστεί να προωθηθεί σε πολλούς κόμβους, το κόστος του αλγορίθμου αναζήτησης διατηρείται ακόμη σε επίπεδο O(logN), αφού οι ερωτήσεις προωθούνται παράλληλα.

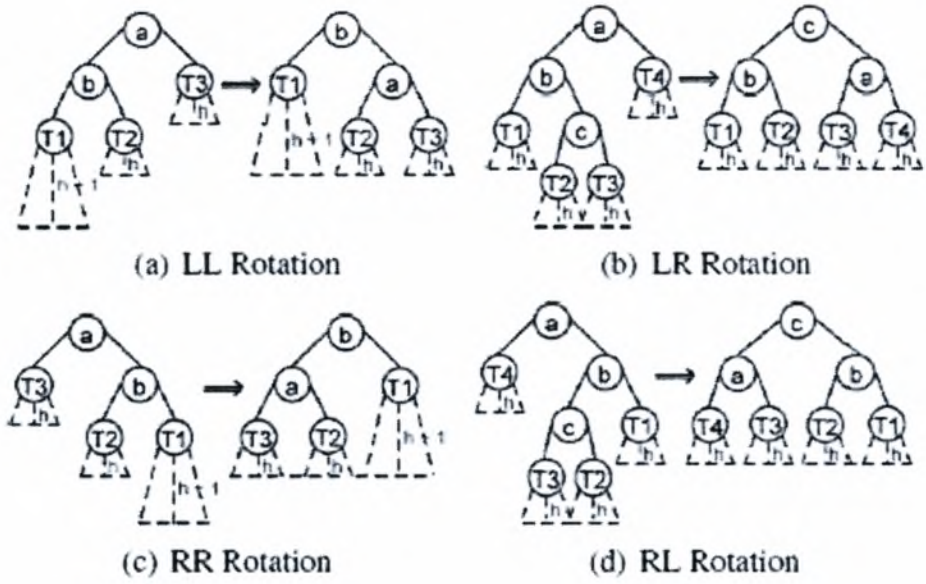
2.3.8: Εξισορρόπηση φόρτου

Υπάρχουν δύο σχήματα εξισορρόπησης φόρτου στη δομή του VBI-δέντρου. Το πρώτο σχήμα είναι απλό, ο υπερφορτωμένος κόμβος προσπαθεί να μοιραστεί το φορτίο του με τα παιδιά του, αν είναι εσωτερικός κόμβος δρομολόγησης, ή με τον αδερφό του αν είναι κόμβος δεδομένων. Από τη στιγμή, που αυτό το σχήμα δεν επαρκεί να αντιμετωπίσει πολύ ανόμοια κατανεμημένα σύνολα δεδομένων, προτείνεται το δεύτερο σχήμα. Στο δεύτερο σχήμα λοιπόν, ένας εσωτερικός κόμβος δρομολόγησης μοιράζεται το φορτίο μόνο με τα παιδιά του. Ωστόσο, αν πρόκειται για έναν κόμβο δεδομένων που είναι φύλλο, προσπαθεί να μοιραστεί το φορτίο του, σε πρώτη φάση, με τον αδερφό του. Αν ο αδερφός του είναι κι αυτός υπερφορτωμένος, τότε προσπαθεί να βρει έναν ελαφρύτερα φορτωμένο κόμβο δεδομένων μέσω των γειτονικών του συνδέσεων όπως και στο BATON. Ο ελαφρά φορτωμένος κόμβος αφήνει την τρέχουσα θέση του και επανεισάγεται στο δέντρο, ως παιδί του υπερφορτωμένου κόμβου, για να μοιραστεί το φορτίο του. Το σχήμα αυτό μπορεί γρήγορα να αποκαταστήσει καθολικές δυσλειτουργίες που υπήρχαν με το φόρτο των κόμβων, αλλά μπορεί να προκαλέσει ανισορροπίες στο δέντρο. Η διαδικασία αναδόμησης δικτύου εφαρμόζεται έπειτα για επανέλθει η ισορροπία στο δέντρο.

2.3.9: Αναδόμηση δικτύου

Όταν ένας κόμβος λάβει ένα μήνυμα από το παιδί του, να ενημερώσει το ύψος του, γιατί έχει αλλάξει, συγκρίνοντας το με το ύψος του άλλου παιδιού, τότε μπορεί να καταλάβει

αν το υποδέντρο του είναι ακόμη ισορροπημένο. Αν το σύστημα έρθει σε ανισορροπία, η διαδικασία αναδόμησης δικτύου ξεκινά από τον κόμβο που ανιχνεύει την ανισορροπία. Η διαδικασία αυτή στηρίζεται στη χρήση πράξεων περιστροφής όπως συμβαίνει και στα AVL δέντρα. Υπάρχουν τέσσερις τρόποι για επαναφορά της ισορροπίας όπως φαίνεται και στην εικόνα 2.20.



Πηγή: H.V.Jagadish [3]

Εικόνα 2.20: Απεικόνιση περιστροφών κατά τη διαδικασία αναδόμησης του δικτύου.

Στην ΑΑ και στην ΔΔ περιστροφή, δύο κόμβοι απαιτείται να υπολογίσουν ξανά τις περιοχές που καλύπτουν. Όπως φαίνεται στην εικόνα 2.20a και 2.20c, πρώτον ο κόμβος a χρειάζεται να υπολογίσει την ελάχιστη περιοχή που καλύπτουν το T2 και το T3. Έπειτα, ο b επανυπολογίζει την ελάχιστη περιοχή που καλύπτουν ο T1 και ο a. Δε χρειάζεται καμία μετακίνηση δεδομένων εκτός κι αν υπάρχουν «ξεχωριστά» δεδομένα αποθηκευμένα στους κόμβους a και b. Παρομοίως, στην ΑΔ και στην ΔΑ περιστροφή, όπως φαίνονται και στις εικόνες 2.20b και 2.20d, πρώτον οι κόμβοι b και a υπολογίζουν ξανά την ελάχιστη περιοχή που καλύπτουν οι T1 και T2, και οι T3 και T4. Έπειτα, ο c υπολογίζει κι αυτός την ελάχιστη περιοχή που καλύπτουν οι a και b. Τελικά, τα «ξεχωριστά» δεδομένα που είναι αποθηκευμένα στους κόμβους a, b και c τοποθετούνται στις θέσεις τους κατάλληλα. Από τη στιγμή που αλλάζουν οι περιοχές κάλυψης, θα πρέπει να ενημερωθούν οι «άνω-πλευρικοί» πίνακες των απογόνων. Όπως και στην

περίπτωση της επέκτασης της περιοχής κάλυψης, όταν είχαμε μια εισαγωγή ενός νέου σταθμού, η «χαλαρή» ενημέρωση μπορεί να χρησιμοποιηθεί κι εδώ. Για την αναδόμηση ενός δικτύου, όπου εμπλέκονται n κόμβοι, απαιτούνται n μηνύματα για το κομμάτι της ενημέρωσης. Επιπρόσθετα, κάθε κόμβος θα πρέπει να ενημερώσει τους νέους γειτονικούς κόμβους για τις αλλαγές, διαδικασία που κοστίζει O(logN). Ως αποτέλεσμα, η αναδόμηση δικτύου στην οποία εμπλέκονται n κόμβοι απαιτεί $O(n \times \log N + n)$ κόστος. Όσο περισσότεροι κόμβοι συμμετέχουν στη διαδικασία αναδόμησης, τόσο μεγαλύτερο και το κόστος της. Ωστόσο, η πιθανότητα της διαδικασίας περιστροφής όπου εμπλέκονται n κόμβοι μειώνεται εκθετικά ως προς το n. Το amortized κόστος μιας ενημέρωσης μπορεί να αποδειχθεί ότι είναι O(logN).

Κεφάλαιο 3: Γράφοι Αναπηδήσεως

Οι γράφοι αναπηδήσεως(skip graphs) [4] είναι μια κατανεμημένη δομή, βασισμένη στις λίστες αναπηδήσεως. Η δομή αυτή παρέχει την πλήρη λειτουργικότητα ενός ισοζυγισμένου δέντρου, σε ένα κατανεμημένο σύστημα, όπου οι πόροι αποθηκεύονται σε διαφορετικούς κόμβους, οι οποίοι μπορεί να «πέσουν» ανά πάσα στιγμή. Επίσης, παρέχοντας τη δυνατότητα υποβολής ερωτήσεων, βασισμένες στη διάταξη κλειδιών, υπερέχουν των δομών DHT, οι οποίες προσφέρουν, μόνο, τη λειτουργικότητα ενός πίνακα κατακερματισμού. Οι γράφοι αναπηδήσεως, αντίθετα από τις λίστες αναπηδήσεως και άλλες δενδρικές δομές, παρέχουν σε υψηλό βαθμό ανθεκτικότητα καθώς η συνδεσιμότητα δεν χάνεται, ακόμα κι αν ένα μεγάλος μέρος των κόμβων «πέσει». Επιπρόσθετα, η κατασκευή, η εισαγωγή, η διαγραφή, η αναζήτηση κόμβων καθώς και πράξεις όπως η ανίχνευση και διόρθωση λαθών στη δομή δεδομένων μπορούν να υλοποιηθούν με απλούς και άμεσους αλγορίθμους.

Ένας γράφος αναπηδήσεως είναι γενίκευση μιας λίστας αναπηδήσεως και γι' αυτό πριν περάσουμε στην ανάλυση του, θα περιγράψουμε με συντομία τη λίστα αναπηδήσεως.

3.1: Λίστα αναπηδήσεως

Οι λίστες αναπηδήσεως προτάθηκαν από τον Pugh το 1990 [19]. Η ιδέα προήλθε από την εξής διαίσθηση: Στα ευρετήρια, τα οποία είναι υλοποιημένα ως μια απλή λίστα, μια αναζήτηση είναι δυνατόν να προκαλέσει την εξέταση κάθε κόμβου της λίστας. Εάν, σε κάθε δεύτερο κόμβο της λίστας, προσθέταμε έναν επιπλέον δείκτη προς τον κόμβο που βρίσκεται δύο θέσεις πιο μπροστά, τότε, στη χειρότερη περίπτωση, θα εξετάζαμε $\lceil n/2 \rceil + 1$ στοιχεία, δηλαδή τα μισά περίπου σε σχέση με πριν. Αυτό ισοδυναμεί με τη δημιουργία μιας δεύτερης λίστας, μεγαλύτερου επιπέδου, με τα μισά στοιχεία, όπου κάθε κόμβος γνωρίζει το στιγμιότυπό του, στη λίστα χαμηλότερου επιπέδου, δια μέσου κατάλληλου δείκτη. Συνεχίζοντας έτσι, αν σε κάθε τέταρτο κόμβο της αρχικής λίστας ξεκινά ένας δείκτη, ο οποίος δείχνει προς κόμβο, ο οποίος βρίσκεται τέσσερις θέσεις πιο

μπροστά, τότε το χειρότερο κόστος πέφτει σε $\lceil n/4 \rceil + 2$ και γενικεύοντας, εάν, κάθε 2^i – στός κόμβος της αρχικής λίστας διαθέτει δείκτη προς κόμβο 2^i θέσεις μπροστά τότε,

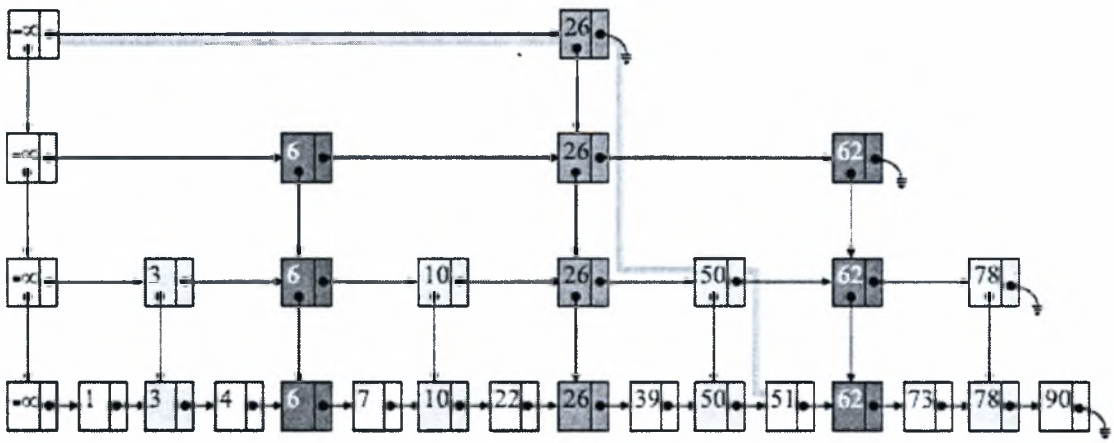
a) το κόστος αναζητήσεως μειώνεται σε λογαριθμικό

b) το μέγεθος της δομής απλώς διπλασιάζεται και επομένως παραμένει γραμμικό στο πλήθος των αποθηκευμένων στοιχείων, καθώς:

$$\sum_{i=0}^{\log n} \frac{n}{2^i} = n \sum_{i=0}^{\log n} \frac{1}{2^i} = O(n),$$

λόγω της προσεκτικής «αντιγραφής».

Στην εικόνα 3.1 φαίνεται ένα παράδειγμα μιας λίστας αναπηδήσεως. Η αναζήτηση του 51 προκαλεί τη σύγκριση με το 26 της κορυφαίας λίστας που βρίσκεται στο επίπεδο 3(το υψηλότερο επίπεδο). Καθώς το 26 είναι το μεγαλύτερο κλειδί και δεν υπάρχει άλλο, κατεβαίνουμε στον κόμβο 26 του επιπέδου 2. Εφόσον το 51 είναι μικρότερο του 62 που είναι το επόμενο κλειδί δεξιά του 26, κατεβαίνουμε στο 26 του επιπέδου 1. Το επόμενο προς τα δεξιά κλειδί είναι το 50, το οποίο είναι μικρότερο του 51 οπότε κατεβαίνουμε στο 50 του επιπέδου 0. Το επόμενο κλειδί δεξιά του 50 είναι το 51 που είναι και το ζητούμενο κλειδί. Για βοηθητικούς λόγους, ο κόμβος κεφαλή της λίστας θεωρείται πως αντιστοιχεί στο -∞, οπότε και σε κλειδί μικρότερο από κάθε μέλος του υπό συζήτηση σύμπαντος.



Πηγή: Π.Μποζάνης, Δομές Δεδομένων [15]

Εικόνα 3.1 Η αναζήτηση του στοιχείου με κλειδί 51 σε μια λίστα αναπηδήσεως

Η παραπάνω θεώρηση, ναι μεν επιταχύνει τη διαδικασία αναζητήσεως, αλλά είναι άκαμπτη ως προς τις μεταβολές του υποκείμενου συνόλου. Διαπιστώνεται εύκολα πως μια ένθεση ή απόσβεση στοιχείου προκαλεί δραματικές, γραμμικές στο μέγεθος της δομής, αλλαγές. Η απαιτούμενη ευελιξία επιτυγχάνεται με την ενσωμάτωση της τυχαιότητας στη διαδικασία αντιγραφής ενός στοιχείου και αυτό γίνεται ως εξής: Για κάθε στοιχείο που είναι να εισαχθεί, εισάγουμε το κλειδί του στη σωστή θέση στο επίπεδο 0 και «στρίβουμε» ένα νόμισμα. Αν το αποτέλεσμα είναι κεφαλή, συνεχίζουμε με την εισαγωγή του κλειδιού στο επίπεδο 2 και «στρίβουμε» ξανά το νόμισμα. Όσο το αποτέλεσμα επιμένει να είναι κεφαλή ,συνεχίζουμε να ενθέτουμε το κλειδί σε επόμενα επίπεδα μέχρι είτε να φέρουμε γράμματα είτε να έχουμε φτάσει στο τελευταίο επίπεδο και να έχουμε φέρει πάλι κεφαλή. Και στις δύο περιπτώσεις, η διαδικασία σταματάει. Συγκεκριμένα στη δεύτερη περίπτωση αυξάνουμε το επίπεδο της δομής κατά ένα και προσθέτουμε μια επιπλέον λίστα με μόνα στοιχεία την κεφαλή(-∞) και το νεοεισαχθέν κλειδί.

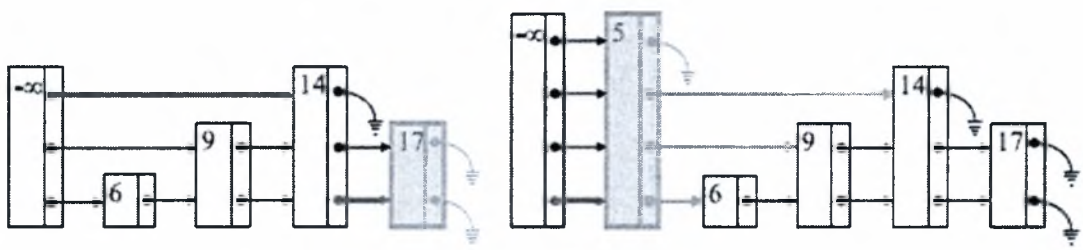
Καθώς η αντιγραφή ενός κλειδιού σε πολλαπλά επίπεδα είναι εξαιρετικά σπάταλη, ο Pugh πρότεινε την υλοποίηση του ενός κόμβου ανά στοιχείο s, ο οποίος θα διαθέτει πολλαπλούς δεξιούς δείκτες, έναν για κάθε επίπεδο συμμετοχής του s. Έτσι, εξαλείφεται και η ανάγκη για «κάτω» δείκτες, προς τα αντίγραφα του ιδίου κόμβου. Ακολουθεί η περιγραφή των βασικών πράξεων της τυχαίας αυτής δομής ευρετηρίου.

3.1.1: Αναζήτηση στοιχείου στη λίστα αναπηδήσεως

Η ανωτέρω θεώρηση ελάχιστα αλλάζει τη διαδικασία αναζητήσεως που παρουσιάστηκε παραπάνω. Ξεκινώντας από το ανώτερο επίπεδο, όσο συναντώνται στοιχεία με κλειδί μικρότερο από αυτό που ψάχνουμε, ακολουθούμε δεξιούς δείκτες του τρέχοντος επιπέδου i. Όταν «κολλήσουμε» σε έναν κόμβο u στο τρέχον επίπεδο, τότε είτε έχουμε εντοπίσει τον κόμβο u, του οποίου ο ακόλουθος φέρει στοιχείο με το εν λόγω κλειδί, είτε η αναζήτηση συνεχίζεται με το δεξιό δείκτη του u του αμέσως μικρότερου επιπέδου i-1. Η διαδικασία θα σταματήσει, είτε στον κόμβο που στεγάζει στοιχείο με το αναζητούμενο κλειδί, εφόσον, βέβαια, υπάρχει τέτοιο στη δομή, είτε προ του αριστερότερου κόμβου, επιπέδου 0, με κλειδί μεγαλύτερο από αυτό που ψάχνουμε.

3.1.2: Εισαγωγή στοιχείου στη λίστα αναπηδήσεως

Η διαδικασία ενθέσεως ενός στοιχείου με κλειδί χ είναι εξαιρετικά απλή. Εκτελούμε μια αναζήτηση με το χ, ώστε να εντοπιστεί η θέση του σε όλα τα επίπεδα, εφόσον δεν υπάρχει ήδη στη δομή. Κατόπιν, αποφασίζεται το πλήθος l των επιπέδων που θα στεγάσουν το χ, «στρίβοντας» ένα νόμισμα, μέχρι να έρθουν γράμματα. Εάν το l αποδειχθεί μεγαλύτερο από το τρέχοντα μέγιστο αριθμό επιπέδου $level$, τότε θέτουμε $l = level + 1$. Στη συνέχεια, παρεμβάλλουμε τον νέο κόμβο στα πρώτα l επίπεδα, στις θέσεις που έχει εντοπίσει η διαδικασία αναζητήσεως. Στην εικόνα 3.2 φαίνεται η μετασχηματισμένη δομή της λίστας ,όπου έχουμε εισάγει δεξιούς δείκτες προς κόμβους σε κάθε επίπεδο, καθώς και ένα παράδειγμα εισαγωγής των στοιχείων με κλειδιά 17 και 5. Παρατηρούμε, ότι στην περίπτωση εισαγωγής του 5, το νόμισμα έφερε διαδοχικά περισσότερες φορές κεφαλή, σε σχέση με το ύψος της δομής, γι' αυτό και το τρέχων μέγιστο επίπεδο αυξάνεται κατά ένα και ενώ πριν την εισαγωγή του 5, η δομή είχε 3 επίπεδα, μετά την εισαγωγή έχει 4.



Πηγή: Π. Μποζάνης, Δομές Δεδομένων [15]

Εικόνα 3.2 Εισαγωγή των στοιχείων με κλειδιά 17 και 5 στη λίστα αναπηδήσεως

3.1.3: Διαγραφή στοιχείου από τη λίστα αναπηδήσεως

Η διαδικασία αποσβέσεως του στοιχείου με κλειδί χ δρα αντιστρόφως προς αυτήν της ενθέσεως. Προηγείται μια αναζήτηση, προκειμένου να εντοπιστεί ο κόμβος που στεγάζει το εν λόγω στοιχείο. Κατόπιν, αφαιρείται από όλα τα επίπεδα όπου αυτός ανήκει.

Το μέγεθος της δομής είναι γραμμικό ως προς το πλήθος των στοιχείων, δηλαδή O(n) όπου n το πλήθος των στοιχείων, και το ύψος της, δηλαδή το πλήθος των επιπέδων, είναι O(logn) με μεγάλη πιθανότητα. Τέλος, οι πράξεις της αναζήτησης, της ένθεσης και της απόσβεσης στοιχείου κοστίζουν σε χρόνο O(logn).

3.1.4: Από τις λίστες αναπηδήσεως, στους γράφους αναπηδήσεως

Καθένας από τους n κόμβους, σε ένα γράφο αναπηδήσεως, είναι μέλος πολλαπλών συνδεδεμένων λιστών. Η λίστα επιπέδου 0 αποτελείται από όλους τους κόμβους σε αύξουσα σειρά. Εκεί που διαφέρει ένας γράφος από μια λίστα αναπηδήσεως είναι ότι σε ένα επίπεδο, μπορεί να υπάρχουν πολλές λίστες και κάθε κόμβος να συμμετέχει σε μια από αυτές, μέχρι που οι λίστες καταλήγουν να αποτελούνται από έναν μοναδικό κόμβο(singletons) μετά από O(logn) επίπεδα κατά μέσο όρο. Σε έναν γράφο αναπηδήσεως μπορούν να εφαρμοστούν οι πράξεις της αναζήτησης, εισαγωγής και διαγραφής ανάλογες με αυτές των λιστών αναπηδήσεως. Επίσης όπως αποδεικνύεται και παρακάτω, οι αλγόριθμοι για τις λίστες αναπηδήσεως μπορούν να εφαρμοστούν κατευθείαν και στους γράφους καθώς ένας γράφος αναπηδήσεως ισοδυναμεί με μια συλλογή από n λίστες αναπηδήσεως που τυχαίνει να μοιράζονται κάποια από τα κατώτερά τους επίπεδα.

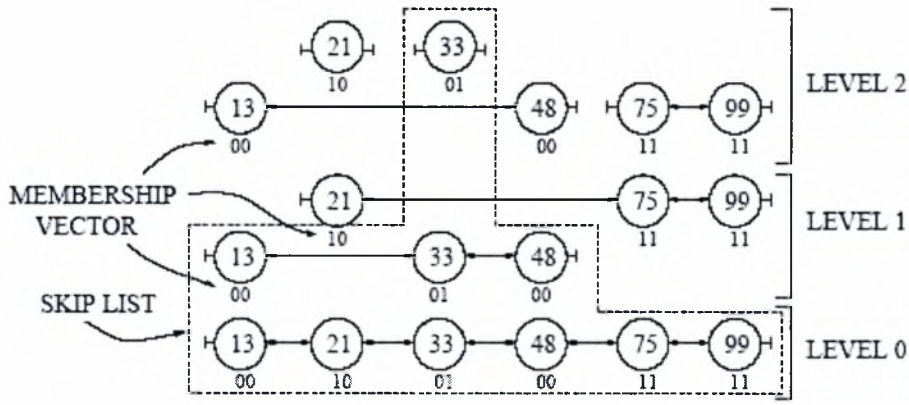
3.2: Ακριβής ορισμός του γράφου αναπηδήσεως

Ακολουθούν μερικοί χρήσιμοι ορισμοί και θεωρήσεις για τον πληρέστερο ορισμό της δομής. Θεωρούμε:

- Σ ένα πεπερασμένο αλφάβητο
- Σ* το πλήθος τω πεπερασμένων λέξεων αποτελούμενων από τα γράμματα του αλφάβητου Σ
- Σ^w το πλήθος άπειρων λέξεων του Σ
- w μια λέξη η οποία αποτελείται από χαρακτήρες $w_0w_1w_2w_3\ldots$ του Σ
- |w| το μέγεθος της λέξης w
- w_i τον i-στο χαρακτήρα της λέξης w
- $w_1 \wedge w_2$ το κοινό πρόθεμα των λέξεων w_1 και w_2
- το σύμβολο ε για την κενή λέξη
- Αν $w \in \Sigma^w$, τότε $|w| \rightarrow \infty$ αλλιώς το |w| είναι πεπερασμένος αριθμός
- Αν ισχύει $|w| \geq i$, τότε ο συμβολισμός $w\uparrow i$ είναι για το πρόθεμα της λέξης w μήκους i

- Αν v και w είναι δύο λέξεις, τότε θα γράφουμε $v \prec w$ αν η λέξη v είναι πρόθεμα της λέξης w, δηλαδή αν ισχύει $w \uparrow |v| = v$

Επιστρέφοντας τώρα στους γράφους αναπηδήσεως, κάθε κόμβος x της δομής έχει ένα διάνυσμα μελών (membership vector), το m(x). Το m(x) είναι μια τυχαία λέξη η οποία αποτελείται από, θεωρητικά, άπειρους χαρακτήρες του αλφάβητου Σ αλλά, πρακτικά, χρειάζεται να αποτελείται κατά μέσο όρο από O(logn) τυχαίους χαρακτήρες του Σ. Επίσης, κάθε κόμβος x ανήκει σε κάποιες λίστες, οι οποίες καθορίζονται από το διάνυσμα μελών m(x). Πιο συγκεκριμένα, κάθε συνδεδεμένη λίστα στο γράφο έχει ως ετικέτα μια πεπερασμένη λέξη w και κάθε κόμβος x ανήκει στη λίστα με ετικέτα w, αν και μόνο αν, το w είναι πρόθεμα του m(x).



Πηγή: "Skip Graphs"[4]

Εικόνα 3.3 Παράδειγμα ενός γράφου αναπηδήσεως

Το κατώτερο επίπεδο του γράφου είναι μια διπλά συνδεδεμένη λίστα S_ε, αποτελούμενη από όλους τους κόμβους του γράφου. Γενικά, για κάθε λέξη w του Σ*, η διπλά συνδεδεμένη λίστα S_w ανήκει στο επίπεδο i, αν |w| = i. Έτσι, προκύπτει μια άπειρη οικογένεια διπλά συνδεδεμένων λιστών από τις οποίες όμως, για πρακτικούς λόγους, αναπαρίστανται στη δομή μόνο αυτές που αποτελούνται από τουλάχιστον δύο κόμβους.
Ένας γράφος αναπηδήσεως είναι ακριβώς μια οικογένεια από διπλά συνδεδεμένες λίστες { S_w } κι επειδή τα διανύσματα μελών είναι τυχαίες μεταβλητές, κάθε λίστα S_w είναι κι αυτή μια τυχαία μεταβλητή. Ένα παράδειγμα γράφου αναπηδήσεως φαίνεται στην εικόνα 3.3.

Μπορούμε επίσης, να φανταστούμε το γράφο αναπηδήσεως σαν έναν τυχαίο γράφο, όπου υπάρχει μια ακμή μεταξύ x και y, οποτεδήποτε τα x και y είναι γειτονικά σε κάποια λίστα S_w. Ορίζουμε τον αριστερό και δεξιό γείτονα του x στο επίπεδο i, ως τον άμεσο πρόγονο και απόγονο του x, αντίστοιχα, στην $S_{m(x)\uparrow i}$, ή αλλιώς το $\perp$, αν δεν υπάρχουν τέτοιοι κόμβοι. Οπότε, θα γράφουμε xL_i για τον αριστερό γείτονα του x και xR_i για το δεξιό του γείτονα στο επίπεδο i. Γενικά, θα θεωρήσουμε ότι το R_i ανήκει σε μια οικογένεια σχεσιακών, εκθετικών τελεστών για να μπορούμε να σχηματίζουμε εκφράσεις του τύπου $xR_iR^2_{i-1}\ldots$ Επίσης, συμβολίζουμε με x.maxLevel το πρώτο επίπεδο l, στο οποίο η λίστα αποτελείται μόνο από τον κόμβο x. Τέλος, να σημειωθεί ότι, για έναν κόμβο x με διάνυσμα μέλους m(x), η λίστα αναπηδήσεως $S_{m(x)}$ ονομάζεται περιοριστική λίστα αναπηδήσεως του x.

Ένας εναλλακτικός τρόπος να δούμε το γράφο αναπηδήσεως είναι σαν ένα trie από λίστες αναπηδήσεως οι οποίες μοιράζονται τα κατώτερά τους επίπεδα. Αν σκεφτούμε τη λίστα αναπηδήσεως ως μια ακολουθία τυχαίων μεταβλητών $S_0, S_1, S_2, \ldots$, όπου με S_i εννοούμε τη λίστα επιπέδου i, τότε προκύπτει το παρακάτω λήμμα.

Λήμμα 1: Έστω { S_w } ένα γράφος αναπηδήσεως με αλφάβητο Σ. Για κάθε $z \in \Sigma^w$ η ακολουθία $S_0, S_1, S_2, \ldots$, όπου κάθε $S_i = S_{z\uparrow i}$, είναι μια λίστα αναπηδήσεως με παράμετρο $p = |\Sigma|^{-1}$.

Απόδειξη: Με επαγωγή ως προς i. Η λίστα S_0 ισοδυναμεί με την λίστα S_ε, η οποία όπως αναφέρθηκε και παραπάνω είναι η διπλά συνδεδεμένη λίστα του χαμηλότερου επιπέδου που περιέχει όλους τους κόμβους του γράφου. Ένας κόμβος x ανήκει στην S_i αν $m(x)\uparrow i = z \uparrow i$, με την προϋπόθεση να συμβαίνει το εξής γεγονός: Η πιθανότητα με την οποία ο x ανήκει στην S_{i+1} είναι η πιθανότητα με την οποία ισχύει $m(x)_{i+1} = z_{i+1}$. Το γεγονός αυτό συμβαίνει με πιθανότητα $p = |\Sigma|^{-1}$ και είναι εύκολο να διαπιστώσει κανείς ότι είναι ανεξάρτητο από αντίστοιχο γεγονός για κάθε άλλο x' το οποίο ανήκει στην S_i. Γι αυτό, κάθε κόμβος που ανήκει στην S_i ανήκει και στην S_{i+1} με ανεξάρτητη πιθανότητα p, και οι $S_0, S_1, S_2, \ldots$ σχηματίζουν μια λίστα αναπηδήσεως.

3.3: Θέματα υλοποίησης

Κατά την υλοποίηση ενός P2P συστήματος με χρήση γράφου αναπηδήσεως κάθε κόμβος στον γράφο αναπηδήσεως αντιστοιχεί και σε έναν πόρο άρα και σε ένα κλειδί. Οι πόροι διατάσσονται σε αύξουσα λεξικογραφική σειρά με βάση την τιμή των κλειδιών τους. Η αντιστοίχηση των κλειδιών στους πραγματικούς σταθμούς του συστήματος μπορεί να γίνει με δύο τρόπους

1. Κάθε σταθμός είναι υπεύθυνος για τους πόρους που διαθέτει
2. Χρήση μιας μεθόδου κατανεμημένου κατακερματισμού για να αποφασιστεί ποιοι κόμβοι είναι υπεύθυνοι για ποια κλειδιά.

Η πρώτη προσέγγιση επιτυγχάνει ασφάλεια και καλή διαχείριση ενώ η δεύτερη, καλή εξισορρόπηση φόρτου. Για αρχή, θεωρούμε ότι οι κόμβοι σε έναν γράφο αναπηδήσεως αντιστοιχούν σε πόρους του συστήματος, χωρίς να μας ενδιαφέρει ο τρόπος που οι πόροι κατανέμονται στους σταθμούς του συστήματος. Η πληροφορία που αποθηκεύει κάθε κόμβος σε έναν γράφο αναπηδήσεως είναι

- το κλειδί του πόρου που είναι υπεύθυνος
- το διάνυσμα μελών m
- τις διευθύνσεις και τα αναγνωριστικά του αριστερού (neighbor[L][level]) και δεξιού του γείτονα (neighbor[R][level]) σε κάθε επίπεδο
- το maxLevel όπως εξηγήθηκε παραπάνω

Αν επιτρέψουμε την ταυτόχρονη εκτέλεση πράξεων, τότε υποθέτουμε ότι οι μεταβλητές δείκτες διατηρούνται από μια κατανεμημένη, διατεταγμένη, διπλά-συνδεδεμένη λίστα, υλοποιημένη έτσι ώστε να λύνει το πρόβλημα των αδιεξόδων. Υποθέτουμε ότι οι πράξεις της αναζήτησης, της εισαγωγής και διαγραφής είναι ατομικές και κοστίζουν O(κ) σε χρόνο και μηνύματα όπου κ είναι ο αριθμός βημάτων από τον κόμβο που ξεκινά η πράξη μέχρι τον κόμβο στόχο. Επίσης, υποθέτουμε ότι οι πράξεις σε διαφορετικές λίστες μπορούν να εφαρμοστούν παράλληλα χωρίς να προκύπτουν προβλήματα συνέπειας.

3.4: Η διαδικασία αναζήτησης στοιχείου (Ερωτήσεις Ακριβείας)

Η διαδικασία της αναζήτησης σε ένα γράφο αναπηδήσεως είναι παρόμοια με αυτή ων λιστών αναπηδήσεως, με μερικές προσαρμογές, για να μπορέσει να εφαρμοστεί σε ένα

κατανεμημένο περιβάλλον. Η αναζήτηση ξεκινά στο μεγαλύτερο επίπεδο που εμφανίζεται ο κόμβος, στον οποίο ανατίθεται η αναζήτηση ενός κλειδιού και συνεχίζει κατά μήκος του επιπέδου, μέχρι να εντοπιστεί το κλειδί, οπότε σταματάει η διαδικασία, ή να φτάσουμε σε κόμβο με μεγαλύτερο κλειδί από αυτό που ψάχνουμε. Στη δεύτερη περίπτωση, η διαδικασία συνεχίζεται με τον ίδιο τρόπο σε κατώτερα επίπεδα, μέχρι να φτάσουμε στο επίπεδο 0, όπου είτε επιστρέφεται ο κόμβος με το ζητούμενο κλειδί είτε ο κόμβος με το μέγιστο κλειδί, που είναι όμως μικρότερο από το ζητούμενο κλειδί. Ο αλγόριθμος 3.1 περιγράφει τη διαδικασία όπου ένας κόμβος (ο startNode) ζητά από τον κόμβο v, να του επιστρέψει τον κόμβο με κλειδί searchKey.

```
Algorithm 3.1: search-for node v
// startNode asks node v to look for the node with the searchKey
upon receiving⟨ searchOp, startNode, searchKey, level ⟩
 if ( v.key = = searchKey )
    send⟨foundOp, v⟩ to startNode
 if ( v.key < searchKey )
     while ( level ≥ 0 )
          if ( v.neighbor[R][level].key < searchKey )
            {
              send⟨searchOp, startNode, search key, level⟩ to v.neighbor[R][level]
              break
            }
          else
             { level = level - 1 }
else
     while ( level ≥ 0 )
          if ( v.neighbor[L][level].key > searchKey )
            {
              send⟨searchOp, startNode, search key, level⟩ to v.neighbor[L][level]
              break
            }
          else
             { level = level - 1 }
if ( level < 0 )
    send ⟨ notFoundOp, v ⟩ to startNode
```

Το λήμμα 2 αποδεικνύει ότι το κόστος αναζήτησης είναι λογαριθμικό στο πλήθος των κόμβων.

Λήμμα 2: Η διαδικασία της αναζήτησης σε έναν γράφο αναπηδήσεως S με n κόμβους διαρκεί O(logn) και χρειάζεται να σταλούν O(logn) μηνύματα.

Απόδειξη: Έστω Σ το αλφάβητο για τα διανύσματα μέλη (membership vectors) των κόμβων στο γράφο αναπηδήσεως S και z ο κόμβος από τον οποίο ξεκινά η αναζήτηση. Από το λήμμα 1, η ακολουθία $S_m(z) = S_0, S_1, S_2, \ldots$,όπου κάθε $S_i = S_z|i$ είναι μια λίστα αναπηδήσεως επιπέδου i . Μια αναζήτηση, η οποία ξεκινά από τον κόμβο z, θα ακολουθήσει το ίδιο μονοπάτι στον S όπως και στην $S_m(z)$. Άρα, μπορούμε να χρησιμοποιήσουμε κατευθείαν την ανάλυση που γίνεται στη διαδικασία αναζήτησης σε μια λίστα αναπηδήσεως, για να αναλύσουμε την ίδια διαδικασία στον S.

Με n κόμβους θα υπάρχουν κατά μέσο όρο $O\left(\log n \frac{1}{\log\left(\frac{1}{p}\right)}\right)$ επίπεδα, για $p = |\Sigma|^{-1}$. Το πολύ $\frac{1}{1-p}$ κόμβοι διαπερνώνται σε κάθε επίπεδο, οπότε αναμένεται να μεταδοθούν συνολικά $O\left(\log n \frac{1}{(1-p)\cdot\log\left(\frac{1}{p}\right)}\right)$ μηνύματα στο δίκτυο και η όλη διαδικασία να διαρκέσει $O\left(\log n \frac{1}{(1-p)\cdot\log\left(\frac{1}{p}\right)}\right)$ χρόνο. Έτσι με σταθερό p, η διαδικασία αναζήτησης κοστίζει O(logn)σε χρόνο και O(logn) μηνύματα.

Η απόδοση του δικτύου εξαρτάται από την τιμή του $p = |\Sigma|^{-1}$. Όσο το p αυξάνει, ο χρόνος αναζήτησης μειώνεται, αλλά ο αριθμός των επιπέδων αυξάνεται. Έτσι, κάθε

κόμβος πρέπει να διατηρεί γείτονες σε περισσότερα επίπεδα. Άρα, υπάρχει ένα trade-off ανάμεσα στο χρόνο αναζήτησης και τις χωρικές απαιτήσεις σε κάθε κόμβο.

Η απόδοση που περιγράφηκε στο λήμμα 2 είναι συγκρίσιμη με την απόδοση των DHTs, όπως για παράδειγμα το Chord. Με n πόρους στο σύστημα, ένας γράφος αναπηδήσεως χρειάζεται O(logn) χρόνο, για μια αναζήτηση, ενώ το Chord O(logm) χρόνο, όπου m ο αριθμός των σταθμών στο σύστημα. Επειδή το n είναι πολυωνυμικό ως προς το m παίρνουμε την ίδια ασυμπτωτική απόδοση και στις δύο προσεγγίσεις (DHTs και Skip Graphs) για τη διαδικασία της αναζήτησης.

3.5: Εισαγωγή νέου κόμβου στο δίκτυο

Για να εισέλθει ένας νέος κόμβος *u* στο δίκτυο, θα πρέπει να γνωρίζει έναν κόμβο *v* που είναι ήδη στο δίκτυο. Ο κόμβος εγγράφει τον εαυτό του σε κάθε συνδεδεμένη λίστα σε κάθε επίπεδο, ξεκινώντας από το χαμηλότερο επίπεδο, μέχρι να φτάσει σε ένα επίπεδο και να ανακαλύψει ότι βρίσκεται σε μια λίστα, όπου ο ίδιος είναι το μοναδικό στοιχείο της λίστας. Η διαδικασία της ενθέσεως αποτελείται από δύο βήματα:

1. Ο κόμβος u ξεκινά μια αναζήτηση από τον κόμβο v (introducer) με κλειδί το u.key, για να ανακαλύψει τον αριστερό του γείτονα στο επίπεδο 0, και εγγράφεται στη λίστα επιπέδου 0.
2. Ο κόμβος u βρίσκει τους πιο «κοντινούς» του κόμβους s και y σε κάθε επίπεδο $l \geq 0$, όπου $s < u < y$, έτσι ώστε να ισχύει $m(u) \uparrow (l+1) = m(s) \uparrow (l+1) = m(y) \uparrow (l+1)$. Αν οι κόμβοι αυτοί υπάρχουν συνδέεται ανάμεσα σε αυτούς στο επίπεδο $l+1$. Δεν είναι απαραίτητο να υπάρχουν και οι δύο «κοντινοί» κόμβοι, και μόνο ένας να υπάρχει ο u θα συνδεθεί σε αυτόν.

Επειδή κάθε υπάρχων κόμβος v δεν χρειάζεται άμεσα την τιμή $m(v)_{l+1}$, εκτός κι αν υπάρχει άλλος κόμβος u, τέτοιος ώστε $m(v) \uparrow (l+1) = m(u) \uparrow (l+1)$, μπορεί να καθυστερήσει να διαμορφώσει την τιμή του διανύσματος μελών, μέχρι να ξεκινήσει η διαδικασία εισαγωγής ενός νέου κόμβου, ο οποίος να πρέπει να μάθει την τιμή του v. Γι'

αυτό, σε κάθε δεδομένη χρονική στιγμή, μόνο ένα πεπερασμένο πρόθεμα της τιμής του διανύσματος μελών χρειάζεται να παράγεται. Ο ψευδοκώδικας για την διαδικασία αναζήτησης φαίνεται στον αλγόριθμο 3.2.

Algorithm 3.2: insert node u
execute search from introducer to find max s < u
$l \leftarrow 0$
while (true)
{
insert v in list at level l starting from s
scan backwards at level l to find s' such that $m(s') \uparrow (l+1) = m(u) \uparrow (l+1)$
if (no such s' exists)
{ exit }
else
{
$s \leftarrow s'$
$l \leftarrow l+1$
}
}

Προχωράμε στο λήμμα 3 όπου αποδεικνύεται το λογαριθμικό κόστος σε χρόνο και μηνύματα της διαδικασίας ενθέσεως.

Λήμμα 3: Με την προϋπόθεση ότι δεν επιτρέπονται ταυτόχρονες πράξεις, η διαδικασία εισαγωγής ενός νέου κόμβου σε έναν γράφο αναπηδήσεως S με n κόμβους διαρκεί O(logn) και χρειάζεται να σταλούν O(logn) μηνύματα.

Απόδειξη: Έστω Σ το αλφάβητο, θεωρούμε p = |Σ|. Με n κόμβους, θα υπάρχουν κατά μέσο όρο O(logn) επίπεδα στο γράφο αναπηδήσεως. Για να συνδεθεί ένας νέος κόμβος u στο επίπεδο 0, εκτελεί μια φορά τη διαδικασία αναζήτησης η οποία όπως ξέρουμε από το λήμμα 1 κοστίζει O(logn) σε χρόνο και μηνύματα. Σε κάθε επίπεδο l, όπου $l > 0$, ο u επικοινωνεί κατά μέσο όρο με 1/p κόμβους πριν βρει τον πρόγονό του s' στο επίπεδο $l+1$. Η διαδικασία αυτή απαιτεί O(1/p) = O(1) χρόνο και μηνύματα. Από εκεί και έπειτα η διαδικασία εισαγωγής θεωρείται ότι κοστίζει σταθερό χρόνο O(1).

Οπότε συνοψίζοντας για όλα τα επίπεδα και λαμβάνοντας υπόψη και το κόστος της αρχικής αναζήτησης καταλήγουμε στο συμπέρασμα ότι η διαδικασία εισαγωγής κοστίζει O(logn) σε χρόνο και μηνύματα.

Αν επιτρέψουμε ταυτόχρονες εισαγωγές κόμβων, το κόστος μιας συγκεκριμένης εισαγωγής μπορεί να γίνει αυθαίρετα μεγάλο. Επιπρόσθετα, με τα κόστη που υπολογίζονται για την υλοποίηση της διπλά-συνδεδεμένης λίστας, που αναφέρθηκε πιο πάνω, υπάρχει μια πιθανότητα λιμοκτονίας στο σημείο, που ο νέος κόμβος ψάχνει για κοντινότερους γείτονες σε μεγαλύτερα επίπεδα, καθώς νέοι κόμβοι μπορεί να ενθέτονται πιο γρήγορα από ότι ο νέος κόμβος μπορεί να τους ανακαλύψει κατά τη διαδικασία της δικής του εισαγωγής.

3.6: Αποχώρηση κόμβου από το δίκτυο

Algorithm 3.3: delete existing node u
for $l \leftarrow 1$ to u.maxLevel in parallel do
{
  delete u from list at level l
  $l \leftarrow l+1$
}
delete u from list at level 0

Η διαδικασία της αποχώρησης είναι πολύ απλή. Όταν ένας κόμβος *u*, επιθυμεί να αποχωρήσει από το δίκτυο, ενημερώνει τον πρόγονό του σε κάθε επίπεδο(εκτός από το επίπεδο 0) να αλλάξει το δείκτη του προηγούμενου του κόμβου και να τον κάνει να δείξει στον επόμενο από τον πρόγονο του *u*, κόμβο. Το ίδιο συμβαίνει και για τον επόμενο κόμβο από τον πρόγονο του *u*. Αλλάζει ο δείκτης του, που έδειχνε στον πρόγονο του *u* και τώρα θα δείχνει στο προηγούμενο κόμβο από τον πρόγονο του *u*. Όταν τελειώσει η διαγραφή του *u* από όλα τα ανώτερα επίπεδα, τότε προχωράμε και στην απόσβεσή του από το επίπεδο 0. Ο αλγόριθμος που περιγράφει τη διαδικασία αποχώρησης του κόμβου *u* είναι ο 3.3.

Ακολουθεί το λήμμα 4 όπου αποδεικνύεται το κόστος της διαδικασίας αποχώρησης του κόμβου u.

Λήμμα 4: Με την προϋπόθεση ότι δεν επιτρέπονται ταυτόχρονες πράξεις, η διαδικασία αποχώρησης ενός κόμβου από έναν γράφο αναπηδήσεως S με n κόμβους διαρκεί O(1) και χρειάζεται να σταλούν O(logn) μηνύματα.

Απόδειξη: Υποθέτουμε ότι κάθε διαδικασία διαγραφής από την κάθε συνδεδεμένη λίστα κοστίζει O(1) σε χρόνο και μηνύματα. Από τη στιγμή που όλες, εκτός από μια, διαγραφές σε κάθε επίπεδο εκτελούνται ταυτόχρονα , ο συνολικός χρόνος είναι O(1) ενώ ο συνολικός αριθμός μηνυμάτων O(logn) αφού πρέπει να σταλούν μηνύματα σε καθένα από τα O(logn) επίπεδα.

Αν επιτρέψουμε ταυτόχρονες πράξεις, τότε το κόστος εξαρτάται από το κόστος διαγραφής της υλοποίησης της κατανεμημένης λίστας, που αναφέρθηκε στην παράγραφο 3.3.

3.7: Ερωτήσεις εύρους

Οι ερωτήσεις εύρους υλοποιούνται εύκολα και αποτελεσματικά σε έναν γράφο αναπηδήσεως καθώς υπάρχει διάταξη των κλειδιών. Η υλοποίηση είναι πολλή απλή αρκεί να γίνουν μικρές αλλαγές στον αλγόριθμο αναζήτησης 3.1. Εκτελούμε τον αλγόριθμο αναζήτησης 3.1 για την κατώτερη τιμή v1 του εύρους τιμών [v1, v2] το οποίο αναζητούμε. Από τη στιγμή που φτάσουμε στον αντίστοιχο κόμβο στη βασική λίστα του επιπέδου 0, προχωράμε μέσω δεικτών προς τα δεξιά επιστρέφοντας τα αποτελέσματα, μέχρι να συναντήσουμε κλειδί με τιμή μεγαλύτερη από v2, οπότε και η διαδικασία ολοκληρώνεται. Το κόστος αναζήτησης ενός διαστήματος τιμών είναι O(logN) μέχρι να εντοπιστεί το αριστερότερο άκρο του διαστήματος και επιπλέον 2r, για τους r κόμβους, όπου οι τιμές τους βρίσκονται μέσα στο διάστημα τιμών που αναζητείται(r μηνύματα προς τους κόμβους και r απαντήσεις στον κόμβο που ξεκίνησε την αναζήτηση). Οπότε, μια ερώτηση εύρους κοστίζει O(logN + 2r) σε χρόνο και μηνύματα.

3.8: Ανοχή Σφαλμάτων

Όπως και σε κάθε σύστημα έτσι κι εδώ, ένας αριθμός κόμβων του γράφου θα αστοχήσει. Μας ενδιαφέρει λοιπόν, ο αριθμός των κόμβων που μπορεί να αποκοπούν από τη δομή, εξαιτίας της αστοχίας άλλων κόμβων. Πολλοί κόμβοι σχετίζονται με έναν μόνο σταθμό,

αφού ένας σταθμός μπορεί να είναι υπεύθυνος για περισσότερους από έναν πόρους. Όταν λοιπόν, ένας σταθμός τίθεται εκτός λειτουργίας, όλοι οι κόμβοι που σχετίζονται με αυτόν, αυτόματα «εξαφανίζονται». Τα αποτελέσματα υπολογίζονται με βάση τον αριθμό των κόμβων που «πέφτουν». Αν οι κόμβοι είναι περίπου ισοκατανεμημένοι στου σταθμούς τότε ένας ανάλογος αριθμός σταθμών «πέφτει». Παρόλα αυτά, θα ήταν χρήσιμο να έχουμε μια καλύτερη άποψη για την ανοχή σφαλμάτων, όταν λαμβάνεται υπόψη και ο τρόπος που κατανέμονται οι κόμβοι στους σταθμούς. Το γεγονός αυτό μπορεί να βελτιώσει κατά πολύ την ανοχή σφαλμάτων, καθώς οι κόμβοι που είναι αποθηκευμένοι στους σταθμούς μπορούν πάντα να εντοπίζουν άλλους κόμβους που βρίσκονται στον ίδιο σταθμό.

Θεωρούμε δύο μοντέλα βλαβών: Ένα τυχαίο μοντέλο βλαβών, όπου τυχαία κόμβοι επιλέγονται να αστοχήσουν και ένα μοντέλο χειρότερης περίπτωσης ,όπου επιλέγονται να αστοχήσουν κόμβοι σε στρατηγικά σημεία για τη δομή.

- Για το μοντέλο τυχαίων βλαβών, τα πειραματικά αποτελέσματα δείχνουν πως για έναν δικαιολογημένα μεγάλο γράφο αναπηδήσεως, σχεδόν όλοι οι κόμβοι παραμένουν στην αρχική σύνθεση, μέχρι περίπου τα 2/3 των κόμβων να αποτύχουν και αυτό είναι δυνατό να κάνει τη διαδικασία της αναζήτησης πολύ ανθεκτική σε βλάβες, ακόμα κι αν δε γίνει χρήση του μηχανισμού ανάκαμψης, με τη χρήση πλεοναζουσών συνδέσεων.
- Για το μοντέλο χειρότερης περίπτωσης, τα θεωρητικά αποτελέσματα δείχνουν ότι ακόμα και στη χειρότερη περίπτωση, όπου συγκεκριμένοι κόμβοι επιλέγονται να «πέσουν», η ζημιά είναι περιορισμένη. Με υψηλή πιθανότητα, ένας γράφος αναπηδήσεως με n κόμβους έχει έναν ρυθμό επέκτασης της τάξης του $\Omega\left(\frac{1}{\log n}\right)$, από τον οποίο συμπεραίνουμε, ότι το πολύ $O(f \cdot \log n)$ κόμβοι μπορούν να αποκοπούν από την αρχική δομή με f αποτυχίες. Δε δίνονται πειραματικά αποτελέσματα για εσκεμμένες εχθρικές βλάβες, καθώς τα πειράματα μπορεί να μην είναι ικανά να ανιχνεύσουν το μοντέλο χειρότερης περίπτωσης.

3.9: Μηχανισμός Ανάκαμψης

Στην προηγούμενη παράγραφο, είδαμε ότι αστοχίες κόμβων δεν είναι πιθανό να αποσυνδέσουν το γράφο αναπηδήσεως, αν αυτό όμως συμβεί, προκύπτει το βραχυπρόθεσμο πρόβλημα της εκτέλεσης αναζητήσεων σε κατεστραμμένο γράφο και το μακροπρόθεσμο πρόβλημα της ανασύστασης των χαμένων συνδέσεων, έτσι ώστε η δομή που προκύπτει, να αποτελεί πάλι έναν γράφο αναπηδήσεως. Το καλύτερο θα ήταν, ένας μηχανισμός ανάκαμψης, ο οποίος ανασυνθέτει έναν κατεστραμμένο γράφο αναπηδήσεως από τα «επιζώντα» του κομμάτια. Σε προηγούμενη έκδοση των γράφων αναπηδήσεως, είχε περιγραφεί από τους J.Aspnes και G.Shah λεπτομερώς, ένας μηχανισμός για την ανάκαμψη του γράφου ταυτόχρονα με ενθέσεις και αποσβέσεις κόμβων. Η χειρότερη περίπτωση επιδιόρθωσης με βάση τον μηχανισμό, ήταν γραμμική στο μέγεθος του γράφου και μπορούσε κάτω από ορισμένες συνθήκες να προκαλέσει μόνιμη απόσχιση κόμβων από το γράφο αλλά και να αποτύχει να συγκλίνει. Ευτυχώς, προέκυψαν καλύτερες μέθοδοι. Η πιο απλή είναι μια γενική προσέγγιση, στην οποία ο γράφος αναπηδήσεως, περιοδικά, χτίζεται πάλι από την αρχή με όλους τους κόμβους να μεταναστεύουν στο νέο γράφο αναπηδήσεως, με μιας, από τη στιγμή που αναδομείται. Αυτό μπορεί να γίνει είτε με το να υπάρχει κάποιος κόμβος αρχικοποίησης, ο οποίος συγκεντρώνει τους εναπομείναντες κόμβους, χρησιμοποιώντας τον κλασσικό αλγόριθμο εισαγωγής είτε με κάποιον γρηγορότερο παράλληλο μηχανισμό, ο οποίος έχει προταθεί [18]. Ο μηχανισμός αυτός μπορεί, καταρχήν, να ανακατασκευάσει ένα γράφο αναπηδήσεως από ένα αυθαίρετο επιζών κομμάτι σε χρόνο $O(\log^2 n)$, υπό εύλογες υποθέσεις σχετικά με το μέγεθος του κλειδιού.

3.10: Εγγυήσεις σε περιπτώσεις συμφόρησης

Επιπρόσθετα με την ανοχή σφαλμάτων, ένας γράφος αναπηδήσεως παρέχει μια περιορισμένη μορφή ελέγχου της συμφόρησης, με το να εξομαλύνει τα σημεία συμφόρησης που προκαλούνται από δημοφιλείς στόχους αναζήτησης. Οι εγγυήσεις που προσφέρει σε αυτή τη περίπτωση είναι παρόμοιες με τις εγγυήσεις που δίνονται για τη ανοχή σφαλμάτων. Όπως, το να συνεχίζει ένας κόμβος να παραμένει συνδεδεμένος στο γράφο εξαρτάται από τη βιωσιμότητα των γειτόνων του, ο φόρτος των μηνυμάτων που θα περάσουν από αυτόν εξαρτάται από την δημοτικότητα των γειτόνων ως στόχων

αναζήτησης. Ωστόσο, μπορούμε να δείξουμε ότι το φαινόμενο αυτό εξαλείφεται ραγδαία με την απόσταση, δηλαδή κόμβοι που βρίσκονται μακριά από έναν δημοφιλή στόχο στο κατώτερο επίπεδο της λίστας ενός γράφου αναπηδήσεως, παίρνουν λιγότερο αυξημένο φόρτο μηνυμάτων κατά μέσο όρο. Παρουσιάζονται δύο εκδόσεις του αποτελέσματος αυτού, η πρώτη είναι για τη μέση συμφόρηση μιας μόνο αναζήτησης και η δεύτερη για την κατανομή της μέσης συμφόρησης. Και από τις δύο εκδόσεις προκύπτει ότι, οι περισσότεροι γράφοι αναπηδήσεως χειρίζονται την συμφόρηση αποτελεσματικά. Οι αποδείξεις των παρακάτω θεωρημάτων και των λημμάτων έχουν παραλειφθεί, ωστόσο ο προσεκτικότερος αναγνώστης μπορεί να ανατρέξει στην πηγή [4].

Μέση συμφόρηση για μια μόνο αναζήτηση

Η πρώτη έκδοση δείχνει ότι η πιθανότητα μια συγκεκριμένη αναζήτηση να χρησιμοποιεί έναν κόμβο μεταξύ πηγής και προορισμού, μειώνεται αντίστροφα με την απόσταση του κόμβου από τον στόχο και πιο συγκεκριμένα, σύμφωνα με το παρακάτω θεώρημα,

Θεώρημα 5: Έστω ένας γράφος αναπηδήσεως S, όπου εκτελείται αναζήτηση με πηγή τον κόμβο *s* και κόμβο στόχο τον *t*. Έστω *u* ένας κόμβος για τον οποίο ισχύει $s < u < t$ ως προς τις τιμές των κλειδιών (η περίπτωση $s > u > t$ είναι συμμετρική), και έστω *d* η απόσταση από τον *u* στον *t*, ως ο αριθμός των κόμβων *v* για τους οποίους ισχύει $u < v \le t$. Τότε, η πιθανότητα μια αναζήτηση, η οποία ξεκινά από τον *s* και καταλήγει στον *t*, να περνά από τον κόμβο *u*, είναι μικρότερη από $\frac{2}{d+1}$.

Το επιχείρημα για την απόδειξη του θεωρήματος 5, βασίζεται στην παρατήρηση ότι ένας κόμβος εμφανίζεται στο μονοπάτι αναζήτησης σε μια λίστα αναπηδήσεως S, εάν είναι ανάμεσα στους ψηλότερους(με μεγάλο maxLevel) κόμβους που βρίσκονται μεταξύ αυτού και του κόμβου στόχου. Πιο συγκεκριμένα, στηρίζεται στην απόδειξη ενός μικρού τεχνικού λήμματος, το οποίο μετράει τον αναμενόμενο αριθμό τέτοιων «ψηλών» κόμβων. Θεωρούμε μια διαδικασία Markov $A_0 \supseteq A_1 \supseteq A_2 \ldots$ όπου A_0 είναι κάποιο όχι άδειο αρχικό σύνολο και κάθε στοιχείο του A_t εμφανίζεται στο A_{t+1} με ανεξάρτητη πιθανότητα ½. Έστω τ ο μεγαλύτερος δείκτης, όπου το A_τ δεν είναι άδειο. Τότε

αποδεικνύεται, ότι ο μέσος αριθμός κόμβων που απαρτίζουν το σύνολο A_τ, δηλαδή το $E[|A_\tau|]$, είναι μικρός ανεξάρτητα από το μέγεθος του αρχικού συνόλου A_0, και συγκεκριμένα $E[|A_\tau|] < 2$.

Το γεγονός ότι, η μέση συμφόρηση είναι αντιστρόφως ανάλογη με την απόσταση, δεν επιβεβαιώνεται απαραίτητα μόνο σε «βαριά» φορτωμένους κόμβους. Από τη στιγμή που η πιθανότητα υπολογίζεται κατά μέσο όρο από όλες τις επιλογές των διανυσμάτων μελών, μπορεί κάποιος «άτυχος» κόμβος να βρεθεί με ένα διάνυσμα μελών που να τον βάζει σχεδόν σε κάθε μονοπάτι αναζήτησης σε κάποιον πολύ δημοφιλή στόχο.

Κατανομή της μέσης συμφόρησης

Η δεύτερη έκδοση, δείχνει ότι τα όρια της μέσης περίπτωσης διατηρούνται με μεγάλη πιθανότητα. Ενώ είναι ακόμη πιθανό, ένας απίστευτα «άτυχος» κόμβος να διαπερνάται στις περισσότερες αναζητήσεις, μια τέτοια κατάσταση εμφανίζεται μόνο σε εξαιρετικά χαμηλών πιθανοτήτων επιλογές των διανυσμάτων μελών. Ορίζουμε τη μέση συμφόρηση L_{tu}, η οποία επιβάλλεται από μια αναζήτηση για τον κόμβο *t*, σε έναν κόμβο *u*, ως τη πιθανότητα μια *s*-*t* αναζήτηση να περάσει από τον *u* με υποθετικά διανύσματα μελών για όλους τους κόμβους στο διάστημα [*u*, *t*], όπου $s < u < t$, ή ισοδύναμα , $s > u > t$. Από τη στιγμή που στο διάστημα [*u*, *t*] δε βρίσκεται ο κόμβος *s*, υποθέτουμε ότι το *m*(s) επιλέγεται τυχαία. Προσεγγίζει δηλαδή, την κατάσταση ενός σταθερού γράφου αναπηδήσεως, όπου ένας συγκεκριμένος κόμβος *t* χρησιμοποιείται σε πολλές αναζητήσεις που μπορεί να περάσουν από τον κόμβο *u*, αλλά οι κόμβοι από τους οποίους ξεκινάνε οι αναζητήσεις, επιλέγονται τυχαία από τους υπόλοιπους κόμβους του γράφου.

Το θεώρημα 5 υπονοεί ότι η αναμενόμενη τιμή του L_{tu} δεν είναι παραπάνω από $\frac{2}{d+1}$.

Το θεώρημα 6 μας δείχνει ότι η κατανομή της μέσης συμφόρησης L_{tu} μειώνεται εκθετικά μέχρι αυτό το σημείο και πιο συγκεκριμένα,

Θεώρημα 6: Έστω ένας γράφος αναπηδήσεως S, και οι σταθεροί κόμβοι t και u για τους οποίους ισχύει $u < t$ και $|\{v : u < v \le t\}| = d$. Τότε, για κάθε ακέραιο $l \ge 0$ ισχύει $\Pr[L_{tu} > 2^{-l}] \le 2e^{-2^{-l}d}$.

3.11: Βελτιστοποίηση του γράφου όταν$k!=\log N$

Η αναζήτηση σε έναν γράφο αναπηδήσεως είναι αρκετά γρήγορη O(log$N$), όταν ισχύει k = log$N$, όπου k το πλήθος των δυαδικών ψηφίων που απαρτίζουν το διάνυσμα μελών. Η απόδοση του όμως πέφτει σημαντικά όταν k < log$N$. Επιπρόσθετα, όταν k > log$N$ η απόδοση της δρομολόγησης δε βελτιώνεται. Στην πραγματικότητα, συμπεριφέρεται ως γραμμική αναζήτηση για πολύ μικρά k, και δεν μπορεί καν να φτάσει το όριο της δυαδικής αναζήτησης όταν ισχύει k > log$N$, καθώς όταν έχουμε πολύ μεγάλο k, αυξάνεται πολύ ο αριθμός των λιστών ενώ τα στοιχεία τους ολοένα και λιγοστεύουν. Με άλλα λόγια, παρόλο που ο γράφος αναπηδήσεως ,όταν ισχύει k = log$N$, φαίνεται να είναι ο καλύτερος τρόπος να «κόψει κανείς δρόμο» μέσω των άλλων λιστών για να φτάσει στον επιθυμητό κόμβο, η κατασκευή του δεν τον βοηθά να αποδίδει το ίδιο για μικρές και μεγάλες τιμές του k.

Με βάση την παραπάνω παρατήρηση έχουν προταθεί και αποδειχθεί θεωρητικά [5] κάποιες βελτιστοποιήσεις για το σχήμα των γράφων αναπηδήσεως, όταν βρισκόμαστε σε μια από τις δύο περιπτώσεις που αναφέρθηκαν ακριβώς από πάνω. Πιο συγκεκριμένα προτείνονται:

- Ένα πιο αραιό σχήμα γράφου αναπηδήσεως στην περίπτωση που ισχύει k < log$N$. Έχουν προταθεί δύο προσεγγίσεις. Στην στατική προσέγγιση κάθε κόμβος είναι συνδεδεμένος σε ανώτερα επίπεδα, όχι με όλους τους κόμβους με τους οποίους έχει κοινό πρόθεμα, αλλά με αυτούς που έχει κοινό πρόθεμα συγκεκριμένων μηκών. Τα συγκεκριμένα μήκη προθέματος επιλέγονται με μια συγκεκριμένη διαδικασία. Στην τυχαία προσέγγιση, απλά τα μήκη προθέματος που θα χρησιμοποιηθούν επιλέγονται τυχαία από το σύνολο {1, 2, 3, …, log$N$}.

- Ένα πιο πυκνό σχήμα γράφου αναπηδήσεως στην περίπτωση που ισχύει $k > \log N$ όπου όμως ο αριθμός των επιπρόσθετων συνδέσεων είναι μικρός σε σύγκριση με τον πλήρη γράφο.

Κεφάλαιο 4: P-Grid

Το P-Grid [6] [7] είναι μια κλιμακούμενη δομή πρόσβασης δεδομένων, η οποία θα λέγαμε ότι, ανήκει στην κατηγορία των δομών κατανεμημένου κατακερματισμού (DHTs). Χτίζεται όμως από την αρχή, χρησιμοποιώντας μια συνάρτηση κατακερματισμού για τον υπολογισμό των κλειδιών, η οποία διατηρεί τη σχετική τους διάταξη και άρα μπορεί να προκύψει ανομοιόμορφη κατανομή. Η δομή P-Grid πιο συγκεκριμένα, στηρίζει την κατασκευή της σε ένα εικονικό δυαδικό δέντρο αναζήτησης (trie), με βάση το οποίο γίνεται η δρομολόγηση των μηνυμάτων στη διαδικασία αναζήτησης. Όταν εφαρμόζονται δεδομένα που δεν κατανέμονται ομοιόμορφα, τότε προκύπτει μη ισορροπημένο δέντρο αναζήτησης. Όμως τελικά φαίνεται πως τα μη ισορροπημένα δέντρα δεν αποτελούν «πανάκεια» για ένα κατανεμημένο P2P σύστημα, από τη στιγμή που η επικοινωνία θεωρείται ως το πιο κρίσιμο κόστος και η δομή κατασκευαστεί προσεκτικά. Εκτός από την απόδειξη της παραπάνω πρότασης αλλά και το κατά πόσο αυτή είναι εφαρμόσιμη, προτείνονται και οι απαραίτητοι κατανεμημένοι, τυχαιοποιημένοι αλγόριθμοι οι οποίοι επιτρέπουν την κατασκευή της δομής P-Grid με έναν αυτόνομο τρόπο, έτσι ώστε η δενδρική αυτή δομή να προσαρμόζεται δυναμικά στην κατανομή των δεδομένων.

Η παρατήρηση κλειδί για τα P2P συστήματα είναι πως κρίσιμο σημείο για την απόδοση μιας αναζήτησης, δεν είναι το βάθος του δέντρου αλλά ο αριθμός των μηνυμάτων που ανταλλάσσονται κατά τη διάρκεια της αναζήτησης. Ο ισχυρισμός, εδώ, είναι ότι δε χρειάζεται να δώσουμε και τόσο σημασία στο κόστος της διαπέρασης των μονοπατιών σε δέντρα, όπου οι πράξεις γίνονται τοπικά σε έναν σταθμό, αλλά αντίθετα να δώσουμε σημασία στον αριθμό μηνυμάτων που ανταλλάσσονται μεταξύ σταθμών. Σε περίπτωση που κάτι τέτοιο δεν αποδειχθεί αληθές, μπορούμε ακόμη να χρησιμοποιήσουμε κάποια τοπική δομή λεξικού για να επιταχύνουμε τις τοπικές πράξεις. Η παρατήρηση αυτή δείχνει επίσης ότι στη κατανεμημένη και γενικά αποκεντρωμένη διαχείριση δεδομένων θα πρέπει να ξανασκεφτούμε από την αρχή κάποιες από τις θεμελιώδεις υποθέσεις πάνω στη διαχείριση δεδομένων, οι οποίες έχουν εξελιχθεί κυρίως παράλληλα με την εξέλιξη των κεντρικοποιημένων βάσεων δεδομένων.

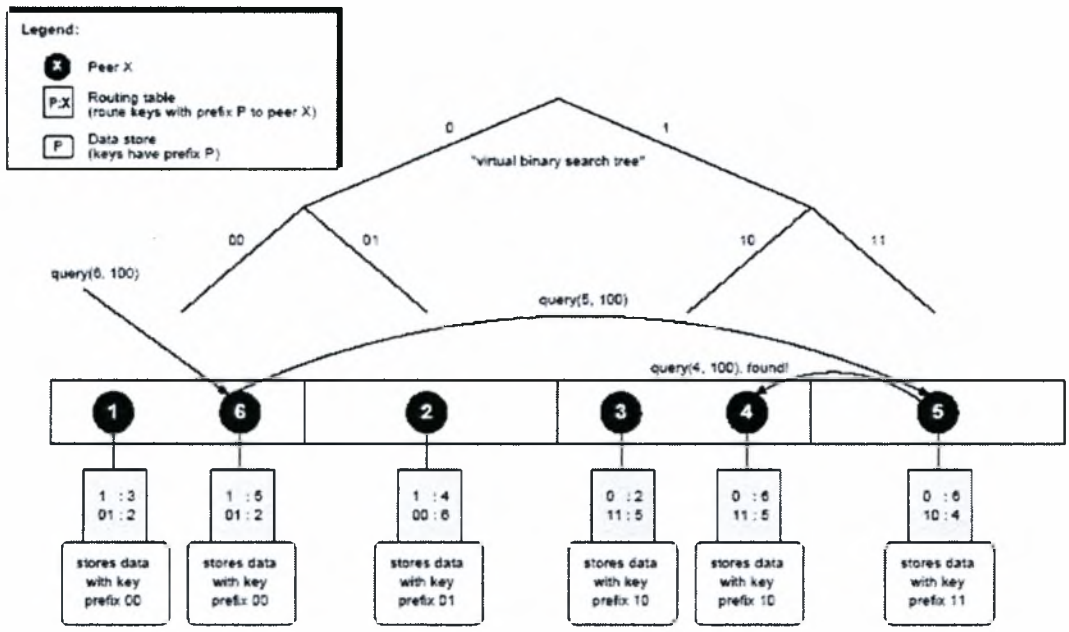
4.1: Περιγραφή της δομής πρόσβασηςP-Grid

Θεωρούμε ένα δεδομένο σύνολο από N σταθμούς $P = \{p_1,..., p_N\}$. Ένα αντικείμενο δεδομένων ή αλλιώς ένας πόρος $d \in D$ χαρακτηρίζεται από ένα δυαδικό κλειδί $key(d) = b_1,...,b_{kd}, b_i \in \{0,1\}$, όπου το μήκος k_d κλειδιού είναι σταθερό. Κάθε σταθμός $p \in P$ συσχετίζεται κι αυτός με ένα δυαδικό κλειδί το $key(p) = b_1,...,b_{kp}, b_i \in \{0,1\}, k_p \leq k_d$. Ένας σταθμός p αποθηκεύει τους πόρους d για τους οποίους ισχύει ότι το $key(p)$ είναι πρόθεμα του $key(d)$. Έτσι, κάθε σταθμός είναι υπεύθυνος για ένα μέρος μόνο των πόρων. Να σημειωθεί επίσης, πως το μήκος του κλειδιού k_p μπορεί να διαφέρει από σταθμό σε σταθμό. Για κάθε πρόθεμα $b_1..b_l$ του $key(p)$ με μήκος l, l = 1,…, k_p , ο σταθμός p διατηρεί έναν πίνακα δρομολόγησης. Ο πίνακας αυτός αποτελείται από μια λίστα αναφορών προς άλλους σταθμούς p', οι οποίοι έχουν το ίδιο πρόθεμα μήκους l αλλά διαφορετική τιμή στη θέση l+1. Θα ονομάσουμε τις αναφορές αυτές, αναφορές προς άλλους σταθμούς p'στο επίπεδο l και θα σημειώνονται ως $refs(p,l)$. Οι αναφορές αυτές χρησιμοποιούνται για τη δρομολόγηση ερωτήσεων αναζήτησης που αφορούν κλειδιά πόρων με πρόθεμα $b_1..b_l$, το οποίο στη θέση l+1 δεν ταιριάζει με το $key(p)$.

4.2 Αναζήτηση κλειδιού στη δομή πρόσβασης P-Grid

Από τη στιγμή που η δομή πρόσβασης κατασκευάζεται με βάση ένα δυαδικό δέντρο αναζήτησης, ο χώρος αναζήτησης είναι διαδοχικά χωρισμένος σε διαστήματα από τιμές κλειδιών. Αφού κάθε κλειδί αντιστοιχεί σε ένα μονοπάτι στο δυαδικό δέντρο αναζήτησης, μπορούμε επίσης να πούμε πως κάθε σταθμός είναι υπεύθυνος και για το μονοπάτι που του αντιστοιχεί με βάση το κλειδί του. Για να γίνει καλύτερα κατανοητή η δομή πρόσβασης P-Grid, παρατίθεται ένα παράδειγμα στην εικόνα 4.1. Η θέση στο εικονικό δυαδικό δέντρο αναζήτησης του σταθμού 4 στο παράδειγμα αυτό, δείχνει ότι είναι υπεύθυνος για όλους τους πόρους με πρόθεμα «10», οπότε και τους αποθηκεύει. Όπως φαίνεται και στην εικόνα , είναι δυνατό πολλοί σταθμοί να είναι υπεύθυνοι για το ίδιο μονοπάτι (όπως ο σταθμός 1 και ο 6). Το γεγονός αυτό βελτιώνει τη στιβαρότητα και

την ικανότητα ανταπόκρισης, όταν οι σταθμοί δε βρίσκονται online συνέχεια αλλά με μια συγκεκριμένη, πιθανά μικρή, πιθανότητα.



Πηγή: K.Aberer [7]

Εικόνα 4.1 Παράδειγμα της δομής πρόσβασης P-Grid

Μια αίτηση αναζήτησης μπορεί να ξεκινήσει σε οποιοδήποτε σταθμό και εκτελείται με τον εξής τρόπο: Ξεκινώντας από το πρώτο δυαδικό ψηφίο, το κλειδί αναζήτησης συγκρίνεται ψηφίο προς ψηφίο με τα κλειδιά των σταθμών. Όταν το τοπικό κλειδί ταιριάζει τότε συγκρίνεται το επόμενο ψηφίο, αν δεν ταιριάζει τότε η αίτηση προωθείται σε έναν από τους σταθμούς που βρίσκονται στον πίνακα δρομολόγησης του ερωτηθέντος σταθμού. Για παράδειγμα, μια ερώτηση για τον πόρο με το κλειδί «100» στέλνεται στο σταθμό 6. Ο 6 την προωθεί στον 5 στο πρώτο επίπεδο(καθώς δεν υπάρχει κοινό πρόθεμα μεταξύ του μονοπατιού που είναι υπεύθυνος ο 6 και του κλειδιού αναζήτησης, οπότε ο σταθμός 6 θα πρέπει να προωθήσει την αίτηση σε ένα σταθμό που γνωρίζει από τον πίνακα δρομολόγησης του και ο οποίος να βρίσκεται στο άλλο μισό του δέντρου, δηλαδή να έχει την τιμή 1 στη θέση του πρώτου ψηφίου). Με το που λαμβάνει στην αίτηση ο σταθμός 5 ο οποίος είναι υπεύθυνος για το πρόθεμα «11» διαπιστώνει ότι πρέπει να την προωθήσει στον σταθμό 4(αφού το μεγαλύτερο κοινό πρόθεμα του σταθμού 5 και του

κλειδιού είναι μήκους 1, το πρώτο ψηφίο. Ο 5 προωθεί την αίτηση σε έναν σταθμό , εδώ τον 4, ο οποίος βρίσκεται στο υπόλοιπο μισό του υποδέντρου στο επίπεδο 2.) Ο σταθμός 4, ο οποίος είναι υπεύθυνος για το πρόθεμα «10» , μπορεί τελικά να στείλει το περιεχόμενο στο σταθμό που ξεκίνησε τη διαδικασία, στο σταθμό 6. Να σημειωθεί ότι σε ένα πραγματικό περιβάλλον, πολλοί σταθμοί θα υπάρχουν που να ικανοποιούν τη συνθήκη του προθέματος (που να βρίσκονται στο άλλο μισό του δέντρου), οπότε επιλέγεται τυχαία ένας από αυτούς για να του προωθηθεί η αίτηση.

Η διαδικασία αναζήτησης περιγράφεται και σε μορφή ψευδοκώδικα στον αλγόριθμο 4.1.

Algorithm 4.1: Search in P-Grid: Retrieve(key, p)
if (key(p) ⊆key) // ⊆means is-prefix-of
{ return (d ∈data(p)| key(d) = key) }
else
{
determine *l* such that prefix(key, *l*) = $\overline{\text{prefix(p, } l)}$ //The *l* bit of prefix(p) is
// inverted
r = randomly selected element from refs(p, *l*)
Retrieve(key, r)
}

Από τη στιγμή που η δομή P-Grid βασίζεται σε δυαδικό δέντρο αναζήτησης , μια αίτηση αναζήτησης κοστίζει O(logN) σε μηνύματα σε ένα ισορροπημένο όμως δέντρο. Η μη ομοιόμορφη κατανομή δεδομένων μπορεί να προκαλέσει ανισορροπία στο δέντρο και άρα λογικά σκεπτόμενοι το κόστος θα πάψει πια να είναι λογαριθμικό σε αριθμό μηνυμάτων. Παρόλα αυτά, όπως θα αποδειχθεί και παρακάτω, εξαιτίας της τυχαίας επιλογής σταθμών, όταν αυτή είναι απαραίτητη, από τις λίστες αναφορών που διατηρεί κάθε σταθμός σε κάθε επίπεδο, το αναμενόμενο κόστος παραμένει λογαριθμικό ανεξάρτητα από το αν το δέντρο είναι ισορροπημένο ή όχι. Μια επιπλέον σημείωση για να εξηγήσουμε διαισθητικά πως κάτι τέτοιο μπορεί να συμβαίνει, είναι ότι, όταν αναλύεται ένα κλειδί κατά τη διαδικασία αναζήτησης, σπάνια προχωράμε ψηφίο-ψηφίο μέχρι την ολοκλήρωση της ανάλυσης, αλλά συνήθως προχωράμε ανά μεγαλύτερο αριθμό ψηφίων(οπότε απαιτούνται λιγότερα μηνύματα), άρα τελικά καταλήγουμε πάλι σε λογαριθμικό αριθμό μηνυμάτων. Με κάθε κλήση της Retrieve(key, p) το μήκος του

κοινού προθέματος του key(p) και του key αυξάνεται τουλάχιστον κατά ένα και άρα ο αλγόριθμος πάντα τερματίζει. Επίσης, αν η κατασκευή του P-Grid είναι πλήρης και τουλάχιστον ένας σταθμός για κάθε διαμέριση του χώρου κλειδιών είναι δυνατόν να προσπελαστεί(επιβεβαιώνεται από τις λίστες αναφορών και το ότι πολλαπλοί σταθμοί είναι υπεύθυνοι για κάθε μονοπάτι), τότε ο αλγόριθμος τερματίζει και επιτυχώς.

4.3 Εισαγωγή δεδομένων στη δομή πρόσβασηςP-Grid

Η διαδικασία εισαγωγής δεδομένων στο P-Grid βασίζεται στην πιο γενική λειτουργία της αναβάθμισης, η οποία προσφέρει πιθανοτικές εγγυήσεις συνέπειας και είναι αποτελεσματική ακόμα και σε περιβάλλοντα με πολύ μικρή αξιοπιστία ως προς τον αριθμό των αντιγράφων. Η διαδικασία της εισαγωγής δεδομένων εκτελείται σε δύο φάσεις: Πρώτον εντοπίζεται ένας σταθμός, ο οποίος είναι υπεύθυνος για το διάστημα του χώρου των κλειδιών που ανήκει το κλειδί του νέου πόρου, μέσω της διαδικασίας αναζήτησης (Retrieve(key)) και αφού εντοπιστεί, ενημερώνει τα αντίγραφά του για το νεοεισαχθέν κλειδί, χρησιμοποιώντας έναν ελαφρύ υβριδικό push-and-pull μηχανισμό gossiping.

4.4 Κατασκευή της δομής πρόσβασης

Έχει προταθεί ήδη [6] ένας αποτελεσματικός, τυχαιοποιημένος και κατανεμημένος αλγόριθμος για την κατασκευή μιας τέτοια δομής πρόσβασης βασισμένης σε τυχαίες αλληλεπιδράσεις μεταξύ των σταθμών. Η βασική ιδέα είναι ότι όποτε «συναντιούνται» δύο σταθμοί (συναντιούνται είτε ενεργά προκειμένου να κατασκευαστεί το P-Grid είτε ως συνέπεια άλλων πράξεων, όπως η ανίχνευση μεταξύ τους(pinging)) φροντίζουν να επιλέγουν κλειδιά προς αντίθετες κατευθύνσεις. Αν ένα καθολικό μέγιστο μήκος κλειδιού, έστω $k_{\max}$ δοθεί, τότε ο αλγόριθμος αυτός δρα σε μια ομοιόμορφη κατανομή των κλειδιών στους σταθμούς. Κάθε κλειδί θα σχετίζεται κατά μέσο όρο με $\frac{N}{2^{k\max}}$ σταθμούς. Από καθολικής άποψης, η προκύπτουσα δομή πρόσβασης αποτελείται από εξολοκλήρου ισορροπημένα δυαδικά δέντρα, όπου κάθε σταθμός υποστηρίζει την αναζήτηση κατά μήκος ενός μονοπατιού από τη ρίζα σε ένα φύλλο του δέντρου. Κατά

συνέπεια, ο συνολικός αριθμός των πόρων κλιμακώνει γραμμικά ως προς τον αριθμό των σταθμών και το κόστος αναζήτησης, ενώ κλιμακώνει λογαριθμικά σε χρόνο και αριθμό μηνυμάτων που παράγονται.

Με μη ομοιόμορφες κατανομές δεδομένων, όπως θα περίμενε κανείς σε ρεαλιστικές εφαρμογές, χρησιμοποιώντας ένα ισορροπημένο δυαδικό δέντρο για την κατανομή των δεδομένων και την αναζήτηση, θα είχε ως αποτέλεσμα έναν ανομοιόμορφο κατανεμημένο φόρτο στους σταθμούς. Γι' αυτό, γίνεται χρήση μιας λίγο διαφοροποιημένης έκδοσης του αλγορίθμου κατασκευής που είχε προταθεί [6]. Ο διαφοροποιημένος αυτός αλγόριθμος δρα σε μη ισορροπημένα δέντρα αναζήτησης. Υποθέτουμε ότι οι σταθμοί αποθηκεύουν ήδη κάποια δεδομένα τα οποία ανταποκρίνονται στην πραγματική κατανομή δεδομένων. Κατά τη διάρκεια της διαδικασίας κατασκευής, όποτε «συναντιούνται» σταθμοί, δεν επεκτείνουν μόνο τα αντίστοιχα κλειδιά τους αλλά ανταλλάσσουν και τα δεδομένα τους. Με τον τρόπο αυτό τα δεδομένα διασπείρονται καθολικά και οι σταθμοί χρησιμοποιούν την πληροφορία αυτή για να κατευθύνουν τη διαδικασία κατασκευής. Δε γίνεται πλέον χρήση της καθολικής παραμέτρου του μέγιστου επιτρεπόμενου μήκους κλειδιού $k_{\max}$ για τον έλεγχο του τερματισμού της διαδικασίας κατασκευής. Οι σταθμοί μπορούν να επεκτείνουν περαιτέρω τα κλειδιά τους, αφού τους γνωστοποιηθεί ένα μικρό μέρος από τα δεδομένα των κόμβων που συνάντησαν και το οποίο όμως μπορεί να δικαιολογήσει την περαιτέρω επέκταση του κλειδιού.

Ο αλγόριθμος 4.2 σε μορφή ψευδοκώδικα περιγράφει τη διαδικασία κατασκευής της δομής πρόσβασης και αποτελείται από τα εξής κυρίως βήματα:

- Ανταλλαγή αναφορών: Το γεγονός αυτό οδηγεί σε μια αυξανόμενη ομοιόμορφη κατανομή των αναφορών στις λίστες αναφορών διαφορετικών κόμβων. Να σημειωθεί ότι κατά τη διάρκεια της ανταλλαγής δε χάνεται καμία αναφορά.
- Ανταλλαγή δεδομένων: Το γεγονός αυτό οδηγεί σε μια αυξανόμενη κατανομή και αντιγραφή των δεδομένων μεταξύ των σταθμών. Το βήμα αυτό επιβεβαιώνει ότι τελικά όλοι οι πόροι αποθηκεύονται σε έναν σταθμό, ο οποίος είναι υπεύθυνος

για τους πόρους αυτούς, δηλαδή όπου το κλειδί του σταθμού αποτελεί πρόθεμα του κλειδιού του πόρου.

- Επέκταση του κλειδιού που σχετίζεται με έναν σταθμό: Το βήμα αυτό εκτελείται μόνο αν ο σταθμός μετά την ανταλλαγή διαθέτει έναν επαρκή αριθμό δεδομένων για να προχωρήσει σε επέκταση. Με αυτό τον τρόπο, τα μήκη των κλειδιών των σταθμών προσαρμόζονται ευέλικτα έτσι ώστε κάθε σταθμός να αποθηκεύει κατά μέσο όρο την ίδια ποσότητα δεδομένων.

Algorithm 4.2: Exchange Algorithm that two peers p1 and p2 perform in order to construct the P-Grid structure

exchange (p1, p2, r)

1: if (length(key(p1)) > length(key(p2)))
2: { swap p1 and p2 (make sure that p2 has the longer key) }
3: *lc* = length of common prefix of key(p1) and key(p2)
4: if (*lc* > 0)
5: { exchange references between *refs*(p1, *lc*) and *refs*(p2, *lc*) }
6: *l*1 = length(key(p1)) – *lc*
7: *l*2 = length(key(p2)) – *lc*
8: if (*l*1 > 0 and *l*2 > 0)
{
9: the peers select from each other data, stores the data that belong to their key
// peers may store data that does not correspond to their key
10: now forward the peer with the shorter path (which is p1) to another peer using *refs*(p2, *l*1)
}
11: if (*l*1 = 0 and *l*2 = 0)
{
12: k1 = key(p1) extended by a random bit
13: k2 = key(p2) extended by the inverse of the random bit
14: d1 = Select Data(data(p1) ∪data(p2), k1)
15: d2 = Select Data(data(p1) ∪data(p2), k2)
// Select Data(d, k) selects from the set d the data objects with a key of
// which k is prefix

```
16:     ld1 = number of data items in d1
17:     ld2 = number of data items in d2
        // now proceed differently depending on the number of data objects each
        // peer holds minstorage is the minimal number of data items required in
        // order to extend the key of a peer by an additional bit
18:     if ( ld1 > minstorage and ld2 <= minstorage )
        {
19:       key(p1) = k1
20:       update reference lists and exchange data objects
        }
21:     if ( ld1 <= minstorage and ld2 > minstorage )
        {
22:       key(p2) = k2
23:       update reference lists and exchange data objects
        }
24:     if ( ld1 > minstorage and ld2 > minstorage )
        {
25:       key(p1) = k1
26:       key(p2) = k2
27:       update reference lists and exchange data objects
        }
28:     if ( ld1 <= minstorage and ld2 <= minstorage )
        {
29:       data(p1) = data(p2) = d1 ∪d2
        }
    }
// now we treat the cases where the peers have paths of different length – extend the
path
// of peer p1 if it has already a sufficient number of data objects corresponding to the
// extended key
30: if ( l1 = 0 and l2 > 1 )
    {
31:     k1 is extended by the inverse bit from key(p2) at position lc + l1
32:     ld1 = number of data objects matching key k1
33:     if ( ld1 >minstorage )
        {
34:       key(p1) = k1
35:       update the reference lists and exchange data objects
        }
    }
```

Πιο αναλυτικά, έχουμε: Όταν δύο σταθμοί p_1 και p_2 «συναντιούνται», πρώτα αποφασίζουν ποιος σταθμός έχει το μεγαλύτερο σε μήκος κλειδί και βρίσκουν το κοινό πρόθεμα των κλειδιών τους (γραμμές 1-3). Έπειτα, ανταλλάσουν λίστες αναφορών από τους πίνακες δρομολόγησης, στο επίπεδο όμως του κοινού τους προθέματος (γραμμές 4-5). Το βήμα αυτό είναι απαραίτητο για τη διασπορά και την ομοιόμορφη κατανομή των αναφορών σε όλους τους πίνακες δρομολόγησης. Η διαδικασία συνεχίζεται ανάλογα με το πώς σχετίζονται τα κλειδιά των σταθμών (γραμμές 6-7). Αν τα κλειδιά επεκτείνουν το κοινό τους πρόθεμα προς διαφορετικές κατευθύνσεις (γραμμές 8-10), τότε οι σταθμοί ανταλλάσσουν δεδομένα, τα οποία μπορεί ο καθένας τους να αποθηκεύσει (μπορεί να αποθηκεύσουν και δεδομένα που τελικά δεν αντιστοιχούν στο κλειδί τους) (γραμμή 9). Ο σταθμός με το μικρότερο κλειδί σε μήκος χρησιμοποιεί τον πίνακα δρομολόγησης του άλλου σταθμού για να εντοπίσει κάποιον άλλο σταθμό, έτσι ώστε να εκτελέσει τον αλγόριθμο ανταλλαγής μαζί του (γραμμή 10). Το γεγονός αυτό επιβεβαιώνει ότι κάθε ανταλλαγή αρχικοποιεί κάποια επέκταση των κλειδιών, αν αυτό είναι δυνατό, και αυτό βοηθά στη γρήγορη κατασκευή της δομής πρόσβασης. Η επόμενη περίπτωση είναι όταν τα κλειδιά των σταθμών είναι ίσα (γραμμές 11-29) και αυτό αποτελεί το κυρίως βήμα της διαδικασίας. Οι σταθμοί επεκτείνουν τα κλειδιά τους κατά ένα δυαδικό ψηφίο αλλά αντίθετο (γραμμές 12-13) και επιλέγουν κάποια δεδομένα, από αυτά που ήδη γνωρίζουν από κοινού, (από τους πίνακες δρομολόγησης) με βάση τα νέα επεκταμένα κλειδιά που δημιουργήθηκαν (γραμμές 14-15). Ανάλογα με το αν υπάρχουν αρκετά δεδομένα (minstorage) θα αποφασίσουν αν θα δεχτούν την επέκταση του κλειδιού (γραμμές 18-27). Όταν ένας σταθμός επεκτείνει το κλειδί, θα πρέπει να αποκτήσει όλα τα δεδομένα που αντιστοιχούν στο νέο του κλειδί και να δημιουργήσει μια καινούργια λίστα αναφορών για το καινούργιο επίπεδο δρομολόγησης, η οποία αρχικά θα περιέχει μόνο μια αναφορά προς το σταθμό που την δημιούργησε. Αργότερα αυτή η λίστα αναφορών θα συμπληρωθεί και με άλλες εγγραφές εξαιτίας των ανταλλαγών αναφορών (γραμμές 4-5). Αν και οι δύο σταθμοί αποφασίσουν να μην επεκτείνουν τα κλειδιά τους τότε ανταλλάσσουν όλα τα δεδομένα που έχουν στη διάθεσή τους μέχρι τώρα (γραμμές 28-29). Με αυτόν τον τρόπο, τα δεδομένα αντιγράφονται, και σε μελλοντικές συναντήσεις με άλλους σταθμούς θα έχουν συγκεντρωθεί αρκετά δεδομένα, έτσι ώστε να είναι δυνατή μια επέκταση του κλειδιού. Η τελευταία περίπτωση που μπορεί να μας απασχολήσει

είναι όταν το κλειδί του ενός σταθμού είναι πρόθεμα του κλειδιού του άλλου (γραμμές 30-35). Τα βήματα που εκτελούνται εδώ είναι παρόμοια με παραπάνω εκτός από το ότι ο σταθμός με το μικρότερο σε μήκος κλειδί αναγκαστικά θα επεκτείνει το κλειδί του με το αντίθετο δυαδικό ψηφίο που έχει ο άλλος σταθμός στη θέση επέκτασης.

4.5 Ανάλυση του κόστους αναζήτησης

Για την ανάλυση του κόστους αναζήτησης θα εξετάσουμε μόνο τον αριθμό μηνυμάτων που ανταλλάσσονται ανάμεσα στους σταθμούς κατά τη διάρκεια της διαδικασίας αναζήτησης και όχι το μήκος του μονοπατιού αναζήτησης. Το κυριότερο ζήτημα το οποίο και θα αποδειχθεί, είναι: Ακόμη και στην περίπτωση που το δέντρο είναι τελείως εκτός ισορροπίας, ο αριθμός των μηνυμάτων παραμένει χαμηλός, της τάξης $O(\log N)$, υποθέτοντας ότι η κατανομή των σταθμών στις λίστες αναφορών είναι ίδια με την καθολική κατανομή των σταθμών που έχουν το αντίστοιχο κλειδί.

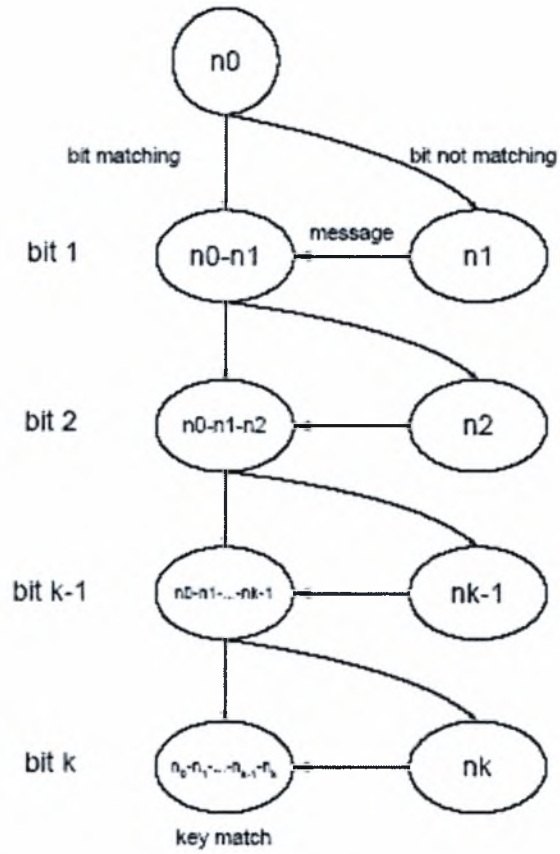
Πιο συγκεκριμένα, για μια λίστα αναφορών *ref*(p, *l*) ενός σταθμού που σχετίζεται με το κλειδί $key(p) = b_1..b_{kp}$, υπάρχει ένα σύνολο από σταθμούς $p' \in \hat{P}$, οι οποίοι έχουν τη δυνατότητα να επιλεχθούν να μπουν στη λίστα αυτή. Πρόκειται για τους σταθμούς για τους οποίους ισχύει $key(p) = b_1..b_{l-1}b_l^{-1}$, όπου $b_l^{-1} = (1 - b_l)$. Χρειάζεται να κάνουμε την υπόθεση ότι η πιθανότητα να ανήκει κάποιος σταθμός στη λίστα αναφορών *ref*(p, *l*) είναι ίδια για όλους τους $p' \in \hat{P}$, δηλαδή $\Pr ob(p' \in ref(p,l)) = const$ για όλους τους σταθμούς $p' \in \hat{P}$. Η ιδιότητα αυτή επιβεβαιώνεται από το βήμα της συγχώνευσης των λιστών αναφορών διαφορετικών σταθμών κατά τη διαδικασία ανταλλαγής(αλγόριθμος 4.2, γραμμές 5-7). Πιο αναλυτικά, στο βήμα αυτό, δύο σταθμοί p_1 και p_2 με λίστες αναφορών $r_1 = ref(p_1, l)$ και $r_2 = ref(p_2, l)$ αντίστοιχα, κατασκευάζουν από αυτές τις λίστες δύο νέες λίστες ανταλλάσσοντας τυχαία εγγραφές από τις r_1 και r_2. Η ανταλλαγή αυτή όμως πρέπει να γίνει με συγκεκριμένο τρόπο, για να φτάσουμε στην επιθυμητή κατανομή, όπου κάθε εγγραφή επιλέγεται με την ίδια πιθανότητα. Άτυπα, ο καθένας από τους σταθμούς κατανέμουν τυχαία τις εγγραφές από το σύνολο $r_1 \cup r_2$ μεταξύ τους, διατηρώντας όμως τη σχετική διάταξη των εγγραφών. Αν μετά από αυτό το βήμα μπορούν να προστεθούν περαιτέρω εγγραφές, οι σταθμοί διαλέγουν ο ένας από τον άλλο

όσες περισσότερες εγγραφές μπορούν, ξεκινώντας από την αρχή της κάθε λίστας. Έτσι, δημιουργούνται αντίγραφα για κάποιες από τις εγγραφές. Παρατηρώντας τη σειρά επιλογής των επιπρόσθετων εγγραφών που επιλέγονται για αντιγραφή, επιβεβαιώνεται το γεγονός ότι αντιγράφονται πρώτα οι εγγραφές που συμβαίνουν λιγότερο συχνά. Γι' αυτό κι αλγόριθμος έχει την τάση να εξισορροπεί την καθολική κατανομή των αναφορών.

Θα προχωρήσουμε στην απόδειξη χρησιμοποιώντας το γεγονός ότι οι σταθμοί επιλέγονται για μια λίστα αναφορών με ίση πιθανότητα, γεγονός που επιβεβαιώνεται όπως είδαμε με τη διαδικασία ανταλλαγής του αλγορίθμου κατασκευής. Κατά τη διαδικασία αναζήτησης, θεωρούμε ένα συγκεκριμένο κλειδί σε έναν κόμβο φύλλο στο δέντρο αναζήτησης. Η αναζήτηση για το κλειδί αυτό ξεκινά σε κάποιο τυχαίο κόμβο. Σε κάθε βήμα (πηγαίνοντας από το ένα δυαδικό ψηφίο του κλειδιού στο επόμενο) είτε το ψηφίο του κλειδιού του σταθμού ταιριάζει με το αντίστοιχο ψηφίο του κλειδιού αναζήτησης, οπότε και απαιτείται κανένα μήνυμα, είτε η αίτηση πρέπει να μεταφερθεί σε έναν άλλο σταθμό, ακολουθώντας μια από τις αναφορές, που βρίσκονται αποθηκευμένες στον πίνακα δρομολόγησης του αντίστοιχου επιπέδου. Έπειτα η ίδια διαδικασία συνεχίζεται μέχρι να φτάσουμε το επίπεδο των φύλλων.

Ας δούμε τα μονοπάτια αναζήτησης και τον αριθμό τον σταθμών που είναι υπεύθυνοι για ένα συγκεκριμένο κλειδί. Στο επίπεδο 0, έχουμε $n_0 = N$ σταθμούς, και αναγκαστικά όλοι οι σταθμοί είναι υπεύθυνοι για το κενό κλειδί. Από αυτούς στους n_1 σταθμούς το πρώτο ψηφίο δεν ταιριάζει με αυτό του κλειδιού αναζήτησης, ενώ στους $n_0 - n_1$ ταιριάζει. Οπότε, με πιθανότητα $\frac{n_1}{n_0}$ απαιτείται ένα μήνυμα, έτσι ώστε να συνεχίσουμε με έναν σταθμό, ο οποίος επιλέγεται τυχαία από τη λίστα αναφορών, όπου να ταιριάζει το ψηφίο. Αφού επιλέξουμε έναν τέτοιο σταθμό συνεχίζουμε με το επόμενο ψηφίο. Πάλι υπάρχουν n_2 σταθμοί ανάμεσα στους $n_0 - n_1$ εναπομείναντες, όπου το δεύτερο ψηφίο δεν ταιριάζει. Από τη στιγμή που μπορούμε να υποθέσουμε ότι η κατανομή των σταθμών στις λίστες αναφορών είναι ίδια με την καθολική κατανομή των σταθμών, με άλλα λόγια, όπως κάθε

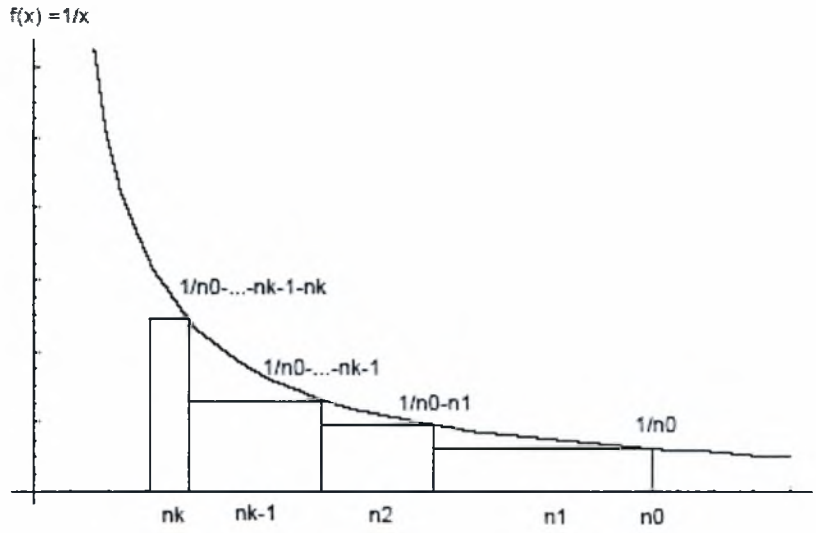
σταθμός επιλέγεται με την ίδια σταθερή πιθανότητα, με πιθανότητα $\frac{n_2}{n_0 - n_1}$ ένα μήνυμα στέλνεται προκειμένου να συνεχίσουμε με ένα σταθμό όπου το δεύτερο ψηφίο θα ταιριάζει. Η διαδικασία μπορεί να συνεχιστεί για k βήματα αν υποθέσουμε ότι k είναι το μέγεθος του κλειδιού. Η όλη διαδικασία φαίνεται στο σχήμα της εικόνας 4.2.



Πηγή: K.Aberer [7]

Εικόνα 4.2 Διαδικασία αναζήτησης k βημάτων για αναζήτηση κλειδιού με μήκος k

Προσθέτοντας τον αναμενόμενο αριθμό μηνυμάτων σε κάθε βήμα, καταλήγουμε στον αναμενόμενο συνολικό αριθμό μηνυμάτων της διαδικασίας αναζήτησης: $\sum_{i=1}^{k} \frac{n_i}{n_0 - \ldots - n_{i-1}}$ για μια αυθαίρετη ακολουθία θετικών αριθμών $n_1,\ldots,n_k$ με $\sum_{i=1}^{k} n_i < n_0$. Προκειμένου να εκτιμήσουμε την τιμή του αναμενόμενου αριθμού μηνυμάτων θα ακολουθήσουμε την εξής διαδικασία: Κάθε όρος μπορεί να αναπαρασταθεί από ένα ορθογώνιο όπως φαίνεται στην ακόλουθη εικόνα 4.3.



Πηγή: K. Aberer [7]

Εικόνα 4.3 Παραγωγή άνω ορίου για τον αριθμό των αναμενόμενων μηνυμάτων

Μπορούμε ευδιάκριτα να παρατηρήσουμε ότι τα παραλληλόγραμμα βρίσκονται ακριβώς κάτω από την καμπύλη της συνάρτησης $f(x)=\frac{1}{x}$. Γι' αυτό η συνολική περιοχή των παραλληλογράμμων είναι μικρότερη από $\int_r^{n_0}\frac{1}{x}dx=\log n_0-\log r$, όπου $r=n_0-\sum_{i=1}^{k}n_i\geq 1$. Οπότε καταλήγουμε στο όριο $\sum_{i=1}^{k}\frac{n_i}{n_0-\ldots-n_{i-1}}<\log n_0=\log 2\log_2 N$, το οποίο δείχνει ότι ανεξάρτητα από το κατά το πόσο το δέντρο είναι εκτός ισορροπίας, ο αριθμός των μηνυμάτων θα είναι πάντα της τάξης του $O(\log N)$. Το όριο αυτό είναι πολύ κοντά στο $\frac{1}{2}\log_2 N$, που είναι ο αναμενόμενος αριθμός μηνυμάτων αν το δέντρο αναζήτησης ήταν ισορροπημένο.

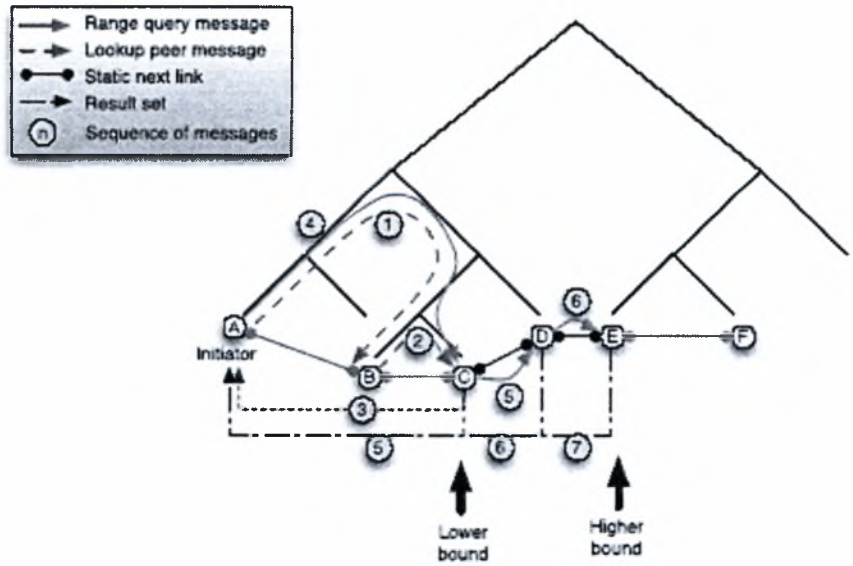
4.6 Υποστήριξη των ερωτήσεων εύρους στη δομήP-Grid

Το P-Grid όπως είδαμε κατασκευάζεται με βάση μια συνάρτηση κατακερματισμού όπου διατηρείται η διάταξη των κλειδιών, οπότε μπορούν να εφαρμοστούν αποτελεσματικά σε αυτό ερωτήσεις εύρους. Παρουσιάζονται δύο αλγόριθμοι [8] για την αναζήτηση που

περιέχει διαστήματα με τιμές κλειδιών και αυτοί είναι ο σειριακός αλγόριθμος διαπέρασης ελάχιστου-μέγιστου (min-max traversal algorithm) και ο παράλληλος αλγόριθμος του «ντους» (shower algorithm).

4.6.1: Αλγόριθμος διαπέρασης min-max

Οι ερωτήσεις εύρους μπορούν να επεξεργαστούν σειριακά ξεκινώντας από έναν σταθμό, ο οποίος περιέχει δεδομένα, τα οποία ανήκουν στο κατώτερο όριο του διαστήματος τιμών της ερώτησης. Έπειτα, προωθεί την ερώτηση σε έναν σταθμό, που είναι υπεύθυνος για το επόμενο κομμάτι του χώρου των κλειδιών, μέχρι να βρεθεί ένας σταθμός υπεύθυνος για το ανώτερο όριο του διαστήματος αναζήτησης. Η πληροφορία για τους σταθμούς, που είναι υπεύθυνοι για το επόμενο διάστημα του χώρου των κλειδιών, μπορεί να ανακτηθεί είτε κατά την διάρκεια κατασκευής του P-Grid (αλγοριθμικά τετριμμένη διαδικασία) είτε κατά την ώρα της εκτέλεσης, χρησιμοποιώντας την ήδη υπάρχουσα πληροφορία από τον πίνακα δρομολόγησης που διατηρούν οι σταθμοί. Στην εικόνα 4.4 φαίνεται ένα παράδειγμα μιας ερώτησης εύρους ενώ παρατίθεται και ο αλγόριθμος σε ψευδοκώδικα.



Πηγή: K. Aberer [8]

Εικόνα 4.4 Παράδειγμα ερώτησης εύρους εφαρμόζοντας τον αλγόριθμο min-max

Αρχικά, ο σταθμός A εκτελεί αναζήτηση για να βρεθεί ο σταθμός, που είναι υπεύθυνος για το κάτω όριο του διαστήματος της ερώτησης και παίρνει ως απάντηση τον σταθμό C. Ο A στέλνει την ερώτηση εύρους στον C και λαμβάνει πίσω τα αποτελέσματα από το κομμάτι του διαστήματος της ερώτησης που κατέχει ο C. Παράλληλα, η ερώτηση εύρους προωθείται και στον D ο οποίος είναι σταθμός υπεύθυνος για το επόμενο στη σειρά κομμάτι του χώρου κλειδιών. Ο D αφού διαπιστώσει ότι πρόκειται για διάστημα τιμών για το οποίο είναι υπεύθυνος στέλνει τα αποτελέσματα στον A και προωθεί με τον ίδιο τρόπο την ερώτηση στον επόμενο κόμβο στη σειρά, τον E. ο E στέλνει κι αυτός τα αποτελέσματα στον A και σταματά τη διαδικασία, καθώς αντιλαμβάνεται ότι ανάμεσα στις τιμές κλειδιών, για τις οποίες είναι υπεύθυνος, βρίσκεται και το άνω όριο του εύρους της ερώτησης.

Algorithm 4.3: Sequential Range Queries: minmax(R, p)
if (key(p) $\subseteq$R)
{
return (d $\in$data(p)| key(d) $\in$R)
determine a peer r such that r is responsible for the next key space partition
minmax(R, r)
}

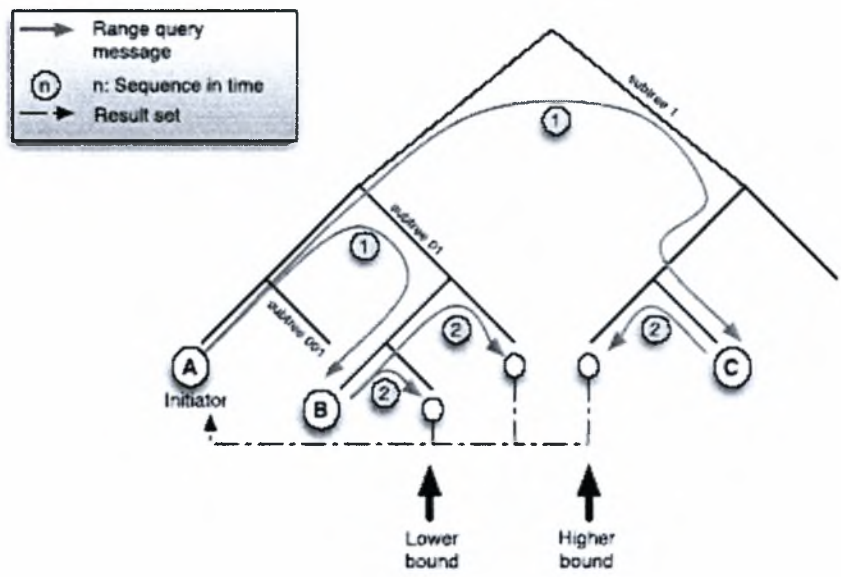
Για να απλοποιήσουμε την ανάλυση του κόστους θεωρούμε ότι ο αλγόριθμος ξεκινά από τον σταθμό, που είναι υπεύθυνος για το κατώτερο όριο του εύρους R της ερώτησης. Επίσης, θεωρούμε ότι οι συνδέσεις προς σταθμούς, που είναι υπεύθυνοι για γειτονικά διαστήματα του χώρου κλειδιών, αποθηκεύονται σε κάθε σταθμό κατά τη διάρκεια κατασκευής της δομής. Επιπλέον, θεωρούμε ότι κατά μέσο όρο υπάρχουν M πόροι ανά διαμέριση του χώρου των κλειδιών. Οπότε, αν υπάρχει μια ερώτηση εύρους που αφορά το διάστημα R, τέτοιο ώστε, να υπάρχουν D πόροι στο δεδομένο διάστημα, τότε το κόστος αναζήτησης και η καθυστέρηση χρησιμοποιώντας την minmax διαπέραση είναι $O(\log_2|\Pi|)+|\Pi_R|-1$, όπου $|\Pi_R|$ είναι ο αριθμός των διαμερίσεων που αφορούν το εύρος R της ερώτησης και $|\Pi|$ ο συνολικός αριθμός των κόμβων φύλλων στο πλήρες δέντρο του P-Grid (στην ουσία συνολικός αριθμός των διαμερίσεων του χώρου των κλειδιών). Το κόστος αναζήτησης και η καθυστέρηση, εδώ, εξαρτώνται από το μέγεθος του

συνόλου D της απάντησης αλλά είναι ανεξάρτητα από το μέγεθος του εύρους R της ερώτησης. Αυτό συμβαίνει επειδή το $|\Pi_R|$ έχει μια αναμενόμενη τιμή D/M, και στην ουσία, χρησιμοποιώντας την ανισότητα Markov, $\Pr[|\Pi_R| \geq {}^{cD}/_{M}] \leq \frac{1}{c}$ για κάθε θετικό c, το οποίο και δίνει ένα ασθενές όριο στην απόκλιση. Δε θεωρούμε την τετριμμένη περίπτωση $D \leq M$ καθώς αυτό θα αφορούσε μόνο ένα ή δύο σταθμούς και επικεντρωνόμαστε στην πιο γενική περίπτωση όπου $D > M$. Χωρίς τις συνδέσεις προς σταθμούς με γειτονικές διαμερίσεις του χώρου των κλειδιών, το κόστος επιβαρύνεται με την ποσότητα $|\Pi_R| O(\log_2|\Pi|)$, η οποία αποτελεί και άνω όριο καθώς ένα μέρος της δρομολόγησης δε χρειάζεται να επαναληφθεί για τους σταθμούς που βρίσκονται στο διάστημα.

4.6.2: Αλγόριθμος Shower

Εδώ, το εύρος R της ερώτησης, πρώτα προωθείται αυθαίρετα σε έναν σταθμό, ο οποίος είναι υπεύθυνος για οποιοδήποτε από τα διαστήματα του χώρου κλειδιών, που αφορούν το εύρος R. Έπειτα, η ερώτηση προωθείται και στους άλλους σταθμούς, που είναι υπεύθυνοι για τα υπόλοιπα διαστήματα, χρησιμοποιώντας πληροφορία που διατηρείται στον πίνακα δρομολόγησης του σταθμού. Η διαδικασία είναι αναδρομική και από τη στιγμή που η ερώτηση χωρίζεται σε διαστήματα υποερωτήσεων, τα οποία φαίνεται να διασχίζουν το δέντρο ταυτόχρονα προκειμένου να απαντηθούν, ο αλγόριθμος ονομάστηκε αλγόριθμος του «ντους». Κατά τη διάρκεια της προώθησης είναι δυνατόν μια ερώτηση να προωθηθεί σε έναν σταθμό, ο οποίος είναι υπεύθυνος για κλειδιά έξω από το εύρος R. Παρόλα αυτά, εγγυάται ότι ο σταθμός αυτός, θα προωθήσει την ερώτηση εύρους πίσω σε έναν σταθμό με διαμέριση που να ανήκει στο εύρος R. Επιπλέον, η δρομολόγηση στο P-Grid διαβεβαιώνει ότι δε θα επιστραφούν τα ίδια αποτελέσματα πολλές φορές, από πολλούς σταθμούς, για το ίδιο διάστημα.

Ένα παράδειγμα του αλγορίθμου παρουσιάζεται στην εικόνα 4.5 και ο ψευδοκώδικας παρατίθεται παρακάτω στον αλγόριθμο 4.4.



Πηγή: Aberer [8]

Εικόνα 4.5 Παράδειγμα ερώτησης εύρους εφαρμόζοντας τον αλγόριθμο shower

Algorithm 4.4: Parallel Range Queries: shower(R, $l_{current}$, p)
if (key(p) $\subseteq$R)
{ return (d $\in$data(p)| key(d) $\in$R) }
determine l_l such that prefix(min(R), l_l) = $\overline{\text{prefix}(p, l_l)}$ //The l_l bit of prefix(p) is inverted
determine l_h such that prefix(max(R), l_h) = $\overline{\text{prefix}(p, l_h)}$ // The l_h bit of prefix(p) is // inverted

l_{min} = max($l_{current}$, min(l_l, l_h))
l_{max} = max(l_l, l_h)
if ($l_{current} < l_{max}$)
for ($l = l_{min}$ to l_{max})
{
r = randomly selected element from refs(p, l)
shower(R, l+1, r)
}

Το κόστος αναζήτησης σε μορφή μηνυμάτων φράσσεται χαμηλά από την ποσότητα $O(x)+|\Pi_R|-1$. Από τη στιγμή που κάθε μήνυμα το οποίο δημιουργείται σε κάποιο

υποδιάστημα του εύρους φτάνει σε διαφορετικό κόμβο φύλλο(από τη στιγμή που τα υποδιαστήματα είναι σαφώς χωρισμένα), και αναμένονται D/M τέτοια υποδιαστήματα, το άνω όριο είναι $O(x)+\min\left(2O(|\Pi_R|),2^{Depth-x}\right)$ όπου *Depth* είναι το μέγιστο μήκος του μονοπατιού για διαμέριση που βρίσκεται μέσα στο εύρος R. Γι' αυτό, η πολυπλοκότητα του αλγορίθμου αυτού είναι επίσης ανεξάρτητη από τον αριθμό των πόρων στο εύρος R, αλλά εξαρτάται από την κατανομή των πόρων (η οποία καθορίζει τον όρο *Depth*). Να σημειωθεί ότι, ο σταθμός που ξεκίνησε τη διαδικασία αναζήτησης, θα αρχίσει να λαμβάνει αποτελέσματα για κάποια διαστήματα του εύρους με μια ελάχιστη καθυστέρηση $O(x)$, καθώς είναι σίγουρο ότι θα συναντήσει κάποιο σταθμό υπεύθυνο για κάποιο κομμάτι του εύρους.

Η αναμενόμενη τιμή για το χ είναι $0.5\log(nM/D)$. Η διαίσθηση για την τιμή του χ στηρίζεται στο ότι αν αυξήσουμε τη μέση μνήμη για κάθε λογική διαμέριση σε D αντί για M, τότε θα υπάρχουν $\frac{n}{D/M}$ διαμερίσεις του χώρου κλειδιών συνολικά.

Κεφάλαιο 5: Συμπεράσματα

Φτάνοντας στο τέλος της εργασίας και συνοψίζοντας, ακολουθεί ένας συγκεντρωτικός πίνακας με την απόδοση σε αριθμό μηνυμάτων, των δομών που μελετήθηκαν.

Απόδοση των δομών ως προς τα μηνύματα που ανταλλάσσονται					
	BATON	BATON*	VBI-TREE	SKIP GRAPH	P-GRID
Ένθεση στοιχείου	O(logN) + max 6logN	$O(m \cdot \log_m N)$	O(logN) + max 7logN	O(logN)	
Απόσβεση στοιχείου	O(logN) + max 12logN	$O(m \cdot \log_m N)$	O(logN) + max 8logN	O(logN)	
Ερώτηση σημείου	O(logN)	$O(\log_m N)$	O(logN)	O(logN)	O(logN)
Ερώτηση εύρους	O(logN + X), X ο αριθμός κόμβων που καλύπτουν το εύρος της ερώτησης	$O(\log_m N + X)$, X ο αριθμός κόμβων που καλύπτουν το εύρος της ερώτησης	O(logN + X), X ο αριθμός κόμβων που καλύπτουν το εύρος της ερώτησης	O(logN+2X)	$O(\log_2 \lvert\Pi\rvert) + \lvert\Pi_R\rvert - 1$ Min-max αλγόριθμος $O(x) + \min\left(2O(\lvert\Pi_R\rvert), 2^{Depth-x}\right)$ Shower αλγόριθμος
Ανοχή σφαλμάτων	O(logN) Το δέντρο κατακερματίζεται δύσκολα	$f \rightarrow$ #failed nodes Από f/(m+1) Έως f/(m-1) αποκομμένοι κόμβοι	Όπως και στο BATON	$O(f \cdot \log N)$ αποκομμένοι κόμβοι με f αστοχίες	Εξαρτάται από την τιμή minstorage η οποία καθορίζει τον αριθμό αντιγράφων
Εξισορρόπηση φόρτου	O(logN) ανά εισαγωγή ή διαγραφή	Όπως και στο BATON	Όπως και στο BATON	Δεν απαιτείται – εγγυήσεις συμφόρησης λόγω δημοφιλών στόχων	Δεν απαιτείται
Αναδόμηση Δικτύου	O(logN) ανά εμπλεκόμενο κόμβο	Όπως και στο BATON	O(n ×logN + n) ανά εμπλεκόμενο κόμβο	Δεν απαιτείται	Δεν απαιτείται -ενημερώσεις γίνονται όταν σταθμοί «συναντιούνται»

Ακολουθούν κάποια συμπεράσματα και σχόλια για τις δομές που μελετήθηκαν στην εργασία αυτή καθώς και μια πρόταση για μια πιο απλή υλοποίηση δενδρικής δομής κοντά στην ιδέα της δομής BATON.

Δενδρικές δομές BATON, BATON* και VBI-δέντρο

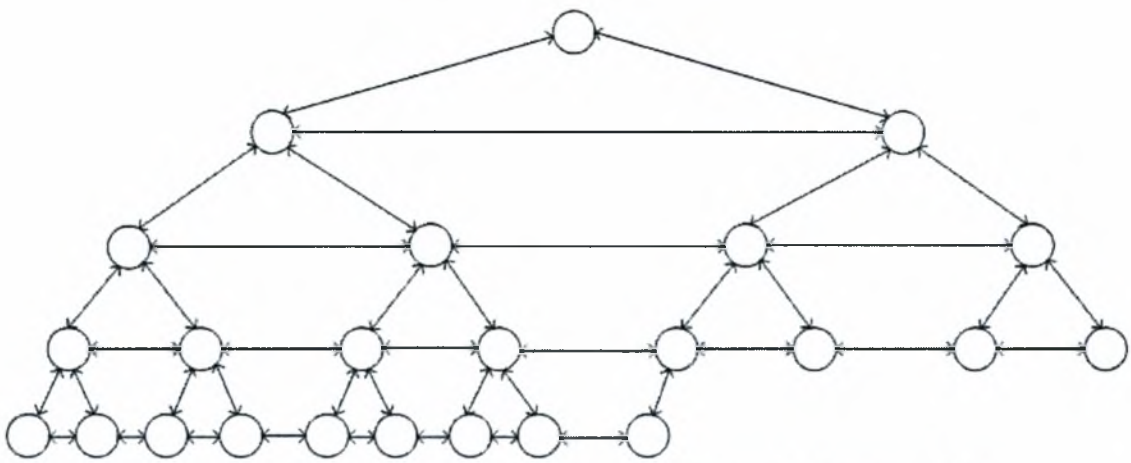
Γενικά, η δομή BATON προσπαθεί να κρατήσει ισορροπημένο το δέντρο της από όλες τις απόψεις(και από άποψη φόρτου και από άποψη τοποθέτησης των κόμβων) για να έχει μια σταθερή απόδοση αλλά και να μην υπερφορτώνει την περιοχή της ρίζας. Το γεγονός αυτό όμως, έχει ως αποτέλεσμα να αυξάνει την πολυπλοκότητα των βασικών πράξεων και να χρειάζεται να ανατρέχουμε συνεχώς σε μηχανισμούς αναδόμησης δικτύου και εξισορρόπησης φόρτου. Έτσι, ενώ αρχικά η δομή BATON φαίνεται αρκετά απλή και στη σύλληψη ως ιδέας αλλά και στην υλοποίηση, τελικά αποδεικνύεται πολύπλοκη λόγω της συνεχούς προσπάθειας επιβολής ισορροπίας.

Το ίδιο συμβαίνει και με τις άλλες δύο δομές BATON* και VBI-δέντρο, οι οποίες στηρίζονται στην πρωταρχική δομή BATON. Πιο συγκεκριμένα, στη δομή BATON* επιτυγχάνεται καλύτερη απόκριση του δικτύου ως προς τις αναζητήσεις, αρκεί να επιλέξουμε μια κατάλληλη τιμή για το fan-out, έτσι ώστε να μην αυξάνεται σε απαγορευτικό βαθμό το κόστος ενημέρωσης των πινάκων δρομολόγησης. Επιπλέον πλεονέκτημα της δομής αυτής είναι και η αποτελεσματική υποστήριξη πολύ-παραμετρικών ερωτήσεων. Το VBI-δέντρο είναι μια παραλλαγή του BATON και σχεδιάστηκε για να υποστηρίζει δημοφιλείς πολυδιάστατες δενδρικές δομές ευρετηρίου όπως το R-δένδρο, το X-δένδρο και άλλες παρόμοιες δομές.

Πρόταση για μια νέα δενδρική δομή

Μια πρόταση για την απλούστευση της δομής BATON χωρίς να χάνουμε σε λειτουργικότητα είναι η εξής [17]. Ένα ημι-πλήρες δυαδικό δέντρο, ύψους $\lceil \log n \rceil$ σαν και αυτό που χρησιμοποιείται, έμμεσα, στην δενδρική δομή σωρού, στη διαδικασία ταξινόμησης με τη χρήση σωρού (heapsort) [16]. Όλοι οι κόμβοι του, με εξαίρεση αυτούς των δύο τελευταίων επιπέδων, έχουν δύο παιδιά. Στο προτελευταίο επίπεδο, όλοι οι κόμβοι έχουν είτε δύο είτε μηδέν παιδιά, με ενδεχομένως εξαίρεση έναν. Όλοι οι

κόμβοι με δύο παιδιά, απλώνονται από αριστερά προς τα δεξιά, ακολουθεί ενδεχομένως αυτός με το ένα (αριστερό) παιδί, και, κατόπιν αυτοί με κανένα παιδί. Κάθε κόμβος ξέρει τον πατέρα, τα παιδιά και τους δύο γείτονές του στο επίπεδό του. Επιπλέον σε κάθε επίπεδο υπάρχουν οριζόντιες συνδέσεις ανάμεσα στους κόμβους γείτονες. Ένα παράδειγμα της δομής αυτής φαίνεται στην εικόνα 5.1.

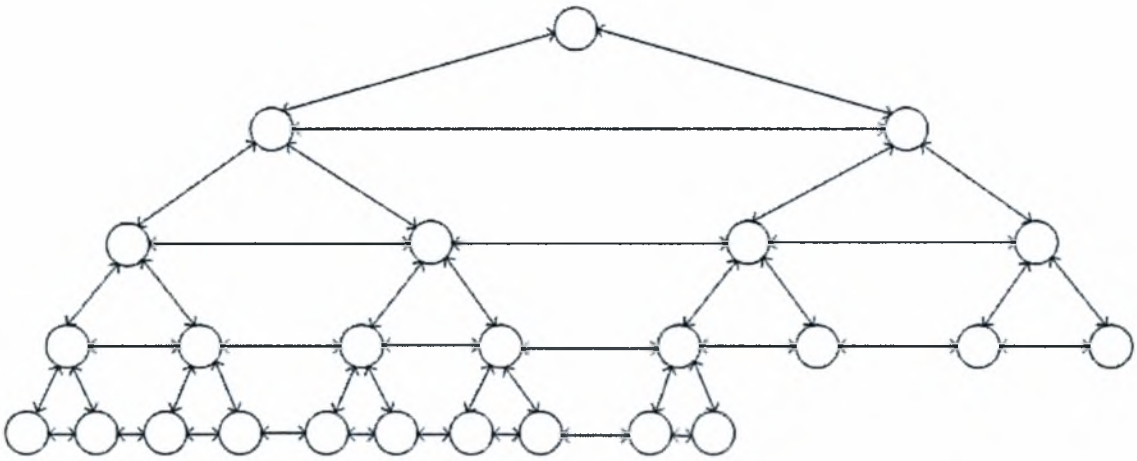


Εικόνα 5.1: Παράδειγμα μιας απλούστερης δομής για τη δομή BATON

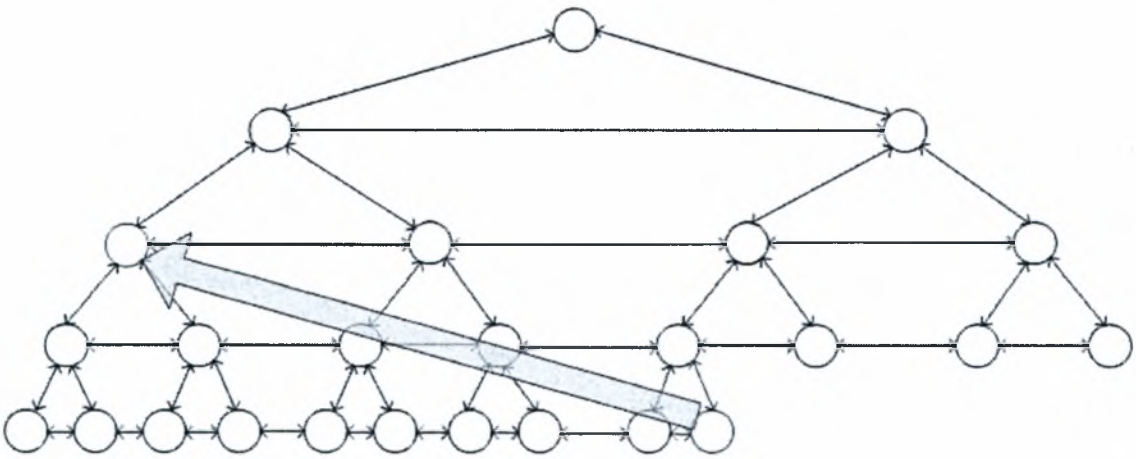
Για να μην εκτελείται κάθε φορά ολόκληρη διαδικασία για να βρούμε σε ποιο σημείο θα εισαχθεί ένας νέος κόμβος ή ποιος θα είναι ο αντικαταστάτης για έναν κόμβο που αποχωρεί, είναι καλύτερα να υπάρχει ένα δεδομένο σημείο εισαγωγής και ένας συγκεκριμένος κόμβος αντικαταστάτης. Πιο συγκεκριμένα, όταν ένας νέος κόμβος εμφανίζεται στο σύστημα, γίνεται παιδί του κόμβου με το ένα παιδί, ή, εάν δεν υπάρχει τέτοιος, παιδί του αριστερότερου κόμβου με μηδέν παιδιά που βρίσκεται στο προτελευταίο επίπεδο, διαμοιραζόμενος το range του (εικόνα 5.2). Κατά την αποχώρηση ενός κόμβου, αυτός αντικαθίσταται από τον δεξιότερο κόμβο του τελευταίου επιπέδου (εικόνα 5.3).

Ως προς την αναζήτηση με εύρος τιμών, αφού βρούμε το κατώτερο όριο του διαστήματος, ανεβαίνουμε προς τα πάνω, μέχρι να βρούμε τον πρώτο κοινό πρόγονο που καλύπτει το εύρος αναζήτησης, ελέγχοντας, όπου χρειάζεται, τα διαστήματα τιμών των γειτόνων κάνοντας χρήση των εκάστοτε γειτονικών συνδέσεων. Έπειτα κατεβαίνουμε

προς τα κάτω χρησιμοποιώντας τους δείκτες προς τα παιδιά. Το κόστος είναι λογαριθμικό ως προς τον αριθμό των κόμβων. Για επιπλέον επιτάχυνση της διαδικασίας μπορούμε να αποθηκεύουμε το μονοπάτι προς την ρίζα, αλλά με κόστος στον χώρο και την διαχείριση.



5.2: Παράδειγμα εισαγωγής νέου κόμβου



5.3: Το δεξιότερο φύλλο είναι ο αντικαταστάτης κατά την αποχώρηση ενός κόμβου

Σχετικά με την εξισορρόπηση φόρτου, ταιριάζει αρκετά η χρήση της τεχνικής της αναδόμησης με βάρη. Πιο συγκεκριμένα, όταν στο υποδένδρο ενός κόμβου χ, περάσουν τόσες εισαγωγές/διαγραφές όσο είναι και το βάρος του (πχ, πλήθος αποθηκευμένων αντικειμένων), τότε όλοι οι κόμβοι στο υποδέντρο, δίχως να αλλάξουν τους δείκτες τους,

ανακατανέμουν τα διαστήματα τιμών για τα οποία είναι υπεύθυνοι, ώστε να υπάρχει ισοκατανομή. Το κόστος που προκύπτει, είναι amortized λογαριθμικό.

Γράφος Αναπηδήσεως

Ο γράφος αναπηδήσεως είναι μια δομή βασισμένη στις γνωστές λίστες αναπηδήσεως, με σαφή και απλή υλοποίηση αλλά συνάμα και αποτελεσματική. Είναι μια στιβαρή δομή που καταφέρνει να ανταποκρίνεται σε λογαριθμικό χρόνο και μηνύματα. Μόνο της μειονέκτημα, το γεγονός ότι πέφτει η απόδοσή της όταν k != log$N$, όπου k ο αριθμός των ψηφίων του διανύσματος μελών.

P-Grid

Η δομή P-Grid φαίνεται να ανταποκρίνεται πολύ καλά και στο λογαριθμικό χρόνο αναζήτησης αλλά και στο γεγονός ότι δε χρειάζεται επιπλέον κόστος για εξισορρόπηση φόρτου. Το γεγονός όμως ότι είναι μια δομή πρόσβασης που στηρίζεται σε ένα μη ισορροπημένο δέντρο δημιουργεί το εξής πρόβλημα. Τα μήκη των κλειδιών τα οποία σχετίζονται με σταθμούς σε ένα συγκεκριμένο διάστημα τιμών, σχετίζονται και με τον αριθμό των δεδομένων που βρίσκονται στο διάστημα αυτό. Για διαστήματα τιμών όπου τυχαίνει να υπάρχει μεγάλος αριθμός δεδομένων, το μήκος των κλειδιών που αντιστοιχούν στην περιοχή αυτή, έχει την τάση να αυξάνεται πολύ περισσότερο σε σχέση με περιοχές τιμών όπου υπάρχει σχετικά μικρός αριθμός δεδομένων. Έτσι προκύπτει και η ανισορροπία στο δέντρο. Γι' αυτό το λόγο, η δομή δεν μπορεί να εγγυηθεί πλέον ότι το μήκος των μονοπατιών αναζήτησης θα παραμείνει μικρό. Στην πραγματικότητα, μπορεί να υπάρξουν δέντρα τα οποία στη χειρότερη περίπτωση περιέχουν μονοπάτια μήκους O(N) γεγονός που καθιστά τη χρήση ενός μη ισορροπημένου δέντρου αναποτελεσματική. Παρόλα αυτά η δομή P-Grid εστιάζει στον αριθμό των μηνυμάτων που ανταλλάσσονται κατά τη διαδικασία αναζήτησης και υποστηρίζει πως μπορεί να γίνει χρήση άλλων μεθόδων για την επιτάχυνση των τοπικών πράξεων και άρα το μήκος του μονοπατιού αναζήτησης να μην αποτελεί έτσι ουσιαστικό πρόβλημα. Επιπλέον, με την αντικατάσταση του μέγιστου ορίου στο μήκος του κλειδιού ($k_{\max}$) με μια καθολική κατανομή δεδομένων, επιτυγχάνεται μείωση της καθολικής γνώσης που απαιτείται για την κατασκευή του ευρετηρίου (του δυαδικού δέντρου) και αυξάνεται ο βαθμός της

αυτόνομης οργάνωσης της δομής. Τέλος, να αναφέρουμε ότι, έχει γίνει μια προσπάθεια ο μηχανισμός του P-Grid να εφαρμοστεί σε ένα P2P κατανεμημένο σύστημα, το οποίο ονομάστηκε Gridella [10] και στόχος είναι η περαιτέρω εξέλιξή του σε ένα εντελώς αποκεντρωμένο και πραγματικά αποτελεσματικό, κατανεμημένο σύστημα διαχείρισης δεδομένων.

Βιβλιογραφία

[1] "BATON: A Balance Tree Structure for Peer-to-Peer Networks", H.V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, In Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.

[2] "BATON*: Speeding up Search in Peer-to-Peer Networks with A Multi-way Tree Structure", H.V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Quang Hieu Vu, Rong Zhang, SIGMOD 2006, June 27-29 / 2006, Chicago, Illinois, USA.

[3] "VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes", H.V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, Rong Zhang, Aoying Zhou.

[4] "Skip Graphs", James Aspnes, Gauri Shah, In Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 384{393, 2003}.

[5] "Extended Skip Graphs for Efficient Key Search in P2P Environment", Satoshi Fujita, Akira Ohtsubo, Massaya Mito, In Proccedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN' 05), 2005.

[6] "P-Grid: A Self-organizing Access Structure for P2P Information Systems", K. Aberer, In Proceedings of the 9th International Conference on Cooperative Information Systems, (CoopIS 2001), Trento, Italy, 2001.

[7] "Scalable Data Access in Peer-to-Peer Systems Using Unbalanced Search Trees", K. Aberer, June 28, 2002.

[8] "Range Queries in Trie-Structured Overlays", A. Datta, M. Hauswitrh, R. John, R. Schmidt, K. Aberer, 2005.

[9] "An Overview on Peer-to-Peer Information systems", K.Aberer, M. Hauswitrh, 2002.

[10] http://www.p-grid.org - Gridella: P-Grid implemented in Java

[11] "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, In Proceedings of the ACM SIGCOMM, 2001.

[12] "A Definition of *Peer-to-Peer* Networking for the Classification of *Peer*-to-Peer Architectures and Applications", R. Schollmeier, in proceedings of the 1st conference on Peer-to-Peer computing, 2002.

[13] Σ. Λάλης, από το μάθημα Κατανεμημένα Συστήματα, σετ διαφανειών: «Εισαγωγή στα κατανεμημένα συστήματα», Πανεπιστήμιο Θεσσαλίας, ΤΜΗΥΤΔ, Εαρινό εξάμηνο 2006.

[14] Σ. Λάλης, από το μάθημα Κατανεμημένα Συστήματα, σετ διαφανειών: «Υπηρεσίες Καταλόγου», Πανεπιστήμιο Θεσσαλίας, ΤΜΗΥΤΔ, Εαρινό εξάμηνο 2006.

[15] Π. Μποζάνης, από το βιβλίο του «Δομές Δεδομένων», κεφάλαιο 19- ενότητα 19.1 «Λίστα Αναπηδήσεως», ΕΚΔΟΣΕΙΣ ΤΖΙΟΛΑ, copyright 2005

[16] Π. Μποζάνης, από το βιβλίο του «Δομές Δεδομένων», κεφάλαιο 4- ενότητα 4.3 «Ταξινόμηση Σωρού», ΕΚΔΟΣΕΙΣ ΤΖΙΟΛΑ, copyright 2005

[17] Π. Μποζάνης, Σχόλια πάνω στη δομή BATON και πρόταση μιας νέας απλής δενδρική δομής, 2006.

[18] "Fast construction of overlay networks", Dana Angluin, James Aspnes, Jianh Chen, Yinghua Wu, Yioting Yin. In Seventeenth Annual ACM Symposium on Parallelism in Algoritms and Architectues, pages 145-154, July 2005

[19] "Skip Lists: A Probabilistic Alternative to Balanced Trees", W. Pugh. Communications of the ACM, 33(6):668-676, June 1990.