ΠΑΝΕΠΗΣΤΙΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΥΚΤΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Επιστήμη και Τεχνολογία Υπολογιστών, Τηλεπικοινωνιών και Δικτύων

ΔΟΜΕΣ ΕΥΡΕΤΗΡΙΟΥ ΓΙΑ CLOUD COMPUTING

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

andpapad@inf.uth.gr

ΒΟΛΟΣ, ΙΟΥΝΙΟΣ 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΔΟΜΕΣ ΕΥΡΕΤΗΡΙΟΥ

ΓΙΑ

CLOUD COMPUTING

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

Παπαδόπουλος Ανδρέας

Επιβλέποντας: Κατσαρός Δημήτριος

Βόλος, Ιούνιος 2011

Περίληψη

Η ραγδαία ανάπτυξη της τεχνολογίας των ηλεκτρονικών υπολογιστών τα τελευταία χρόνια έχει οδηγήσει στην αύξηση της ταχύτητας μετάδοσης δεδομένων και των δυνατοτήτων των υπολογιστών. Λόγω των υψηλών ταχυτήτων σύνδεσης έχει ευνοηθεί και αναπτυχθεί η μελέτη και χρήση κατανεμημένων συστημάτων. Παράλληλα η ραγδαία ανάπτυξη του παγκόσμιου ιστού έχει δημιουργήσει την ανάγκη της γρήγορης και αποδοτικής επεξεργασίας πολύ μεγάλου όγκου δεδομένων (το Google επεξεργάζεται καθημερινά περίπου 20 petabytes δεδομένων). Για την απαραίτητη επεξεργασία δεν είναι πλέον δυνατή η χρήση ενός απλού υπολογιστή και πολλές φορές δεν είμαστε σε θέση να αποφασίσουμε εκ των προτέρων την υπολογιστική ισχύ που θα χρειαστούμε. Οι αλλαγές αυτές σε συνδυασμό με τις συνεχώς αυξανόμενες σύγχρονες ανάγκες έχουν οδηγήσει στην ανάπτυξη του cloud computing. Το cloud computing μπορεί απλά να περιγραφεί ως ένα σύνολο άπειρων δυνατοτήτων (επεξεργαστική ισχύ, χωρητικότητα κλπ) από το οποίο ο κάθε χρήστης χρησιμοποιεί ότι του είναι απαραίτητο για την κάλυψη των αναγκών του.

Η ισχύς και η αποδοτικότητα των συστημάτων αυτών σίγουρα είναι ένα θετικό στοιχείο αλλά δεν είναι επαρκής. Καθώς ο όγκος των δεδομένων είναι πολύ μεγάλος είναι επιθυμητή και αναπόφευκτη η δυνατότητα να μπορεί ο χρήστης να θέτει ερωτήματα και να λαμβάνει απάντηση σε σύντομο χρονικό διάστημα. Τέτοια ερωτήματα μπορεί να είναι κατά πόσο κάποιο σημείο υπάρχει στα διαθέσιμα δεδομένα ή ποια είναι τα διαθέσιμα δεδομένα σε κάποιο εύρος τιμών. Σε πολλές εφαρμογές όπως network monitoring και geographical information systems τα δεδομένα έχουν πολλές ιδιότητες (attributes) και ανήκουν σε μεγαλύτερες διαστάσεις. Μια τυπική πλατφόρμα cloud computing αποτελείται από εκατοντάδες ή και χιλιάδες από κόμβους (υπολογιστές χαμηλού κόστους) οι οποίοι είναι ικανοί να αποθηκεύσουν τεράστιο όγκο δεδομένων. Για την διαχείριση τόσο μεγάλου όγκου δεδομένων είμαστε αναγκασμένοι να αναπτύξουμε αποδοτικές δομές ευρετηρίου που θα μας επιτρέπουν την γρήγορη και παράλληλη αναζήτηση και θα απαιτούν όσο το δυνατόν λιγότερο αποθηκευτικό χώρο και υπολογιστική ισχύ.

Στα κεφάλαια που ακολουθούν περιγράφεται αναλυτικά το cloud computing καθώς και μια δομή ευρετηρίου που ανταποκρίνεται στις πιο πάνω ανάγκες. Επιπλέον ως αντιμετώπιση του προβλήματος προτείνεται μια αποδοτική δομή ευρετηρίου η οποία κλιμακώνει και μπορεί να χρησιμοποιηθεί σε πραγματικά συστήματα cloud computing. Σύμφωνα με τα πειράματα που έχουν γίνει η δομή αυτή είναι ικανή να ανταποκριθεί σε πολλά point και range queries με ελάχιστη καθυστέρηση. Η προτεινόμενη δομή βασίζεται σε R-Tree και Bloom Filters.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Cloud computing, Databases

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Cloud computing, multi-dimensional index, distributed index, query processing

Περιεχόμενα

Λίστα αλγορίθμων

Λίστα πινάκων

Λίστα σχημάτων

Κεφάλαιο 1 - Εισαγωγή

1.1. Εισαγωγή

Η ραγδαία ανάπτυξη των δικτύων υπολογιστών, και του παγκόσμιου ιστού, έχουν οδηγήσει και στην ανάπτυξη πολλών χρήσιμων εφαρμογών που παρέχουν διάφορες υπηρεσίες πάνω από το δίκτυο. Βασισμένα σε ένα μοντέλο παροχής υπηρεσιών το οποίο αναπτύχθηκε, παρά την ραγδαία ανάπτυξη της επιστήμης, της τεχνολογίας και της βιομηχανίας των ηλεκτρονικών υπολογιστών, μας δίνουν την δυνατότητα παροχής υπηρεσιών στους χρήστες όταν το χρειάζονται (on demand), όμοια με άλλες υπηρεσίες στην κοινωνία. Οι χρήστες πρέπει να πληρώσουν τους παρόχους μόνο όταν χρησιμοποιούν τις υπολογιστικές υπηρεσίες (computing services) που προσφέρουν. Επιπλέον οι καταναλωτές δεν χρειάζεται να επενδύσουν χρήματα και να αντιμετωπίσουν πολλές δυσκολίες αγοράζοντας και συντηρώντας το δικό τους εξοπλισμό.

Με βάση αυτό το μοντέλο οι χρήστες έχουν πρόσβαση σε διάφορες υπηρεσίες όταν τις χρειάζονται χωρίς να ενδιαφέρονται που βρίσκονται οι υποδομές και που στην πραγματικότητα θα γίνουν οι υπολογισμοί για να πάρουν τα επιθυμητά αποτελέσματα.

Το μοντέλο που μόλις περιγράφτηκε αναφερόταν ως *utility computing* και αργότερα ως *Cloud computing*[1]. Ο όρος *Cloud computing* περιγράφει την υποδομή ως ένα σύννεφο, *"Cloud"*, μέσω του οποίου απλοί χρήστες και εταιρίες έχουν πρόσβαση σε διάφορες εφαρμογές ως υπηρεσίες από οπουδήποτε στο κόσμο όταν το χρειάζονται. Το *Cloud computing* είναι η δυναμική παροχή υπολογιστικών υπηρεσιών ή πόρων και υποστηρίζεται από τα προηγμένης τεχνολογίας κέντρα δεδομένων (*data centers*) που χρησιμοποιούν τεχνολογία εικονικών μηχανών (Virtual Machines – VM - Virtualization) με στόχο την απομόνωση των διάφορων εφαρμογών[2]. Πολλές εταιρίες, ανάμεσα τους οι Google, Yahoo, IBM και Microsoft, αναπτύσσουν γρήγορα κέντρα δεδομένων (data centers) για να παρέχουν υπηρεσίες πάνω από το cloud (Cloud computing services). Οι προοπτικές από αυτή την τάση είναι μεγάλες και γίνονται ακόμα πιο κατανοητές από τη δήλωση «Τα κέντρα δεδομένων είναι ο ηλεκτρονικός υπολογιστής» - *«The Data Center Is The Computer»* από τον David Patterson [3].

Το *Cloud computing* παρέχει υποδομή (infrastructure), πλατφόρμα (platform), και λογισμικό (software - applications) σαν υπηρεσίες, διαθέσιμες στους καταναλωτές γνωστές και ως Infrastructure as a Service (IaaS), Platform as a Service (PaaS), και Software as a Service (SaaS) αντίστοιχα.

Λόγω της ύπαρξης του cloud computing, καινοτόμες ιδέες και εφαρμογές δεν θα απαιτούν να δαπανηθεί χρόνος για την αγορά και εγκατάσταση προηγμένου και ακριβού εξοπλισμού από τους προγραμματιστές ή τις εταιρίες ανάπτυξης λογισμικού[4].

1.2. Κίνητρο

Καθώς το cloud είναι μια ευέλικτη υποδομή, αποτελούμενη από πολλούς υπολογιστές διαφορετικών συνήθως δυνατοτήτων, όπου διαφορετικές και εύκολα μεταβαλλόμενες δυνατότητες παρέχονται στους χρήστες, πρέπει να εκμεταλλευόμαστε όσο καλύτερα γίνεται τις δυνατότητες του. Πρέπει αναπόφευκτα να γίνεται αποδοτική και γρήγορη διαχείριση των δεδομένων ούτως ώστε να μην καθυστερεί η απόκριση σε διάφορα ερωτήματα από τις εφαρμογές που τρέχουν στο cloud και έτσι να μπορούν να εξυπηρετηθούν παράλληλα πολλοί τελικοί χρήστες. Για να πετύχουμε αποδοτική διαχείριση των δεδομένων πρέπει η δομή ευρετηρίου που θα χρησιμοποιήσουμε να μας παρέχει χαμηλό κόστος συντήρησης, να κλιμακώνει, να μπορεί να απαντά σε πολλά ερωτήματα στην μονάδα του χρόνου και να μπορεί να εκτελείται παράλληλα εκμεταλλευόμενη τις δυνατότητες του data center [7].

Επιπλέον λαμβάνοντας υπόψη το γεγονός ότι η χρήση των πόρων που παρέχονται από κάποιο data center κοστίζει δεν είναι δυνατόν η υλοποίηση και έλεγχος των διάφορων αλγορίθμων – εφαρμογών να γίνει εύκολα και συστηματικά. Για το λόγο αυτό είναι πάρα πολύ χρήσιμο, έως αναπόφευκτο, να γίνεται προσομοίωση και ανάλυση των διάφορων αλγορίθμων πριν την χρήση τους στα data centers. Με την χρήση εργαλείων προσομοίωσης μπορούμε να αξιολογήσουμε τις υποθέσεις μας πριν την πραγματική υλοποίηση σε ένα περιβάλλον όπου εύκολα μπορούμε να αναπαράγουμε και να συγκρίνουμε τα αποτελέσματα χωρίς να ξοδέψουμε άσκοπα χρήματα για δοκιμές[6]. Ακόμα ένας σημαντικός λόγος να χρησιμοποιήσουμε εργαλεία προσομοίωσης είναι το γεγονός ότι η βελτιστοποίηση μπορεί να γίνει πριν την διάθεση τις εφαρμογής – αλγορίθμου στο cloud. Χωρίς την ύπαρξη και χρήση τέτοιων εργαλείων οι χρήστες και οι πάροχοι θα έπρεπε να βασίζονται σε θεωρητικές και συνήθως ανακριβής αποτιμήσεις ή σε αποτελέσματα από δοκιμές (trial and error) που οδηγούν σε ανεπιθύμητη συμπεριφορά και μείωση των εσόδων.

1.3. Στόχοι

Στόχος αυτής της μεταπτυχιακής διατριβής είναι η παρουσίαση μιας αποδοτικής δομής ευρετηρίου για διαχείριση δεδομένων σε ένα περιβάλλον cloud computing. Η προτεινόμενη δομή θα προσομοιωθεί και θα συγκριθεί με μια δομή ευρετηρίου βασισμένη σε KD-Trees και R-trees όπως προτάθηκε από τους Xiangyu Zhang, Jing Ai, Zhongyuan Wang, Jiaheng Lu και Xiaofeng Meng στην δημοσίευση τους An Efficient Multi-Dimensional Index for Cloud Data Management [7] .

1.4. Οργάνωση

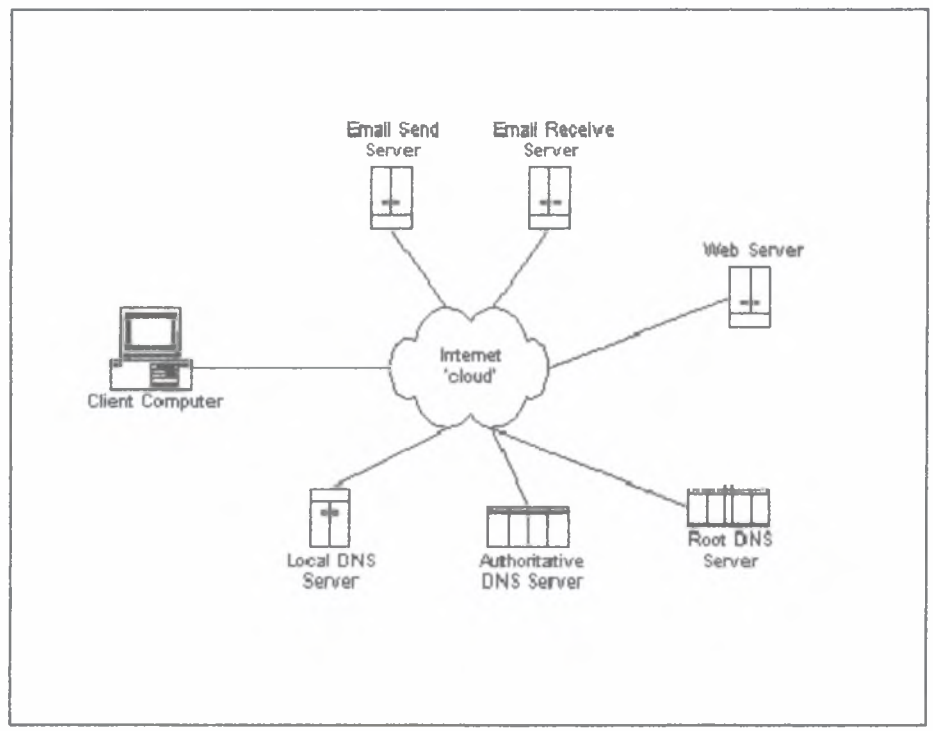
Στο επόμενο κεφάλαιο παρουσιάζεται και αναλύεται το Cloud Computing. Το κεφάλαιο 3 αναφέρεται στο πρόβλημα της διαχείρισης του μεγάλου όγκου δεδομένα στο cloud computing καθώς και αλγόριθμοι που έχουν προταθεί. Στην συνέχεια στο κεφάλαιο 4 περιγράφεται ο πρώτος αλγόριθμος που θα προσομοιωθεί [7] και στο κεφάλαιο 5 προτείνεται και παρουσιάζεται μια αποδοτική δομή ευρετηρίου. Ακολουθεί η περιγραφή της υλοποίησης - προσομοίωσης στο κεφάλαιο 6 και η πειραματική αξιολόγηση στο κεφάλαιο 7. Στο κεφάλαιο 8 παρατίθενται τα συμπεράσματα και προτείνονται κάποιες βελτιώσεις και μελλοντικές επεκτάσεις στην παρούσα εργασία.

Κεφάλαιο 2 - Cloud Computing

2.1. Εισαγωγή

Σε αυτό το κεφάλαιο αναλύεται η προέλευση του cloud computing καθώς και η εξέλιξη και η χρήση του σήμερα.

Για αρκετά χρόνια, και ακόμη και σήμερα, στην αναπαράσταση δικτύων χρησιμοποιείται το σχήμα του σύννεφου για να αναπαραστήσουμε το διαδίκτυο ή ένα «άγνωστο - ασαφή» ενδιάμεσο δίκτυο το οποίο όμως απαραίτητα πρέπει να συμπεριλάβουμε στα διαγράμματα μας. Όλες οι γραμμές – συνδέσεις παρουσίαζαν την μεταφορά δεδομένων μέσα στο απροσδιόριστο αυτό δίκτυο.



Σχήμα 2.1. Τυπικό διάγραμμα δικτύου (Jeffrey, 2009)

Πλέον με την ανάπτυξη του Cloud Computing οι γραμμές δεν αναπαριστούν το γεγονός ότι δεδομένα – πληροφορίες περνάνε μέσα από αυτό το δίκτυο αλλά συνδέουν τους υπολογιστές - χρήστες σε αυτό το «σύννεφο». Οι πληροφορίες, τα αρχεία και οι εφαρμογές που παραδοσιακά ήταν αποθηκευμένα στον υπολογιστή του χρήστη ή σε κάποιο τοπικό εξυπηρετητή πλέον προσπελάζονται, αποθηκεύονται και δημιουργούνται στο cloud. Έτσι οι χρήστες έχουν πρόσβαση σε αυτά από διαφορετικές τοποθεσίες και με διαφορετικούς τρόπους.

2.2. Ιστορική αναδρομή

Το μοντέλο πίσω από το cloud computing δεν είναι μια νέα επαναστατική έννοια. Το 1960 ο John McCarthy διατύπωσε την θεωρία ότι η χρήση ηλεκτρονικών υπολογιστών για υπολογισμούς ίσως μια μέρα να οργανωθεί σαν οργανισμοί που θα κατέχουν και θα συντηρούν τις υποδομές για μια δημόσια υπηρεσία - *"computation may someday be organized as a public utility."*

Σχήμα 2.2. Διάγραμμα δικτύου Cloud Computing ("DOE Deploys Cloud Computing," 2010)

Παρά το γεγονός αυτό, στα τελευταία χρόνια πολλά στοιχεία έχουν συνδυαστεί και δημιούργησαν την κατάλληλη υποδομή όπου το cloud computing μπορεί να αναπτυχθεί (π.χ. τεχνολογικά άλματα, Web 2.0). Για πολλά χρόνια το μεγαλύτερο εμπόδιο στην ανάπτυξη του cloud computing ήταν η ταχύτητα του δικτύου που δεν ευνοούσε την ανάπτυξη μεγάλων κέντρων υπολογιστών (large scale cloud computing). Μετά το 1990 και το dotcom boom όπου έγιναν μεγάλες επενδύσεις σε υποδομές με υψηλές ταχύτητες, με χρήση οπτικών ινών, ο αριθμός χρηστών με υψηλές ταχύτητες πρόσβασης στο διαδίκτυο αυξήθηκε.

Μετά το τέλος αυτής της εποχής, το 2000, εδραιώθηκε το μοντέλο του cloud computing στις επιχειρήσεις. Καινούριες εταιρίες έπρεπε να αγοράσουν ακριβό εξοπλισμό, λογισμικό και άδειες χρήσης για να αναπτύξουν την επιχείρηση τους. Το δίλημμα που επικρατούσε ήταν κατά πόσο να επενδύσουν στον εξοπλισμό καθώς μη επαρκή υλικό θα οδηγούσε σε δυσανασχέτηση των πελατών – χρηστών ενώ σε αντίθεση υπεραρκετός εξοπλισμός θα οδηγούσε σε σπατάλη χρόνου και χρήματος και τα κεφάλαια ήταν περιορισμένα έως ανύπαρκτα. Την ίδια περίοδο μεγάλες εταιρίες στο διαδίκτυο, όπως η Google και η Amazon, διέθεταν μεγάλα και ισχυρά υπολογιστικά κέντρα για να ικανοποιήσουμε τις απαιτήσεις τους, η Google για να ενημερώνει τα ευρετήρια της για το γρήγορα αναπτυσσόμενο παγκόσμιο ιστό και παράλληλα να δίνει αποτελέσματα στις αναζητήσεις των χρηστών και η Amazon για να εδραιώσει την συνεχώς αναπτυσσόμενη παρουσία της στο χώρο του ηλεκτρονικού εμπορίου, καθώς αυτές προέκυπταν ειδικά σε ώρες αιχμής. Έχοντας στην διάθεση τους πανίσχυρα για την εποχή υπολογιστικά κέντρα, τα οποία

χρησιμοποιούταν αποδοτικά μόνο σε ώρες ακμής, οδηγούσε σε μη αποδοτική χρησιμοποίηση του υλικού τον περισσότερο χρόνο.

Παρά το γεγονός ότι ο ισχυρός εξοπλισμός τους και τα συστήματα τους δεν χρησιμοποιούταν αποδοτικά έπρεπε να συντηρούνται, από ειδικά εκπαιδευμένο προσωπικό, να τροφοδοτούνται και να ψύχονται δαπανώντας έτσι τεράστια ποσά για να λειτουργήσουν.

Σχήμα 2.3. Large Data Center ("Computer History," 2010)

Το 2006 η Amazon, προσπαθώντας να αναπτυχθεί και να αξιοποιήσει καλύτερα την μεγάλη υπολογιστική ισχύ που διέθετε, και εκμεταλλευόμενη την ήδη υπάρχουσα υψηλής ταχύτητας σύνδεση που υπήρχε ανάμεσα στα υπολογιστικά της κέντρα και το διαδίκτυο, ξεκίνησε να ενοικιάζει υπολογιστική ισχύ και αποθηκευτικό χώρο. Η υπηρεσία αυτή έγινε γρήγορα αποδεκτή και άρχισε να χρησιμοποιείται από εταιρίες, ειδικά πρωτοεμφανιζόμενες, καθώς ήταν η λύση στο δίλημμα που αντιμετώπιζαν. Πλέον ήταν σε θέση να χρησιμοποιήσουν εξοπλισμό, τον οποίο δεν θα μπορούσαν να αποκτήσουν, πληρώνοντας μόνο για ότι χρειάζονται – χρησιμοποιούν. Περίπου το 2006 ήταν και η είσοδος της Google, παράλληλα με την Amazon, στο cloud computing και συγκεκριμένα στο Infrastructure as a Service (IaaS).

Σχήμα 2.4. A Google data center, το 2000. (Credit: Stephen Shankland-CNET News.com/Jeff Dean-Google)

Η βασική χρήση του διαδικτύου έχει αλλάξει αρκετά, σε σχέση με τα προηγούμενα χρόνια, σε μεγάλο βαθμό λόγο της ανάπτυξης στην σχεδίαση λογισμικού και την διάδοση των προτύπων (standards). Αυτή η αλλαγή οδήγησε στην γρήγορη ανάπτυξη εφαρμογών, κυρίως σε υπηρεσίες διαδικτύου (Web Services), που με την σειρά τους συνέβαλαν δραστικά στην ανάπτυξη των social media. Σελίδες κοινωνικής δικτύωσης, όπως MySpace, FaceBook, YouTube, Twitter κ.α., έχουν γίνει πασίγνωστα και χρησιμοποιούνται από εκατομμύρια χρήστες στο κόσμο. Άλλες γνωστές υπηρεσίες είναι τα Google Docs, Office Live ή ακόμα και online λειτουργικά συστήματα, π.χ. eyeOS, iCloud, GlideOS, Jolicloud, ajaxWindows κ.α. με χαρακτηριστικό παράδειγμα την διάθεση στην αγορά του Chrome Book από την Google το 2011. Οι χρήστες πλέον φαίνεται να έχουν συνηθίσει και δεν έχουν πρόβλημα στο να αποθηκεύσουν και να επεξεργαστούν τα δεδομένα κάπου αλλού εκτός από τον προσωπικό τους υπολογιστή.

Όλα τα πιο πάνω, μαζί με άλλους παράγοντες όπως η οικονομική κρίση που ξεκίνησε το 2007, αποκορύφωσαν το σημερινό ενδιαφέρον στο cloud computing.

2.3. Ορισμός

Το Cloud Computing είναι ένας γενικός όρος που αναφέρεται σε οτιδήποτε προσφέρει υπηρεσίες πάνω από το διαδίκτυο. Αυτές οι υπηρεσίες χωρίζονται σε τρεις βασικές κατηγορίες: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) και Software-as-a-Service (SaaS). Το όνομα cloud computing είναι εμπνευσμένο από το σύννεφο που αναπαριστά το διαδίκτυο σε γραφήματα και σχεδιαγράμματα.[10]

Το Cloud Computing είναι η παροχή υπηρεσιών, υπολογισμών, εφαρμογών, πρόσβαση και αποθήκευση δεδομένων, χωρίς ο τελικός χρήστης να χρειάζεται να γνωρίζει οτιδήποτε για την τοποθεσία ή τις ρυθμίσεις των υποδομών του συστήματος που προσφέρει τις υπηρεσίες. Μπορεί κανείς να το παρομοιάσει με το δίκτυο παροχής ηλεκτρισμού, όπου οι χρήστες χρησιμοποιούν τον ηλεκτρισμό χωρίς να χρειάζεται, και πολλές φορές, να γνωρίζουν πως παράγεται και ποια υποδομή χρησιμοποιείται για την μεταφορά ή οποιεσδήποτε επιπλέον πληροφορίες αφορούν την υπηρεσία.

Το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας - National Institute of Standards and Technology(NIST) διατύπωσε το πιο κάτω ορισμό:

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential **characteristics,** three **service models**, and four **deployment models**."[9]

Πιο απλά το cloud computing πρόκειται για τη χρήση των υπηρεσιών του Web για τις ανάγκες των χρηστών, η οποία μπορεί να περιλαμβάνει τη χρήση εφαρμογών λογισμικού, την αποθήκευση δεδομένων ή ακόμα και τη χρήση μίας πλατφόρμας για τη δημιουργία εφαρμογών.

Παρόλο που κάποιοι χρήστες μπορεί να μην είναι εξοικειωμένοι με τον όρο, στη πραγματικότητα και στην καθημερινότητά μας ήδη εκμεταλλευόμαστε τα πλεονεκτήματα του cloud computing. Οι υπηρεσίες που παρέχει το cloud computing μέσω των εφαρμογών του Web 2.0 συμπεριλαμβάνουν:

- ηλεκτρονικό ταχυδρομείο (για παράδειγμα, το Gmail, Hotmail και Yahoo!)
- διανομή φωτογραφιών και βίντεο (π.χ., Flickr και το YouTube)
- διαδικτυακές εφαρμογές (π.χ., Google Docs και το Adobe Express)
- αποθήκευση των αρχείων δεδομένων και την επεξεργασία μεγάλου όγκου δεδομένων (π.χ. Amazon S3)
- σελίδες κοινωνικής δικτύωσης (π.χ. Facebook, MySpace, Tweeter)
- ανάπτυξη εφαρμογών (π.χ. Google App Engine)

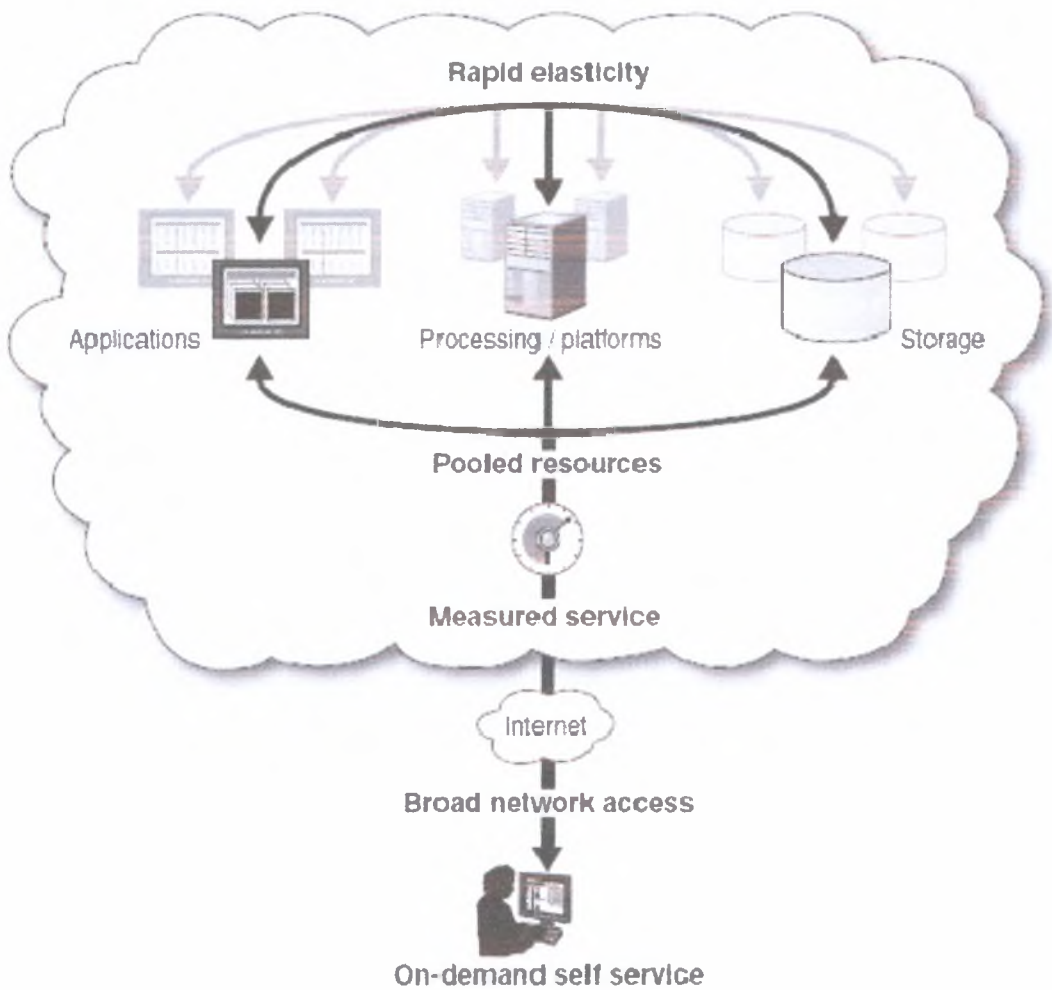
2.4. Απαραίτητα χαρακτηριστικά

2.4.1. Εισαγωγή

Όπως αναφέρεται και στον ορισμό πιο πάνω ένα περιβάλλον cloud computing πρέπει να χαρακτηρίζεται από πέντε απαραίτητα χαρακτηριστικά. Τα απαραίτητα αυτά χαρακτηριστικά παρουσιάζονται στο πιο κάτω πίνακα και στο Σχήμα 2.5.

Γρήγορη Ελαστικότητα - Rapid elasticity
Διαθεσιμότητα πόρων - Resource pooling.
Measured Service
Broad network access
On-demand self-service

Πίνακας 2.1. Απαραίτητα χαρακτηριστικά ενός συστήματος Cloud Computing



Σχήμα 2.5. Απαραίτητα χαρακτηριστικά Cloud Computing ("Federal Guidance Needed to Address Control Issues with Implementing Cloud Computing," 2010)

2.4.2. Rapid Elasticity

Ο όρος rapid elasticity αναφέρεται στο απαραίτητο χαρακτηριστικό της κλιμάκωσης, προς τα πάνω ή προς τα κάτω, όταν χρειάζεται. Πόροι που μπορούν να κλιμακώνουν ανάλογα των

απαιτήσεων του χρήστη είναι ο αριθμός των επεξεργαστών, ταχύτητα σύνδεσης πάνω από το δίκτυο, αποθηκευτικός χώρος καθώς και ο αριθμός των στιγμιότυπων των εφαρμογών. Η κλιμάκωση αυτή είναι σημαντικό να πραγματοποιείται σε σύντομο χρονικό διάστημα.

Ο χρήστης πρέπει να είναι σε θέση να προμηθευτεί – εκμεταλλευτεί ακόμα ένα εξυπηρετητή (server) όταν τον χρειάζεται χωρίς να πρέπει να ενδιαφερθεί πως αυτός ο εξυπηρετητής θα τροφοδοτηθεί, ψυχθεί ή θα τοποθετηθεί στο χώρο και που. Το μόνο που στην πραγματικότητα πρέπει να κάνει ο χρήστης, που θεωρητικά βλέπει το cloud να αποτελείται από άπειρο εξοπλισμό, είναι να πληρώσει και να εφοδιαστεί με όσες υπολογιστικές μονάδες υπηρεσιών θεωρεί ότι του είναι αναγκαίες όταν και όποτε το θελήσει.

2.4.3. Resource Pooling

Με τον όρο Resource Pooling αναφερόμαστε σε άλλο ένα απαραίτητο χαρακτηριστικό. Οι πόροι (resources) ενός παροχέα υπηρεσιών πρέπει να είναι οργανωμένοι έτσι ώστε να είναι σε θέση να εξυπηρετήσει – ενοικιάσει τις υποδομές του δυναμικά σε πολλούς χρήστες ταυτόχρονα. Αυτή είναι η σημασία της ανεξάρτητης τοποθεσίας (location independence) στην οποία ο χρήστης δεν μπορεί να επέμβει, καθώς δεν έχει έλεγχο ούτε γνωρίζει την ακριβή τοποθεσία του εξοπλισμού.

Οι πόροι μπορούν να είναι αποθηκευτικός χώρος, επεξεργαστική ισχύ, μνήμη, εύρος δικτύου ή εικονικές μηχανές. Η χρήση των πόρων αυτών ταυτόχρονα από πολλούς πελάτες είναι που δίνει νόημα στο κομμάτι «ως υπηρεσία (as a service)» του cloud computing καθιστώντας το σαν μια προσφερόμενη επιχειρηματική ευκαιρία. Εξυπηρετώντας ταυτόχρονα πολλούς πελάτες, χρησιμοποιώντας την ίδια πλατφόρμα, μειώνεται το κόστος συντήρησης και το κόστος ανά χρήστη παραμένει χαμηλό οδηγώντας έτσι σε κλιμακωτή ανάπτυξη των οικονομικών της επιχείρησης.

2.4.4. Measured Service

Ο όρος αναφέρεται στην παρακολούθηση και τον έλεγχο από τον παροχέα όλων των πτυχών που αφορούν την προσφερόμενη cloud υπηρεσία.

Τα συστήματα cloud computing αυτόματα ελέγχουν και βελτιστοποιούν την χρήση των πόρων χρησιμοποιώντας κάποιο σύστημα μετρήσεων σε αφηρημένο επίπεδο κατάλληλο για τον τύπο της προσφερόμενης υπηρεσίας (π.χ. αποθήκευση, επεξεργασία, εύρος ζώνης, ενεργοί χρήστες). Η χρησιμοποίηση των πόρων από τις ενεργές υπηρεσίες μπορεί να παρακολουθηθεί και να παρουσιαστεί και στο παροχέα και στον χρήστη με διαφάνεια. Η καταγραφή της χρήσης πόρων

είναι ζωτική σημασίας για τον έλεγχο προσπέλασης, την βελτιστοποίηση και την κοστολόγηση υπηρεσιών και άλλων σημαντικών εργασιών.

2.4.5. Broad Network Access

Οι υπηρεσίες είναι διαθέσιμες μέσω δικτύου και μπορούν να χρησιμοποιηθούν αποκτώντας πρόσβαση μέσω απλών συσκευών(π.χ. κινητά, φορητούς υπολογιστές και PDAs).

2.4.6. On-Demand Self Service

Ακόμα ένα απαραίτητο χαρακτηριστικό είναι η δέσμευση υπολογιστικών πόρων, όπως αποθηκευτικός χώρος και υπολογιστική ισχύ, όταν χρειάζεται να μπορεί να γίνει από τον πελάτη χωρίς να χρειάζεται η ανθρώπινη παρέμβαση από το μέρος του παροχέα υπηρεσιών.

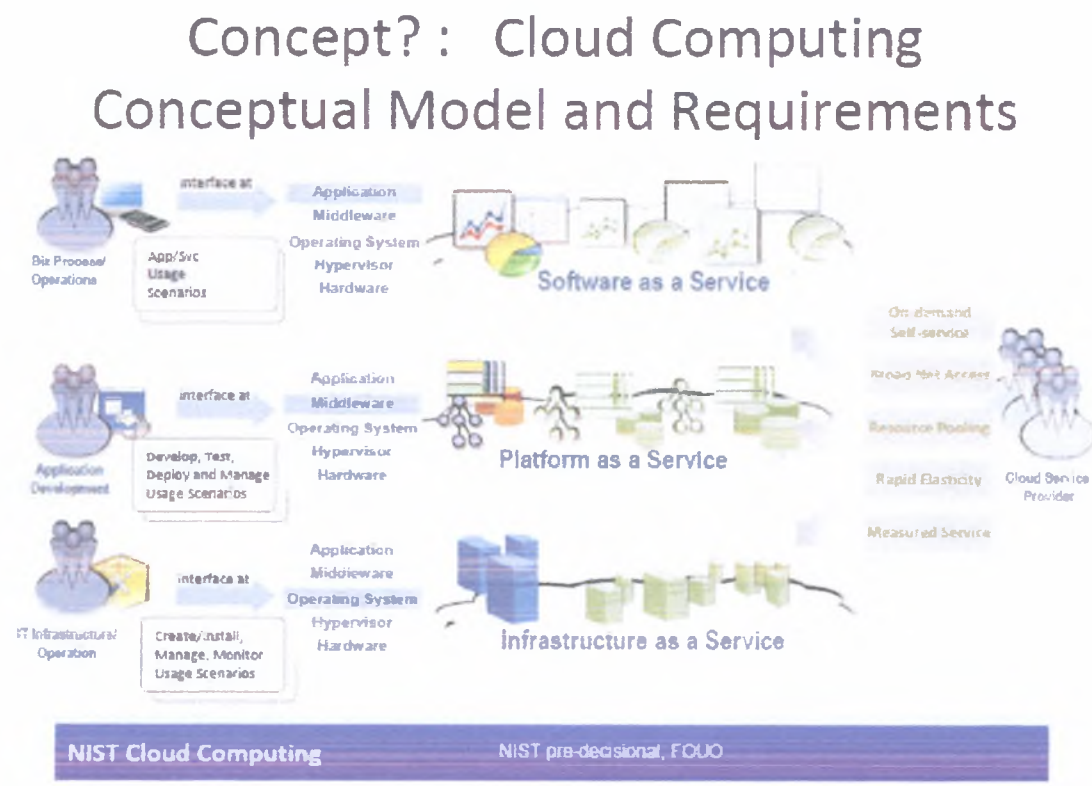
2.5. Μοντέλα παροχής υπηρεσιών

2.5.1. Εισαγωγή

Το cloud computing αποτελείται από τρία βασικά μοντέλα παροχής υπηρεσιών (deployment models) τα οποία παρουσιάζονται στο πιο κάτω πίνακα και στο Σχήμα 2.6.

Infrastructure as a Service (IaaS)
Platform as a Service (PaaS)
Software as a Service (SaaS)

Πίνακας 2.2. Μοντέλα παροχής υπηρεσιών στο Cloud Computing



Σχήμα 2.6. Μοντέλα παροχής υπηρεσιών στο Cloud Computing

Κάθε μοντέλο υπηρεσιών του cloud computing εξυπηρετεί διαφορετικές ανάγκες αλλά όλα μοιράζονται κοινά χαρακτηριστικά όπως μοντέλο αγορών pay-as-you-go ή pay-as-you-use-it, απαιτήσεις κλιμάκωσης, την ανάγκη χρήσης από πολλούς χρήστες ταυτόχρονα και virtualization.

Η συνδυασμένη χρήση και των τριών μοντέλων αναφέρεται ως το μοντέλο SPI (SPI model (SaaS, PaaS, IaaS)).

2.5.2. Infrastructure as a Service (IaaS)

Ο όρος Infrastructure as a Service (IaaS) αναφέρεται στην προσφορά βασικών πόρων (επεξεργαστική ισχύ, εξυπηρετητές, αποθηκευτικός χώρος, εξοπλισμός δικτύου και εύρος ζώνης) σαν υπηρεσία. Στο μοντέλο αυτό η συνηθισμένη πολιτική χρέωσης είναι pay as you use που σημαίνει ότι ο χρήστης πληρώνει για ότι ζήτησε προς μίσθωση (υπολογιστική ισχύ, αποθήκευση κλπ.) προηγουμένως είτε με μακροχρόνιο συμβόλαιο είτε με συνδρομή. Ο καταναλωτής έχει την ικανότητα να εκμεταλλευτεί τον εξοπλισμό για να εκτελέσει οποιοδήποτε λογισμικό επιθυμεί που μπορεί να είναι ακόμα και κάποιο λειτουργικό σύστημα. Παρ' όλα αυτά δεν έχει κανένα έλεγχο στην υποδομή που χρησιμοποιείται από τον παροχέα (cloud infrastructure) αλλά μόνο τον πλήρη έλεγχο σε επίπεδο εφαρμογών και μερικές φορές περιορισμένο έλεγχο σε στοιχεία δικτύου όπως ισορροπιστές φόρτου (load balancers) και τοίχοι ασφαλείας (firewalls).

2.5.3. Platform as a Service (PaaS)

Ο όρος Platform as a Service (PaaS) αναφέρεται στην προσφορά μιας υπολογιστικής πλατφόρμας σαν υπηρεσία. Η πλατφόρμα συνήθως είναι μια εφαρμογή πλαισίου (application framework) που τυπικά καταναλώνει πόρους της cloud υποδομής και υποστηρίζει άλλες εφαρμογές (cloud applications). Ο καταναλωτής έχει την ικανότητα να αναπτύξει δικές του εφαρμογές με βάση τις γλώσσες και τα εργαλεία που είναι διαθέσιμα από τον παροχέα. Το PaaS παρέχει την διευκόλυνση της ανάπτυξης εφαρμογών χωρίς το κόστος και την πολυπλοκότητα της αγοράς και συντήρησης του απαραίτητου υλικού και λογισμικού. Ο χρήστης δεν έχει απολύτως κανένα έλεγχο στην υποδομή που χρησιμοποιείται από τον παροχέα (cloud infrastructure) αλλά μπορεί να έχει πλήρη έλεγχο στις εφαρμογές του και πιθανώς και στις ρυθμίσεις του περιβάλλοντος εκτέλεσης τους.

2.5.4. Software as a Service (SaaS)

Σε αυτή την κατηγορία ο χρήστης έχει την ικανότητα να χρησιμοποιήσει τις εφαρμογές όπως αυτές προσφέρονται από τον παροχέα. Οι εφαρμογές αυτές μπορούν να προσπελαστούν και να χρησιμοποιηθούν από απλές συσκευές (κινητά κλπ.) μέσω μιας ορισμένης διεπαφής όπως για παράδειγμα ενός web browser(π.χ. Web-based ηλεκτρονικό ταχυδρομείο). Ο χρήστης δεν έχει απολύτως κανένα έλεγχο στην υποδομή που χρησιμοποιείται από τον παροχέα (cloud infrastructure) ούτε στις προσφερόμενες εφαρμογές παρά μόνο από εξατομικευμένες περιορισμένες επιλογές για τις εφαρμογές.

2.5.5. Anything as a service (XaaS)

Το XaaS είναι ένας συγκεντρωτικός όρος που αναφέρεται σε οτιδήποτε ως υπηρεσία – "X as a service", με βάση την συνεχή και ραγδαία ανάπτυξη των υπηρεσιών που προσφέρονται στο διαδίκτυο. Το XaaS είναι η ουσία του Cloud Computing.

Τα μοντέλα που αναφέρθηκαν πιο πάνω, Software as a Service (SaaS), Infrastructure as a Service (IaaS) και Platform as a Service (PaaS), είναι τα πιο κοντινά παραδείγματα στο Anything as a service (XaaS). Άλλα παραδείγματα του Anything as a service (XaaS) είναι storage as a service (SaaS), communications as a service (CaaS), network as a service (NaaS) και monitoring as a service (MaaS).

2.6. Μοντέλα ανάπτυξης - Deployment Models

2.6.1. Εισαγωγή

Λόγω των διαφορετικών αναγκών και απόψεων από πλευράς οργανισμών έχουν διατυπωθεί και αναπτυχθεί διαφορετικά μοντέλα που χαρακτηρίζουν το cloud computing. Σύμφωνα με την διατύπωση του NIST τα μοντέλα αυτά είναι τέσσερα και παρουσιάζονται στο πιο κάτω πίνακα.

Ιδιωτικό - Private Cloud
Community Cloud
Δημόσιο - Public Cloud
Hybrid Cloud

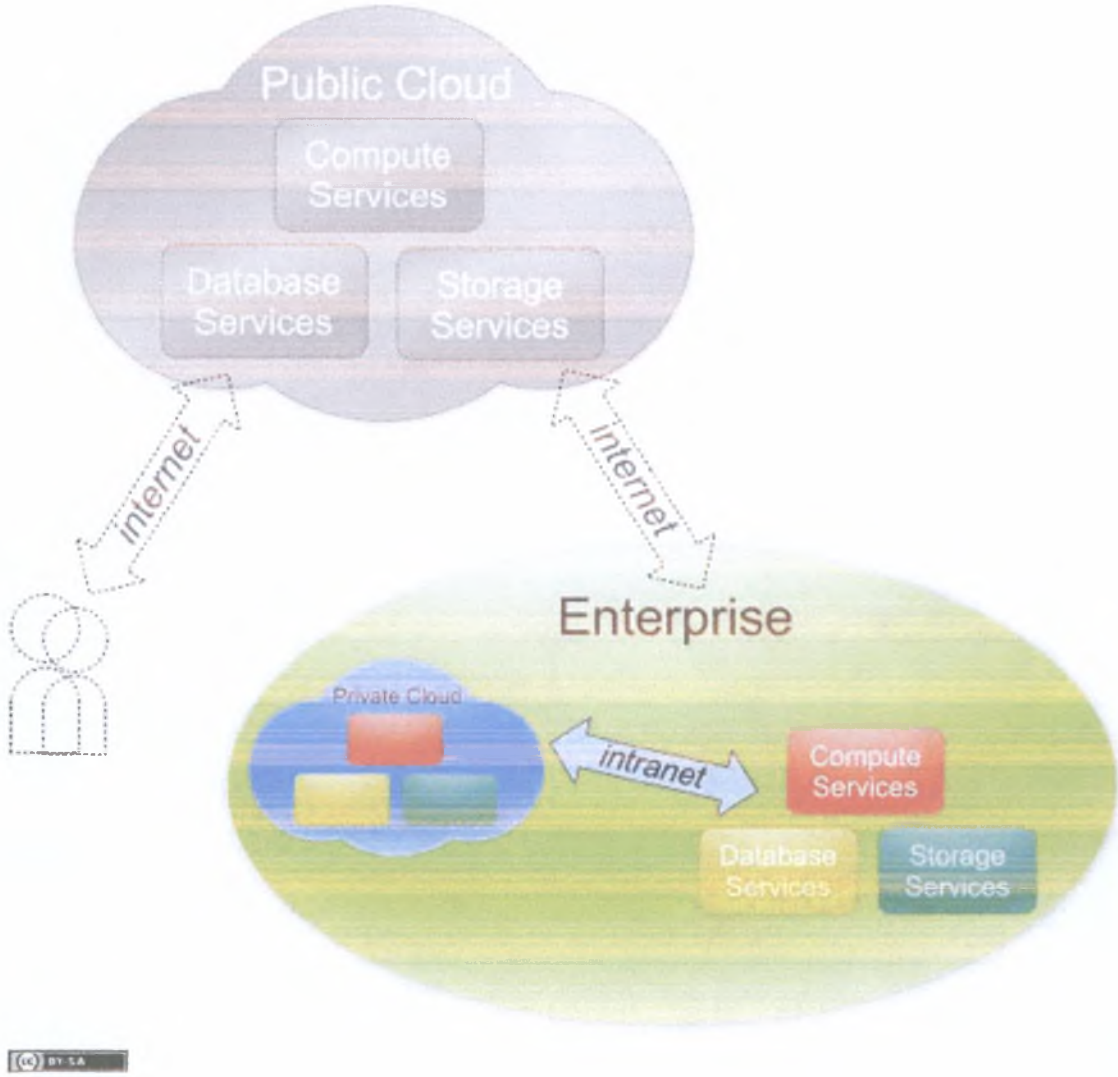
Πίνακας 2.3. Μοντέλα ανάπτυξης Cloud Computing

Κάθε μοντέλο έχει τα δικά του χαρακτηριστικά τα οποία διαφοροποιούνται συνήθως με βάση τις απαιτήσεις ασφαλείας για κάθε μοντέλο.

2.6.2. Ιδιωτικό - Private Cloud

Το **Private Cloud** αποτελεί ένα σύνολο από υπολογιστικούς πόρους που προσφέρονται ως ένα προτυποποιημένο σύνολο υπηρεσιών οι οποίες καθορίζονται, σχεδιάζονται και ελέγχονται από ένα συγκεκριμένο οργανισμό. Η επιλογή ανάπτυξης ενός Private Cloud συνήθως καθοδηγείται από την ανάγκη για τη διατήρηση του πλήρους ελέγχου ενός παραγωγικού περιβάλλοντος εξ' αιτίας ιδιαίτερων απαιτήσεων των εφαρμογών από πλευράς απόδοσης, ωριμότητας ή νομικού πλαισίου λειτουργίας. Σημαντικό χαρακτηριστικό του είναι πολύ υψηλό κόστος απόκτησης και λειτουργίας του. Το Private cloud συχνά συγχέεται με το Virtualization, το όποιο όμως αποτελεί μόνο ένα μικρό μέρος αυτού, αφού ακόμα και ως private θα πρέπει να έχει τα χαρακτηριστικά αυτόματης ανάκαμψης, αυτό-επιτήρησης, αυτό-διαχείρισης, αυτόματης επαναδιαμόρφωσης, δυνατότητας καθορισμού SLAs, και δυνατότητες (αυτό)κλιμάκωσης. Είναι αξιοσημείωτο ότι και πάλι τα δεδομένα μπορεί να ταξιδεύουν πάνω από το δίκτυο.

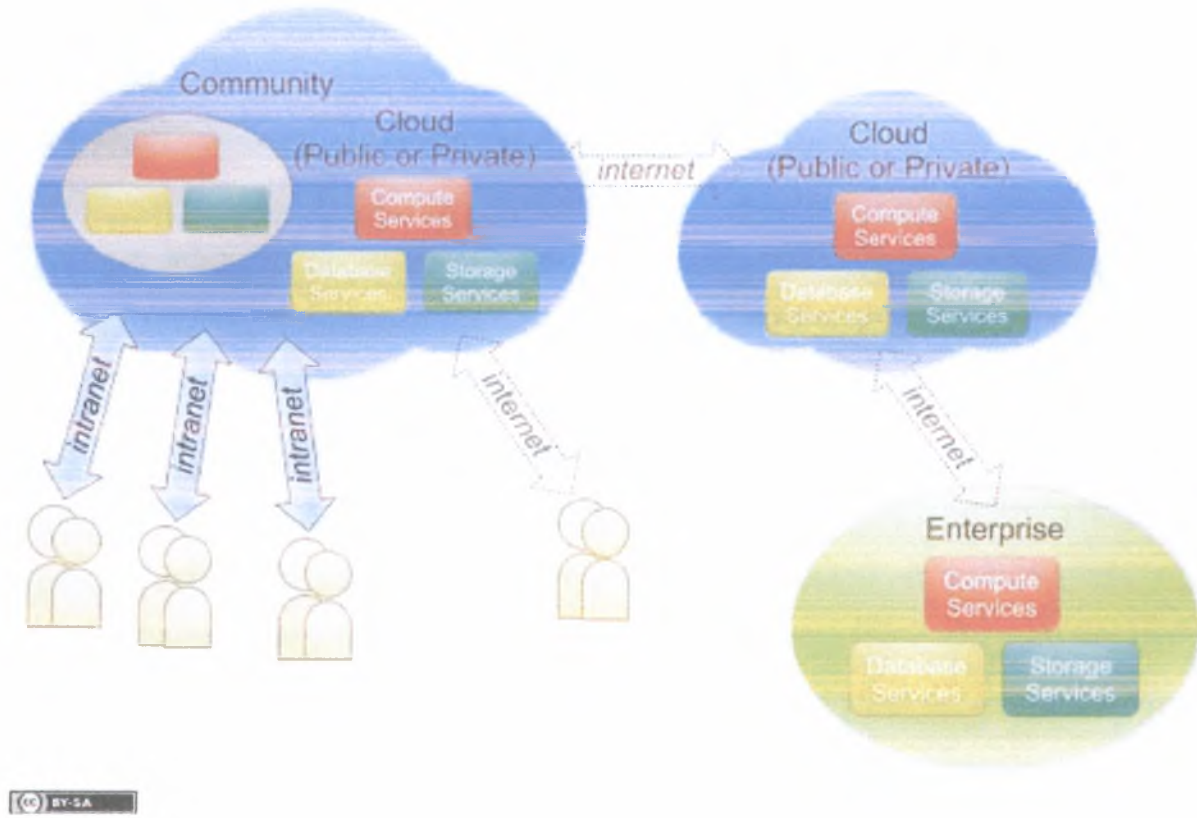
Η Microsoft μέσω του Dynamic Datacenter Toolkit προσφέρει τη δυνατότητα ανάπτυξης Private Clouds.



Σχήμα 2.7. Private Cloud ("Cloud Computing Use Cases White Paper v3.0," 2010)

2.6.1. Community Cloud

Το **Community Cloud** αποτελεί ένα σύνολο από υπολογιστικούς πόρους που διαμοιράζονται μεταξύ συγκεκριμένων οργανισμών για να υποστηρίξουν μια συγκεκριμένη κοινότητα με κοινά ενδιαφέροντα και ανησυχίες ή μια κοινή αποστολή. Μπορεί να διαχειρίζεται από τους ίδιους τους οργανισμούς ή από κάποιο τρίτο παροχέα. Όλα τα μέλη της κοινότητας έχουν πρόσβαση σε κοινά δεδομένα και εφαρμογές.

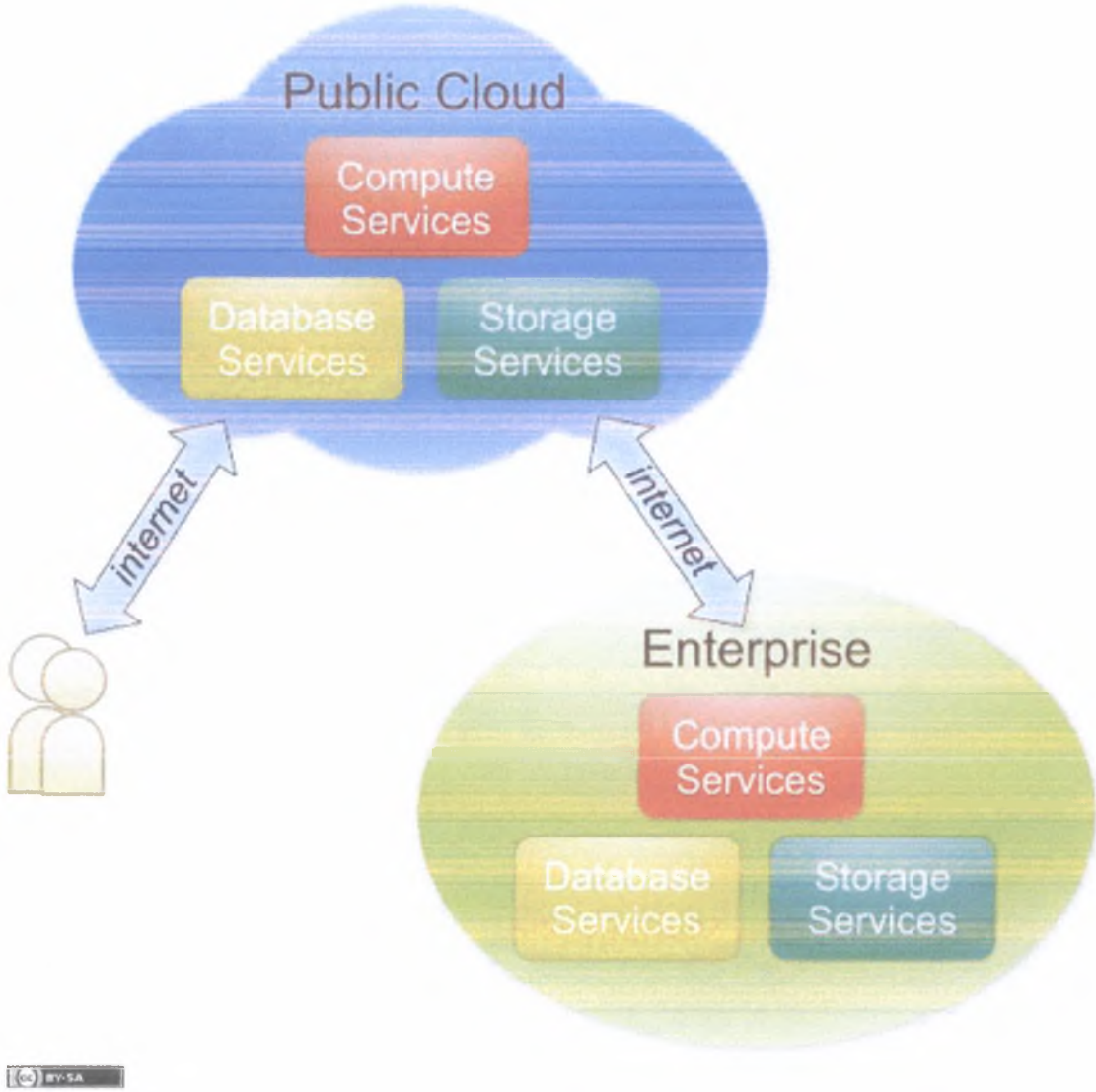


Σχήμα 2.8. Community Cloud ("Cloud Computing Use Cases White Paper v3.0," 2010)

2.6.2. Δημόσιο - Public Cloud

Το **Public cloud** αποτελεί ένα σύνολο από υπολογιστικούς πόρους οι οποίοι διατίθενται πάνω από το διαδίκτυο προς το ευρύ κοινό. Προσφέρονται από έναν πάροχο συνήθως με μοντέλο «pay as you go». Το Public Cloud computing έχει τα ακόλουθα πλεονεκτήματα: Η χρέωση της υπηρεσίας είναι για ότι χρησιμοποιηθεί, μεγάλη ευελιξία λόγω της άμεσης διάθεσης υπηρεσιών, υπάρχει άμεση κλιμάκωση σε μεγαλύτερη ή μικρότερη χωρητικότητα σε μόλις μερικά λεπτά, και όλες οι υπηρεσίες προσφέρονται με βελτιωμένη και συνεχή διαθεσιμότητα, ελαστικότητα, ασφάλεια και διαχειρισιμότητα. Σημειώνεται ότι στο public cloud δεν σημαίνει ότι τα δεδομένα των χρηστών είναι διαθέσιμα και ορατά προς όλους και μάλιστα πολλοί πάροχοι προσφέρουν στον χρήστη τον έλεγχο για πρόσβαση στα δεδομένα του χρησιμοποιώντας μηχανισμούς ελέγχου πρόσβασης.

Η Microsoft προσφέρει τις υπηρεσίες BPOS-S & Windows Azure Platform, ενώ σύντομα και το CRM Online, για την ανάπτυξη public cloud.

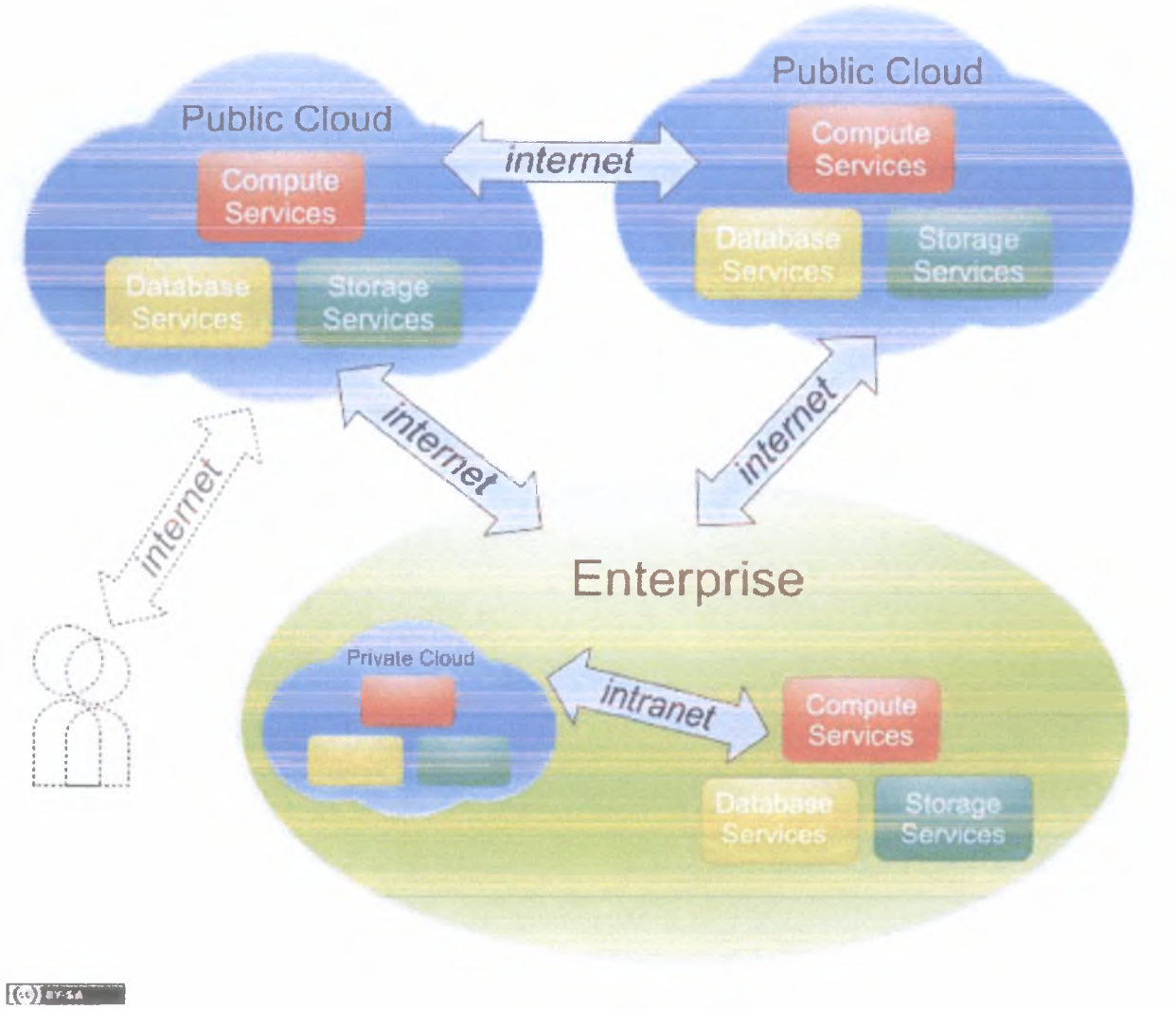


Σχήμα 2.9. Public Cloud ("Cloud Computing Use Cases White Paper v3.0,"2010)

2.6.3. Υβριδικό - Hybrid Cloud

Το **Hybrid cloud** είναι ένας συνδυασμός από δύο ή περισσότερα μοντέλα cloud (private, community ή public) τα οποία συνδέονται ισχυρά μεταξύ τους βάση προτυποποιημένων τεχνολογιών που επιτρέπουν την φορητότητα των δεδομένων και των εφαρμογών (π.χ. cloud bursting για εξισορρόπηση του φόρτου εργασίας).

Το υβριδικό μοντέλο χρησιμοποιείται αρκετά από μεγάλους οργανισμούς που θέλουν να εκμεταλλευτούν τις δυνατότητες του cloud computing και ταυτόχρονα να έχουν πλήρη έλεγχο στο υλικό και λογισμικό για τα πολύ σημαντικά δεδομένα τους και τις κρίσιμες εργασίες τους χωρίς να βασίζονται πλήρως στην επιτυχία κάποιου συγκεκριμένου παροχέα.



Σχήμα 2.10. Hybrid Cloud ("Cloud Computing Use Cases White Paper v3.0," 2010)

2.7. Πλεονεκτήματα του Cloud Computing

Το cloud computing δουλεύει με παρόμοια με τις υπόλοιπες υπηρεσίες που παρέχονται σήμερα στην κοινωνία. Για παράδειγμα στα παλιά χρόνια κάθε χωριό ή σπίτι είχε το δικό του πηγάδι από όπου αντλούσε νερό για τις ανάγκες του. Σήμερα δεν χρειάζεται τίποτα άλλο παρά μόνο να ανοίξουμε την βρύση στο σπίτι μας οποιαδήποτε ώρα έχουμε ανάγκη το νερό. Επιπλέον πληρώνουμε μόνο για το νερό που χρησιμοποιήσαμε όταν η βρύση μας ήταν ανοικτή. Τα πλεονεκτήματα του cloud computing είναι πολλά και μερικά από αυτά παρατίθενται πιο κάτω.

- Οικονομικά. Σύμφωνα με το μοντέλο του cloud computing πληρώνουμε για ότι πραγματικά χρησιμοποιούμε όταν και όποτε το θέλουμε ανάλογα με τις ανάγκες μας.

Ίσως αρχικά να ξοδέψουμε κάποιο ποσό για να ξεκινήσουμε αλλά στην συνέχεια τα έξοδα μειώνονται ή αυξάνονται όταν είναι απαραίτητο ανάλογα με την ανάπτυξη μας.

- Ευελιξία. Οι συνεχής διακυμάνσεις στο φόρτο εργασίας μιας εταιρίας δεν απαιτούν την επίπονη και χρονοβόρα διαδικασία της αγοράς, εγκατάστασης και συντήρησης νέου εξοπλισμού και λογισμικού. Η προσθήκη ή αφαίρεση εξοπλισμού γίνεται πολύ εύκολα με βάση τις ανάγκες της εταιρίας και πληρώνει μόνο για ότι χρησιμοποιεί.
- Γρήγορη υλοποίηση. Πλέον για την ανάπτυξη εφαρμογών – υπηρεσιών μπορεί να γίνει πολύ πιο γρήγορα αφού δεν χρειάζεται να έχουμε στην κατοχή μας τον απαραίτητο εξοπλισμό ούτε να ξοδέψουμε χρόνο για την συντήρηση του και σε αντίθεση έχουμε στην διάθεση σχεδόν άπειρα εργαλεία και υπηρεσίες.
- Συνεχής υποστήριξη. Μια εταιρία ανάπτυξης λογισμικού μπορεί να δαπανήσει πολύτιμο χρόνο σε προβλήματα εξοπλισμού και δικτύου. Πλέον με το cloud computing υπάρχει καλύτερης ποιότητας υποστήριξη και αξιοπιστία καθώς και άμεση ανταπόκριση σε περιπτώσεις άμεσης ανάγκης.
- Αυξημένη αποδοτικότητα. Για τους ίδιους λόγους με πιο πάνω ο χρήστης δεν απαιτείται να γνωρίζει πώς να ρυθμίσει και να συντηρήσει τα συστήματα που χρησιμοποιεί δαπανώντας έτσι το χρόνο του σε πιο σημαντικές εργασίες που πρέπει να ολοκληρωθούν.
- Εξοικονόμηση ενέργειας. Καθώς ο εξοπλισμός βρίσκεται αλλού δεν είναι ανάγκη ο κάθε οργανισμός να έχει στην κατοχή του την δική του υποδομή εξοπλισμού αλλά στην πραγματικότητα να την διαμοιράζεται με άλλους χρήστες - οργανισμούς. Αυτό οδηγεί στην ανάγκη για λιγότερους servers που θα χρησιμοποιούνται περισσότερο αποφέροντας έτσι σημαντική μείωση στην σπατάλη ενέργειας.

Κεφάλαιο 3 - Διαχείριση των δεδομένων στο cloud

3.1. Εισαγωγή

Στα κατανεμημένα συστήματα, και ειδικά στις κατανεμημένες βάσεις δεδομένων, είναι επιθυμητή και αναπόφευκτη η δυνατότητα να μπορεί ο χρήστης να θέτει ερωτήματα και να λαμβάνει απάντηση σε σύντομο διάστημα. Τέτοια ερωτήματα μπορεί να είναι κατά πόσο κάποιο σημείο υπάρχει στα διαθέσιμα δεδομένα ή ποια είναι τα διαθέσιμα δεδομένα σε κάποιο εύρος τιμών. Για παράδειγμα σε ένα κατανεμημένο σύστημα όπου αποθηκεύουμε μισθό και ηλικία. Ο χρήστης πρέπει να μπορεί να πάρει απάντηση στο ερώτημα υπάρχει κάποιος με ηλικία 35 και μισθό 30000 (point query), ή αντίστοιχα πόσοι είναι μεταξύ 25 και 35 και έχουν μισθό από 25000 έως 40000 (range query). Σε πολλές εφαρμογές όπως network monitoring και geographical information systems τα δεδομένα έχουν πολλές ιδιότητες (attributes) και ανήκουν σε μεγαλύτερες διαστάσεις.

Μια τυπική πλατφόρμα cloud computing αποτελείται από εκατοντάδες ή και χιλιάδες από κόμβους (υπολογιστές χαμηλού κόστους) οι οποίοι επεξεργάζονται παράλληλα τον φόρτο εργασίας. Όταν ο χρήστης θέσει κάποιο ερώτημα (submit query) με βάση τις δομές που χρησιμοποιούνται το ερώτημα προωθείται στους υπεύθυνους κόμβους για επεξεργασία και επιστροφή των αποτελεσμάτων.

3.2. Αποδοτική διαχείριση των δεδομένων στο cloud computing

Η προφανής λύση στο πρόβλημα είναι να υπάρχουν δομές ευρετηρίου πολλών διαστάσεων στα δεδομένα (multidimensional data indexing structures). Μια τέτοια δομή ευρετηρίου όμως πρέπει να είναι αποδοτική ως προς το χρόνο απάντησης, τις δυνατότητες που παρέχει (point, range and cover queries) καθώς και τον αποθηκευτικό χρόνο που απαιτεί αλλιώς η επεξεργασία ερωτημάτων θα χρειάζεται πολύ χρόνο ειδικά για μεγάλες συλλογές δεδομένες (data sets).

Αρκετές από τις προτεινόμενες λύσεις βασίζονται σε συναρτήσεις κατακερματισμού (hashing) όπως για παράδειγμα Distributed Hash Table [11][12][12]. Αυτές οι λύσεις είναι αποδοτικές στο να απαντήσουν ερωτήματα ύπαρξης ενός σημείου σε κάποιο dataset αλλά δεν είναι σε θέση να απαντήσουν range queries.

Άλλη μια προτεινόμενη λύση που μπορεί να υποστηρίξει και range queries είναι Scalable distributed B-tree[14]. Η λύση αυτή δεν μπορεί να εφαρμοστεί σε δεδομένα πολλών διαστάσεων.

Δομές που μπορούν να ανταποκριθούν στις πιο πάνω ανάγκες έχουν προταθεί. Μεταξύ αυτών είναι BR-Tree: A Scalable Prototype for Supporting Multiple Queries of Multidimensional

Data[8] και RTCAN: An Indexing Framework for Efficient Retrieval on the Cloud[15][16] οι οποίες κλιμακώνουν και είναι ευέλικτες (scalable and flexible). Το μειονέκτημα τους είναι ότι οργανώνουν τους κόμβους σαν ένα δομημένο peer-to-peer σύστημα. Παρόλο που αύτη η επιλογή μειώνει σημαντικά το κόστος συντήρησης του ευρετηρίου, δεν είναι κατάλληλη για cloud συστήματα.

Στο επόμενο κεφάλαιο περιγράφεται η *EEMINC*: Extended Efficient Multi-dimensional Index with Node Cube[7] και στο κεφάλαιο 5 η προτεινόμενη δομή. Οι δομές αυτές δεν οργανώνουν τους κόμβους σαν ένα δίκτυο peer-to-peer και μπορούν να ανταποκριθούν στις πιο πάνω απαιτήσεις.

Κεφάλαιο 4 - An Efficient Multi-Dimensional Index for Cloud Data Management

4.1. Εισαγωγή

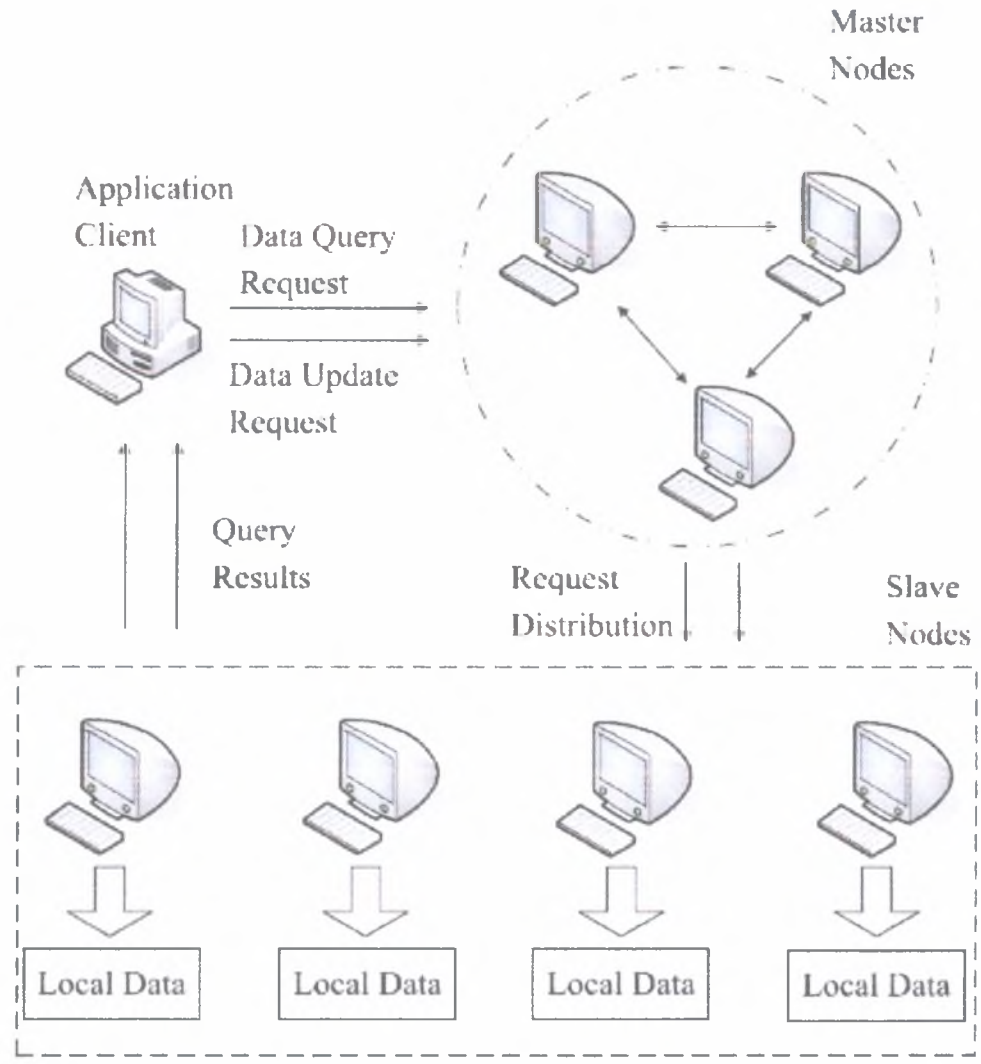
Ο αλγόριθμος που περιγράφεται σε αυτό το κεφάλαιο [7] διαχωρίζει τους κόμβους σε δύο είδη, master και slave. Η διαφορά σε αυτές τις δύο κατηγορίες κόμβων είναι ότι οι master κόμβοι είναι υπεύθυνοι και για την αποθήκευση κάποιων μετα-δεδομένων. Παρόλο που ένα από τα στοιχεία του μοντέλου του cloud computing είναι ότι δεν χρειάζονται κεντρικοί servers η επιλογή αυτή είναι αναγκαία για διατήρηση των μέτα-δεδομένων και καθιστά αποδοτικότερη την διαχείριση του συστήματος.

Οι χρήστες προωθούν τα ερωτήματα τους στους master nodes και αυτοί τα προωθούν στους υπεύθυνους slaves nodes. Στο σημείο αυτό η επικοινωνία μεταξύ του χρήστη και του master node τελειώνει και πλέον περιμένει να πάρει το αποτέλεσμα από τους υπεύθυνους slaves nodes. Η διαδικασία αυτή παρουσιάζεται στο Σχήμα 4.1.

Αφού οι κόμβοι έχουν χωριστεί στις πιο πάνω κατηγορίες η επεξεργασία ενός ερωτήματος αποτελείται από δύο στάδια:

- Εύρεση των σχετικών κόμβων με το ερώτημα και
- Επεξεργασία του ερωτήματος στους κόμβους που επιλέχτηκαν.

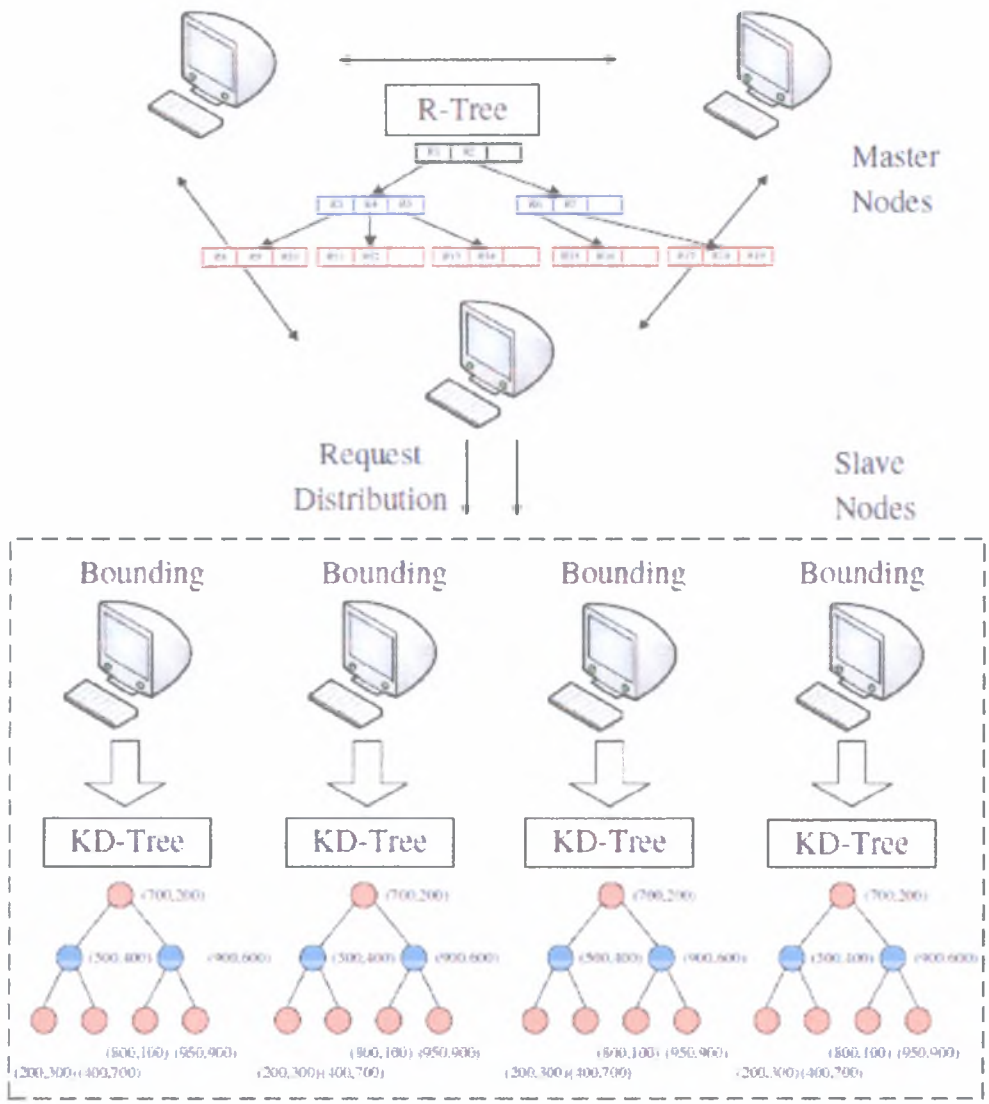
Για να είμαστε σε θέση να εντοπίσουμε τους σχετικούς κόμβους χρειαζόμαστε κάποια δομή ευρετηρίου στους master nodes (global index). Επίσης για να ανακτήσουμε τα σχετικά αποτελέσματα στους κόμβους χρειαζόμαστε ακόμη μια τοπική δομή ευρετηρίου. Επιπλέον πρέπει να μπορούμε να προσθέσουμε ή να διαγράψουμε κάποια εγγραφή από το σύστημα που είναι ισοδύναμο με το να βρούμε τους σχετικούς κόμβους όπου θα γίνει η εισαγωγή ή η διαγραφή και προώθηση του ερωτήματος σε αυτούς.



Σχήμα 4.1 Επεξεργασία ερωτημάτων στο Cloud

4.2. Βασική δομή

Οι δομές που χρησιμοποιούνται είναι R-Tree[17] για το global index και KD-Tree[18] για το local index. Μπορούμε πλέον να κτίσουμε ένα global index πάνω στα local indices. Το global index βρίσκεται στους master nodes και τα local indices στους slave nodes. Με το συνδυασμό R-Tree και KD-Tree επιτυγχάνεται η δυνατότητα επεξεργασίας point και range queries και παρουσιάζεται στο Σχήμα 4.2.



Σχήμα 4.2 Συνδυασμός R-Tree και KD-Tree για επεξεργασία ερωτημάτων

4.2.1. R-Tree

Η δομή R-Tree είναι ευρέως διαδεδομένη για πολυδιάστατα δεδομένα και χρησιμοποιείται συνήθως σε spatial και multi-dimensional εφαρμογές. Η δομή R-Tree είναι παρόμοια με την δομή B-Tree και μπορεί επιπλέον να διαχειριστεί δεδομένα πολλών διαστάσεων. Κάθε κόμβος καλύπτει μια περιοχή (π.χ. στο δυσδιάστατο χώρο μπορεί να είναι ένα ορθογώνιο, στο τρισδιάστατο χώρο κύβος κ.λπ.).

4.2.2. KD-Tree

Η δομή KD-Tree είναι ένα δυαδικό δένδρο όπου ο κάθε κόμβος έχει μια ιδιότητα (attribute) και μία τιμή (value - V) διαχωρίζοντας έτσι τα σημεία σε δύο κατηγορίες: μικρότερα

του V και μεγαλύτερα ή ίσα του V. Σε διαφορετικά επίπεδα οι ιδιότητες είναι διαφορετικές και αντιστοιχούν σε μια διάσταση.

4.2.3. Δομή ευρετηρίου

Για την επεξεργασία των ερωτημάτων είναι απαραίτητο να αποφεύγεται η αχρείαστη επεξεργασία από διάφορους κόμβους έτσι ώστε να μην σπαταλούμε υπολογιστικούς πόρους αλλά και πόρους δικτύου. Για το λόγο αυτό εισάγεται η έννοια του *node cube* ως εξής:

Ένα *node cube* αποτελείται από διαστήματα τιμών και κάθε διάστημα αναπαριστά το εύρος τιμών μιας ιδιότητας.

Definition. A node cube is a sequence of value intervals, and each interval represents the value range of one indexed attribute on this node. If there's only one value on some dimension, the corresponding interval regresses to a value point.

Για παράδειγμα εάν έχουμε δύο ιδιότητες: μισθός και ηλικία τότε το αντίστοιχο node cube μπορεί να είναι {(1000, 10000) , (25, 50)} και ερμηνεύεται ως: το όρισμα μισθός έχει τιμές από 1000 έως 10000 και το όρισμα ηλικία έχει τιμές από 25 έως 50.

Κάθε κόμβος που περιέχει δεδομένα διατηρεί το δικό του *node cube* το οποίο και γίνεται index από τους master κόμβους. Αυτή η δομή ονομάζεται *EMINC*: Efficient Multi-dimensional Index with Node Cube.

Definition. EMINC index structure consists of a R-tree in master nodes and one KD-tree on each slave node. Each leaf node of the R-tree contains a node cube and one or more pointers that point to the slave nodes corresponding to the node cube.

4.3. Επεξεργασία ερωτημάτων

Όταν κάποιο ερώτημα πρέπει να απαντηθεί αρχικά υπολογίζουμε το *query cube* για το ερώτημα. Ένα *query cube* είναι ανάλογο με το *node cube* και ορίζεται ως εξής:

Ένα *query cube* αποτελείται από διαστήματα τιμών και κάθε διάστημα αναπαριστά το εύρος τιμών μιας ιδιότητας. Εάν κάποια τιμή δεν έχει οριστεί τότε αναθέτουμε την μικρότερη και την μεγαλύτερη τιμή για κάτω και πάνω όριο αντίστοιχα.

Definition. A query cube is a sequence of intervals, and each interval represents the value range of one attribute in this query. If either side of the attribute is not specified, we assign it the biggest negative or positive value accordingly.

Αφού έχουμε φτιάξει το *query cube* μπορούμε να αναζητήσουμε στο R-Tree τους κόμβους που έχουν σχετικά δεδομένα οι οποίοι είναι όσοι από τους slave nodes το node cube τους τέμνεται με το query cube για κάθε ιδιότητα (attribute).

Αφού εντοπιστούν οι σχετικοί κόμβοι εκτελούμε αναζήτηση τοπικά για να πάρουμε το αποτέλεσμα.

4.4. Διατήρηση της δομής σε έγκυρη κατάσταση

Αφού έχει αναλυθεί ο τρόπος επεξεργασίας των ερωτημάτων το πρόβλημα που υπάρχει είναι πως θα διατηρηθεί η δομή σε έγκυρη κατάσταση και συγκεκριμένα πως οι master nodes θα είναι σε θέση να γνωρίζουν και να απαντούν σωστά ποιοι είναι οι σχετικοί κόμβοι.

Για να επιδεχθεί το πιο πάνω οι slave nodes πρέπει να στέλνουν στους master nodes κάποιες ενημερώσεις που ονομάζονται cube updates. Όταν ο node cube του κόμβου έχει αλλάξει τότε στέλνεται ενημέρωση στους master nodes. Το node cube μπορεί να αλλάξει μόνο κατά την εισαγωγή ή διαγραφή στοιχείων από την βάση δεδομένων, πράξεις που δεν είναι τόσο συνηθισμένες σε μεγάλα συστήματα.

Επιπλέον για να περιοριστεί, έστω και ελάχιστα, ο αριθμός των cube updates οι master nodes εάν ο χρήστης θέλει να κάνει εισαγωγή επιλέγονται με το ίδιο τρόπο που γίνεται η επιλογή και για ερώτηση.

4.5. Βελτιωμένη έκδοση

Η πιο πάνω δομή μπορεί να απαντήσει point και range queries αλλά μπορεί και να βελτιωθεί. Ο λόγος είναι ότι το node cube μπορεί να είναι αρκετά ευρύ και να καλύπτει και μεγάλο σύνολο δεδομένων τα οποία όμως δεν υπάρχουν στο συγκεκριμένο κόμβο.

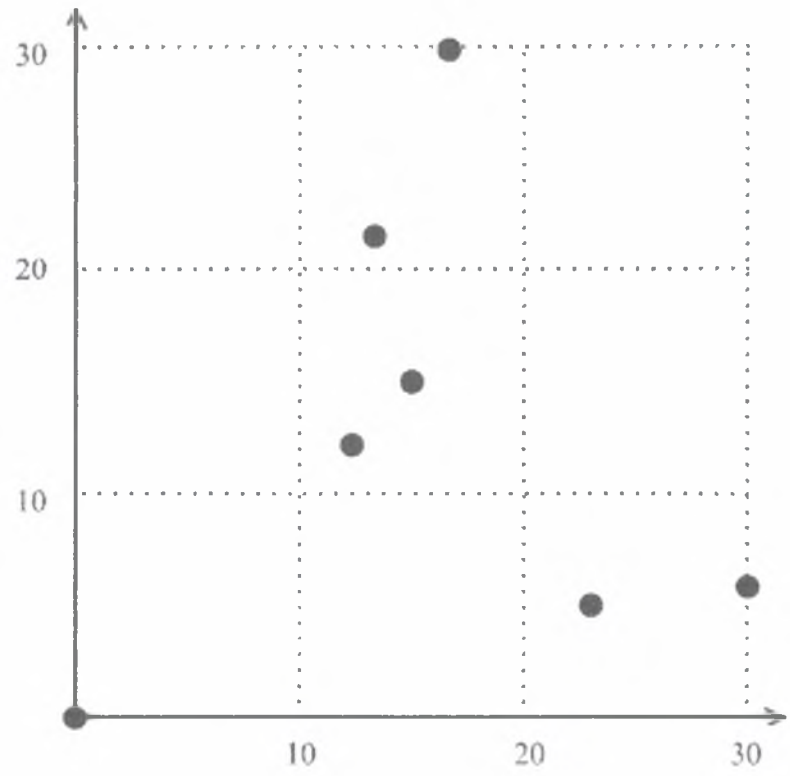
Ως βελτίωση του EMINC index προτείνεται η δομή EEMINC: Extended EMINC. Η διαφορά είναι ότι λύνει το πιο πάνω πρόβλημα και ορίζεται ως εξής:

Definition. EEMINC is an extension of EMINC. The difference from EMINC is that in EEMINC data records on one slave node will be represented by multiple node cubes. The shape and amount of node cubes is dependent on the method used for cutting the original single node cube.

Σε αυτή την έκδοση ο κάθε κόμβος οριοθετεί τα δεδομένα του σε περισσότερα του ενός node cubes. Η μέθοδος που ακολουθείται μπορεί να είναι:

- Τυχαία επιλογή: Τα σημεία όπου θα σπάσει κάθε attribute επιλέγονται τυχαία. Η μέθοδος αυτή είναι αρκετά αποδοτική στη πράξη και εύκολα υλοποιήσιμη.
- Ομοιόμορφα: Το node cube σπάει σε ίσα κομμάτια. Στο Σχήμα 4.3 φαίνεται ένας ομοιόμορφος διαμερισμός του node cube στο δυσδιάστατο χώρο σε εννέα (μικρότερα) node cubes. Η μέθοδος αυτή είναι πολύ εύκολη στην υλοποίηση.
- Ομαδοποίηση: Τα δεδομένα ομαδοποιούνται για να δημιουργηθούν τα αντίστοιχα node cubes. Όπως είναι προφανές είναι η πιο αποδοτική μέθοδος για αποφυγή

περιττής επεξεργασίας ερωτημάτων. Στην πράξη όμως μια τέτοια υλοποίηση απαιτεί $O(n^2)$ πράξεις και αυτό αυξάνει κατά πολύ το κόστος.



Σχήμα 4.3 Ομοιόμορφος διαμερισμός δεδομένων στο δυσδιάστατο χώρο

Όπως είναι αναμενόμενο με αυτή την αλλαγή οι slave nodes πρέπει να στέλνουν πιο πολλά cube updates στους master nodes. Για παράδειγμα στο Σχήμα 4.3 ο κόμβος θα στείλει τέσσερεις ενημερώσεις {{(0, 10), (0, 10)}, {(10, 20), (10, 20)}, {(10, 20), (20, 30)}, {(20, 30), (0, 10)} ενώ στην προηγούμενη έκδοση θα έστελνε μόνο μια {{(0, 30), (0, 30)}). Αυτό το επιπλέον κόστος όμως είναι πολύ μικρότερο από το κόστος που θα χρειαζόταν για να απαντήσει ερωτήματα που είναι σίγουρο ότι δεν θα έχουν αποτέλεσμα. Τέτοια ερωτήματα είναι όσα αφορούν δεδομένα στα {(0, 10), (20, 30)} , {(10, 20), (0, 10)} και {(20, 30), (10, 30)}.

Σε κάθε περίπτωση μεγάλος διαμοιρασμός του node cube αυξάνει το κόστος αναζήτησης σχετικών κόμβων στους master nodes. Για το λόγο αυτό και για καλύτερη απόδοση ο συνολικός αριθμός node cubes πρέπει να διατηρείται πάρα πολύ μικρός σε σχέση με το μέγεθος της συλλογής.

Επιπλέον ο χρόνος για να σπάσει το node cube σε μικρότερα έχει κόστος στην καλύτερη περίπτωση O(n). Για καλύτερη αντιμετώπιση του tradeoff που υπάρχει μεταξύ χρόνου επεξεργασίας ερωτημάτων και χρόνου για διαμοιρασμό του node cube ο κάθε κόμβος πρέπει να είναι σε θέση να επιλέγει το σωστό χρόνο για να προβεί σε διαμερισμό.

4.6. Στρατηγική ανανεώσεων

Η ιδέα της στρατηγικής ανανεώσεων είναι απλή και ορίζεται ως: benefit > cost όπως είναι αναμενόμενο. Το πρόβλημα ανάγεται στο καθορισμό του benefit και του cost. Για να μπορούν να συγκριθούν μεταξύ τους και τα δύο εκφράζονται ως αριθμός από queries. Για τον ορισμό των δύο αυτών ποσοτήτων ορίζεται και η βοηθητική έννοια του volume cube που είναι πόσα queries καλύπτει ένας node cube.

Definition. Volume of a node cube is defined as the maximum number of unique records that this cube can cover. We note volume of a cube by *v*.

Το κέρδος μιας ενημέρωσης ορίζεται ως ο αριθμός των άσχετων queries που δεν θα τεθούν στο κόμβο μετά την συγκεκριμένη ενημέρωση.

$$benefit = (\delta v/v) * n_q * \delta T$$

όπου δν είναι πόσο μειώνεται ο node cube μετά την ανανέωση, και n_q ο αριθμός των query cubes που τέμνονται με τον node cube ανά μονάδα χρόνου.

Το κόστος είναι η αναδιαμόρφωση του node cube και το κόστος εισαγωγής του στα R-Trees στους master nodes.

$$cost = (m_t + \varepsilon)/q_t \sim= m_t/q_t$$

όπου ε το κόστος αναδιαμόρφωσης του node cube, q_t ο μέσος χρόνος επεξεργασίας ενός query και m_t ο χρόνος για να ολοκληρωθεί η ενημέρωση.

Αντικαθιστώντας στην εξίσωση benefit > cost έχουμε:

$$\delta T > (m_t * v)/(q_t * \delta v * n_q)$$

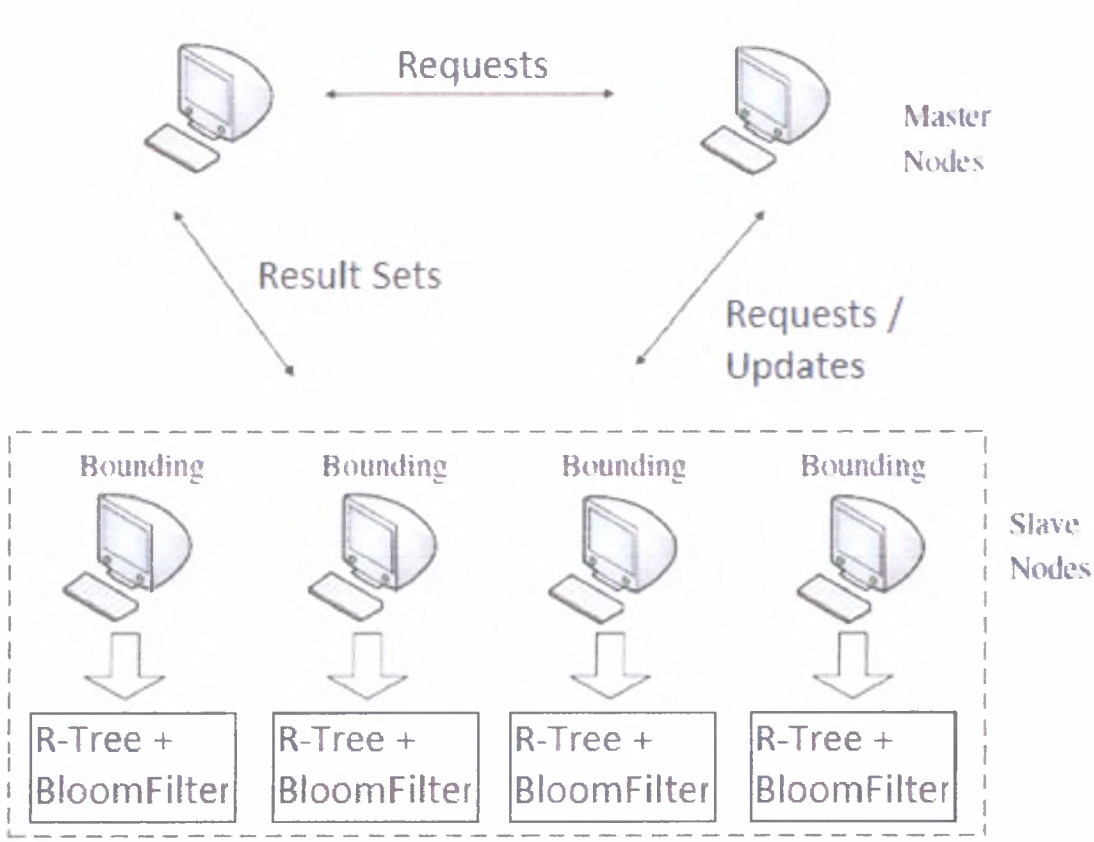
που σημαίνει ότι κάποιος slave node πρέπει να περιμένει τουλάχιστον δT πριν την επόμενη ενημέρωση.

Κεφάλαιο 5 - Προτεινόμενη δομή – A-Tree: Distributed Index for Multidimensional Data

5.1. Εισαγωγή

Οι κόμβοι, όπως και πριν, διαχωρίζονται σε δύο κατηγορίες: master nodes και slave nodes και παρουσιάζεται στο Σχήμα 4.1. Η περιγραφή είναι ίδια με το προηγούμενο.

Η βασική δομή που χρησιμοποιείται είναι η BR-Tree[8] που είναι βασισμένη σε R-Tree[19] και Bloom filter[21]. Με το συνδυασμό αυτό επιτυγχάνεται η αποδοτική επεξεργασίας point και range queries και παρουσιάζεται στο Σχήμα 5.1.

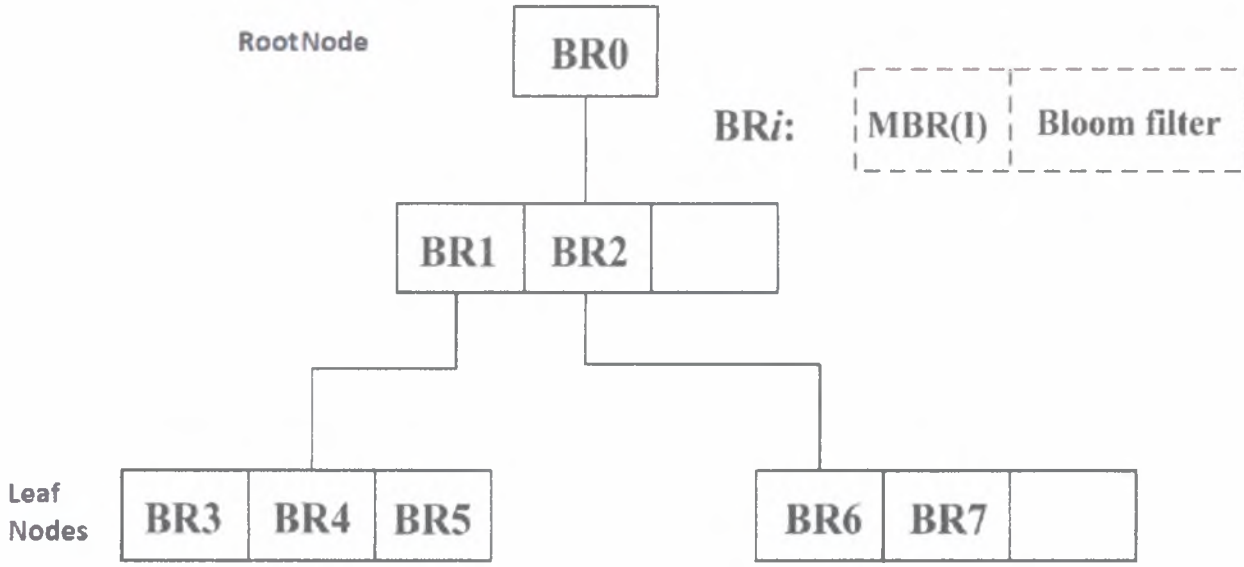


Σχήμα 5.1 Συνδυασμός R-Tree και Bloom filter για επεξεργασία ερωτημάτων

5.1.1. BR-Tree

Η δομή BR-Tree είναι μια δενδρική δομή βασισμένη στις δομές R-Tree και Bloom Filter. Κάθε κόμβος του δένδρου περιέχει όπως και στο R-Tree ένα bounding box και επιπλέον ένα Bloom

Filter που καλύπτει όλα τα δεδομένα κάτω από αυτόν. Το Σχήμα 5.2 παρουσιάζει ένα παράδειγμα της δομής BR-Tree.

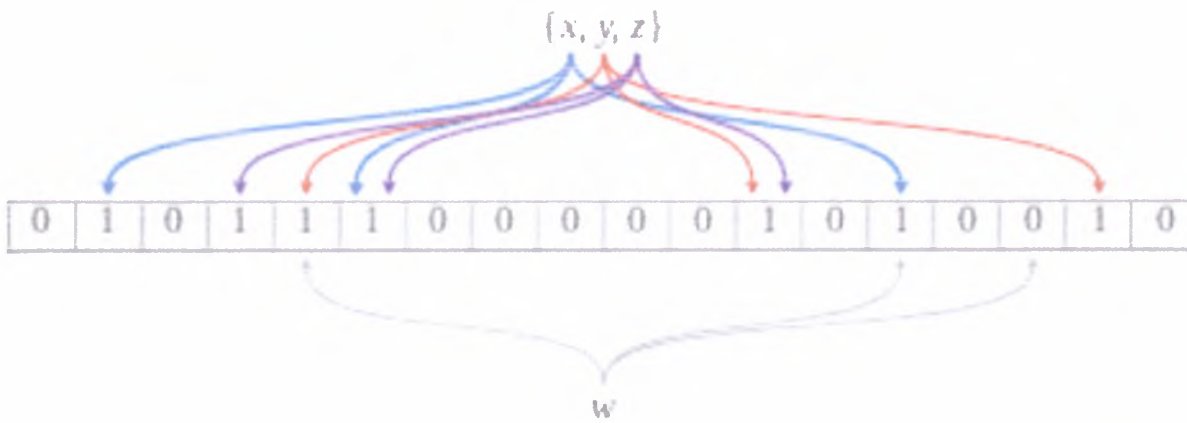


Σχήμα 5.2 Δομή BR-Tree - Bloom-filter-based R-tree

5.1.2. Bloom Filter

Η δομή Bloom Filter είναι μια δομή που σε O(1) απαντάει εάν κάποιο στοιχείο ανήκει σε μια συλλογή και χρειάζεται ελάχιστο αποθηκευτικό χώρο. Είναι πιθανόν να υπάρχουν στοιχεία τα οποία λανθασμένα θα απαντηθεί ότι ανήκουν στη συλλογή (false positives).

Ένα Bloom Filter είναι ένας πίνακας από m bits. Για κάθε στοιχείο υπολογίζεται η τιμή από k hash functions και τα αντίστοιχα bits γίνονται 1. Το Σχήμα 5.3 παρουσιάζει ένα παράδειγμα για μια συλλογή με τρία στοιχεία (x, y, z) και αναζήτηση για το στοιχείο w που δεν ανήκει στη συλλογή. Στο σχήμα m = 18, k = 3 και τα χρωματιστά βέλη δείχνουν ποια bits ενεργοποιούνται από πιο στοιχείο.



Σχήμα 5.3 Παράδειγμα Bloom Filter

5.2. Βασική δομή ευρετηρίου

Κάθε slave node κτίζει στα δεδομένα του ένα R-Tree στο οποίο κάθε κόμβος του δένδρου περιέχει και ένα Bloom Filter που καλύπτει όλα τα δεδομένα κάτω από αυτόν όμοια με την δομή BR-Tree[8]. Κάθε slave node είναι υπεύθυνος να κρατά σε συνεπή κατάσταση το δικό του R-Tree και Bloom Filter καθώς και να στέλνει ανανεώσεις στους master nodes όταν το κρίνει απαραίτητο.

Ο slave node στέλνει στους master nodes τα bounding boxes από κάποιους κόμβους του R-tree του μαζί με το Bloom Filter της ρίζας. Η διαδικασία αυτή είναι αντίστοιχη με τις μεθόδους για διαμοιρασμό του node cube στον αλγόριθμο που περιγράφεται στο Κεφάλαιο 4 - An Efficient Multi-Dimensional Index for Cloud Data Management.

Οι master nodes οργανώνει τα bounding boxes που παίρνει και διαχειρίζεται τα Bloom filters έτσι ώστε να αποφεύγεται η προώθηση άσχετων ερωτημάτων στους κόμβους.

5.3. Τοπικές λειτουργίες στους master nodes

5.3.1. Επεξεργασία ερωτημάτων

Οι master nodes είναι υπεύθυνοι να μεταβιβάσουν τα requests στους slave nodes. Λαμβάνοντας ένα query (εισαγωγή ή αναζήτηση) πρέπει να βρουν τους σχετικούς κόμβους. Ο γενικός αλγόριθμος επεξεργασίας των queries, αναζήτησης και εισαγωγής, παρουσιάζονται πιο κάτω.

Algorithm 1 Process query on cloud

```
1. procedure SET PROCESSQUERY(Query q, User u_id)
2.
3. Set nodes = GetRelativeNodes(q);
4. for each (node n in nodes) do
5.         send(QUERY,n,q,u_id);
6. end for
7. return #nodes;
```

Αλγόριθμος 5.1. Επεξεργασία ερωτημάτων στο cloud

Για κάθε query που φτάνει στο σύστημα ο master node που το δέχθηκε πρέπει να βρει τους κόμβους που πιθανόν να περιέχουν σχετικά αποτελέσματα. Αφού εντοπίσει τους κόμβους τότε στέλνει σε όλους το query καθώς και το αναγνωριστικό του χρήστη για να μπορεί ο κόμβος να επικοινωνεί κατευθείαν με το χρήστη χωρίς να είναι απαραίτητη η απασχόληση του master node. Αφού ενημερωθούν οι σχετικοί κόμβοι το πλήθος τους επιστρέφεται στο χρήστη – εφαρμογή για να γνωρίζει από πόσους κόμβους να περιμένει αποτέλεσμα.

Για εισαγωγή νέων εγγραφών ο αλγόριθμος είναι παρόμοιος με πριν με την διαφορά ότι πρέπει να υπάρχουν ακριβώς REPLICATION κόμβοι στο nodes set. Αρχικά επιλέγονται οι σχετικοί κόμβοι έτσι ώστε να αποφθεχθούν κάποιες ενημερώσεις στην συνέχεια. Εάν οι σχετικοί κόμβοι είναι πιο λίγοι τότε προσθέτουμε κόμβους στο nodes set μέχρι να γεμίσει. Εάν αντίθετα έχουμε πιο πολλούς κόμβους τότε αφαιρούμε από το nodes set. Ο Αλγόριθμος 5.2 περιγράφει αυτή την διαδικασία.

Algorithm 2 Record insertion to cloud

```
1.  procedure BOOLEAN INSERTRECORD(Record r, User u_id)
2.
3.  Set nodes =  GetRelativeNodes(r);
4.  while (#nodes < REPLICATION)
5.          nodes.add(getAnode());
6.  while (#nodes > REPLICATION)
7.          nodes.removeFirst();
8.  for each (node n in nodes) do
9.          send(INSERT,n,r,u_id);
10. end for
11. return #nodes;
```

Αλγόριθμος 5.2. Εισαγωγή εγγραφών στο cloud

5.3.2. Επιλογή των σχετικών κόμβων ως προς τα queries

Καθώς χρησιμοποιούμε και Bloom filters τα οποία δεν μπορούν να χρησιμοποιηθούν για range queries ο αλγόριθμος είναι διαφορετικός ανάλογα με το τύπο του query. Ο Αλγόριθμος 5.3 παρουσιάζει την αναζήτηση σχετικών κόμβων ως προς κάποιο point query.

Algorithm 3 getRelativeNodes for Point Query

```
1.  procedure SET GetRelativeNodesPointQuery(Point p)
2.
3.  SET nodes = {};
4.  for each (UPDATE u in global Index) do
5.          if u.bloomfilter.membershipTest(p) then
6.                  nodes.add(node_id);
7.          end if
8.  end for
9.  return nodes;
```

Αλγόριθμος 5.3. Εύρεση σχετικών κόμβων για point query

Για εύρεση των σχετικών κόμβων για κάποιο point query αρκεί να κοιτάξουμε εάν το συγκεκριμένο σημείο καλύπτεται από τα bloom filters των κόμβων. Ανάλογα με το μέγεθος της συλλογής αυξάνει και η πιθανότητα false positives. Για να το αντιμετωπίσουμε αυτό μπορούμε αρχικά να μεγαλώσουμε το μέγεθος του bloom filter. Αυτό είναι εφικτό αφού το bloom filter είναι δεν χρειάζεται πολύ χώρο. Για παράδειγμα σε ένα σύστημα με 1000 slave nodes και μέγεθος bloom filter 100000 bits ο master node χρειάζεται 10^8 bits =~ 12MB για να αποθηκεύσει όλα τα bloom filters. Σε αντάλλαγμα χρειάζεται να πληρώνουμε περισσότερο χρόνο επεξεργασίας στους master κόμβους, κάτι το οποίο δεν είναι επιθυμητό ειδικά όταν υπάρχουν πάρα πολλά queries στο σύστημα.

Algorithm 4 getRelativeNodes for Range Query

```
1.  procedure SET GetRelativeNodesRangeQuery(Query r)
2.
3.  SET nodes = {};
4.  for each (UPDATE u in global Index) do
5.          for each (Bounding Box box in u) do
6.                  if (box.overlaps(q) then
7.                          nodes.add(u.node_id);
8.                          continue with next Update;
9.                  end if
10.         end for
11. end for
12. return nodes;
```

Αλγόριθμος 5.4. Εύρεση σχετικών κόμβων για range query

Ο Αλγόριθμος 5.4 παρουσιάζει το αντίστοιχο αλγόριθμο για range query. Για εύρεση των σχετικών κόμβων για κάποιο range query τα bloom filters δεν μας παρέχουν καμία χρήσιμη πληροφορία και έτσι πρέπει να κοιτάξουμε τα bounding boxes από κάθε κόμβο. Για κάποιο κόμβο αφού βρούμε το πρώτο bounding box που τέμνεται με το query δεν υπάρχει λόγος να ψάξουμε τα υπόλοιπα και ο αντίστοιχος κόμβος αυτομάτως γίνεται υποψήφιος για να περιέχει σχετικά αποτελέσματα.

5.4. Τοπικές λειτουργίες στους slave nodes

Οι κόμβοι αυτοί πρέπει να είναι σε θέση να αναζητούν και να προσθέτουν εγγραφές στο σύστημα.

Η εισαγωγή εγγραφής περιλαμβάνει την εισαγωγή της εγγραφής στο R-tree και ανανέωση των Bloom Filters από τον κόμβο εισαγωγής μέχρι την ρίζα του δένδρου. Ο αλγόριθμος που

χρησιμοποιείται είναι ίδιος με αυτόν της BR-Tree. Ο χρόνος εισαγωγής είναι O(log N) εάν ο κόμβος δεν είναι γεμάτος. Εάν ο κόμβος έχει γεμίσει τότε χρειάζεται όσο είναι το κόστος να μοιράσουμε τον κόμβο.

Για αναζήτηση point και range και πάλι οι αλγόριθμοι είναι ίδιοι με την BR-Tree. Για αναζήτηση σημείου μπορούμε σε οποιοδήποτε κόμβο του δένδρου σε O(1) χρησιμοποιώντας τα Bloom filters να απαντήσουμε εάν υπάρχει πιο κάτω ή όχι με μικρή πιθανότητα λάθους. Λόγω του μεγάλου αριθμού στοιχείων στην ρίζα υπάρχει αυξημένο ποσοστό false positives. Αφού γνωρίζουμε εάν το στοιχείο υπάρχει εκτελούμε την αναζήτηση με κόστος O(log N).

5.5. Στρατηγική ανανεώσεων και διατήρηση της δομής σε έγκυρη κατάσταση

5.5.1. Αρχική προσέγγιση

Η στρατηγική ανανεώσεων αρχικά είναι πολύ απλή. Ο κάθε slave κόμβος στέλνει το bloom filter της ρίζας του ευρετηρίου του μαζί με τα bounding boxes όλων των παιδιών της ρίζας. Αυτή η στρατηγική είναι πολύ απλή στην υλοποίηση και έτσι έχουμε σχεδόν μηδενικό overheat για αποστολή ενημερώσεων. Ο Αλγόριθμος 5.5 παρουσιάζει τα βήματα για την αποστολή ενημερώσεων. Η αποστολή ενημερώσεων είναι απαραίτητη όταν εισάγουμε κάποιο καινούριο στοιχείο και γίνει αλλαγή στα bounding boxes.

Algorithm 5 sendUpdate

```
1.  procedure SendUpdate()
2.
3.  UPDATE u;
4.  u. BloomFilter = root.bloomfilter;
5.  for each (child of root) do
6.          u.boundingBoxes.add(child.getBoundingBox())
7.  end for
8.  for each (MASTER Node n) do
9.          send(INDEXUPDATE, u, my_id, n)
10. endfor
11. LastSentUpdate = u;
```

Αλγόριθμος 5.5. Αποστολή ενημερώσεων

Στους master nodes υπάρχει ένας πίνακας που κρατά όλα τις ενημερώσεις και κάθε θέση στο πίνακα αντιστοιχεί σε ένα slave node. Κάθε γραμμή του πίνακα αντιστοιχεί σε ένα slave node. Ο αλγόριθμος είναι πολύ απλώς και φαίνεται πιο κάτω.

Algorithm 6 receiveUpdate

```
1. Procedure receiveUpdate(UPDATE u, NODE node_id)
2.
3. global_index[node_id] = u;
```

Αλγόριθμος 5.6. Ανανέωση global index

5.5.2. Αποστολή ενημερώσεων με βάση το κέρδος – Cost Modeling

Παρόλο που η πιο πάνω λύση δουλεύει σωστά αντιμετωπίζουμε το πρόβλημα ότι έχουμε πολλούς κόμβους που λανθασμένα αναζητούνται για αποτελέσματα. Ο αριθμός των false positives αυξάνει καθώς αυξάνει το μέγεθος της συλλογής και καθώς μειώνεται το fan out του δένδρου και αυτό οφείλεται στο ότι το global index περιέχει κόμβους οι οποίοι βρίσκονται πολύ ψηλά στα τοπικά ευρετήρια και καλύπτουν μεγάλο αριθμό ερωτήσεων.

Για να διατυπώσουμε την στρατηγική ενημερώσεων ορίζουμε το κέρδος που θα έχουμε εάν στο global index προσθέσουμε τα παιδιά κάποιου κόμβου ως τον αριθμό των queries που ενδεχομένως να μην λάβει ο κόμβος. Πιο συγκεκριμένα καθώς ο κάθε κόμβος του BR-Tree περιλαμβάνει ένα bounding box, που καλύπτει όλα τα παιδιά του, ένα μέτρο για τον αριθμό ερωτήσεων που θα προωθηθούν στο κόμβο αυτό είναι ο όγκος του bounding box. Αντίστοιχα κάθε παιδί του κόμβου αυτού έχει το δικό του bounding box. Ο Αλγόριθμος 5.7 παρουσιάζει πως υπολογίζουμε το κέρδος που θα έχουμε εάν κάνουμε index τα παιδιά ενός κόμβου.

Algorithm 7 Benefit For Indexing Children of Node n

```
1.  procedure double BenefitForIndexingChildren(Node n)
2.
3.  children_volume = 0.0;
4.  children_overlap_volume = 0.0;
5.
6.  for ( i = 0; i < #children; i++ ) do
7.          childrens_volume += child[i].getVolume();
8.          for ( j = i+1; j < #children; j++ ) do
9.                  children_overlap_volume += child[i].overlapVolume(child[j]);
10.         end for
11. end for
12.
13. benefit = n.getVolume() – (children_volume – children_overlap_volume);
14. return benefit;
```

Αλγόριθμος 5.7. Υπολογισμός κέρδους για αφαίρεση κόμβου από το ευρετήριο

Είναι προφανές ότι για κάθε κόμβο θα έχουμε κάποιο θετικό κέρδος αλλά επίσης θα έχουμε και κάποιο κόστος καθώς εάν αυξήσουμε πολύ τον αριθμό των indexed nodes στους master nodes τότε θα πληρώνουμε περισσότερο χρόνο και χώρο σε αυτούς. Έτσι ορίζουμε σαν κόστος μια ενημέρωσης των αριθμό των bounding boxes που περιέχει. Το κόστος αυτό είναι αντιπροσωπευτικό για το χρόνο που απαιτείται για επεξεργασία στους master nodes καθώς και την καθυστέρηση στο δίκτυο για την αποστολή της ενημέρωσης.

Ο αλγόριθμος επιλογής των κόμβων που θα σταλούν σε μια ενημέρωση πρέπει να εγγυάται ότι για κάθε μονοπάτι από την ρίζα μέχρι κάθε φύλλο του δένδρου υπάρχει ακριβώς ένας κόμβος. Αυτό είναι απαραίτητο καθώς εάν δεν υπάρχει τέτοιος κόμβος για κάθε φύλο του δένδρου τότε η ενημέρωση μας είναι ελλιπής και δεν καλύπτει όλα μας τα δεδομένα. Χρειάζεται να υπάρχει ακριβώς ένας τέτοιος κόμβος για κάθε φύλο καθώς δεν έχει νόημα να στείλουμε κάποιο κόμβο εάν πρόκειται να στείλουμε κάποιο πρόγονο του. Σε αντίθεση αν στείλουμε περισσότερους θα χάσουμε καθώς θα υπάρχει μεγαλύτερη καθυστέρηση στο δίκτυο, θα απαιτείται περισσότερος χρόνος από τους master nodes για επιλέξουν τους σχετικούς κόμβους και θα έχουμε μεγαλύτερο αριθμό false positives. Πιο κάτω παρουσιάζεται ο αλγόριθμος επιλογής των κόμβων.

Algorithm 8 SelectIndexingNodes

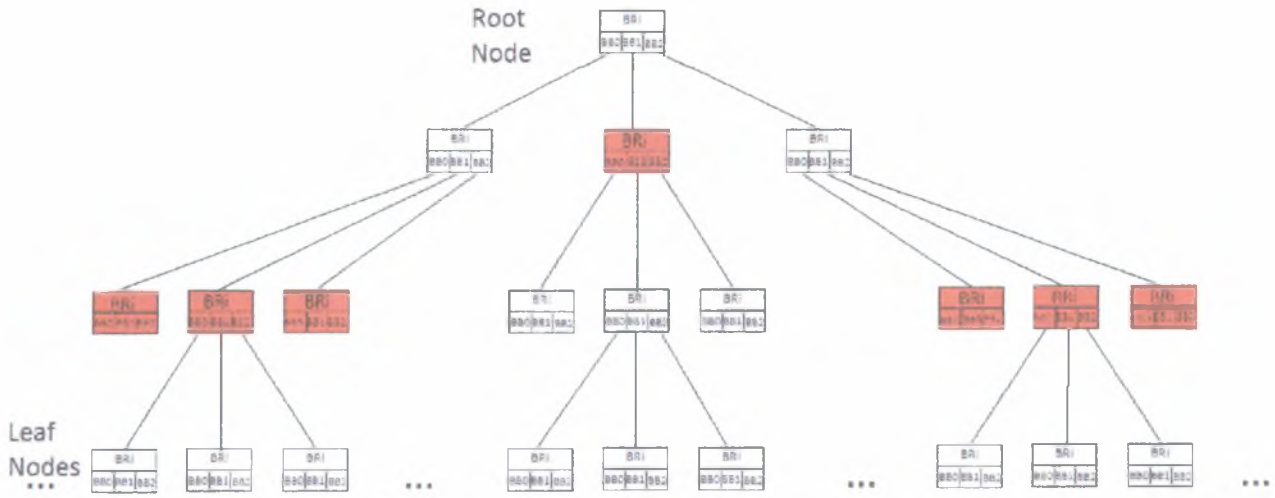
```
1.  procedure SelectIndexNodes( Node n, SET<BoundingBox> indexNodes,
2.                                  int remainingSpace, double minBenefit)
3.
4.  if ( n.getBenefitForIndexingChildrens() > minBenefit && !n.hasLeafChild()
5.          && n.#childrens <= remainingSpace ) then
6.          int currentSize = res.size();
7.          int more_free_slots = 0;
8.          int perChild = remainingSpace/node. #children;
9.          for ( int i = 0; i < n. #children; i++ ) do
10.                 SelectIndexNodes(n.child[i], indexNodes, perChild, minBenefit);
11.                 more_free_slots = (res.size() – currentSize);
12.                 perChild += more_free_slots/(n. #children – i);
13.                 currentSize = res.size();
14.         end for
15. else
16.         indexNodes.add(n.boundingBox);
17. endif
```

Αλγόριθμος 5.8. Επιλογή κόμβων για προσθήκη στο ευρετήριο

Κάθε κόμβος εκτελεί το αλγόριθμο 8 αρχικά από την ρίζα του δένδρου του, το μέγιστο αριθμό κόμβων που θα σταλούν και το ελάχιστο κέρδος που πρέπει να έχουμε για κάποιο κόμβο εάν προσθέσουμε τα παιδιά του. Ο κάθε κόμβος μοιράζει ίσα τον αριθμό των υπολειπόμενων θέσεων σε όλα τα παιδιά του (γραμμή 8). Κάθε παιδί μπορεί να προσθέσει μέχρι και perChild κόμβους στο indexNodes set. Εάν κάποιο παιδί προσθέσει λιγότερους κόμβους (π.χ. εάν το

κόστος είναι μικρότερο από minBenefit προσθέτει μόνο τον εαυτό του) τότε οι υπόλοιπες θέσεις ανακατανέμονται ίσα στα υπόλοιπα παιδιά του (γραμμές 11-13). Το βέλτιστο θα ήταν οι θέσεις να κατανέμονται ανάλογα με το κέρδος που θα έχουμε για κάθε παιδί και να καλούμε αναδρομικά αρχικά με το παιδί με το μικρότερο αναμενόμενο κέρδος. Έτσι το παιδί με μεγαλύτερο αναμενόμενο κέρδος θα γινόταν index στην τελευταία αναδρομική κλίση και συνεπώς ίσως να είχε στην διάθεση του και κάποιες επιπλέον θέσεις που περίσσεψαν από προηγούμενες κλήσεις. Αυτό όμως αυξάνει την πολυπλοκότητα του αλγορίθμου και το χρόνο εκτέλεσης αφού σε μεγάλα συστήματα, οι κόμβοι του δένδρου με μεγάλη πιθανότητα βρίσκονται στο δίσκο, πρέπει να ταξινομηθούν όλοι οι κόμβοι με βάση το αναμενόμενο κέρδος τους.

Ο χρόνος που απαιτεί ο Αλγόριθμος 5.8 είναι στην χειρότερη περίπτωση O(|V|) όπου |V| ο αριθμός κόμβων του R-Tree. Το Σχήμα 5.4 παρουσιάζει ένα παράδειγμα εκτέλεσης του αλγορίθμου 5.8 όπου με κόκκινο χρώμα φαίνονται οι κόμβοι που έχουν επιλεγεί.



Σχήμα 5.4 Επιλογή κόμβων για προσθήκη στο global index

Ο βελτιωμένος αλγόριθμος αποστολής ενημερώσεων παρουσιάζεται πιο κάτω. Το κόστος του στην χειρότερη περίπτωση είναι O(|V|) + Θ(n) όπου n ο αριθμός των master nodes στο σύστημα.

Algorithm 9 sendUpdate

```
1.  procedure SendUpdate()
2.
3.  UPDATE u;
4.  u.BloomFilter = root.bloomfilter;
5.  SET indexNodes = {}
6.  SelectIndexNodes(root, indexNodes, Max_Update_Size, minBenefit)
7.
8.  u.boundingBoxes = indexNodes;
9.  if(!LastSentUpdate.equals(u) then
10.         for each (MASTER Node n) do
11.                 send(INDEX_UPDATE, u, my_id, n)
12.         endfor
13. endif
14. LastSentUpdate = u;
```

Αλγόριθμος 5.9. Επιλογή κόμβων και αποστολή ενημερώσεων

Ο Αλγόριθμος 5.9 είναι σχεδόν απαραίτητο να κληθεί όταν το σύστημα εκκινήσει για να μπορούν να απαντηθούν τα ερωτήματα με όσο το δυνατόν λιγότερη καθυστέρηση. Παρόλα αυτά εάν οι master nodes δεν έχουν πληροφορίες για τους slaves nodes μπορούν να προωθούν τα ερωτήματα σε όλους τους κόμβους μαζί με αίτηση για ενημέρωση.

Αφού το σύστημα έχει ξεκινήσει και οι master nodes έχουν πάρει τουλάχιστον μια ενημέρωση από κάθε κόμβο τότε είναι απαραίτητο να στείλουμε καινούριες ενημερώσεις μόνο όταν εισάγουμε νέα στοιχεία στην συλλογή και δεν καλύπτονται από προηγούμενη ενημέρωση. Αυτό μας απαλλάσσει από την ανάγκη να στέλνουμε ενημέρωση εάν οι κόμβοι που έχουμε στείλει έχουν αλλάξει. Σε περίπτωση διαγραφής στοιχείων μπορούμε να μετρούμε τις διαγραφές και όταν φτάσουν κάποιο κατώφλι να στέλνουμε ενημέρωση. Όταν κάποιο στοιχείο διαγραφεί δεν μας πειράζει εάν έχουν αλλάξει οι κόμβοι του δένδρου καθώς όλα τα δεδομένα που κρατά ο κόμβος εξακολουθούν να καλύπτονται από την προηγούμενη ενημέρωση.

Συνοψίζοντας τα πιο πάνω, ο Αλγόριθμος 5.9 πρέπει να κληθεί:

- Σε σύντομο χρονικό διάστημα μετά την εκκίνηση του συστήματος
- Όταν προστεθούν εγγραφές οι οποίες δεν καλύπτονται από την προηγούμενη ενημέρωση
- Όταν οι διαγραφές που έγιναν έχουν ξεπεράσει κάποιο προκαθορισμένο κατώφλι

Ο αλγόριθμος για επεξεργασία των ενημερώσεων από τους master nodes παραμένει ο ίδιος και περιγράφεται στον αλγόριθμο 5.6.

Παρατηρούμε ότι από τις τρεις πιο πάνω περιπτώσεις μπορούμε να αποφύγουμε την εκτέλεση του αλγορίθμου 5.8 και έτσι να ελαττώσουμε το κόστος – χρόνο επεξεργασίας που χρειάζεται για αποστολή ενημερώσεων. Με βάση αυτή την παρατήρηση επεκτείνουμε τα πιο πάνω προσθέτοντας την στρατηγική των μερικών ενημερώσεων που περιγράφεται πιο κάτω.

5.5.3. Αποστολή μερικών ενημερώσεων μετά από αλλαγές στην συλλογή

Όπως αναφέρεται και πιο πάνω είμαστε υποχρεωμένοι να διατηρούμε στους master nodes την ενημέρωση η οποία καλύπτει όλα τα δεδομένα που διαχειριζόμαστε. Συνεπώς σε περίπτωση εισαγωγής ή διαγραφής στοιχείων πρέπει να στείλουμε ενημερώσεις. Καθώς ο Αλγόριθμος 5.8. Επιλογή κόμβων για προσθήκη στο ευρετήριο απαιτεί αρκετό χρόνο μπορούμε να αποφύγουμε κάποιες εκτελέσεις του. Πιο κάτω περιγράφονται οι αλγόριθμοι εισαγωγής και διαγραφής στοιχείων καθώς και οι αλγόριθμοι διαχείρισης και αποστολής μερικών ενημερώσεων (partial updates).

5.5.3.1. Εισαγωγή νέων εγγραφών

Κάθε UPDATE περιέχει ένα επιπλέον hyper bounding box το οποίο αναλαμβάνει να καλύψει όλες τις εγγραφές που έχουν προστεθεί αργότερα στο σύστημα. Κατά την εισαγωγή εάν η προηγούμενη ενημέρωση δεν καλύπτει την νέα εγγραφή τότε το επιπλέον hyper bounding box μεγαλώνει έτσι ώστε να καλύπτει και την καινούρια εγγραφή. Εάν ο όγκος του υπερβεί ένα κατώφλι τότε εκτελούμε τον αλγόριθμο 9 για να στείλουμε ενημέρωση στους master nodes αλλιώς στέλνουμε κάθε φορά αυτό το hyper bounding box μαζί με το bloom filter της ρίζας εάν έχει αλλάξει. Αντί για ολόκληρο το bloom filter θα μπορούσαμε να στείλουμε μόνο τα bits που έχουν αλλάζει αλλά αυτό θα αύξανε το κόστος και έτσι αποφεύγεται καθώς το bloom filter απαιτεί ελάχιστο χώρο. Η επιλογή για το κατώφλι είναι πολύ σημαντική καθώς πολύ μικρός αριθμός θα οδηγήσει σε εκτέλεση του αλγόριθμου 9 και αντίθετα πολύ μεγάλη τιμή θα αυξήσει τον αριθμό των αριθμό ερωτήσεων που προωθούνται στο κόμβο (περισσότερα false positives). Ο αλγόριθμος εισαγωγής νέων εγγραφών είναι:

Algorithm 10 Record insertion on Slave Nodes

```
1.  procedure INSERTRECORD(Record r, User u_id)
2.
3.  result = local_rtee.insert(r.boundingBox)
4.  send(INSERT_QUERY_REPLY,result,u_id)
5.  if( result && !LastSentUpdate.covers( r ) ) then
6.          BoundingBox bb = LastSentUpdate.ExtraBB
7.          bb.add( r )
8.          If(bb.volume > THRESSHOLD) then
9.                  invoke Algorithm 9
10.                 return;
```

```
11.         endif
12.         LastSentUpdate.ExtraBB = bb
13.         sendPartialUpdate()   // invoke Algorithm 11
14. endif
```

Αλγόριθμος 5.10. Εισαγωγή νέας εγγραφής στην συλλογή

Algorithm 11 Slave Nodes Send Partial Index Update

```
1.  procedure sendPartialUpdate()
2.  If( root.bloomfilter changed ) then
3.          for each (MASTER Node n) do
4.                  send(P_UPDATE1, LastSentUpdate.ExtraBB, root.bloomfilter, my_id, n)
5.          endfor
6.  else
7.          for each (MASTER Node n) do
8.                  send(P_UPDATE2, LastSentUpdate.ExtraBB, my_id, n)
9.          endfor
10. endif
```

Αλγόριθμος 5.11. Αποστολή μερικών ενημερώσεων

5.5.3.2. Διαγραφή εγγραφών

Ο αλγόριθμος για διαγραφή παρουσιάζεται πιο κάτω. Για κάθε επιτυχή διαγραφή αυξάνουμε ένα μετρητή. Εάν φτάσουμε μια προκαθορισμένη τιμή τότε εκτελείται ο Αλγόριθμος 5.9 για αποστολή ενημερώσεων. Η επιλογή για την τιμή αυτή προτείνεται όπως ρυθμίζεται δυναμικά με βάση το φόρτο εργασίας στο σύστημα.

Algorithm 12 Record deletion on Slave Nodes

```
1.  procedure DELETERECORD(Record r, User u_id)
2.
3.  result = local_rtee.delete(r.boundingBox)
4.  send(DELETE_QUERY_REPLY,result,u_id)
5.  if( result ) then //successful deletion
6.          deletionCounter++;
7.  endif
8.  if( deletionCounter > Delete_Threshold ) then
9.          invoke Algorithm 9
10.         deletionCounter = 0;
11. endif
```

Αλγόριθμος 5.12. Διαγραφή εγγραφής από την συλλογή

5.5.3.3. Διαχείριση μερικών ενημερώσεων

Ο αλγόριθμος για διαχείριση των μερικών ενημερώσεων από τους master nodes είναι πολύ απλός και παρόμοιος με τον αλγόριθμο 6 που παρουσιάζεται πιο πάνω. Όταν κάποιος master node λάβει μερική ενημέρωση τότε ανανεώνει μόνο τα αντίστοιχα πεδία της προηγούμενης ενημέρωσης που έλαβε από τον αντίστοιχο κόμβο.

5.6. Απαιτήσεις σε χώρο και χρόνο

5.6.1. Απαιτήσεις στους Master κόμβους

5.6.1.1. Απαιτήσεις σε χώρο

Στην πιο πάνω δομή κάθε master κόμβος χρειάζεται να αποθηκεύει μια ενημέρωση για κάθε slave κόμβο στο σύστημα. Το μέγεθος μιας ενημέρωσης είναι:

Size (Update) = Size (Bloom Filter) + #Bounding Boxes * Size (Bounding Box)

Το μέγεθος του bloom filter είναι ο αριθμός bits που χρειάζεται και εξαρτάται από το αριθμό εγγραφών που διαχειρίζεται ο κάθε κόμβος. Για να μην έχουμε μεγάλο αριθμό false positives πρέπει να έχουμε αρκετά μεγάλο αριθμό bits.

Το μέγεθος ενός bounding box διάστασης d είναι:

Size (Bounding Box) = 2 * d * Size (double)

Έστω ότι σε ένα σύστημα με N = 1000 κόμβους στο οποίο κάθε κόμβο είναι υπεύθυνος για 500000 εγγραφές, μέγεθος Bloom Filter 250000 bits, μέγιστο αριθμό Bounding Box/Update = 50 και εγγραφές στο 3-διάστατο χώρο με κάθε ιδιότητα να είναι double (8Kb). Τότε έχουμε:

$Cost_{Space}$ = N * size (Update)

= 1000 * (30,5 + 50 * 2 * 3 * 8) Kb

= 1000 * (30,5 + 2400) Kb

=~ 2,3 Gb

Ο πιο πάνω υπολογισμός φανερώνει πόσο αποδοτική είναι η δομή αυτή καθώς η συλλογή μας αποτελείται από 1000 * 500000 = 5*10^8 εγγραφές.

5.6.1.2. Απαιτήσεις σε χρόνο

Ο χρόνος που απαιτείται για την επεξεργασία ερωτημάτων, στους master nodes, είναι ο χρόνος που χρειάζεται για να βρούμε τους σχετικούς κόμβους. Ο χρόνος αυτός δεν είναι ίδιος για range και για point queries.

Για εύρεση των σχετικών ως κάποιο range query κόμβων απαιτείται να ψάξουμε όλα τα updates που διαχειριζόμαστε. Ο χρόνος αναζήτησης είναι γραμμικός ως προς τον αριθμό των κόμβων και τον αριθμό των bounding boxes/update.

Στην καλύτερη περίπτωση το query καλύπτεται από όλους τους κόμβους από το πρώτο bounding box. Τότε ο χρόνος είναι Θ(N).

Στην χειρότερη περίπτωση το query δεν καλύπτεται από κανένα κόμβο οπότε και πρέπει να ψάξουμε όλα τα bounding boxes για όλα τα updates. Ο χρόνος σε αυτή την περίπτωση είναι Θ(N*B) όπου B ο αριθμός των bounding boxes/update.

Στην μέση περίπτωση απαιτείται χρόνος O((N*B)/2).

Για εύρεση των σχετικών ως κάποιο point query κόμβων απαιτείται να ψάξουμε όλα τα bloom filter των updates που διαχειριζόμαστε. Καθώς σε O(1) μπορούμε να αποφασίσουμε εάν κάποιος κόμβος καλύπτει το σημείο που ψάχνουμε τότε χρειαζόμαστε Θ(N) για να βρούμε τους σχετικούς κόμβους για κάποιο point query.

5.6.2. Απαιτήσεις στους Slave κόμβους

Οι απαιτήσεις στους slave κόμβους είναι ο χώρος και ο χρόνος που απαιτούνται από την δομή R-Tree. Μετά από πειράματα παρατηρήθηκε ότι η δομή bloom filter δεν είναι απαραίτητη σε κάθε κόμβο του δένδρου και απλά αυξάνει τις απαιτήσεις σε χώρο. Έτσι χρησιμοποιείται ένα R-Tree και ένα bloom filter σε κάθε slave node.

Ο χώρος που απαιτείται είναι ο χώρος που χρειάζεται η δομή R-Tree και επιπλέον το μέγεθος των bloom filters * |V| όπου |V| ο αριθμός κόμβων του δένδρου. Παρόλο που μπορούμε να μειώσουμε τα όρια των MBR στους leaf nodes θα έχουμε μικρότερο αριθμό false positives, το ύψος του δένδρου θα μεγαλώσει και θα χρειαζόμαστε περισσότερο αποθηκευτικό χώρο.

Για αναζήτηση ενός σημείου και χρήση των bloom filters ο χρόνος που απαιτείται είναι O(1) ενώ εάν δεν θέλουμε να έχουμε false positives απαιτείται ο χρόνος αναζήτησης που είναι O(log n).

Καθώς η δομή βασίζεται κυρίως σε R-Tree ο χρόνος που απαιτείται για απάντηση σε κάποιο range query είναι O(log n).

5.7. Σύνοψη των παραμέτρων

Όπως αναφέρθηκε και πιο πάνω η δομή έχει αρκετές παραμέτρους οι οποίες εξαρτώνται από το σύστημα στο οποίο θα χρησιμοποιηθεί καθώς και το μέγεθος της συλλογής. Ο πιο κάτω πίνακας συνοψίζει τις παραμέτρους αυτές.

Παράμετρος	Περιγραφή
Αριθμός των master nodes	Ο αριθμός των απαιτούμενων master nodes καθορίζεται από τον συνολικό αριθμό των κόμβων στο σύστημα και τον ρυθμό άφιξης ερωτήσεων.
Αριθμός των slave nodes	Ο αριθμός των απαιτούμενων slave nodes καθορίζεται από το μέγεθος της συλλογής καθώς ο αριθμός δεδομένων που μπορεί να διαχειριστεί κάθε κόμβος περιορίζεται από το διαθέσιμο υλικό.
Μέγεθος bloom filter	Το μέγεθος του bloom filter πρέπει να είναι αρκετά μεγάλο έτσι ώστε να καλύπτει όλα τα δεδομένα με χαμηλό αριθμό false positives.
Μέγιστος αριθμός των bound boxes σε κάθε ενημέρωση (Max_Update_Size)	Όσο περισσότερα bounding boxes στέλνουμε σε κάθε ενημέρωση τόσο λιγότερα false positives έχουμε (για range queries). Σε αντίθεση αν στείλουμε πάρα πολλά το κόστος διαχείρισης τους από τους master nodes αυξάνει και έτσι θα έχουμε μεγαλύτερη καθυστέρηση σε range queries. Εάν έχουμε πολύ ψηλό όριο είναι πολύ πιθανόν ότι δεν θα το φτάσουμε καθώς επηρεάζεται από το ύψος και το fan out του δένδρου. Π.χ. για πλήρες δένδρο ύψους τρία και fan-out 10 μπορούμε να έχουμε το πολύ 10*10=100 bounding boxes (όλοι οι κόμβοι πάνω από τα φύλλα)
Μέγιστος όγκος του επιπλέον bounding box για να στείλουμε πλήρη ενημέρωση	Ο αριθμός αυτός είναι καλό να είναι σχετικά μικρός για να αποφύγουμε επεξεργασία περιττών ερωτήσεων. Στην πράξη τις πιο πολλές φορές εισαγωγή νέας εγγραφής, λόγω του τρόπου που επιλέγονται οι κόμβοι, δεν οδηγεί σε προσθήκη ή μεγάλη αύξηση του όγκου του επιπλέον bounding box.
Ελάχιστο αναμενόμενο κέρδος για να κάνουμε index τα παιδιά ενός κόμβου	Ο αριθμός αυτός φανερώνει πόσα false positives είναι διατεθειμένος ο κάθε κόμβος να επεξεργαστεί. Για πολύ χαμηλό όριο στέλνουμε όλους τους κόμβους του δένδρου πάνω από τα φύλλα και για πολύ ψηλό

(Min_Benefit)	στέλνουμε μόνο την ρίζα.
Μέγιστο fan out στους κόμβους του δένδρου (#children)	Ο μέγιστος αριθμός παιδιών ανά κόμβο του δένδρου καθορίζει το πόσο ψηλό θα είναι το δένδρο (περισσότερο αποθηκευτικό χώρο) καθώς και το πόσο περιοριστικά θα είναι τα bounding boxes στους κόμβους. Για μικρό fan out έχουμε πιο περιοριστικά όρια και έτσι λιγότερα false positives.
Αριθμός διαγραφών για να στείλουμε καινούρια ενημέρωση (Delete_Threshold)	Το κατώφλι αυτό είναι και πάλι ένα μέτρο του πόσα false positives είναι διατεθειμένος να δεχθεί ο κάθε κόμβος με αντάλλαγμα να μην υπολογίσει από την αρχή το σύνολο των κόμβων που πρέπει να στείλει και το κόστος αποστολής στο δίκτυο. Σημαντικός παράγοντας για τον καθορισμό του είναι ο φόρτος εργασίας που υπάρχει στο σύστημα. Για υψηλό φόρτο εργασίας είναι καλύτερα να είναι σχετικά μεγάλο για να μην σπαταλούμε χρόνο τον οποίο διαφορετικά θα χρησιμοποιούσαμε για επεξεργασία ερωτημάτων.

Πίνακας 5.1. Παράμετροι που επηρεάζουν την επίδοση του συστήματος

Κεφάλαιο 6 - Περιγραφή της προσομοίωσης – υλοποίησης

6.1. Εισαγωγή

Οι αλγόριθμοι που περιγράφονται στα κεφάλαια 4 και 5 έχουν προσομοιωθεί σε Java. Για προσομοίωση χρησιμοποιήθηκε το GridSim [22][23].

Το GridSim είναι ένα λογισμικό ανοικτού κώδικα για προσομοίωση κατανεμημένων συστημάτων που αναπτύχθηκε στο πανεπιστήμιο της Μελβούρνης από το Cloud Computing and Distributed Systems (CLOUDS) Laboratory και είναι βασισμένο στην βιβλιοθήκη SimJava[24].

Κάθε resource είναι μια κλάση που κληρονομεί την κλάση GridResource που με την σειρά της κληρονομεί την κλάση Sim_entity. Η επικοινωνία μεταξύ των κλάσεων (entities) γίνεται με αποστολή και παραλαβή events.

Κάθε event έχει ένα αποστολέα, ένα παραλήπτη, προκαθορισμένο μέγεθος σε bytes και κάποια καθυστέρηση σε εικονικό χρόνο – χρόνο προσομοίωσης. Έτσι τα events την στιγμή που δημιουργούνται καταχωρούνται σε μια ουρά με βάση τον χρόνο προσομοίωσης που θα είναι διαθέσιμα στο παραλήπτη τους. Καθώς ο χρόνος προχώρα σε κάθε χρονική στιγμή τα events που μπορούν να παραδοθούν τοποθετούνται σε μια άλλη ουρά από την οποία και κάθε entity παίρνει το επόμενο event που πρέπει να χειριστεί.

Χρησιμοποιώντας κατάλληλα την network υποστήριξη που παρέχει το GridSim στα μηνύματα που ανταλλάσσονται προστίθεται μια καθυστέρηση ανάλογα με την ταχύτητα σύνδεσης των resources και έτσι προσομοιώνεται και η καθυστέρηση του δικτύου.

6.2. QueryGrid package

6.2.1. Γενικά χαρακτηριστικά

Για την προσομοίωση των πιο πάνω αλγορίθμων αναπτύχθηκε το πακέτο querygrid. Το Σχήμα 6.1 παρουσιάζει το βασικό διάγραμμα κλάσεων σε UML για τους δύο αλγορίθμους.

Η κλάση QueryGridUser αντιπροσωπεύει κάποιο χρήστη ο οποίος θέτει ερωτήματα. Ο κάθε χρήστης θέτει ένα ερώτημα και περιμένει μέχρι να πάρει απάντηση πριν προχωρήσει επόμενο.

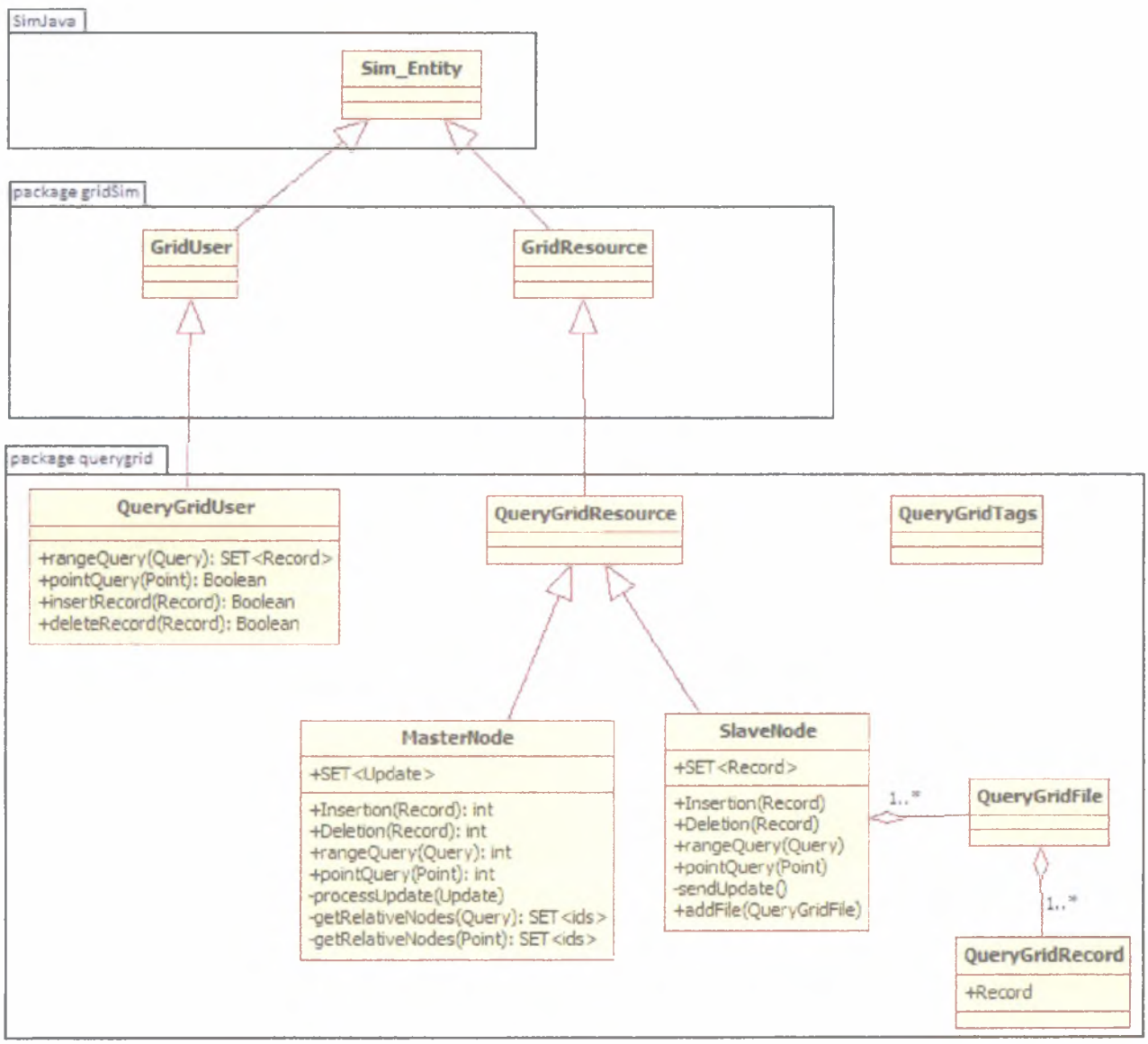
Οι κλάσεις MasterNode και SlaveNode αντιπροσωπεύουν αντίστοιχα ένα master ή ένα slave κόμβο στο σύστημα καθώς και τις βασικές συναρτήσεις που χρειάζονται για την επεξεργασία των ερωτήσεων.

Ο κάθε slave κόμβος είναι υπεύθυνος για κάποια αρχεία (QueryGridFile) όπου κάθε αρχείο περιλαμβάνει κάποιες εγγραφές. Πάνω στις εγγραφές αυτές κτίζεται το τοπικό ευρετήριο.

Ο κάθε master κόμβος είναι υπεύθυνος να διαχειριστεί ένα σύνολο από ενημερώσεις που δέχεται από τους slave nodes καθώς και να προωθεί τις ερωτήσεις των χρηστών στους αντίστοιχους κόμβους.

Η κλάση QueryGridTags περιλαμβάνει τα tags που χρησιμοποιούνται για τα events που ανταλλάζονται (π.χ. RANGE_QUERY, INDEX_UPDATE κλπ).

Επιπλέον το πακέτο querygrid εκμεταλλεύεται την δυνατότητα για προσομοίωση του δικτύου που παρέχει το GridSim για να προσομοιώσει την καθυστέρηση πάνω από το δίκτυο. Για να προσομοιωθεί η καθυστέρηση στην επεξεργασία των ερωτημάτων οι κόμβοι μετά την επεξεργασία κάποιου event - ερωτήματος περιμένουν όσο χρόνο χρειάστηκε πριν απαντήσουν και προχωρήσουν στην επεξεργασία του επόμενου event.



Σχήμα 6.1. Διάγραμμα κλάσεων UML του πακέτου querygrid

6.2.2. Προσομοίωση του EEMINC

Για την προσομοίωση της δομής ευρετηρίου που περιγράφεται στο Κεφάλαιο 4 χρησιμοποιούμε το πακέτο querygrid με τις πιο κάτω τροποποιήσεις.

Κάθε master node διατηρεί ένα R-Tree για τις ενημερώσεις που λαμβάνει και κάθε slave node διατηρεί ένα KD-Tree για τα δεδομένα και ως ενημερώσεις στέλνει τα bounding boxes όπως αυτά υπολογίζονται με βάση την μέθοδο της ίσης διαμοίρασης του χώρου.

6.2.3. Προσομοίωση της δομής A-Tree: Distributed Index for Multidimensional Data

Για την προσομοίωση της δομής A-Tree ο κάθε master κόμβος διατηρεί τις ενημερώσεις σε ένα πίνακα n θέσεων και ο κάθε slave κόμβος διατηρεί ένα R-Tree και ένα bloom filter για τα δεδομένα του.

Κεφάλαιο 7 - Πειραματική αξιολόγηση

7.1. Εισαγωγή

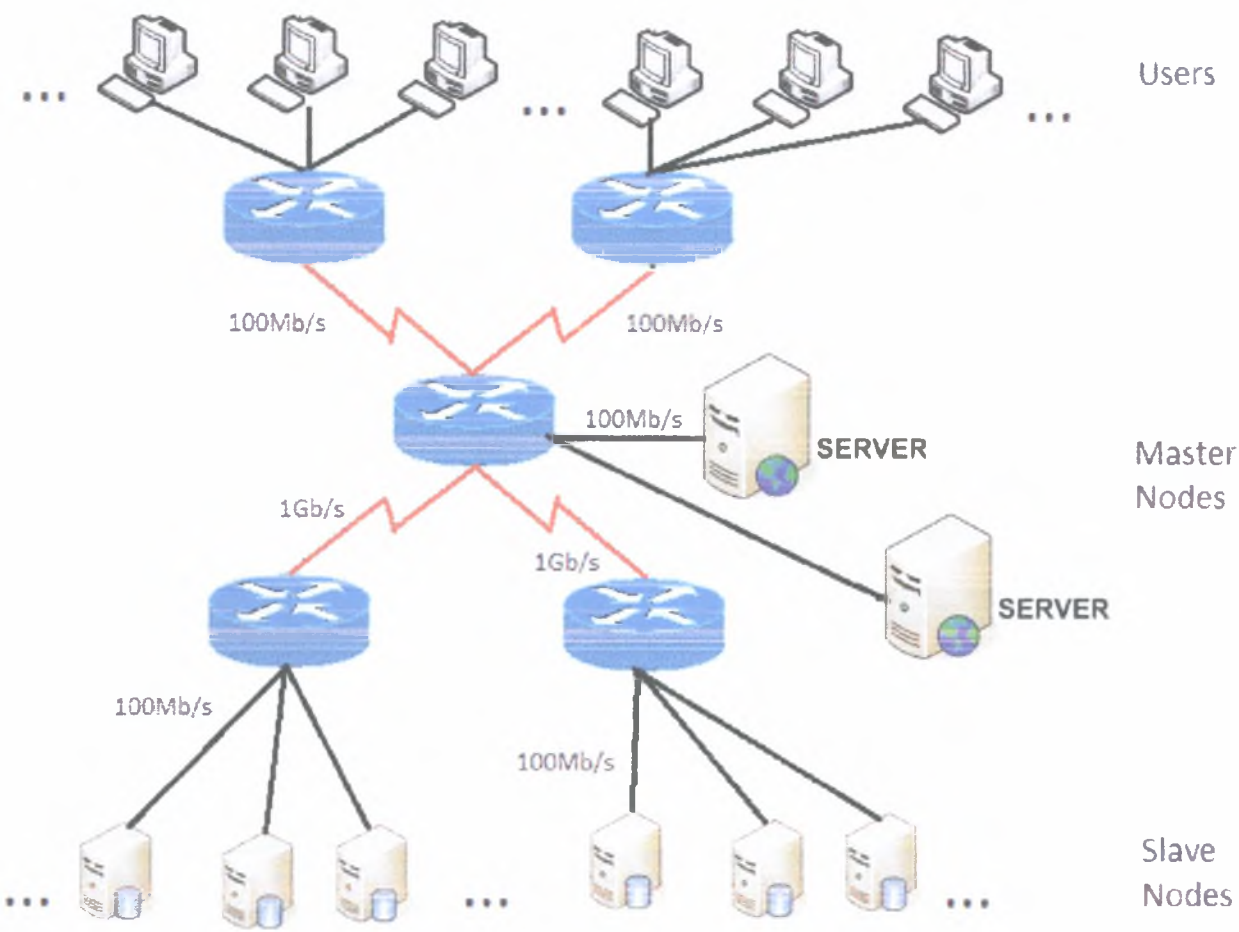
Για να αξιολογήσουμε τις πιο πάνω δομές μετρούμε την καθυστέρηση (latency) που έχουμε για point και range queries καθώς και τον αριθμό των false positives που έχουμε σε κάθε περίπτωση.

Πριν από κάθε πείραμα ορίζουμε τις πιο κάτω παραμέτρους:

- Τοπολογία δικτύου
- Αριθμός εγγραφών που διαχειρίζεται ο κάθε κόμβος – μέγεθος της συλλογής
- Αριθμός χρηστών και αριθμός ερωτήσεων που θα υποβάλουν
- Αριθμό κόμβων (masters και slaves)

Για τις πιο κάτω μετρήσεις καθορίζουμε την τοπολογία του δικτύου ως εξής. Συνολικά υπάρχουν πέντε routers. Στους δύο routers είναι συνδεδεμένοι οι χρήστες, σε ένα router είναι οι master nodes και στους άλλους δύο routers συνδέονται οι slave nodes. Η επιλογή αυτή μπορεί να θεωρηθεί αντιπροσωπευτική για ένα πραγματικό σύστημα καθώς οι ερωτήσεις των χρηστών φτάνουν από διαφορετικά μονοπάτια στους master nodes. Για τους slaves nodes χρησιμοποιούνται δύο routers καθώς ο αριθμός τους είναι πολύ μεγαλύτερος από ότι των master κόμβων και μπορούν να θεωρηθούν ότι αντιπροσωπεύουν το εσωτερικό ιδιωτικό δίκτυο. Το Σχήμα 7.1 παρουσιάζει την τοπολογία αυτή μαζί με την ταχύτητα κάθε σύνδεσης.

Ο υπολογιστής που χρησιμοποιήθηκε στα πειράματα είναι εξοπλισμένος με 16GB ram, τέσσερεις 4-πύρηνους επεξεργαστές Quad-Core AMD Opteron(tm) Processor 8350 και η χωρητικότητα του συστήματος είναι 200GB. Το λειτουργικό σύστημα είναι SUSE Linux Enterprise Server 10 (x86_64) με έκδοση πυρήνα 2.6.16.21-0.8-smp. Ο υπολογιστής διαμοιράζεται και με άλλους χρήστες.



Σχήμα 7.1. Τοπολογία δικτύου που χρησιμοποιήθηκε στην προσομοίωση

7.2. Πείραμα 1 – Μέση καθυστέρηση ως προς το μέγεθος της συλλογής

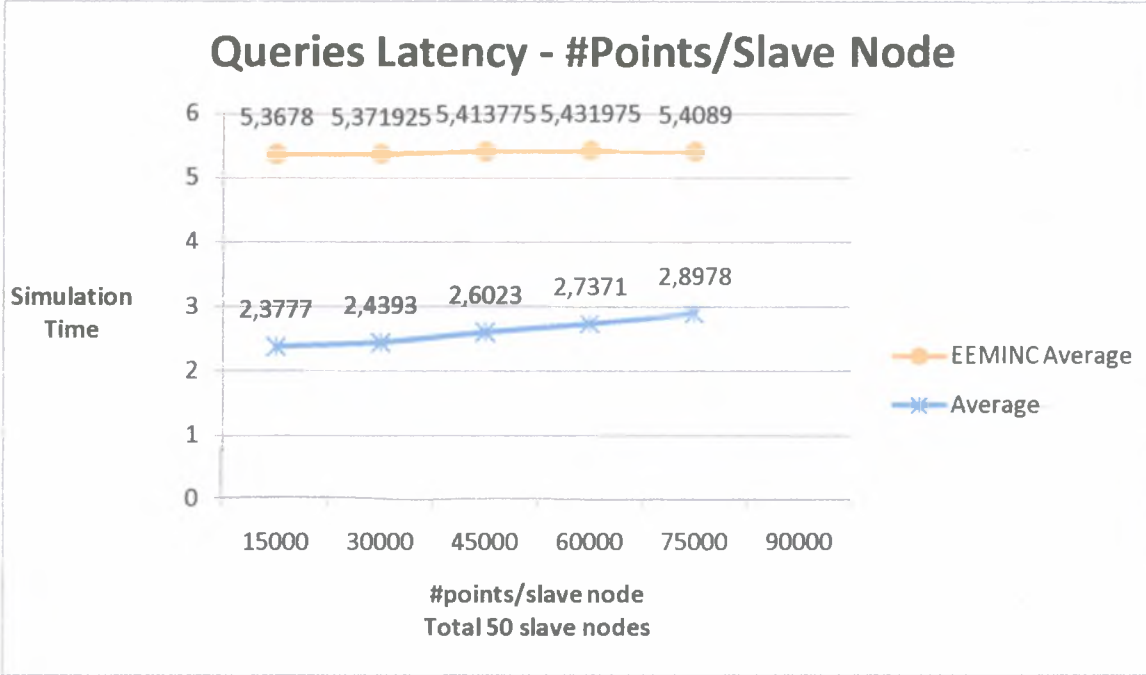
Στο πείραμα αυτό θέλουμε να αξιολογήσουμε πόσο αποδοτική είναι η δομή καθώς αυξάνει το μέγεθος της συλλογής.

Για το πιο κάτω πείραμα έχουμε 15 χρήστες οι οποίοι θέτουν 1000 point queries και 1000 range queries και υπολογίζουμε το μέσο όρο καθυστέρησης. Υπάρχουν πέντε master nodes και 50 slave nodes. Το fan out για το BR-Tree είναι 50 και το μέγεθος του bloom filter 200000 bits. Για τον πρώτο αλγόριθμο κάθε node cube μοιράζεται ίσα σε 20 υποδιαιρέσεις σε κάθε διάσταση.

Η πιο κάτω γραφική παράσταση παρουσιάζει την μέση καθυστέρηση ανά query. Παρατηρούμε ότι υπάρχει πολύ μικρή αύξηση στην καθυστέρηση καθώς το μέγεθος της συλλογής αυξάνει. Η κλίση είναι ελάχιστα μεγαλύτερη στον δεύτερο αλγόριθμο καθώς αυξάνει ο αριθμός των false positives από τα bloom filters. Παρά την μικρή αυτή αύξηση η δομή κλιμακώνει και μπορεί να διαχειριστεί αποδοτικά πολύ μεγάλο όγκο δεδομένων.

Επιπλέον μπορούμε να συμπεράνουμε ότι οι δομές που χρησιμοποιούνται στους slaves κόμβους είναι αποδοτικές όσο αφορά την διαχείριση των δεδομένων και ότι η μέση

καθυστέρηση αυξάνει λόγω των false positives, της καθυστέρησης στο δίκτυο και το κόστος διαχείρισης των ενημερώσεων – επιλογής σχετικών κόμβων στους master κόμβους.



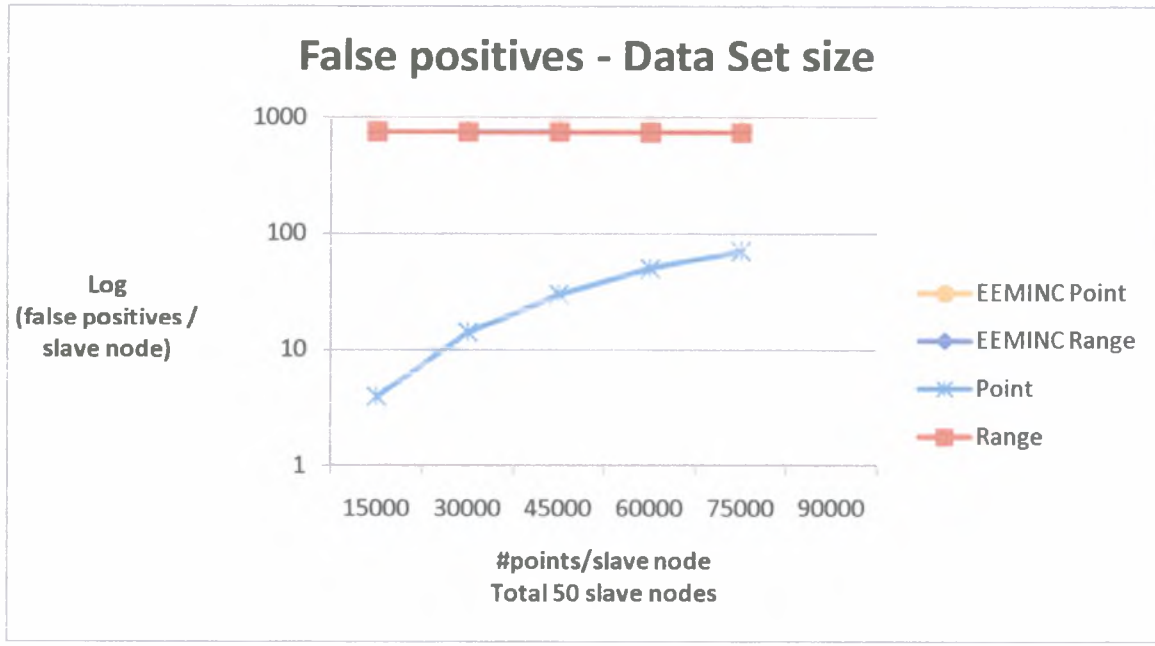
Σχήμα 7.2. Μέση καθυστέρηση ως προς το αριθμό εγγραφών σε κάθε slave κόμβο

7.3. Πείραμα 2 – Μέση καθυστέρηση ως προς τον μέγεθος του συστήματος

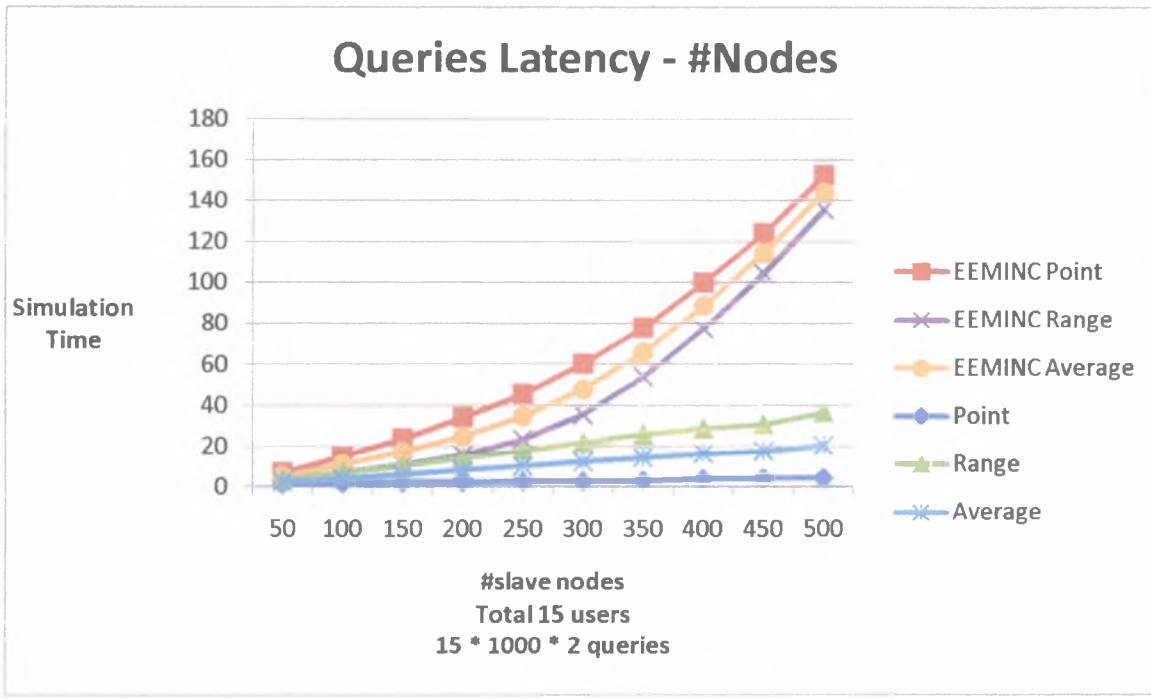
Σε αυτό το πείραμα διατηρούμε σταθερές όλες τις παραμέτρους και ίδιες με πριν και εξετάζουμε την επίδραση που έχει στην καθυστέρηση η αύξηση του αριθμού των slave nodes.

Κάθε κόμβος διαχειρίζεται 10000 εγγραφές. Το μέγεθος της συλλογής αυξάνει καθώς αυξάνει ο αριθμός των slave nodes. Τα αποτελέσματα παρουσιάζονται στο Σχήμα 7.4.

Πιο κάτω φαίνετε ο μέσος αριθμός των false positives ανά slave κόμβο σε λογαριθμική κλίμακα καθώς αυξάνει το μέγεθος της συλλογής. Η πιο κάτω γραφική παράσταση δικαιολογεί την αύξηση στην καθυστέρηση καθώς αυξάνει το μέγεθος της συλλογής και εξαρτάται από το αριθμό των false positive για point queries.



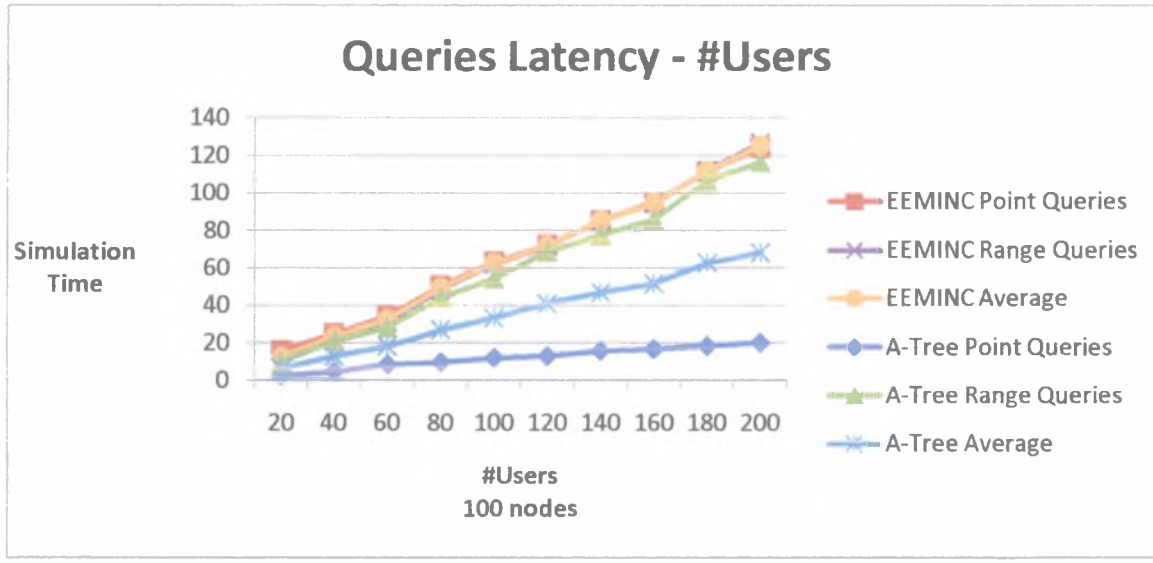
Σχήμα 7.3. Μέσος αριθμός false positives ανά κόμβο



Σχήμα 7.4. Μέση καθυστέρηση ως προς το αριθμό κόμβων στο σύστημα

Το Σχήμα 7.4 φανερώνει πόσο αποδοτική είναι η δομή που προτείνεται. Η καθυστέρηση ανά ερώτημα αυξάνει ελάχιστα καθώς αυξάνει το μέγεθος του συστήματος – δεδομένων και κόμβων.

7.4. Πείραμα 3 – Μέση καθυστέρηση ως προς τον αριθμό των χρηστών στο σύστημα



Σχήμα 7.5. Μέση καθυστέρηση ως προς το αριθμό χρηστών στο σύστημα

Κεφάλαιο 8 - Συμπεράσματα και μελλοντική εργασία

8.1. Συμπεράσματα

Για να πετύχουμε αποδοτική διαχείριση των δεδομένων πρέπει η δομή ευρετηρίου που θα χρησιμοποιήσουμε να μας παρέχει χαμηλό κόστος συντήρησης, να κλιμακώνει, να μπορεί να απαντά σε πολλά ερωτήματα στην μονάδα του χρόνου και να μπορεί να εκτελείται παράλληλα εκμεταλλευόμενη τις δυνατότητες του data center.

Στο κεφάλαιο 5 προτείνεται η δομή A-Tree για αντιμετώπιση του πιο πάνω προβλήματος βασισμένη στις δομές R-Tree και bloom filter. Με βάση την στρατηγική ανανεώσεων που προτείνεται μπορεί να απαντήσει σωστά σε point και range queries και απαιτεί ελάχιστο αποθηκευτικό χώρο.

Σύμφωνα με τις μετρήσεις που έχουν γίνει παρατηρείται ότι η δομή A-Tree χαρακτηρίζεται από όλα τα χαρακτηριστικά που περιγράφονται πιο πάνω. Μπορεί να ανταποκριθεί πλήρως στις πιο πάνω ανάγκες όπως αυτές περιγράφονται στο κεφάλαιο 3, κλιμακώνει και μπορεί να χρησιμοποιηθεί σε συστήματα μεγάλης κλίμακας καθώς η καθυστέρηση παραμένει χαμηλή και αυξάνει γραμμικά ως προς το πλήθος των κόμβων και το συνολικό μέγεθος της συλλογής.

8.2. Μελλοντική εργασία

Είναι βέβαιο ότι η βελτιστοποίηση και ανάπτυξη της πιο πάνω δομής δεν απορρίπτεται. Παρόλο που είναι πλήρης και χαρακτηρίζεται από όλα τα απαραίτητα χαρακτηριστικά πιο κάτω περιγράφονται κάποιες πιθανές βελτιστοποιήσεις.

- Αξιολόγηση της πιο πάνω δομής με υλοποίηση της δομής BR-Tree ως παραλλαγή άλλων δομών ευρετηρίου όπως για παράδειγμα R*-Tree[19][20]. Η δομή R*-Tree είναι μια παραλλαγή της δομής R-Tree που χρησιμοποιεί ένα πιο πολύπλοκο αλγόριθμο για node splitting. Σκοπός της δομής αυτής είναι να μοιράσει το overlap και το coverage των κόμβων χρησιμοποιώντας splits που ελαχιστοποιούν την περίμετρο των bounding boxes των νέων κόμβων.
- Αξιολόγηση και επέκταση της στρατηγικής ανανεώσεων που προτείνεται με σκοπό την ελαχιστοποίηση των false positives, ειδικά για range queries, καθώς και της χρήσης πόρων δικτύου για τις ενημερώσεις.
- Έλεγχος κατά πόσο συμφέρει η χρήση dynamic bloom filters για μηδενισμό των false positives όσο αφορά τα point queries.

Πέρα από τις πιο πάνω δοκιμές θα ήταν χρήσιμο να αναλυθούν οι παράμετροι που παρουσιάζει ο Πίνακας 5.1 με σκοπό να βρεθούν οι βέλτιστες τιμές ανάλογα με το κάθε σύστημα.

Βιβλιογραφία

[1] http://en.wikipedia.org/wiki/Cloud_computing

[2] Weiss. *Computing in the Clouds*. *netWorker*, 11(4):16-25, ACM Press, New York, USA, Dec. 2007.

[3] P. Barham *Xen and the Art of Virtualization*. In Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP 2003), Bolton Landing, USA.

[4] D. A. Patterson. *The Data Center Is The Computer*. Communications of the ACM, 51(1):105-105, Jan. 2008

[5] M. Armbrust *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, Feb. 10, 2009.

[6] CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services - http://www.cloudbus.org/cloudsim/

[7] Xiangyu Zhang, Jing Ai, Zhongyuan Wang, Jiaheng Lu, Xiaofeng Meng *An Efficient Multi-Dimensional Index for Cloud Data Management*. CloudDB '09, School of Information, Renmin University of China, 2009

[8] Yu Hua, Bin Xiao, Jianping Wang *BR-Tree: A Scalable Prototype for Supporting Multiple Queries of Multidimensional Data* IEEE Transactions on computers, vol. 58, no. 12, December 2009

[9] Peter Mell, Tim Grance *The NIST Definition of Cloud Computing*, National Institute of Standards and Technology, October 2009, http://csrc.nist.gov/groups/SNS/cloud-computing/

[10] TechTarget Search cloud computing (http://www.searchcloudcomputing.com) – Ορισμός Cloud computing - http://searchcloudcomputing.techtarget.com/definition/cloud-computing

[11] R. Devine, *Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm* Proc. Fourth Int'l Conf. Foundations of Data Organizations and Algorithms, pp. 101-114, 1993

[12] "Distributed Hash Tables Links" http://www.etse.urv.es/

[13] M. Harren, J.M. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker, and I. Stoica, *Complex Queries in DHT-Based Peer-to-Peer Networks* Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS), 2002

[14] M. K. Aguilera, W. Golab, and M. A. Shah, *A practical scalable distributed b-tree* in Proceedings of VLDB'08, Auckland, New Zealand, August 2008, pp. 598–609.

[15] S. Wu and K.-L. Wu, *An indexing framework for efficient retrieval on the cloud* IEEE Data Eng. Bull., vol. 32, pp.75–82, 2009.

[16] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi., *Indexing multi-dimensional data in a cloud system*, In SIGMOD Conference, pages 591–602, 2010.

[17] Antonin Guttman, *R-trees: A dynamic index structure for spatial searching*, SIGMOD Record, Vol. 14, No. 2, pp. 47-57, June 1984

[18] J. L. Bentley, *Multidimensional binary search trees used for associative searching,* Communications of the ACM - CACM, vol.18, no. 9, pp. 509-517, 1975.

[19] Hans-peter Kriegel, Ralf Schneider, Bernhard Seeger, Norbert Beckmann, *The R*-tree: an efficient and robust access method for points and rectangles,* SIGMOD Record, Vol. 19, No. 2, pp. 322-331, 1990

[20] R-Tree demo - http://donar.umiacs.umd.edu/quadtree/docs/rtree_split_rules.html

[21] Burton H. Bloom, *Space/Time Trade-offs in Hash Coding with Allowable Errors*, Communications of the ACM - CACM, vol. 13, no. 7, pp. 422-426, 1970

[22] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic, Rajkumar Buyya, *A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim*, Concurrency and Computation: Practice and Experience (CCPE), Online ISSN: 1532-0634, Printed ISSN: 1532-0626, 20(13): 1591-1609, Wiley Press, New York, USA, Sep. 2008

[23] GridSim: A Grid Simulation Toolkit for Resource Modeling and Application Scheduling for Parallel and Distributed Computing - http://www.cloudbus.org/gridsim/

[24] Howell F, McNab R., *SimJava: A discrete event simulation package for Java with applications in computer systems modeling*, Proceedings of the 1st International Conference on Web-based Modeling and Simulation, San Diego, CA. Society for Computer Simulation, 1998.