Πανεπιστήμιο Πειραιά
Τμήμα Πληροφορικής

Ένα πλαίσιο για την ανάλυση του φόρτου εργασίας του SQL optimizer στο Σύστημα Διαχείρισης Βάσεων Δεδομένων MySQL

Πτυχιακή Εργασία

Κατσικάρος Ευάγγελος του Ιωάννη (ΑΜ: Π 01053)

Επιβλέπων καθηγητής: Θεοδωρίδης Ιωάννης

Πειραιάς, Ιούνιος 2008

Πίνακας Περιεχομένων

1. Εισαγωγή

Τα υπολογιστικά συστήματα έχουν διεισδύσει πλέον στην καθημερινή ζωή και το πλήθος των δεδομένων που παράγουν είναι ολοένα και αυξανόμενο. Η διαχείριση του ολοένα αυξανόμενου όγκου δεδομένων, η οποία είναι ευθύνη των Συστημάτων Διαχείρισης Βάσεων Δεδομένων, είναι ένα θέμα το απασχολεί τον χώρο της πληροφορικής τόσο ερευνητικά όσο και επιχειρηματικά. Το αποτέλεσμα είναι ένα μεγάλο πλήθος από προγράμματα διαχείρισης βάσεων δεδομένων και ποικίλη έρευνα η οποία έχει δημιουργήσει νέου κλάδους όπως η Εξόρυξη Γνώσης, τα Γεωγραφικά Συστήματα Πληροφοριών και άλλα.

Τα Σχεσιακά Συστήματα Βάσεων Δεδομένων (RDBMS) είναι ο καθιερωμένος τρόπος, αυτήν την στιγμή, για την αποθήκευση μεγάλου όγκου των δεδομένων. Υπάρχουν πολλοί παράγοντες που επηρεάζουν την απόδοση ενός RDBMS, κατά την ανάγνωση και την εγγραφή των δεδομένων. Η ύπαρξη αυτών των παραγόντων έχουν οδηγήσει στην αυτοματοποίηση της εύρεση του καλύτερου τρόπου με τον οποίο μπορεί το RDBMS να προσπελάσει τα δεδομένα. Ένα από τα πιο σημαντικά τμήματα του RDBMS, το οποίο είναι υπεύθυνο για αυτήν την εύρεση του βέλτιστου τρόπου προσπέλασης των δεδομένων, είναι ο SQL optimizer.

Αντικείμενο της παρούσης εργασίας είναι η μελέτη της λειτουργίας του SQL optimizer και πιο συγκεκριμένα μελετάται ο τρόπος με τον οποίο μπορεί να υλοποιηθεί ένα framework για την μέτρηση του φόρτου εργασίας του SQL optimizer στο Σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (RDBMS) MySQL.

1.1. Κίνητρο της μελέτης

Ο SQL optimizer ενός RDBMS είναι ένα νευραλγικό σημείο του συστήματος, στο οποίο οφείλεται σημαντικό μέρος της απόδοσης του συστήματος. Η εύρεση ενός ικανοποιητικού ή ακόμα καλύτερα βέλτιστου πλάνου εκτέλεσης ενός αιτήματος, μπορεί να ελαχιστοποιήσει την χρήση των πόρων του συστήματος και να μειώσει δραματικά τον χρόνο εκτέλεσής του.

Η εύρεση των πιο χρονοβόρων τμημάτων του SQL optimizer μπορεί να βοηθήσει σημαντικά την περαιτέρω ανάπτυξη του RDBMS συστήματος. Το ανθρώπινο δυναμικό και η προσπάθεια που μπορεί να συγκεντρωθεί στην περαιτέρω ανάπτυξη είναι πεπερασμένη και συγκεκριμένη κάθε φορά. Επομένως, όταν βελτιώνονται ήδη υπάρχοντα τμήματα ενός προγράμματος, είναι προτιμότερο να επικεντρώνεται η προσπάθεια στην βελτίωση των πιο προβληματικών τμημάτων ή των τμημάτων τα οποία μπορούν να προσφέρουν τα μέγιστα στην βελτίωση της απόδοσης του προγράμματος.

Συμπερασματικά, η ενασχόληση με την απόδοση των τμημάτων ενός SQL optimizer αγγίζει ένα σημαντικό παράγοντα της υλοποίησης ενός RDBMS. Η εύρεση

των τμημάτων του SQL optimizer, των οποίων η βελτίωση μπορεί να επιφέρει το μεγαλύτερο όφελος στην ανάπτυξη είναι ένα σημαντικό κίνητρο για την εκπόνηση της μελέτης αυτής.

1.2. Στόχοι της μελέτης

Για τις ανάγκες της ανάλυσης υλοποιείται ένα framework (πλαίσιο) το οποίο είναι υπεύθυνο για την ενεργοποίηση του SQL optimizer της MySQL και τη συλλογή δεδομένων κατά την εκτέλεσή του SQL optimizer.

Το πλαίσιο αποτελείται από έναν client, ο οποίος στέλνει αιτήματα στο RDBMS MySQL και μιας διαδικασίας, η οποία παρεμβαίνει στην λειτουργία του RDBMS MySQL προκειμένου να συλλέξει και να τελικά να καταγράψει τα ζητούμενα μεγέθη.

Ο client πρέπει να είναι τέτοιος ώστε να παράγει ερωτήματα που καταφέρνουν να ενεργοποιήσουν τον SQL optimizer και τα οποία χρίζουν βασικής βελτιστοποίησης. Για τον λόγο αυτό χρησιμοποιούνται benchmark προορισμένα για βάσεις δεδομένων.

Οι παρεμβάσεις στην MySQL πρέπει να είναι σχεδιασμένες έτσι ώστε:

- να απαιτείται η ελάχιστη παρέμβαση στον πηγαίο κώδικα της MySQL.
- να επιβαρύνεται η εκτέλεση της MySQL όσο το δυνατό λιγότερο.
- να υπάρχει επεκτασιμότητα και ευελιξία, όσον αφορά τα μεγέθη τα οποία καταγράφονται και μετρούνται.
- να υπάρχει επεκτασιμότητα, όσον αφορά τον client/benchmark, ο οποίος χρησιμοποιείται για την αποστολή αιτημάτων στην MySQL.
- να μην απαιτούνται αλλαγές στο client/benchmark, το οποίο χρησιμοποιείται, κάνοντας την διαδικασία της καταγραφής των δεδομένων διαφανή στο client/benchmark.

Συμπερασματικά, ο στόχος μας είναι η δημιουργία ενός framework για την ανάλυση του φόρτου εργασίας (χρόνου εκτέλεσης) των τμημάτων από τα οποία αποτελείται ο SQL optimizer της MySQL και η εύρεση των πιο χρονοβόρων από αυτά, λαμβάνοντας υπόψιν τις παραπάνω προϋποθέσεις.

1.3. Δομή της μελέτης

Στο κεφάλαιο 2 αναπτύσσονται θέματα, που αφορούν το υπόβαθρο, το οποίο είναι απαραίτητο για την κατανόηση του περικειμένου του θέματος που μας απασχολεί. Στο 3ο κεφάλαιο περιγράφουμε την δομή και την υλοποίηση του πλαισίου της μελέτης. Στο 4ο κεφάλαιο δίνεται ένα παράδειγμα χρήσης του πλαισίου. Στο κεφάλαιο 5 αναφέρουμε συμπεράσματα και περαιτέρω έρευνα που μπορεί να εκκινήσει με αφορμή την παρούσα μελέτη. Στο κεφάλαιο 6 δίνονται οι βιβλιογραφικές αναφορές και στο κεφάλαιο 7 ένα γλωσσάρι με την αντιστοιχία ελληνικών και αγγλικών όρων που εμφανίστηκαν στο κείμενο.

2. Υπόβαθρο μελέτης

Για τις ανάγκες της παρούσας ανάλυσης, υλοποιείται ένα framework το οποίο πραγματοποιεί την συλλογή των δεδομένων, όσον αφορά τον φόρτο εργασίας του SQL optimizer στο RDBMS MySQL. Το framework αποτελείται από ένα client/benchmark το οποίο στέλνει αιτήματα στο RDBMS MySQL και μιας διαδικασίας, η οποία παρεμβαίνει στην λειτουργία του RDBMS MySQL προκειμένου να καταγράψει ορισμένα μεγέθη. Στο παρόν εισαγωγικό κεφάλαιο παρουσιάζονται θέματα, τα οποία είναι απαραίτητα για την κατανόηση του χώρου, στον οποίο ανήκει η μελέτη και το framework, των εργαλείων που εμπλέκονται καθώς και των προκλήσεων που παρουσιάζονται.

Η δομή του κεφαλαίου έχει ως εξής: στην ενότητα 2.1 παρουσιάζουμε τους λόγους ύπαρξης των συστημάτων διαχείρισης βάσεων δεδομένων. Στην ενότητα 2.2 αναφερόμαστε στα benchmark που αφορούν τον χώρο των βάσεων δεδομένων, και πως καταλήξαμε στην χρήση του OSDB benchmark για την παρούσα μελέτη. Στην ενότητα 2.3 παρουσιάζουμε το RDBMS MySQL, το οποίο είναι αντικείμενο της μελέτης, και τους λόγους για τους οποίου χρησιμοποιείται στην μελέτη. Στην ενότητα 2.4 αναπτύσσουμε την ανάγκη για την ύπαρξη του SQL optimizer και το περίγραμμα της υλοποίησής του στην MySQL. Στην ενότητα 2.5 δίνεται η σύνοψη του κεφαλαίου.

2.1. Συστήματα διαχείρισης βάσεων δεδομένων

Ορισμένα από τα ζητήματα που πρέπει να επιλυθούν κατά την διαχείριση μεγάλου όγκου δεδομένων εμφανίζονται συχνά. Τέτοια ζητήματα είναι (Ramakrishnan & Gehrke, 2002):

- η εγγραφή και η ανάγνωση των δεδομένων από και προς διαφορετικά μέσα αποθήκευσης: μέσω δικτύου, τοπικά, σε ένα αρχείο, σε πολλά αρχεία, κατανεμημένα σε ένα υπολογιστικό σύστημα, κατανεμημένα σε διαφορετικά υπολογιστικά συστήματα (cluster) κτλ.
- η διαχείριση της μνήμης και η μεταφορά των δεδομένων από το μέσο όπου είναι αποθηκευμένα μόνιμα.
- ο συγχρονισμός πολλών χρηστών και κλείδωμα των δεδομένων έτσι ώστε να μην εμφανιστούν ασυνέπειες στα δεδομένα από την εγγραφή και την ανάγνωση από διαφορετικούς χρήστες.
- ο έλεγχος της πρόσβασης στα δεδομένα ανάλογα με τα δικαιώματα κάθε χρήστη.

Ακριβώς επειδή τα ζητήματα αυτά εμφανίζονται πολύ συχνά, δημιουργήθηκαν κάποια συστήματα, τα οποία υλοποιούν εσωτερικά αυτές τις λειτουργίες και παρουσιάζουν στον χρήστη μια διεπαφή για την διαχείριση αυτών των ζητημάτων. Τα συστήματα αυτά ονομάζονται Συστήματα Διαχείρισης Βάσεων Δεδομένων ή DBMS

(DataBase Management System) και ουσιαστικά παρέχουν ένα επίπεδο αφαίρεσης πάνω από τα πραγματικά δεδομένα και την διαχείρισή τους. Με αυτόν τον τρόπο, κρύβουν τις λεπτομέρειες της υλοποίησης των ζητημάτων αυτών και προσφέρουν στον τελικό χρήστη του DBMS μια απλούστερη εικόνα για την διαχείριση των δεδομένων.

Το 1970 προτάθηκε ένα σχήμα αναπαράστασης των δεδομένων το οποίο ονομάζεται σχεσιακό μοντέλο δεδομένων (Codd , 1970). Τα συστήματα βάσεων δεδομένων τα οποία υλοποιούν αυτό το σχήμα για την αναπαράσταση των δεδομένων ονομάζονται Σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων ή αλλιώς RDBMS (Relational DataBase Management System).

2.1.1. SQL

Η γλώσσα SQL έχει καθιερωθεί ως το προγραμματιστικό πρότυπο, μέσω του οποίου ο χρήστης διαχειρίζεται ένα RDBMS και τα δεδομένα τα οποία είναι αποθηκευμένα σε αυτό. Όπως αναφέρεται στα (Hare, 2006) και (Lans, 1989), το 1986 ολοκληρώθηκε το πρώτο standard το οποίο έγινε δεκτό από τα ινστιτούτο ANSI (American National Standards Institute) και τον οργανισμό ISO (International Standards Organization). Επεκτάσεις και αναθεωρήσεις εκδόθηκαν το 1989 και 1992. Στην συνέχεια η προσπάθεια της προτυποποίησης διαχωρίστηκε σε διαφορετικές ομάδες, οι οποίες κατά καιρούς συγχώνευαν τις εργασίες που είχαν ολοκληρωθεί. Σημαντικοί σταθμοί είναι το πρότυπο SQL-92, το SQL:1999 και το SQL:2003 >>>(αυθεντικό reference).

2.2. Τα benchmark στον χώρο των βάσεων δεδομένων

Στον χώρο της πληροφορικής, το benchmark είναι η διαδικασία κατά την οποία εκτελούνται ένα ή περισσότερα προγράμματα, με σκοπό την μέτρηση της απόδοσης ενός υπολογιστικού συστήματος ή προγράμματος. Χαρακτηριστικά παραδείγματα benchmark είναι η σύγκριση της απόδοσης μεταξύ διαφορετικών επεξεργαστών, compiler, συστημάτων βάσεων δεδομένων κτλ. (Gray, 1993).

Ένα benchmark, βοηθά στην απάντηση ερωτημάτων όπως:

- Ποιο υπολογιστικό σύστημα ή τμήμα ενός συστήματος είναι το πιο συμφέρον προς επένδυση.
- Ποιο μέρος ενός συστήματος δημιουργεί την μεγαλύτερη συμφόρηση.

Αυτό το οποίο επιβάλλεται, είναι ο ορισμός των μεγεθών πάνω στο οποίο θα μετρηθεί η απόδοση και ο ορισμός της διαδικασίας με την οποία θα γίνει αυτό (Gray, 1993).

Επίσης είναι σημαντική η εξέταση του περιβάλλοντος στο οποίο βρίσκεται το υπό εξέταση υπολογιστικό σύστημα ώστε να μελετηθούν παράγοντες οι οποίοι μπορούν να επηρεάσουν την μέτρηση της απόδοσης (Patterson &Hennessy, 2005). Στην παρούσα ανάλυση το μέγεθος το οποίο αποτελεί αντικείμενο έρευνας είναι ο χρόνος, ο οποίος δαπανάται συνολικά στην εκτέλεση του SQL optimizer, και ο χρόνος ο

οποίος δαπανάται σε κάθε τμήμα του. Ο χρόνος αυτός μπορεί να επηρεαστεί από την απόδοση I/O. Αυτή με την σειρά της επηρεάζεται από hardware και software παράγοντες: αρχιτεκτονική του συστήματος δοκιμής, ταχύτητα προσπέλασης δίσκου και μνήμης, άλλες I/O δραστηριότητες, ρυθμίσεις του λειτουργικού συστήματος και άλλων υπο-συστημάτων που παίζουν ρόλο όπως συστημάτων αρχείων, κτλ. Επίσης σημαντικό ρόλο διαδραματίζουν οι επιλογές κατά την διαδικασία της μεταγλώττισης του πηγαίου κώδικα του RDBMS, για την παραγωγή του εκτελέσιμου προγράμματος, καθώς και οι ρυθμίσεις των παραμέτρων του RDBMS κατά την εκτέλεσή του. Επομένως, τα αποτελέσματα που θα εξαχθούν πρέπει να μελετηθούν λαμβάνοντας υπόψιν όλα αυτά τα θέματα.

Για να είναι εφικτό να εξαχθούν ασφαλή συμπεράσματα, πρέπει να ληφθούν υπόψιν οι εξής παράγοντες που ενδέχεται να επηρεάσουν την απόδοση ενός συστήματος:

- αρχιτεκτονική της μηχανής
- λειτουργικό σύστημα
- compilation της εφαρμογής
- caching σε διάφορα επίπεδα (πυρήνας, σύστημα αρχείων)
- σύστημα αρχείων

Η δοκιμή πολλών συνδυασμών αυτών των παραμέτρων είναι ο μόνος τρόπος για να επιβεβαιωθούν τα κοινά, σε όλες ή στις περισσότερες περιπτώσεις, χρονοβόρα τμήματα. Η δοκιμή οφείλει να περιλαμβάνει πολλές αρχιτεκτονικές, μηχανήματα, πυρήνες, συστήματα αρχείων, ρυθμίσεων στο compilation, ρυθμίσεις στο σύστημα αρχείων κτλ.

Δίχως την δοκιμή πολλών συνδυασμών, τα δεδομένα τα οποία εξάγονται, δεν μπορούν παρά να έχουν ενδεικτικό χαρακτήρα και να μην μπορούν να χρησιμοποιηθούν ως επιβεβαιωμένα.

2.2.1. Wisconsin benchmark

Στις αρχές της δεκαετίας του 80 στο πανεπιστήμιο του Wisconsin αναπτύχθηκε μια ένα υπολογιστικό σύστημα επιφορτισμένο με την εκτέλεση συστημάτων βάσεων δεδομένων το οποίο ονομαζόταν DIRECT (Gray, 1993). Το 1983 δημιουργήθηκε το Wisconsin benchmark το οποίο αποσκοπούσε στην αξιολόγηση της μηχανής DIRECT, τόσο ως προς τον εαυτό της όσο και προς σχεσιακά συστήματα βάσεων δεδομένων τα οποία έτρεχαν σε υπολογιστές γενικής χρήσης (Date, 2004). Λίγο μετά την δημοσίευσή του, το benchmark αυτό έγινε δημοφιλές επειδή ήταν το πρώτο benchmark στον χώρο των βάσεων δεδομένων το οποίο προσέφερε απλότητα και ταυτόχρονα αποτελέσματα από διαφορετικά συστήματα και αρχιτεκτονικές. Η αποδοχή του ήταν τέτοια ώστε για αρκετά χρόνια ήταν βιομηχανικό στάνταρ. Οι περιορισμοί του αφορούσαν την έλλειψη επεκτασιμότητας και παραμετροποίησης, επειδή τόσο το μέγεθος της βάσης δεδομένων όσο και το πλήθος των χρηστών (ένας) έμεναν σταθερά και αμετάβλητα.

2.2.2. AS3AP benchmark

Αν και πετυχημένο, το Wisconsin benchmark παρουσίαζε ελλείψεις. Το κίνητρο για την δημιουργία του AS3AP (ANSI SQL Standard Scaleable and Portable) >>>(αυθεντικό reference) ήταν η δημιουργία ενός benchmark το οποίο θα κάλυπτε αυτές τις ελλείψεις, θα παρείχε την απλότητα του Wisconsin και θα πρόσφερε επιπλέον επεκτασιμότητα, συνάφεια και ευκολία στην διαχείριση (Gray, 1991).

Πιο συγκεκριμένα οι στόχοι του AS3AP είναι:

- η παροχή ενός κατανοητού και εύκολου στην διαχείριση συνόλου από τεστ για την μελέτη της επεξεργαστικής ισχύος, όσον αφορά τις βάσεις δεδομένων.
- να είναι επεκτάσιμο και φορητό σε διαφορετικές αρχιτεκτονικές.
- να απαιτείται ελάχιστη ανθρώπινη παρέμβαση στην υλοποίηση και την εκτέλεση.
- η παροχή ενός ομοιόμορφου μετρήσιμου μεγέθους.

Για ένα συγκεκριμένο RDBMS, το AS3AP ορίζει ως αποτέλεσμα το μέγεθος μιας βάσης δεδομένων. Το μέγεθος αυτό είναι το μέγιστο μέγεθος της βάσης δεδομένων, για το οποίο το σύστημα είναι εφικτό να εκτελέσει το AS3AP τεστ, για έναν και πολλούς χρήστες, σε χρονικό διάστημα κάτω των 12 ωρών. Το μέγεθος της βάσης δεδομένων είναι ένα απόλυτο μέγεθος μέτρησης της απόδοσης το οποίο με την σειρά του μπορεί να χρησιμοποιηθεί για την παραγωγή και άλλων μεγεθών.

Το AS3AP χωρίζεται σε δύο τμήματα:

- Το τεστ ενός χρήστη, τα οποία περιλαμβάνουν:
    1. εργαλεία για το φόρτωμα και την κατασκευή της βάσης
    2. αιτήματα τα οποία είναι σχεδιασμένα να ελέγξουν τις μεθόδους πρόσβασης στα δεδομένα και βασικές SQL βελτιστοποιήσεις όπως: επιλογές, απλές συνδέσεις, προβολές, συναθροίσεις και τροποποιήσεις.
- Το τεστ πολλών χρηστών, τα οποία μοντελοποιούν διαφορετικούς τύπους φόρτου της βάσης:
    1. εργασίες άμεσης επεξεργασίας συναλλαγών (OLTP)
    2. εργασίες ανάκτησης πληροφοριών (information retrieval)
    3. μεικτές εργασίες που περιλαμβάνουν σύντομες συναλλαγές, αιτήματα για την σύνταξη αναφορών (report queries), πλήρη σάρωση μιας σχέσης (relation scan) και μεγάλες συναλλαγές.

2.2.3. OSDB benchmark

Το OSDB benchmark (Open Source Database Benchmark) δημιουργήθηκε για μια μελέτη στην εταιρία Compaq Computer Corporation με σκοπό την μέτρηση της απόδοσης στην I/O διαπερατότητα και στη γενικότερη επεξεργαστική δύναμη του συστήματος GNU Linux/Alpha (OSDB, 2008).

Για την υλοποίηση της μελέτης αποφασίστηκε να μην γίνει χρηματική επένδυση σε κάποιο ήδη υπάρχον εμπορικό benchmark. Επιλέχθηκε η υλοποίηση ενός benchmark βασισμένου στην επίσημη περιγραφή του AS3AP benchmark όπως αυτό είναι τεκμηριωμένο στο (Gray, 1993).

Αν και το OSDB είναι βασισμένο στην περιγραφή του AS3AP υπάρχουν ορισμένες διαφορές (OSDB, 2008):

- Μετρήσιμα μεγέθη: το AS3AP αναφέρει ένα μόνο μέγεθος: το μέγεθος της μεγαλύτερης βάσης δεδομένων η οποία μπορεί να χρησιμοποιηθεί για να ολοκληρωθεί το benchmark σε λιγότερες από 12 ώρες. Αντίθετα, το OSDB αναφέρει πληθώρα ξεχωριστών αποτελεσμάτων για την κάθε φάση της δοκιμής.
- Έλλειψη λειτουργικότητας: το AS3AP απαιτεί συμφωνία με μια πλήρως συμβατή SQL υλοποίηση ώστε να εκτελεστεί. Αντίθετα, το OSDB μπορεί να εκτελεστεί και σε μερικώς συμβατές SQL υλοποιήσεις καθώς και σε υλοποιήσεις που δεν ακολουθούν την SQL.
- Αποσαφηνίσεις: στο specification του AS3AP ορισμένα σημεία είναι διφορούμενα. Επειδή το OSDB αποτελεί υλοποίηση, αυτές τις περιπτώσεις έχουν αποσαφηνιστεί με συγκεκριμένο τρόπο.
- Αυθαίρετες αλλαγές: έχουν γίνει διάφορες αλλαγές με σκοπό να διαχωριστούν οι επαναλαμβανόμενες ενέργειες από αυτές που πραγματοποιούνται μόνο μια φορά π.χ. αρχική δημιουργία της βάσης δεδομένων και ενέργειες που επαναλαμβάνονται κάθε μέρα.

Αυτό που πρέπει να τονιστεί είναι πως, στην παρούσα εργασία, το OSDB δεν χρησιμοποιείται για τα αποτελέσματα τα οποία παράγονται από την εκτέλεση του. Τα αποτελέσματα αυτά δεν λαμβάνονται καθόλου υπόψιν για 2 λόγους:

- Μας ενδιαφέρουν αποτελέσματα από συγκεκριμένο τμήμα της εκτέλεσης του αιτήματος (SQL optimizer) που στέλνει ο OSDB client. Το OSDB benchmark λαμβάνει υπόψιν του το συνολικό χρονικό διάστημα από την στιγμή, που ο OSDB client έστειλε το αίτημα, έως ότου λάβει την απάντηση για την εκτέλεσή του, μέγεθος το οποίο δεν αφορά την παρούσα μελέτη.
- Τα αποτελέσματα που λαμβάνει το benchmark είναι παραποιημένα. Ανάμεσα στον RDBMS server και στον OSDB client παρεμβάλλεται ένας proxy server. Προκειμένου αυτός να πετύχει συγχρονισμό για την αποθήκευση των αποτελεσμάτων, που χρειάζονται για την παρούσα ανάλυση, υπάρχει καθυστέρηση χρονικά στην απάντηση που στέλνει ο server στον client. Επομένως ο OSDB client παραλαμβάνει παραποιημένα χρονικά αποτελέσματα.

Τα αιτήματα που στέλνει το OSDB benchmark, κατά την διάρκεια της εκτέλεσής του, εκτός από κάποια CREATE, JOINs, INSERT, DELETE ανήκουν στην συντριπτική τους πλειοψηφία τους (πάνω από 98%) σε μια από τις επόμενες 2 κατηγορίες:

```
1. SELECT col_key, col_code, col_date, col_signed,
2. col_name
3. FROM updates
4. WHERE col_key = CONSTANT

5. UPDATE updates
6. SET col_signed=col_signed+1
7. WHERE col_key = CONSTANT
```

όπου 0 <= CONSTANT <= 9999.

Επίσης να σημειωθεί πως στην στήλη updates.col_keys, δημιουργείται ένα ευρετήριο UNIQUE.

Τα αιτήματα που αφορούν την δημιουργία της βάσης δεδομένων του OSDB καθώς και τα υπόλοιπα αιτήματα, που στέλνονται από το OSDB κατά την διάρκεια της εκτέλεσής του, βρίσκονται στο Παράρτημα 8.

Συμπερασματικά, το OSDB benchmark καλύπτει τους στόχους της ανάλυση, ανάλυσης που προτείνουμε στην παρούσα εργασία, όσον αφορά τις προϋποθέσεις που πρέπει να πληρεί ο client/benchmark του framework. Το OSDB benchmark χρησιμοποιείται στην παρούσα μελέτη ως βάση, μόνο για τα SQL αιτήματα που στέλνει στον server. Το σύνολο των SQL αιτημάτων τα οποία περιλαμβάνονται στο OSDB benchmark καλύπτουν ένα δείγμα καθημερινών ενεργειών που λαμβάνουν χώρα σε ένα RDBMS.

Αυτό που δεν καλύπτεται είναι ευρεία χρήση αιτημάτων που χρίζουν σοβαρής βελτιστοποίησης, έτσι ώστε να δοκιμαστούν οι αντοχές του SQL optimizer.

2.3. MySQL και γιατί χρησιμοποιείται στην μελέτη

Η MySQL είναι ένα RDMBS το οποίο αναπτύσσεται από την εταιρία “MySQL AB”. Το σύστημα αυτό είναι ένα δημοφιλές client/server RDBMS στον χώρο του ανοικτού/ελεύθερου λογισμικού αλλά και στον ευρύτερο χώρο των βάσεων δεδομένων (MySQL, 2007), (MySQL, 2008).

Η MySQL διανέμεται υπό διπλή άδεια: διανέμεται υπό την άδεια GPL (GNU General Public License) αλλά διατίθεται και με εμπορική άδεια σε περίπτωση που ζητούμενο είναι να παρακαμφθούν οι όροι της GPL ([MySQL, 2008]). Σύμφωνα με το (FSF, 2008 a), η άδεια GPL πληρεί τις προϋποθέσεις για να χαρακτηριστεί μια άδεια διανομής ως άδεια ελεύθερου λογισμικού, οι οποίες είναι οι εξής, σύμφωνα με το (FSF, 2008 b):

- ελευθερία χρήσης του προγράμματος για οποιοδήποτε λόγο
- ελευθερία μελέτης του τρόπου λειτουργίας του προγράμματος και μετατροπής του ώστε να καλύπτει τις προσωπικές ανάγκες του καθενός. Η πρόσβαση στον πηγαίο κώδικα είναι μια προϋπόθεση για αυτό.
- ελευθερία διανομής αντιγράφων του προγράμματος
- ελευθερία βελτίωσης του προγράμματος και διανομής των βελτιώσεων στο κοινό. Η πρόσβαση στον πηγαίο κώδικα είναι μια προϋπόθεση για αυτό.

Η MySQL χρησιμοποιείται στην παρούσα μελέτη, αντί άλλων γνωστών εμπορικών λύσεων (π.χ. Oracle, MS SQL Server) για δύο λόγους.

1. Η άδεια GPL, υπό την οποία κυκλοφορεί η MySQL, μας δίνει την δυνατότητα να δούμε τον κώδικα του προγράμματος. Αυτό μας επιτρέπει, να μελετήσουμε την υλοποίηση των τμημάτων του προγράμματος που μας ενδιαφέρουν. Επιπλέον η χρήση του προγράμματος είναι ελεύθερη, άρα και η δημοσίευση αποτελεσμάτων από την χρήση του είναι ελεύθερη. Επομένως, μας δίνεται η δυνατότητα να τρέξουμε το benchmark και να δημοσιεύσουμε τα αποτελέσματά

του δίχως περιορισμούς. Σε αντίθεση, σε ένα σύστημα κλειστού κώδικα δεν παρέχεται η δυνατότητα μελέτης του πηγαίου κώδικα, εφόσον δεν παρέχεται ο κώδικας. Επιπλέον, απαγορεύεται η μελέτη του κώδικα που βρίσκεται στην μνήμη μέσω της διαδικασίας του reverse engineering. Απαγορεύεται δηλαδή η μελέτη της υλοποίησης τμημάτων του προγράμματος. Επιπλέον, απαγορεύεται η δημοσίευση αποτελεσμάτων benchmark δίχως την συμφωνία της εταιρίας που το εκδίδει.

2. Η άδεια GPL, υπό την οποία κυκλοφορεί η MySQL, μας δίνει την δυνατότητα να μεταβάλουμε τον κώδικα του προγράμματος επομένως μπορούμε να εισάγουμε κώδικα ο οποίος μας βοηθά στην εύρεση των λεπτομερειών που ζητά η ανάλυση. Σε αντίθεση, σε ένα σύστημα κλειστού κώδικα απαγορεύεται η μετατροπή του προγράμματος και του κώδικα μηχανής στην μνήμη. Επομένως δεν είναι εφικτή η παρέμβαση κατά την εκτέλεση του προγράμματος.

Συμπερασματικά, η MySQL χρησιμοποιείται ως αντικείμενο μελέτης επειδή:

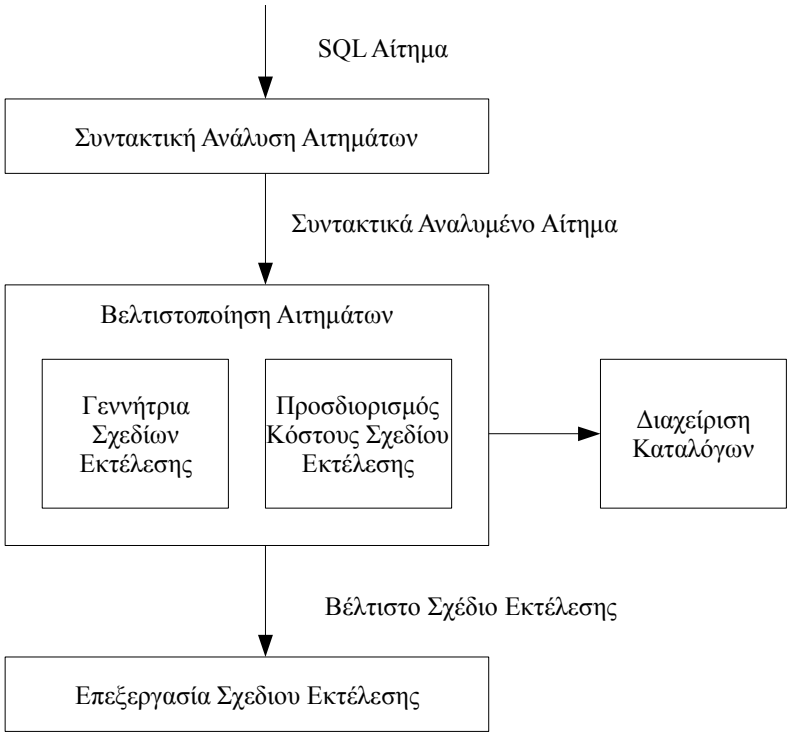
1. είναι ένα δημοφιλές RDBMS, επομένως η μελέτη του εμφανίζει ενδιαφέρον για μεγάλο τμήμα του χώρου της πληροφορικής, και
2. μπορούμε να μελετήσουμε και να παρέμβουμε στο πρόγραμμα με τρόπο που διευκολύνει και διευρύνει τους στόχους της μελέτης.

2.4. Ο SQL optimizer

Η βελτιστοποίηση παρουσιάζει τόσο μια πρόκληση όσο και μια ευκαιρία για ένα RDBMS σύστημα. Πρόκληση παρουσιάζει, επειδή η βελτιστοποίηση είναι απαραίτητη αν είναι επιθυμητό ένα αποδεκτό επίπεδο απόδοσης. Ευκαιρία παρουσιάζει, επειδή η δομή του σχεσιακού μοντέλου ευνοεί την αυτόματη βελτιστοποίηση. Το σχεσιακό μοντέλο είναι ένα μοντέλο το οποίο παρέχει αφαίρεση σε υψηλό επίπεδο, σε αντίθεση με τα μη-σχεσιακά μοντέλα όπου για την βελτιστοποίηση είναι υπεύθυνος ο ίδιος ο χρήστης (Date, 2004),(Ramakrishnan & Gehrke, 2002).

Ο SQL optimizer είναι το τμήμα ενός RDBMS το οποίο ευθύνεται για την εύρεση ενός ικανοποιητικού σχεδίου υπολογισμού για την εκτέλεση κάθε αιτήματος. Ο λόγος για τον οποίο η εύρεση ενός ικανοποιητικού σχεδίου υπολογισμού καθίσταται αναγκαία, είναι η ύπαρξη πολλών διαφορετικών τρόπων για την ανάκτηση των δεδομένων προκειμένου να δοθεί απάντηση στο αίτημα. Ο χώρος των πιθανών εναλλακτικών σχεδίων σχετίζεται με την αλγεβρική αναπαράσταση του κάθε αιτήματος. Όσο μεγαλύτερος είναι ο αριθμός των τελεστών που λαμβάνουν χώρα στο αίτημα, τόσο περιπλοκότερη γίνεται η αλγεβρική παράσταση. Επιπλέον, κάθε RDBMS χρησιμοποιεί κάποιου είδους ευρετήρια προκειμένου να βελτιστοποιήσει τον χρόνο αναζήτησης των αποθηκευμένων δεδομένων. Ένας ακόμα στόχος του optimizer είναι η βέλτιστη διαχείριση των υπαρχόντων ευρετηρίων.

Στο επόμενο σχήμα παρουσιάζεται μία σύνοψη της δομής ενός SQL optimizer:



Εικόνα 1: Συντακτική ανάλυση, βελτιστοποίηση και επεξεργασία SQL αιτήματος

Αρχικά λαμβάνει χώρα η συντακτική ανάλυση, η οποία ευθύνεται για τον γραμματικό έλεγχο των αιτημάτων και την μετατροπή της SQL σε μία δομή, η οποία είναι κατανοητή από το RDBMS. Μετά την συντακτική ανάλυση, το αίτημα δίνεται στον βελτιστοποιητή αιτημάτων (query optimizer).

Ευθύνη του βελτιστοποιητή είναι ο προσδιορισμός ενός αποδοτικού σχεδίου εκτέλεσης για το τρέχον αίτημα. Ο βελτιστοποιητής παράγει εναλλακτικά σχέδια για την εκτέλεση του αιτήματος και επιλέγει εκείνο το οποίο αντιστοιχεί στο ελάχιστο αναμενόμενο κόστος. Στον υπολογισμό του αναμενόμενου κόστους χρησιμοποιείται πληροφορία από τους καταλόγους του συστήματος.

Για να γίνει κατανοητή η ανάγκη για την ύπαρξη του βελτιστοποιητή θα δώσουμε ένα παράδειγμα. Ας υποθέσουμε πως έχουμε μια βάση δεδομένων, όπου είναι καταχωρημένα τα τηλέφωνα, το επίθετο και η περιοχή όλων των κατοίκων της Ελλάδας. Ας υποθέσουμε οτι ζητούμε από το RDBMS όλα τα τηλέφωνα των κατοίκων της Αττικής των οποίο το επίθετο αρχίζει από το γράμμα ωμέγα. Ας υποθέσουμε πως υπάρχουν 10.000.000 τηλέφωνα και πως το καθένα αντιστοιχεί σε ένα επίθετο. Από αυτά εμπειρικά είναι ασφαλές να θεωρήσουμε πως το 40% βρίσκεται στην Αττική και το 5% έχει επίθετο που ξεκινά από ωμέγα.

Στην περίπτωση το RDBMS δεν έχει ευρετήρια τα οποία του δίνουν πρόσβαση σε στατιστικά στοιχεία είναι υποχρεωμένο να προβεί σε πλήρη σάρωση των δεδομένων,

για να επιλέξει τα επίθετα που ξεκινούν από ωμέγα και ταυτόχρονα η περιοχή να είναι η Αττική. Κόστος ανάγνωσης 10.000.000 εγγραφές.

Στην περίπτωση που το RDBMS έχει πρόσβαση σε ένα ευρετήριο που αφορά μόνο την περιοχή, μπορεί να προβεί στην ανάγνωση του 40% των εγγραφών. Με αυτόν τον τρόπο μπορεί να επιλέξει τα τηλέφωνα της Αττικής και σε αυτά να βρει ποια επίθετα ξεκινούν από ωμέγα. Κόστος ανάγνωσης 4.000.000 εγγραφές.

Στην περίπτωση που το RDBMS έχει πρόσβαση σε 2 ευρετήρια, ένα για το επίθετο και ένα για την περιοχή, τότε 2 επιλογές:

1. Να επιλέξει πρώτα τα τηλέφωνα ανά περιοχή και μετά να βρεί ποια από αυτά έχουν επίθετο που ξεκινά από ωμέγα. Κόστος ανάγνωσης 4.000.000 εγγραφές.
2. Να επιλέξει πρώτα τα τηλέφωνα ανά επίθετο και μετά να βρει ποια από αυτά βρίσκονται στην Αττική. Κόστος ανάγνωσης 500.000 εγγραφές.

Από το παραπάνω απλό παράδειγμα, καταρχάς, είναι φανερό πως η ίδια ερώτηση μπορεί να απαντηθεί με διαφορετικά σχέδια εκτέλεσης και πως η διαφορά στην απόδοση ενδέχεται να είναι σημαντική. Επομένως τόσο οι μέθοδοι για την επιτάχυνση της ανάγνωσης και εγγραφής των δεδομένων, όσο και η εύρεση ενός βέλτιστου σχεδίου εκτέλεσης ενός SQL αιτήματος είναι αναγκαία θέματα που πρέπει να λύσει ένα RDBMS.

2.4.1. Ο sql optimizer στην MySQL

Ο sql optimizer, στην MySQL, υλοποιείται στα αρχεία sql/sql_select.cc και πιο συγκεκριμένα στην συνάρτηση JOIN::optimize(). Το παρακάτω διάγραμμα δείχνει την δομή του κώδικα που αναλαμβάνει ένα αίτημα στον server:

```
1. handle_select()
2.   mysql_select()
3.     JOIN::prepare()
4.       setup_fields()
5.     JOIN::optimize() /* ο optimizer ξεκινά εδώ... */
6.       optimize_cond()
7.       opt_sum_query()
8.       make_join_statistics()
9.         get_quick_record_count()
10.       choose_plan()
11.         optimize_straight_join()
12.         /* Εύρεση μερικώς βέλτιστου πλάνου     */
13.         /* μεταξύ όλων & υποσυνόλων από όλα τα */
14.         /* πιθανά πλάνα. ελέγχεται το βάθος    */
15.         /* (exhaustiveness)της έρευνας         */
16.         greedy_search()
17.         /* exhaustive έρευνα για βέλτιστο πλάνο*/
18.         find_best()
19.      make_join_select()       /* ... έως εδώ */
20.    JOIN::exec()
```

Στο διάγραμμα, ένα κομμάτι κώδικα καλεί τον κώδικα που βρίσκεται στοιχισμένος πιο δεξιά.

Όπως φαίνεται από την παραπάνω δομή ο SQL optimizer της MySQL αποτελείται από τις συναρτήσεις optimize_cond, opt_sum_query και make_join_statistics. Η optimize_cond (γραμμή 6) είναι υπεύθυνη για θέματα που αφορούν τις συνθήκες WHERE και HAVING. Η συνάρτηση opt_sum_query (γραμμή 7) είναι υπεύθυνη για την βελτιστοποίηση των συναθροιστικών συναρτήσεων min, max και count. Η make_join_statistics (γραμμή 8) είναι η συνάρτηση όπου αποφασίζεται ο τρόπος με τον οποίο θα εκτελεστεί το JOIN. Πιο συγκεκριμένα, με την συνάρτηση get_quick_record_count (γραμμή 9) βρίσκεται μια πρώτη εκτίμηση του πλήθους των πιθανών σειρών που θα επιστραφούν από τους πίνακες. Με την συνάρτηση choose_plan (γραμμή 10) γίνεται ο υπολογισμός του βέλτιστου πλάνου. Υπάρχουν δύο περιπτώσεις:

- να περιλαμβάνει το query STRAIGHT JOIN. Σε αυτήν την περίπτωση, με την optimize_straight_join (γραμμή 11) ο βελτιστοποιητής προσπαθεί να κάνει την βελτιστοποίηση λαμβάνοντας υπόψιν την σειρά με την οποία έχουν δοθεί οι πίνακες του STRAIGHT JOIN.
- να μην περιλαμβάνει το query STRAIGHT JOIN. Σε αυτήν, την πιο συνηθισμένη περίπτωση, υπάρχουν 2 τρόποι με τους οποίος βρίσκεται το βέλτιστο πλάνο, με βάση το πλήθος των σειρών που θα αντληθούν από τους πίνακες.
    1. αν οι πίνακες είναι λίγοι στο πλήθος, ερευνώνται όλα τα πιθανά πλάνα εκτέλεσης με την συνάρτηση find_best (γραμμή 18).
    2. αν οι πίνακες είναι πολλοί στο πλήθος τότε και τα πιθανά πλάνα εκτέλεσης είναι πολλά. Επομένως ο βελτιστοποιητής δεν τα ελέγχει όλα, αλλά περιορίζεται από κάποιες παραμέτρους που αφορούν το εύρος της έρευνας. Με την greedy_search (γραμμή 16) ο βελτιστοποιητής προσπαθεί να βρει το μερικώς βέλτιστο πλάνο, ένα πλάνο δηλαδή το οποίο είναι βέλτιστο, λαμβάνοντας υπόψιν αυτές τις παραμέτρους. Ανάλογα με τις παραμέτρους, όσο πιο μεγάλο είναι το δυνατό εύρος της έρευνας, τόσο το μερικώς βέλτιστο πλάνο πλησιάζει την απόδοση του συνολικά βέλτιστου πλάνου.

2.5. Σύνοψη

Στην ενότητα αυτή αναλύθηκαν οι λόγοι για τους οποίους το framework δομείται με τα εργαλεία και τα προγράμματα που αναφέρθηκαν πιο πάνω:

- Η MySQL χρησιμοποιείται ως αντικείμενο μελέτης διότι είναι ένα δημοφιλές RDBMS σύστημα και διότι μπορούμε να επέμβουμε σε βάθος στον πηγαίο κώδικά της.
- Ο SQL optimizer της MySQL παρουσιάζει ενδιαφέρον επειδή έχουμε πρόσβαση στον πηγαίο κώδικά του και μπορούμε να μελετήσουμε εκτεταμένα.
- Το OSDB benchmark χρησιμοποιείται για τα SQL αιτήματα που στέλνει στην MySQL, επειδή βασίζεται στο AS3AP, ένα καθιερωμένο και γνωστό

benchmark, το οποίο έχει ένα καλό δείγμα SQL αιτημάτων, τα οποία απαιτούν βασικές ικανότητες βελτιστοποίησης.

Επίσης δόθηκε και μια δομή του SQL optimizer της MySQL, σε υψηλό επίπεδο.

3. Το πλαίσιο ανάλυσης φόρτου εργασίας του SQL optimizer

Στην ενότητα αυτή αναλύονται η δομή του framework το οποίο προτείνουμε καθώς και λεπτομέρειες υλοποίησής του.

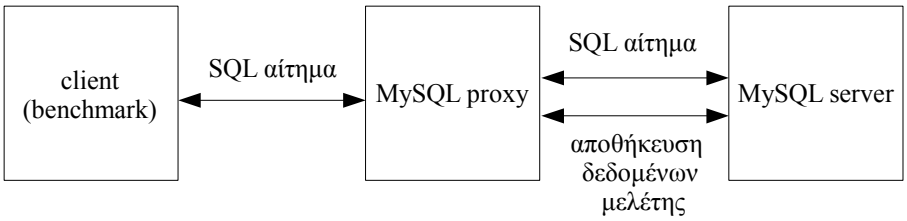
Η δομή του κεφαλαίου έχει ως εξής: στην ενότητα 3.1 γίνεται μια εισαγωγή στην δομή του framework. Στην ενότητα 3.2 περιγράφονται αναλυτικά τα τμήματα από τα οποία αποτελείται το framework καθώς και ο τρόπος με τον οποίο αλλελεπιδρούν. Στην ενότητα 3.3 δίνεται η σύνοψη του κεφαλαίου.

3.1. Η δομή του framework

Το framework αποτελείται από 3 προγράμματα τα οποία αλληλεπιδρούν προκειμένου να παραχθούν και να αποθηκευτούν τα επιθυμητά αποτελέσματα. Αυτά είναι:

- ένας MySQL server.
- ένας MySQL proxy server.
- ένας client ο οποίος στέλνει αιτήματα στον server, στην περίπτωσή της ανάλυσης ένα benchmark.

Το παρακάτω σχήμα παρουσιάζει την δομή του framework:



Εικόνα 2: Δομή των τμημάτων της μελέτης και ροή δεδομένων

Η ροή των δεδομένων έχει ως εξής:

- το benchmark στέλνει αιτήματα στον server, δίχως να γνωρίζει την ύπαρξη του proxy server. Τα αιτήματα αυτά χρησιμοποιούνται από την μελέτη για βρεθεί ο φόρτος εργασίας του SQL optimizer.
- ο MySQL proxy αναλαμβάνει να προωθήσει τα αιτήματα στον MySQL server. Όταν επιστρέφει στον MySQL proxy η απάντηση του αιτήματος, τότε ο MySQL proxy δεν στέλνει αμέσως την απάντηση στον client. Αντιθέτως, στέλνει στην βάση ένα σύνολο αιτημάτων, με το οποίο γίνεται η αποθήκευση μετρήσεων της μελέτης, σε μία βάση δεδομένων στον server. Όταν πραγματοποιηθεί η αποθήκευση, τότε ο MySQL proxy επιστρέφει στον client την απάντηση του server.

- ο MySQL server δέχεται τα αιτήματα τα οποία και προσπαθεί να εξυπηρετήσει. Ο MySQL server, που χρησιμοποιείται στην μελέτη, είναι ένας τροποποιημένος server. Χάρη σε αυτήν την μετατροπή γίνονται οι μετρήσεις για την ανάλυση του φόρτου εργασίας στον SQL optimizer. Όμως τα δεδομένα που παράγονται από την ανάλυση, είναι προσωρινά αποθηκευμένα στην μνήμη και θα χαθούν μόλις ο client αποσυνδεθεί. Για αυτόν τον λόγο, ο MySQL proxy αναλαμβάνει να αποθηκεύσει τα δεδομένα στον server, μόνιμα σε μία βάση δεδομένων. Στην συνέχεια έχουμε την δυνατότητα να τα επεξεργαστούμε και να προχωρήσουμε στην ανάλυση των δεδομένων.

3.2. Περιγραφή των τμημάτων του benchmark

Στις επόμενες ενότητες θα παρουσιάσουμε με λεπτομέρειες τα τμήματα που αποτελούν το benchmark καθώς και τον τρόπο με τον οποίο αλληλεπιδρούν.

3.2.1. MySQL server και code profiling

Όπως αναφέραμε πιο πάνω, χρειάζεται να επέμβουμε στον πηγαίο κώδικα του server προκειμένου να προσθέσουμε κώδικα, ο οποίος καταγράφει τα δεδομένα που ζητούνται. Για την επέμβαση αυτή χρησιμοποιούμε το SHOW PROFILES API (ή πιο απλά profiling API) το οποίο ήδη υπάρχει στην MySQL. Το API αυτό αποτελεί μέρος των “community features” τα οποία δεν περιλαμβάνονται στις binary εκδόσεις των server, πέραν της τρέχουσας έκδοσης παραγωγής.

Η χρήση του API είναι απλή. Γενικά, το API καταμετρά το χρονικό διάστημα, το οποίο περνά ένα αίτημα μέσα στον MySQL server, από την στιγμή που μπαίνει στον server μέχρι την στιγμή που ο server επιστρέφει την απάντηση. Μέσα στον κώδικα του MySQL server υπάρχει ένα πλήθος από σημεία ελέγχου. Το API μετρά το χρονικό διάστημα από την στιγμή που η εκτέλεση του κώδικα συναντήσει ένα σημείο ελέγχου έως ότου συναντήσει το επόμενο σημείο ελέγχου. Στον κώδικα της MySQL, υπάρχουν ήδη τοποθετημένα κάποια τέτοια σημεία ελέγχου, τα οποία καταμετρούν το χρονικό διάστημα που δαπανάται σε σημαντικά σημεία κατά την εκτέλεση ενός αιτήματος.

Το κάθε σημείο ελέγχου υλοποιείται με την κλήση της συνάρτησης thd_proc_info(). Κάθε φορά που το profiling API συναντά αυτή την συνάρτηση μηδενίζει ένα χρονόμετρο και μετρά τον χρόνο έως ότου συναντήσει ξανά μια κλήση της συνάρτησης. Για να προστεθεί ένα καινούργιο σημείο ελέγχου μέσα στον κώδικα, χρειάζεται η προσθήκη μιας συνάρτησης thd_proc_info()στον κώδικα.

Τα δεδομένα, τα οποία καταμετρούνται από το profiling API, αποθηκεύονται στον πίνακα profiling της βάσης δεδομένων information_schema. Ο πίνακας αυτός χρησιμοποιεί την μηχανή αποθήκευσης MEMORY η οποία αποθηκεύει τον πίνακα αποκλειστικά στην μνήμη. Επομένως, μετά την αποσύνδεση του client από τον server τα δεδομένα που είναι αποθηκευμένα στον πίνακα χάνονται. Επίσης για κάθε client υπάρχει διαφορετικό στιγμιότυπο του πίνακα στην μνήμη.

Το profiling API χρησιμοποιήθηκε γιατί καλύπτει τις προϋποθέσεις που τέθηκαν στους στόχους της μελέτης όσον αφορά την μετατροπή του MySQL server.

- Με την χρήση του profiling API δεν χρειάζεται η προσθήκη καινούργιων ορισμών, συναρτήσεων ούτε η μετατροπή στον πηγαίο κώδικα, σε οποιοδήποτε σημείο του προγράμματος. Εφόσον το profiling API είναι τμήμα του κώδικα της MySQL απαιτείται απλά η κλήση του. Επιπλέον η χρήση του είναι ιδιαιτέρως απλή (κλήση μίας συνάρτησης). Επομένως, απαιτείται ελάχιστη παρέμβαση στον πηγαίο κώδικα της MySQL
- Το profiling API, αποθηκεύει τα δεδομένα που συλλέγονται σε μία βάση δεδομένων μέσα στον MySQL server. Η μηχανή αποθήκευσης των πινάκων είναι η "MEMORY"η οποία αποθηκεύει τα δεδομένα στην μνήμη του συστήματος δίχως να γίνεται χρήση του σκληρού δίσκου. Επομένως η I/O επιβάρυνση κατά την εκτέλεση του αιτήματος είναι η ελάχιστη και κατά συνέπεια, επιβαρύνεται η εκτέλεση της MySQL όσο το δυνατό λιγότερο.
- Είναι δυνατή η προσθήκη σημείων ελέγχου σε οποιοδήποτε σημείο του κώδικα, δίνοντας έτσι την δυνατότητα για μετρήσεις σε οποιαδήποτε τμήματα της εκτέλεσης ενός αιτήματος. Επομένως, προσφέρει επεκτασιμότητα και ευελιξία, όσον αφορά τα μεγέθη τα οποία καταγράφονται και μετρούνται.

3.2.2. Μετατροπές στον MySQL server

Στον πηγαίο κώδικα της MySQL υπάρχουν ήδη 2 σημεία ελέγχου του profiling API, τα οποία καλύπτουν την εκτέλεση του SQL optimizer, τα "optimizing" και "statistics". Τα 2 αυτά σημεία έχουν αφαιρεθεί και έχουν προστεθεί περισσότερα (καινούργια) τα οποία καλύπτουν τον ίδιο κώδικα. Για τις ανάγκες της μελέτης πρέπει να εξεταστεί ο κώδικας τον οποίο καλύπτουν τα σημεία αυτά, με περισσότερη λεπτομέρεια.

Για υπάρχει ομοιότητα με τα παλιά σημεία ελέγχου, τα καινούργια έχουν ονομαστεί "optimizing: XYZ" και "statistics: XYZ" όπου XYZ μια ονομασία που χαρακτηρίζει το σημείο αυτό. Συγκεκριμένα:

- Στην συνάρτηση JOIN::optimize επεκτείνεται το σημείο ελέγχου "optimizing" στα επιμέρους σημεία:
    1. "optimizing: init": ξεκίνημα της JOIN::optimize, δέσμευση μεταβλητών και αρχικοποίηση αυτών.
    2. "optimizing: conds": βελτιστοποίηση των conditionals (συνάρτηση optimize_cond)
    3. "optimizing: sums": βελτιστοποίηση των count, min, max (συνάρτηση opt_sum_query)
- Στη συνάρτηση make_join_statistics επεκτείνεται το σημείο ελέγχου "statistics":
    1. "statistics: init": ξεκίνημα της make_join_statistcs, δέσμευση μεταβλητών και αρχικοποίηση αυτών..
    2. "statistics: const tables": Εύρεση των const πινάκων (η έννοια εξηγείται στην συνέχεια).

3. "statistics: index stats": Ανανέωση των πληροφοριών για τα ευρετήρια που μπορούν να χρησιμοποιηθούν.
4. "statistics: count rows": Υπολογισμός του πιθανού πλήθος των γραμμών κάθε πίνακα.
5. "statistics: join order": Εύρεση του βέλτιστου πλάνου για τα JOIN.

Αυτό που πρέπει να σημειωθεί είναι πως το τελευταίο σημείο ελεγχου μετρά όντως τον χρόνο του κώδικα της συνάρτησης make_join_statistics και όχι και επιπλέον κώδικα μετά το τέλος της συνάρτησης αυτής. Αυτό συμβαίνει διότι αμέσως μετά το τέλος της make_join_statistics υπάρχει ένα επόμενο σημείο ελέγχου, το "preparing".

Επιπλέον σκόπιμο είναι να εξηγηθεί τι ορίζεται ως πίνακας const:

- Ένας πίνακας με 0 ή με 1 μόνο γραμμή.
- Μια έκφραση, που περιέχει πίνακα, η οποία περιορίζεται από μια συνθήκη WHERE, η οποία περιέχει συνθήκες της μορφής στήλη=“σταθερά” για:
  1. όλες τις στήλες του πρωτεύοντος κλειδιού ή
  2. για όλες τις στήλες οποιουδήποτε από τα unique κλειδιά του πίνακα (με την προϋπόθεση πως οι στήλες έχουν οριστεί ως NOT NULL).

Αφού γίνει το compile του server ο server χρησιμοποιείται για το benchmark.

3.2.3. Η αποθήκευση των πληροφοριών

Το profiling API παρέχει ένα πλήθος από πληροφορίες. Μία πλήρης καταγραφή των πληροφοριών βρίσκεται στο Παράρτημα 7. Από αυτές τις πληροφορίες, στην ανάλυση χρειάζονται συγκεκριμένες μόνο. Στον MySQL server δημιουργείται μια βάση δεδομένων όπου θα αποθηκευτούν μόνιμα τα δεδομένα, που παρέχει το profiling API, για περαιτέρω επεξεργασία.

Η βάση αυτή περιλαμβάνει τους εξής πίνακες:

```
1. queries( ID, query, session, session_time)
2. details( ID, seq, state, duration)
```

Ο Πίνακας queries αποθηκεύει το “αποτύπωμα” του κάθε αιτήματος, δίχως τις λεπτομέρειες που το αφορούν. Αποτελείται από τα πεδία:

- ID: ένας μοναδικός αριθμός (id) του κάθε αιτήματος που εξυπηρετεί ο server. Ο αριθμός αυτός είναι ανεξάρτητος από τον client που στέλνει το αίτημα, σχετίζεται μόνο με την σειρά με την οποία ο server εξυπηρετεί τα αιτήματα που δέχεται
- QUERY: η SQL που αποτελεί το αίτημα.
- SESSION: το id του thread το οποίο εξυπηρετεί τον κάθε client. Είναι μοναδικό αν ο server δεν κλείσει. Αν γίνει επανεκκίνηση του server η αρίθμηση ξεκινά από την αρχή. Αυτό το δεδομένο μπορεί να χρησιμεύσει αν θέλουμε να εξετάσουμε τη σειρά με την οποία καταφθάνουν στον server οι client.
- SESSION_TIME: η χρονοσφραγίδα στην οποία έγινε η αυθεντικοποίηση του client από τον server. Αυτό το δεδομένο μπορεί να χρησιμεύσει αν θέλουμε να εξετάσουμε τη σειρά και τη χρονική απόσταση με την οποία οι client καταφθάνουν στον server.

Ο Πίνακας details αποθηκεύει τις λεπτομέρειες που αφορούν το κάθε αίτημα. Για το κάθε αίτημα αποθηκεύεται ένα πλήθος από σειρές. Η κάθε σειρά περιέχει τις πληροφορίες από την εκτέλεση του κώδικα που αντιστοιχεί από ένα σημείο ελέγχου του profiling API έως το επόμενο. Αποτελείται από τα πεδία:

- ID: το id του κάθε αιτήματος που εξυπηρετεί ο server (βλ πίνακα queries).
- SEQ: ένας αύξων αριθμός που δηλώνει την σειρά του σημείου ελέγχου ανάμεσα στα σημεία ελέγχου ενός συγκεκριμένου αιτήματος.
- STATE: το όνομα που δίνεται από το profiling API στο σημείο ελέγχου το οποίο χρονομετρεί η γραμμή.
- DURATION: ο χρόνος που χρειάστηκε για την ολοκλήρωση του σημείου ελέγχου.

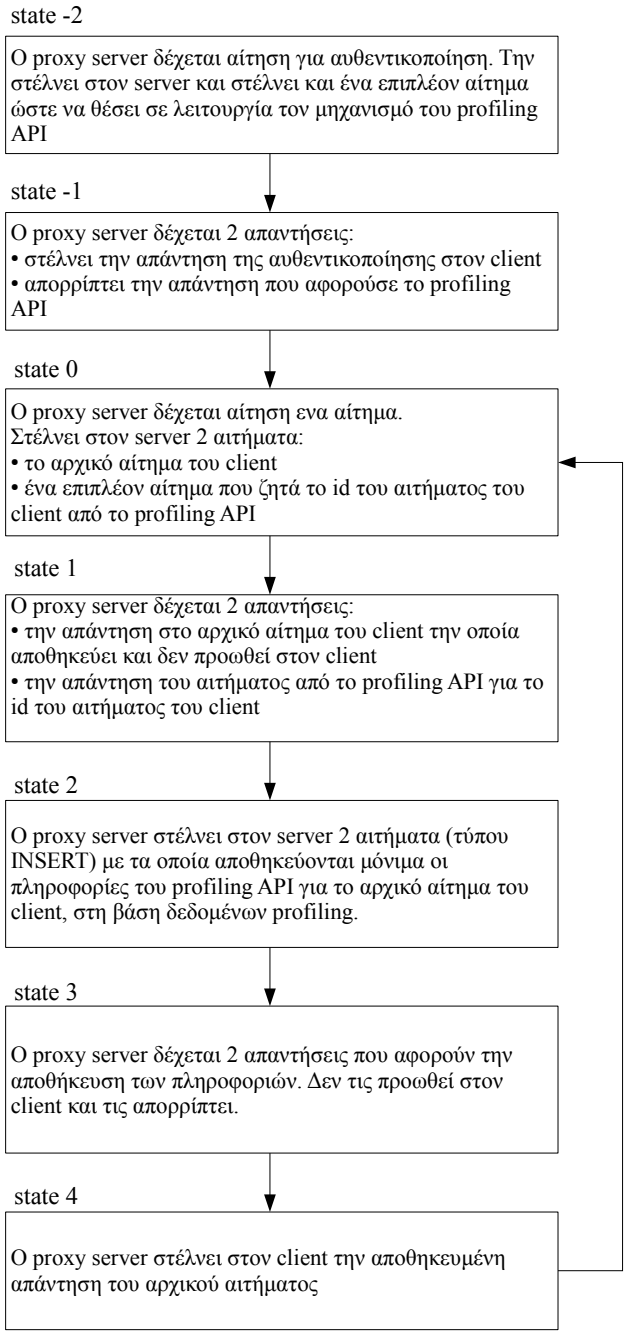
Ο κώδικας του script της δημιουργίας της βάσης δεδομένων για την μόνιμη αποθήκευση των δεδομένων, βρίσκεται στο Παράρτημα 9.1.

3.2.4. MySQL proxy

Ο MySQL proxy είναι ένας proxy server ο οποίος παρεμβάλλεται ανάμεσα στον MySQL server και τους πελάτες του. Ο proxy server μπορεί να μεταβάλλει και να επεξεργαστεί οποιαδήποτε κίνηση δεδομένων από τον client προς στον server και αντίστροφα, η οποία χρησιμοποιεί το MySQL Network protocol και η οποία περνά μέσω της port στην οποία ακούει ο server. Η διαχείριση και η επεξεργασία γίνεται από ένα script, το οποίο διαβάζει ο proxy κάθε φορά που ένας καινούργιος πελάτης προσπαθεί να συνδεθεί.

Για τις ανάγκες της ανάλυσης δημιουργήθηκε ένα script το οποίο αναλαμβάνει την προώθηση του αιτήματος του client στον server και ύστερα την αποθήκευση των δεδομένων του profiling API για το συγκεκριμένο αίτημα.

Ο αλγόριθμος που υλοποιεί το script είναι ο ακόλουθος (ο πλήρης κώδικας του script βρίσκεται στο Παράρτημα 10):



Εικόνα 3: ο αλγόριθμος του script του MySQL proxy

Σχετικά με τον παραπάνω αλγόριθμο επεξηγούνται στην συνέχεια ορισμένα σημεία:

- Ο αλγόριθμος αρχίζει όταν ένας client αρχίζει την διαδικασία αυθεντικοποίησης στον server.
- Ο αλγόριθμος τερματίζει όταν διακοπεί η σύνδεση του client με τον server είτε με κανονικό τρόπο είτε λόγω σφάλματος.
- Η αρίθμηση των καταστάσεων (-2 έως 4) έχει καθαρά ενδεικτικό χαρακτήρα. Στις καταστάσεις πριν το 0 δεν έχει ενεργοποιηθεί ακόμα το profiling API ενώ στις καταστάσεις μετά το 0 έχει ενεργοποιηθεί.
- Ο αλγόριθμος κατά την κύρια λειτουργία του (καταστάσεις 0 έως 4) είτε εξυπηρετεί ένα αίτημα (καταστάσεις 1 έως 4) είτε αναμένει ένα αίτημα (κατάσταση 0)

3.2.5. benchmarking client

Το benchmark συνδέεται στη θύρα όπου ακούει ο proxy server και αρχίζει να στέλνει αιτήματα στον MySQL server (μέσω του proxy server). Τότε, τα δεδομένα του profiling API, για κάθε αίτημα, αποθηκεύονται στον MySQL server.

Μόλις τελειώσει το benchmark στην βάση δεδομένων profiling είναι αποθηκευμένα όλα τα στατιστικά για όλα τα αιτήματα που στέλνονται. Το πλήθος των αιτημάτων είναι περίπου 450.000, αριθμός ο οποίος, περιλαμβάνει και ένα αίτημα BEGIN και ένα COMMIT για κάθε αίτημα τύπου SELECT και UPDATE. Το πλήθος των αιτημάτων δίχως τα BEGIN, COMMIT είναι περίπου 150.000.

3.2.6. Στατιστικά στοιχεία

Μετά την εκτέλεση του benchmark υπάρχει πλέον η δυνατότητα να μετρηθούν τα μεγέθη που χρειάζεται η ανάλυση. Στην παρούσα μελέτη σχεδιάσαμε 2 στατιστικά μεγέθη. Συγκεκριμένα, για κάθε αίτημα υπολογίζονται οι λόγοι:

- Συνολικός χρόνος εκτέλεσης του αιτήματος προς τον χρόνο εκτέλεσης όλου του SQL optimizer (σε ποσοστό %)
- Συνολικός χρόνος εκτέλεσης του αιτήματος προς τον χρόνο εκτέλεσης του κάθε μέρους του SQL optimizer (σε ποσοστό %). Αυτό περιλαμβάνει τα 8 μέρη του SQL optimizer όπως αυτά περιγράφηκαν σε προηγούμενη ενότητα (3.2.2).

3.2.7. Χρήση όλων των στοιχείων μαζί

Για την αυτόματη εκτέλεση όλων των μερών του framework χρησιμοποιείται ένα shell script, ο κώδικας του οποίου παρουσιάζεται στο παράρτημα της ενότητας 12. Το script αυτό αναλαμβάνει να εκτελέσει αυτόματα τα διαφορετικά σενάρια του MySQL server, ένα συγκεκριμένο αριθμό φορών το καθένα.

3.3. Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκε η δομή και η υλοποίηση του framework και των τμημάτων που το αποτελούν. Στον MySQL server έχει ενεργοποιηθεί το profiling API το οποίο κρατά στατιστικά κατά την διάρκεια της εκτέλεσης των SQL αιτημάτων. Τα SQL αιτήματα τα στέλνει ένας client, στην περίπτωση της μελέτης το OSDB benchmark. Προκειμένου να καταγράφονται τα στατιστικά στοιχεία που θέλουμε, στον κώδικα της MySQL έγιναν οι κατάλληλες προσθήκες, εισάγοντας κλήσεις του profiling API. Τα δεδομένα όμως, που μας παρέχει το API αυτό, αποθηκεύονται προσωρινά, επομένως φτιάξαμε μια βάση δεδομένων στην οποία θα αποθηκεύονται μόνιμα τα δεδομένα. Προκειμένου να υλοποιηθεί η αυτόματη αποθήκευση των δεδομένων, χρησιμοποιήθηκε ο MySQL proxy, ο οποίος παρεμβάλλεται ανάμεσα στον server και στον client. Κάθε φορά που έρχεται ένα SQL αίτημα, ο MySQL proxy φροντίζει να επιλέξει από το profiling API τα δεδομένα που θέλουμε και να τα αποθηκεύσει μόνιμα στην βάση δεδομένων που έχουμε δημιουργήσει. Για την αυτόματη εκτέλεση διαφορετικών περιπτώσεων και την συλλογή των στατιστικών χρησιμοποιείται ένα shell script το οποίο φροντίζει για την ομαλή και διαδοχική εκτέλεση όλων των λειτουργιών.

4. Χρήση του framework

Στο κεφάλαιο αυτό, το framework που αναλύθηκε στο προηγούμενο κεφάλαιο, χρησιμοποιείται με στόχο καταρχάς να γίνει επίδειξη της λειτουργίας του και κατά δεύτερον την εξαγωγή κάποιων πρώτων συμπερασμάτων από την χρήση με το OSDB benchmark.

Η δομή του κεφαλαίου είναι η εξής: στην ενότητα 4.1 θα αναφέρουμε τα σενάρια τα οποία χρησιμοποιήθηκαν στην μελέτη. Στην ενότητα 4.2 γίνεται μια παρουσίαση του περιβάλλοντος στο οποίο έγινε η χρήση του framework και η διαδικασία εγκατάστασης. Στην ενότητα 4.3 παρουσιάζουμε τα αποτελέσματα της χρήσης του OSDB benchmark και την επιβάρυνση που επιφέρει η χρήση του framework και του profiling API στην εκτέλεση των SQL αιτημάτων από τον server. Στην ενότητα 4.4 αξιολογείται το framework και στην ενότητα 4.5 δίνεται η σύνοψη του κεφαλαίου.

4.1. Σενάρια χρήσης

Το framework χρησιμοποιήθηκε στα εξής 3 σενάρια, με το profiling API ενεργοποιημένο σε όλα:

- Σενάριο A: δεν έχουν προστεθεί σημεία ελέγχου εκτός από αυτά που ήδη υπάρχουν στον πηγαίο κώδικα.
- Σενάριο B: έχουν προστεθεί επιπλέον σημεία ελέγχου μέσα στον κώδικα του sql optimizer. Από αυτά που αναφέρονται στην ενότητα (3.2.2) χρησιμοποιήθηκαν μόνο όσα αφορούν το τμήμα “statistics”.
- Σενάριο C: έχουν προστεθεί επιπλέον σημεία ελέγχου μέσα στον κώδικα του sql optimizer. Από αυτά που αναφέρονται στην ενότητα (3.2.2) χρησιμοποιήθηκαν όσα αφορούν τόσο το τμήμα “statistics” όσο και το τμήμα “optimizing”.

Στο σενάριο A ο MySQL server έγινε build από τον πηγαίο κώδικα, αυτούσιο όπως διανέμεται από το site της εταιρίας. Στα σενάρια B και C ο MySQL server έγινε build αφού πρώτα έχει προστεθεί το κατάλληλο patch στον πηγαίο κώδικα. Τα patch παρουσιάζονται στο Παράρτημα 9.

Ο λόγος για τον οποίο εκτελούνται τα 3 διαφορετικά σενάρια είναι για να βρεθεί η επιβάρυνση που επιφέρει το profiling API στην εκτέλεση του προγράμματος. Δεν βρέθηκε τρόπος να μετρηθεί η επιβάρυνση για όλο το API, συγκρίνοντας δηλαδή τον χρόνο εκτέλεσης ενός αιτήματος με το API ενεργοποιημένο και απενεργοποιημένο. Ωστόσο είναι εφικτές οι μετρήσεις με διαφορετική, σε έκταση, χρήση του API. Έτσι στο σενάριο A υπάρχει η μικρότερη χρήση του API, στο B υπάρχει περισσότερη χρήση και στο C ακόμα περισσότερη. Βρίσκοντας τις διαφορές στον χρόνο εκτέλεσης των αιτημάτων στα διαφορετικά σενάρια μπορεί να μετρηθεί η σχετική επιβάρυνση μεταξύ των σεναρίων.

Τα δεδομένα που συλλέγονται στη βάση δεδομένων ομαδοποιούνται και τελικά περιλαμβάνει τις εξής μετρήσεις:

- Συνολικό χρονικό διάστημα εκτέλεσης του τμήματος “statistics”.
- Συνολικό χρονικό διάστημα εκτέλεσης του τμήματος “optimizing”.
- Χρονικό διάστημα εκτέλεσης του τμήματος "statistics: init".
- Χρονικό διάστημα εκτέλεσης του τμήματος "statistics: const tables".
- Χρονικό διάστημα εκτέλεσης του τμήματος "statistics: index".
- Χρονικό διάστημα εκτέλεσης του τμήματος "statistics: count rows".
- Χρονικό διάστημα εκτέλεσης του τμήματος "statistics: join order".
- Χρονικό διάστημα εκτέλεσης του τμήματος “optimizing: init”.
- Χρονικό διάστημα εκτέλεσης του τμήματος “optimizing: conds".
- Χρονικό διάστημα εκτέλεσης του τμήματος “optimizing: sums”.

Επιπλέον για την εκτέλεση των παραπάνω τμημάτων καταγράφονται οι μέσοι όροι του % ποσοστού του χρόνου εκτέλεσης του τμήματος σε σχέση με τον χρόνο εκτέλεσης όλου του αιτήματος. Εδώ πρέπει να σημειωθεί ότι το άθροισμα αυτών των μέσων όρων δεν δίνει υποχρεωτικά άθροισμα 100. Η χρήση των μέσω όρων δεν μπορεί να γίνει χρησιμοποιώντας το % ποσοστό ως απόλυτο μέγεθος αλλά ως ένα μέτρο που δίνει την τάξη του μεγέθους.

Επομένως για την εκτέλεση καθενός ενός εκ των σεναρίων καταγράφονται 20 μεγέθη (10 χρονικά διαστήματα σε απόλυτες τιμές και 10 σε % ποσοστό). Ο σκοπός του κάθε τμήματος έχει αναλυθεί στην ενότητα 3.2.2.

4.2. Εγκατάσταση του framework

Στην ενότητα αυτή θα παρουσιαστεί ο τρόπος με τον οποίο γίνεται η εγκατάσταση των τμημάτων του πλαισίου. Έχει ληφθεί υπόψιν πως στο μηχάνημα όπου θα εγκατασταθεί το framework υπάρχει σοβαρή πιθανότητα να υπάρχει ήδη εγκατεστημένος ένας MySQL server, τον οποίο ο χρήστης δεν θέλει να απενεργοποιήσει.

Το μηχάμημα όπου έγινε η δοκιμή ήταν με CPU: AMD AthlonX2 64 3600+, RAM 1G RAM 667MHz και HDD 40G (IDE). Χρησιμοποιήθηκε λειτουργικό σύστημα Linux, διανομή Debian. Για όλο το σύστημα και τα προγράμματα που χρειάστηκαν για το build της MySQL χρησιμοποιήθηκαν τα έτοιμα precompiled πακέτα του συστήματος apt. Πιο συγκεκριμένα χρησιμοποιήθηκε πυρήνας 2.6.24-1-amd64, gcc 4.2.4, make 3.81, libncurses5-dev 5.6, m4 1.4.11.

Για το framework χρησιμοποιήθηκαν το OSDB 0.21 και η MySQL 5.0.24 τα οποία και κατεβάσαμε από τα αντίστοιχα site. Και τα δύο αυτά προγράμματα έγιναν build χειροκίνητα. Επίσης χρησιμοποιήθηκε ο MySQL proxy 0.60 ο οποίος εγκαταστάθηκε μέσω του apt.

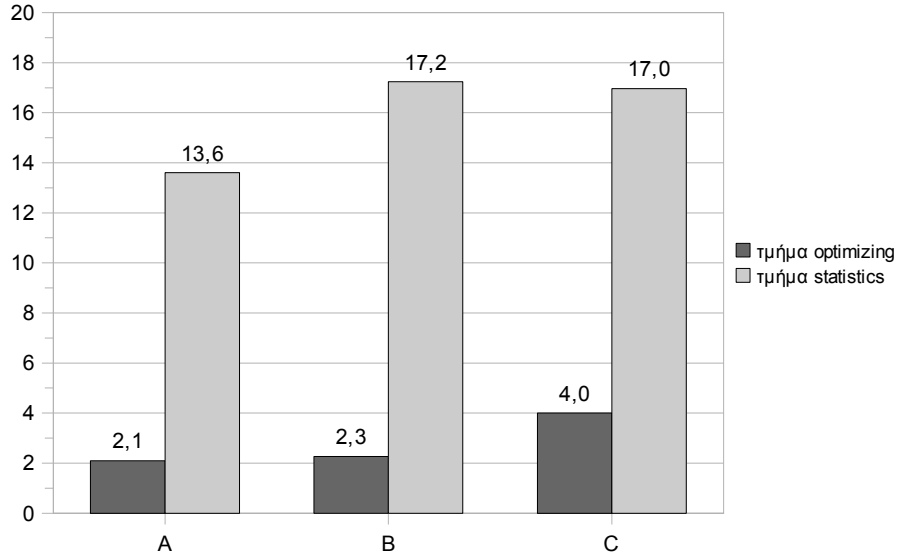
Όσον αφορά την διαρρύθμιση του server και του client, ο MySQL server και το benchmark τρέχουν στο ίδιο μηχάνημα (localhost) και το δίκτυο δεν δέχεται κάποια άλλη κίνηση. Επίσης και όλα τα services απενεργοποιήθηκαν.

Κατά την εκτέλεση της MySQL, δεν έγιναν ρυθμίσεις πέραν των προεπιλεγμένων για τον server.

4.3. Αποτελέσματα χρήσης του benchmark

Το framework εκτελέστηκε 4 φορές για κάθε σενάριο. Τα δεδομένα που συλλέχθηκαν παρουσιάζονται στην επόμενη ενότητα. Το framework εκτελέστηκε παραπάνω από μια φορά με τις ίδιες ρυθμίσεις ώστε τα στατιστικά να βασιστούν σε μέσους όρους πολλών εκτελέσεων (με τις ίδιες ρυθμίσεις). Με αυτόν τον τρόπο δίνονται δεδομένα απαλλαγμένα από αστάθμητους παράγοντες που σχετίζονται με την αλληλεπίδραση του MySQL server, του benchmark και του MySQL proxy με το λειτουργικό σύστημα.

Στα επόμενα διαγράμματα εμφανίζεται το % ποσοστό που καταναλώνεται σε κάθε τμήμα του optimizer (σε σχέση με το συνολικό χρόνο εκτέλεσης του αιτήματος) ως προς το σενάριο (το build του server)



Εικόνα 4: Επίδραση της χρήσης του profiling API, στο % του χρόνου εκτέλεσης του κάθε τμήματος του SQL optimizer που παρακολουθούμε.

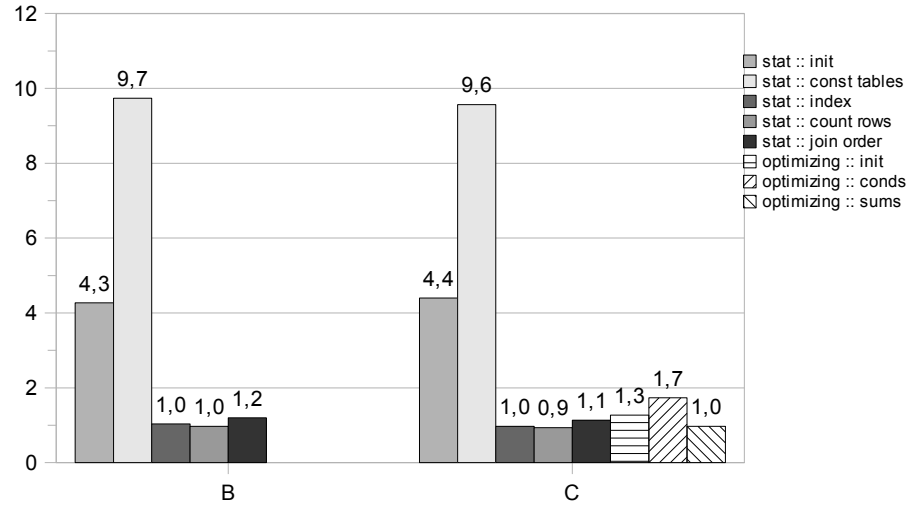
Στην εικόνα 4 εμφανίζεται η επίδραση του profiling API στα διαφορετικά σενάρια (τα build του server) A, B και C, στα οποία 3 σενάρια, το profiling API είναι ενεργοποιημένο. Η επίδραση του profiling API έχει ως εξής:

- στο τμήμα statistics, μεταξύ του σεναρίου A και των B και C η επιβάρυνση είναι σταθερή της τάξεως του 4%, αναμενόμενο εφόσον έχουν προστεθεί επιπλέον

σημεία ελέγχου στα σενάρια αυτά. Μεταξύ των B και C για το ίδιο τμήμα δεν υπάρχουν διαφορές εφόσον τα σημεία ελέγχου είναι ίδια.

- στο τμήμα optimizing, μεταξύ του σεναρίου A και B δεν υπάρχει διαφορά, πράγμα αναμενόμενο αφού τα σημείο ελέγχου είναι τα ίδια. Μεταξύ των A και B και του C υπάρχει επιβάρυνση της τάξεως του 2%, αναμενόμενο εφόσον στο σενάριο C έχουν προστεθεί περισσότερα σημεία ελέγχου

Επομένως η επιβάρυνση είναι πλέον γνωστή και κυμαίνεται μέσα σε αποδεκτά όρια, αν ληφθεί υπόψιν πως ο server υλοποιείται με thread.



Εικόνα 5: Σύγκριση της του φόρτου εργασίας στους server B και C.

Στην εικόνα 5 έχουμε να παρατηρήσουμε τα εξής:

Όπως είναι αναμενόμενο μεταξύ των σεναρίων B και C δεν παρουσιάζονται διαφορές μεταξύ του % ποσοστού του χρόνου στα επιμερους τμήματα του τμήματος statistics. Στο σενάριο B δεν υπάρχουν στοιχεία για το τμήμα optimizing εφόσον δεν υπάρχουν, σε αυτό το τμήμα, σημεία ελέγχου. Αντιθέτως, στο σενάριο C, όπου υπάρχουν σημεία ελέγχου, παρουσιάζεται η ανάλυση του φόρτου εργασίας στο τμήμα optimizing.

Τα σημεία όπου δαπανάται το περισσότερο χρονικό διάστημα είναι:

1. στην εύρεση των πινάκων const
2. στην αρχικοποίηση των μεταβλητών της συνάρτησης

Ο λόγος για τον οποίο εμφανίζεται η κατάταξη αυτή, είναι η φύση των αιτημάτων του OSDB benchmark, τα οποία αναλύονται ως αιτήματα με πινάκες const (ενότητα 2.2.3). Επομένως ορθώς εμφανίζεται το τμήμα του optimizer, που ασχολείται με την εύρεση και την διαχείριση των πινάκων const, να καταναλώνει τον περισσότερο χρόνο.

4.4. Αξιολόγηση του framework

Το framework κρίνεται πετυχημένο διότι μπορεί να δώσει με μικρή επιβάρυνση λεπτομερέστατη ανάλυση των εργασιών που λαμβάνουν χώρα κατά την διάρκεια της εκτέλεσης των SQL αιτημάτων. Επιπλέον είναι πλήρως επεκτάσιμο δίνοντας της δυνατότητα να αναλυθούν με ελάχιστες αλλαγές διαφορετικά τμήματα της εκτέλεσης ενός αιτήματος.

Στην περίπτωση των σεναρίων του OSDB αντέδρασε σωστά και ανέφερε τα αναμενόμενα αποτελέσματα. Επομένως η επέκταση της χρήσης του κρίνεται εφικτή και ελπιδοφόρα όσον αφορά τα αποτελέσματα και τα συμπεράσματα που μπορούν να εξαχθούν από την χρήση του.

Συνολικά, οι στόχοι της μελέτης που είχαν τεθεί στην ενότητα 1.2 έχουν επιτευχθεί και το framework έχει δοκιμαστεί επιτυχώς.

4.5. Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκε η χρήση του framework με client το OSDB benchmark. Χρησιμοποιήθηκαν 3 διαφορετικά build (σενάρια) του MySQL server έτσι ώστε να καταγράψουμε κάθε φορά τα ίδια μεγέθη με διαφορετικούς server και να τους συγκρίνουμε. Χρησιμοποιήσαμε ένα server χωρίς την χρήση του profiling API, ένα με μερική χρήση του profiling API και ένα με εκτεταμένη χρήση του profiling API. Το μηχάνημα το οποίο χρησιμοποιήθηκε τρέχει λειτουργικό Linux/Debian και επιλέχθηκε να χρησιμοποιηθούν όσο περισσότερο γίνεται τα έτοιμα precompiled πακέτα του λειτουργικού, εκτός από την MySQL και το OSDB. Στην συνέχεια αφού εκτελέσαμε για κάθε build του server αρκετές φορές το framework καταγράψαμε του μέσους όρους για τον κάθε server. Καταλήξαμε στο συμπέρασμα ότι το profiling API επιφέρει μια επιβάρυνση η οποία όμως κρίνεται αποδεκτή. Επομένως το framework δοκιμάστηκε με επιτυχία και από την χρήση του φάνηκε πως η επιβάρυνση που δημιουργεί μας δίνει την δυνατότητα να το χρησιμοποιήσουμε επιτυχώς για την μετρήσεις και την παρακολούθηση της εκτέλεσης SQL αιτημάτων στον MySQL server.

5. Συμπεράσματα - Περαιτέρω έρευνα

Στο κεφάλαιο αυτό γίνεται μια σύνοψη όσον αφορά τα συμπεράσματα της τρέχουσας ανάλυσης και προτείνεται περαιτέρω έρευνα που μπορεί να εκκινήσει με αφορμή την παρούσα έρευνα.

5.1. Σύνοψη της έρευνας

Ο SQL optimizer είναι ένα νευραλγικό κομμάτι της λειτουργίας ενός RDBMS. Στην παρούσα έρευνα χρησιμοποιήθηκε το RDBMS MySQL, του οποίου ο κώδικας διατίθεται ελεύθερα, ώστε να γίνει λεπτομερής ανάλυση του χρόνου που καταναλώνεται σε κάθε τμήμα του SQL optimizer κατά την εκτέλεση ενός SQL αιτήματος.

Χρησιμοποιήθηκε το OSDB benchmark, το οποίο είναι μια ανοικτού λογισμικού υλοποίηση του ASA3P benchmark, ώστε να στέλνονται μαζικά στον MySQL server SQL αιτήματα. Επιπλέον, μεταξύ του OSDB benchmark και του MySQL server, παρεμβάλλεται ένας MySQL proxy, ο οποίος είναι υπεύθυνος για την μόνιμη αποθήκευση των στατιστικών, για κάθε αίτημα που στέλνεται στον MySQL server. Στον MySQL το profiling API φροντίζει για την παροχή στατιστικών κατά την εκτέλεση του κάθε SQL αιτήματος. Με το profiling API στατικά, πριν την μεταγλώττιση του server, είναι εφικτό να εισαχθούν σημεία ελέγχου έτσι ώστε κατά την λειτουργία του MySQL να είναι δυνατόν να καταγραφούν στατιστικά από την λειτουργία του MySQL server.

Το framework υλοποιήθηκε και χρησιμοποιήθηκε επιτυχώς δείχνοντας πως είναι εφικτή η καταγραφή των στατιστικών αυτών, δίχως να δημιουργείται μεγάλη επιβάρυνση από την καταγραφή αυτή. Από την χρήση του framework με client το OSDB benchmark φάνηκε πως ένα ποσοστό 17% του χρόνου εκτέλεσης ενός SQL αιτήματος καταναλώνεται στον SQL optimizer και πως το τμήμα που βρίσκει τους πίνακες const καταναλώνει 9% του χρόνου εκτέλεσης ενός SQL αιτήματος.

5.2. Περαιτέρω έρευνα

Η χρήση του framework μπορεί να επεκταθεί σε λεπτομερέστερη ανάλυση των διεργασιών του sql optimizer.

Συγκεκριμένα είναι σημαντικό να δοκιμαστούν διαφορετικοί παράμετροι που ενδέχεται να επηρεάζουν την απόδοση, όπως:

- διαφορετικές αρχιτεκτονικές
- διαφορετικά λειτουργικά συστήματα. Στόχος είναι να καλυφθούν τα:

1. Linux x86 και x86-64: με σύγκριση μεταξύ διανομών όπως Debian, Ubuntu, Fedora, SuSe
2. FreeBSD
3. MS Windows 64 και 32 bit: με έμφαση στις εκδόσεις για server.
4. Open Solaris: με έμφαση στην αρχιτεκτονική SPARC

- διαφορετικά συστήματα αρχείων.

Επίσης μπορούν να δοκιμαστεί μεγαλύτερο εύρος SQL αιτημάτων. Τα αιτήματα του OSDB benchmark είναι περιορισμένα όσον αφορά την ποικιλία τους και την δυσκολία βελτιστοποίησης του αιτήματος. Επομένως κρίνεται αναγκαίο να επεκταθεί το εύρος των αιτημάτων που στέλνονται στην MySQL.

Η πρώτη πρόταση περιλαμβάνει την χρήση δύο forum engines, των phpBB και SimpleMachinesForums (SMF). Η εξομοίωση της χρήσης των forum μπορεί να δώσει ρεαλιστικότερα αποτελέσματα, εφόσον εξομοιώνει την χρήση πραγματικών συστημάτων παραγωγής. Η ποικιλία των αιτημάτων κρίνεται αρκούντως ικανοποιητική και περιλαμβάνει αιτήματα που καλύπτουν ένα μέρος των βελτιστοποιήσεων που μπορεί να αντιμετωπίσει ο server. Σε μια σύντομη ανάλυση που έγινε στην forum engine SMF βρέθηκε πως υπάρχουν:

Εντολή	SELECT	UPDATE	INSERT	DELETE
Πλήθος	905	246	12	200

Από τις SELECT εντολές 257 περιλαμβάνουν συνθήκη IN και 502 JOIN. Επομένως είναι φανερό πως το εύρος των SQL αιτημάτων είναι μεγαλύτερο από αυτό του OSDB client.

Η δεύτερη πρόταση περιλαμβάνει την υλοποίηση ενός client που στέλνει αιτήματα στον server με στόχο αποκλειστικά την επιβάρυνση του SQL optimizer της MySQL. Οι βελτιστοποιήσεις που είναι εφικτές από τον SQL optimizer είναι καταγεγραμμένες στην τεκμηρίωση και επιπλέον μπορούν να βρεθούν στον πηγαίο κώδικα της MySQL. Επομένως είναι δυνατόν να κατασκευαστεί ένας client ο οποίος να δημιουργεί SQL αιτήματα που καλύπτουν όλες τις δυνατές βελτιστοποιήσεις που μπορούν να λάβουν χώρα από το RDBMS.

6. Βιβλιογραφικές Αναφορές

C.J. Date (2003) . *An Introduction to Database Systems* . Addison Wesley, 2003.

E. F. Codd (1970). A relational model of data for large shared data banks. *Communications of the ACM,* 13(6): 377-387, June 1970.

Free Software Foundation (2008) . Licenses - Free Software Foundation . URL: http://www.fsf.org/licensing/licenses/ (προσπέλαση: 30/05/2008)

Free Software Foundation (2008) . The Free Software Definition . URL: http://www.gnu.org/philosophy/free-sw.html (προσπέλαση: 30/05/2008)

Jim Gray (1993) . *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)* . Morgan Kaufmann Pub, 1993.

Keith W. Hare (2006) . JCC's SQL Standards Page . URL: http://www.jcc.com/sql.htm (προσπέλαση: 30/05/2008)

Rick F. van der Lans (1989) . *The SQL standard: a complete guide reference* . Prentice Hall International (UK) Ltd., 1989.

Microsoft (2008) . MS SQL Server 2005 End User Licence Agreement . 2008 . (από αρχείο %ProgramFiles%\Microsoft SQL Server\100\EULA\ )

MySQL (2007) . MySQL AB Completes Record Year . URL: http://www.mysql.com/news-and-events/press-release/release_2007_02.html (προσπέλαση: 30/05/2008)

MySQL (2008) . MySQL 6.0 Reference Manual . URL: http://dev.mysql.com/doc/refman/6.0/en/index.html (προσπέλαση: 30/05/2008)

OSDB (2008) . The Open Source Database Benchmark - FAQ . URL: http://osdb.sourceforge.net/index.php?page=faq (προσπέλαση: 30/05/2008)

David A. Patterson, John l. Hennessy (2005) . *Computer Organization and Design: The Hardware/Software Interface* . Morgan Kaufmann, 2005.

Raghu Ramakrishnan, Johannes Gehrke (2002) . *Database Management Systems (3rd Edition)* . McGraw Hill Higher Education, 2002.

7. Γλωσσάρι

Στο κεφάλαιο αυτό γίνεται αναφορά στην αντιστοιχία της ελληνικής και της αγγλικής ορολογίας.

Ελληνικός Όρος	Αγγλικός Όρος
Αίτημα (SQL)	Query (SQL)
Ανάκτηση Πληροφοριών	Information retrieval
Ανάλυση Κώδικα	Code Profiling
Ανοικτό/Ελεύθερο λογισμικό	Open source
Αρχείο	File
Αρχείο Διαφορών	Patch
Βελτιστοποιητής	Optimizer
Βελτιστοποιητής Αιτημάτων	Query optimizer
Γεωγραφικά Συστήματα Πληροφοριών	Geographical Information Systems
Διαδικτυακός Τόπος	Site
Δυαδικό/Εκτελέσιμο (Αρχείο)	Binary (File)
Είσοδος/Εξοδος	I/O (Input/Output)
Εξυπηρετητής	Server
Ευρετήριο	Index
Λειτουργικό Σύστημα	Operating System
Μεταγλώττιση	Compilation
Μετταγλωττιστής	Compiler
Μηχανή αποθήκευησης	Storage Engine
Παραγωγή Εκτελέσιμου	Build (compilation)
Πελάτης	Client
Πηγαίος Κώδικας	Source Code
Πλαίσιο	Framework
Πληρεξούσιος Εξυπηρετητής	Proxy Server
Πρόγραμμα Αξιολόγησης	Benchmark
Προγραμματιστική Διεπαφή	Application Programming Interface
Προσωρινή Αποθήκευση	Caching
Σύστημα Διαχείρισης Βάσεων Δεδομένων	Database Management System
Τοπική Φιλοξενία	Localhost
Υπηρεσία	Service

8. Παράρτημα Ι: τα SQL αιτήματα του OSDB benchmark

Στο παράρτημα αυτό παρουσιάζονται τα SQL αιτήματα που δίνονται από το OSDB benchmark. Ο κώδικας έχει παρθεί από τον πηγαίο κώδικα του benchmark.

8.1. Δημιουργία της βάσης

Τα παρακάτω αιτήματα δίνονται κατά την δημιουργία της βάσης στην οποία γίνονται οι μετρήσεις.

```
1. create table uniques (
2. col_key    int not null,
3. col_int    int not null,
4. col_signed                int ,
5. col_float    float4 not null,
6. col_double    float not null,
7. col_decim    numeric(18,2) not null,
8. col_date    char(20) not null,
9. col_code    char(10) not null,
10.col_name    char(20) not null,col_address
11. varchar(80) not null
12.);
13.
14.create unique index uniques_key_bt on uniques
   (col_key);
15.create index uniques_code_h on uniques (col_code);
16.
17.create table updates (
18.col_key    int not null,
19.col_int    int not null,
20.col_signed                int ,
21.col_float    float4 not null,
22.col_double    float not null,
23.col_decim    numeric(18,2) not null
24.col_date    char(20) not null,
25.col_code    char(10) not null,
26.col_name    char(20) not null,
27.col_address    varchar(80) not null
28.);
29.
30.create index updates_code_h on updates (col_code);
```

```
31.create index updates_decim_bt on updates
  (col_decim);
32.create index updates_double_bt on updates
  (col_double);
33.create index updates_int_bt on updates (col_int);
34.create unique index updates_key_bt on updates
  (col_key);
35.
36.create table hundred (
37.col_key    int not null,
38.col_int    int not null,
39.col_signed                int ,
40.col_float    float4 not null,
41.col_double    float not null,
42.col_decim    numeric(18,2) not null,
43.col_date    char(20) not null,
44.col_code    char(10) not null,
45.col_name    char(20) not null,
46.col_address    varchar(80) not null
47.);
48.
49.create index hundred_code_h on hundred (col_code);
50.create unique index hundred_key_bt on hundred
  (col_key);
51.createIndexForeign("hundred", "fk_hundred_updates",
  "col_signed", "updates", "col_key");
52.
53.create table tenpct (
54.col_key    int not null,
55.col_int    int not null,
56.col_signed                int ,
57.col_float    float4 not null,
58.col_double    float not null,
59.col_decim    numeric(18,2) not null,
60.col_date    char(20) not null,
61.col_code    char(10) not null,
62.col_name    char(20) not null,
63.col_address    varchar(80) not null
64.);
65.
66.create index tenpct_code_h on tenpct (col_code);
67.create index tenpct_decim_bt on tenpct (col_decim);
68.create index tenpct_double_bt on tenpct
  (col_double);
69.create index tenpct_float_bt on tenpct (col_float);
70.create index tenpct_int_bt on tenpct (col_int);
```

```
71.create unique index tenpct_key_bt on tenpct
   (col_key);
72.create unique index tenpct_key_code_bt on tenpct
   (col_key);
73.create index tenpct_name_h on tenpct (col_name);
74.create index tenpct_signed_bt on tenpct
   (col_signed);
75.
76.create table tiny (
77.col_key    int not null
78.);
79.
80.create unique index tiny_key_bt on tiny (col_key);
```

8.2. SQL αιτήματα εκτός της δημιουργίας της βάσης δεδομένων

Στην συνέχεια παρατείθονται τα SQL αιτήματα που στέλντονται από το OSDB benchmark, εκτός από αυτά που αφορούν την δημιουργία της βάσης δεδομένων.

```
1. select * from uniques where col_int = 1
2.
3. select avg(col_signed), min(col_signed),
4.  max(col_signed),       max(col_date),
5.  min(col_date),           count(distinct col_name),
6.  count(col_name),              col_code, col_int
7. from reportview
8. where col_decim >980000000
9. group by col_code, col_int
10.
11.select avg(col_signed), min(col_signed),
12. max(col_signed),       max(col_date),
13. min(col_date),           count(distinct col_name),
14. count(col_name),              count(col_code),
15. count(col_int)      from reportview
16.where col_decim >980000000
17.
18.select avg(updates.col_decim)
19.from updates
20.where updates.col_key in (
21.  select updates.col_key
22.  from updates, hundred
23.  where hundred.col_key = updates.col_key
24.  and updates.col_decim > 980000000
```

```
25.)


26.select col_key, col_code, col_date,
27. col_signed, col_name
28.from updates
29.where col_key = 0


30.select col_key, col_int, col_signed, col_code,
31. col_double, col_name
32.from updates
33.where col_int <= 100
34.
35.select col_key, col_int, col_signed,
36.col_code, col_double, col_name
37.from updates
38.where col_key <= 100
39.
40.select col_key, col_int, col_signed,
41.col_code, col_double, col_name
42.from updates
43.where col_key = 1000
44.
45.select col_key, col_int, col_signed,
46.col_code, col_double, col_name
47.from tenpct
48.where col_signed < -250000000
49.
50.select col_key, col_int, col_signed,
51.col_code, col_double, col_name
52.from tenpct
53.where col_signed <-500000000
54.
55.select count(*)
56.from updates, sel100rand
57.where updates.col_key=sel100rand.col_key
58.and not (updates.col_double=sel100rand.col_double)
59.
60.select count(*) from updates, sel100seq
61.where updates.col_key=sel100seq.col_key
62.and not (updates.col_double=sel100seq.col_double)
63.
64.select distinct col_address, col_signed from hundred
65.
```

```
66.select distinct col_signed from tenpct
67.
68.select min(col_key) from uniques
69.select uniques.col_date, hundred.col_date,
70.tenpct.col_date, updates.col_date
71.from uniques, hundred, tenpct, updates
72.where uniques.col_key = hundred.col_key
73.and uniques.col_key = tenpct.col_key
74.and uniques.col_key = updates.col_key
75.and uniques.col_key = 1000
76.
77.select uniques.col_signed, uniques.col_date,
78.hundred.col_signed, hundred.col_date,
79.tenpct.col_signed, tenpct.col_date
80.from uniques, hundred, tenpct
81.where uniques.col_key = hundred.col_key and
  uniques.col_key = tenpct.col_key
82.and uniques.col_key = 1000
83.
84.select uniques.col_signed, uniques.col_name,
85.hundred.col_signed, hundred.col_name
86.from uniques, hundred
87.where uniques.col_key = hundred.col_key
88.and uniques.col_key =1000
89.
90.update hundred set col_signed = '-500000000'
91.where col_int = 0
92.
93.update updates
94.set col_double=col_double-100000000
95.where col_key between 1001
96.and 1100
97.
98.update updates
99.set col_double=col_double-100000000
100.where col_key between 1001 and 1100
101.
102.update updates
103.set col_double=col_double+100000000
104.where col_int between 1001 and 1100
105.
106.update updates
107.set col_code = 'SQL+GROUPS'
108.where col_key = 5005
109.
110.update updates
```

```
111.set col_int = 50015
112.where col_key = 5005
113.
114.update updates
115.set col_key = '-1000'
116.where col_key = 1000000001
117.
118.update updates
119.set col_key = '-5000'
120.where col_key = 5005
```

9. Παράρτημα ΙΙ: MySQL server και profiling API

Στην ενότητα αυτή αναλύονται οι λεπτομέρειες της βάσης όπου αποθηκεύονται τα δεδομένα και του τρόπου με τον οποίο εξάγονται τα στατιστικά.

9.1. Η βάση δεδομένων του profiling API

Οι πληροφορίες που παρέχει το profiling API αποθηκεύονται προσωρινά στην βάση δεδομένων information_schema, στον πίνακα profiling. Η δομή του είναι η εξής:

```
+----------------------+--------------+------+-----+----------+-------+
| Field                | Type         | Null | Key | Default  | Extra |
+----------------------+--------------+------+-----+----------+-------+
| QUERY_ID             | int(20)      | NO   |     | 0        |       |
| SEQ                  | int(20)      | NO   |     | 0        |       |
| STATE                | varchar(30)  | NO   |     |          |       |
| DURATION             | decimal(9,6) | NO   |     | 0.000000 |       |
| CPU_USER             | decimal(9,6) | YES  |     | NULL     |       |
| CPU_SYSTEM           | decimal(9,6) | YES  |     | NULL     |       |
| CONTEXT_VOLUNTARY    | int(20)      | YES  |     | NULL     |       |
| CONTEXT_INVOLUNTARY  | int(20)      | YES  |     | NULL     |       |
| BLOCK_OPS_IN         | int(20)      | YES  |     | NULL     |       |
| BLOCK_OPS_OUT        | int(20)      | YES  |     | NULL     |       |
| MESSAGES_SENT        | int(20)      | YES  |     | NULL     |       |
| MESSAGES_RECEIVED    | int(20)      | YES  |     | NULL     |       |
| PAGE_FAULTS_MAJOR    | int(20)      | YES  |     | NULL     |       |
| PAGE_FAULTS_MINOR    | int(20)      | YES  |     | NULL     |       |
| SWAPS                | int(20)      | YES  |     | NULL     |       |
| SOURCE_FUNCTION      | varchar(30)  | YES  |     | NULL     |       |
| SOURCE_FILE          | varchar(20)  | YES  |     | NULL     |       |
| SOURCE_LINE          | int(20)      | YES  |     | NULL     |       |
+----------------------+--------------+------+-----+----------+-------+
```

Για την μόνιμη αποθήκευση των πληροφοριών, δημιουργείται από το framework μια μόνιμη βάση δεδομένων. Η SQL που δημιουργεί την βάση είναι η εξής:

```
1. CREATE DATABASE profiling;
2.
3. USE profiling;
4.
5. CREATE TABLE queries (
6. ID  int(20) AUTO_INCREMENT NOT NULL PRIMARY KEY,
7. QUERY VARCHAR(1000),
8. SESSION int(20),
9. SESSION_TIME DATETIME
10.);
11.
```

```
12.CREATE TABLE details (
13.ID  int(20),
14.SEQ int(20),
15.STATE varchar(30),
16.DURATION decimal(9,6)
17.) engine = CSV;
```

9.2. Η συλλογή των αθροιστικών στατιστικών

Για την συλλογή των στατιστικών δεδομένων γίνεται με ένα script σε perl. Το script αυτό πέρνει ως είσοδο το αρχείο CSV που παράγεται από τον πίνακα details και αθροίζει τα δεδομένα με τρόπο ώστε να υπολογιστούν συνολικά στατιστικά. Η διαδικασία αυτή δεν γίνεται με SQL διότι η μορφή των αιτημάτων θα απαιτούσε την δημιουργία ευρετηρίων, και διαδοχικών ερωτήσεων. Με το script όμως μειώνεται ο χρόνος επεξεργασίας αφού τα στατιστικά υπολογίζονται σε ένα μόνο πέρασμα του αρχείου.

Το αρχείο είναι το get_stats.pl και είναι το επόμενο:

```
1. #/usr/bin/perl -w
2.
3. #------------
4. # Copyright, Vangelis Katsikaros, 2008
5. # Published under the GPLv3
6. #------------
7.
8. # input: a file with data like :
9. # 27,5,"cleaning up",0.000004
10.# 28,1,"starting",0.000047
11.# 28,2,"Opening tables",0.000143
12.# 28,3,"System lock",0.000006
13.# 28,4,"Table lock",0.000010
14.# 28,5,"init",0.000018
15.# 28,6,"optimizing",0.000008
16.# 28,7,"statistics: init",0.000007
17.# 28,8,"statistics: const tables",0.000004
18.# 28,9,"statistics: index stats",0.000005
19.#
20.# process of the input:
21.# caclulate the sum of the 4th column for each id
   (for example for the id 28
22.# sum all the decimal numbers)
23.# calculate the sum of the 4th column for each id
   only if the stage is "stat*"
24.# store the number foreach of the "stat*" stages
```

```
25.# then caclulate the pecentage of each number
   (towards the duration sum)
26.#
27.# output:
28.# for each query (each id) the output is:
29.#
30.# total_sum   total_stats_sum   stats_1
   stats_2 ...  stats_5
31.#
32.# and average percentages :
33.#             total_stats_sum_%  stats_1%   stats_2%
   ...  stats_5%
34.
35.use strict;
36.
37.my $line;
38.my $state;
39.my $current_query = 1;
40.my $old_query = 1;
41.my $duration = 0;
42.
43.my $lines_count = 0;
44.my $queries_count = 0;
45.my $stat_query_count = 0;
46.
47.my @duration; # for each ID
48.#  0: total duration
49.#  1: total stats duration
50.#  2: total opt duration
51.#  3: stats 1 init
52.#  4: stats 2 const
53.#  5: stats 3 index
54.#  6: stats 4 count
55.#  7: stats 5 join
56.#  8: opt 1 init
57.#  9: opt 2 conds
58.#  10: opt 3 sums
59.
60.my @total_duration = ();  # for all the IDs
61.foreach(0..10)
62.{
63.  push( @total_duration, 0);
64.}
65.
66.my @percentage;
67.#  0: total stats percentage
```

```
68.#  1: total opt percentage
69.#  2: stats 1 init
70.#  3: stats 2 const
71.#  4: stats 3 index
72.#  5: stats 4 count
73.#  6: stats 5 join
74.#  7: opt 1 init
75.#  8: opt 2 conds
76.#  9: opt 3 sums
77.
78.#open(DATAFILE, "<", "../test5.1.24-
  rcA/var/profiling/details.CSV")
79.open(DATAFILE, "<", "details.CSV")
80.  or die "can't open file: $!";
81.
82.while ($line = <DATAFILE>) {
83.  $lines_count++;
84.  $line =~ /^(\d+) , (\d+) , (\".+\") , (\d+) .
  (\d+)$/x;
85.  $current_query = $1;
86.  $state = $3;
87.  $duration = "$4.$5";
88.
89.  # new query:
90.  # clear counter and start sums over again
91.  if($old_query ne $current_query)
92.  {
93.    $old_query = $current_query;
94.    $queries_count++;
95.
96.    # the query had an optimization stage
97.    # update stats
98.    if($duration[1] != 0)
99.    {
100.      # counters
101.      $stat_query_count++;
102.
103.      # update global stats + opt durations
104.      foreach (0..10)
105.      {
106.        $total_duration[$_] += $duration[$_];
107.      }
108.
109.      # update global stats & opt percentages
110.      foreach(0..9)
111.      {
```

```
112.        $percentage[$_] += ($duration[$_ + 1] /
  $duration[0]) * 100;
113.      }
114.    }
115.
116.    @duration = ();
117.    foreach(0..10)
118.    {
119.      push(@duration, 0);
120.    }
121.  }
122.  # let's sum statistics (we work on the same query
  now)
123.
124.  $duration[0] += $duration;
125.
126.  if($state =~ /statistics/)
127.  {
128.    $duration[1] += $duration;
129.
130.    if($state =~ /statistics: init/)
131.    {
132.      $duration[3] = $duration;
133.    }
134.    elsif($state =~ /statistics: const/)
135.    {
136.      $duration[4] = $duration;
137.    }
138.    elsif($state =~ /statistics: index/)
139.    {
140.      $duration[5] = $duration;
141.    }
142.    elsif($state =~ /statistics: count/)
143.    {
144.      $duration[6] = $duration;
145.    }
146.    elsif($state =~ /statistics: join/)
147.    {
148.      $duration[7] = $duration;
149.    }
150.  }
151.  elsif($state =~ /optimizing/)
152.  {
153.    $duration[2] += $duration;
154.
155.    if($state =~ /optimizing: init/)
```

```
156.    {
157.      $duration[8] = $duration;
158.    }
159.    elsif($state =~ /optimizing: conds/)
160.    {
161.      $duration[9] = $duration;
162.    }
163.    elsif($state =~ /optimizing: sums/)
164.    {
165.      $duration[10] = $duration;
166.    }
167.  }
168.}
169.
170.print "# Time & Percentage & Counts:\n";
171.foreach(0..10)
172.{
173.  printf "%.2f", $total_duration[$_];
174.  print "\t";
175.}
176.print "\n";
177.
178.print "\t";
179.foreach(0..9)
180.{
181.  printf "%.1f", ($percentage[$_]/
  $stat_query_count);
182.  print "\t";
183.}
184.
185.print "\n";
186.print
  "$lines_count\t$queries_count\t$stat_query_count\t";
187.print "\n";
```

10. Παράρτημα: Το script του MySQL proxy

Το script του MySQL Proxy βρίσκεται στο αρχείο proxy_script.lua:

```
1. --[[
2.    Copyright (C) 2008 Vangelis Katsikaros
3.
4.     This program is free software: you can
   redistribute it and/or modify
5.     it under the terms of the GNU General Public
   License as published by
6.     the Free Software Foundation, either version 3 of
   the License, or
7.     (at your option) any later version.
8.
9.     This program is distributed in the hope that it
   will be useful,
10.    but WITHOUT ANY WARRANTY; without even the
   implied warranty of
11.    MERCHANTABILITY or FITNESS FOR A PARTICULAR
   PURPOSE.  See the
12.    GNU General Public License for more details.
13.
14.    You should have received a copy of the GNU
   General Public License
15.    along with this program.  If not, see
   <http://www.gnu.org/licenses/>.
16.
17.]]--
18.--[[
19.================================================
   ======================
20.This is a mysql proxy script. It was created for
   mysql-proxy version 0.6.0 .
21.It needs a mysql server compiled with "community-
   features" & "profiling"
22.enabled.
23.
24.Script's Task:
25.For each query that is executed by the server, the
   script retrieves data
26.from the SHOW PROFILES API of the mysql server. The
   API's data are saved in
```

```
27.a temporary table. The script saves the data in a
  permanent table.
28.
29.
30.
31.=====================================================
  ======================
32.Detailed Description:
33.
34.States of the script:
35.
36.-2 ==> -1 ==> 0 ==> 1 ==> 2 ==> 3 ==>|
37.              ^                      |
38.              |==<======<======<======<|
39.
40.Meaning of each state:
41.START: state -2
42.
43.  -2  proxy receives connection with server
44.    (server not ready: proxy must turn ON
  profiling.)
45.    Inject query
46.    goto state -1
47.
48.  -1  proxy receives 2 resultsets.
49.    Return the connection query resultset to client
  and drop
50.    the turnON profiling resultset.
51.    goto state 0
52.
53.  0  proxy recieves 1 query and will send 2 queries
  to server
54.    (proxy ready to receive new query. server ready
  for profiling.)
55.
56.    It sends to the server 2 queries. The original
  and a second which asks
57.    the profiling.QUERY_ID
58.    goto state 1
59.
60.  1  proxy receives 2 resultsets.
61.    proxy saves the client's and then sends another
  query to the server.
62.    proxy must write the profiling info of the
  specific QUERY_ID to a
63.    permanent database int the server.
```

```
64.    goto state 2
65.
66.  2  proxy sends query to server (write the
  profiling info)
67.    goto state 3
68.
69.  3  proxy recieves 1 resultset (confirmation:
  profiling info is saved).
70.    proxy must send to the client the original
  resultset
71.    goto state 4
72.
73.  4  proxy sends to the client the original
  resultset
74.    goto state 0
75.
76.]]--
77.
78.-- the state of the script
79.local state=-2
80.
81.-- the original query's resultset
82.local original_query_resultset=" "
83.local original_query_OK_affected_rows=" "
84.local original_query_OK_warning_count=" "
85.local original_query_OK_query_status=" "
86.local original_query_OK_insert_id=" "
87.local original_query_err_code = " "
88.local original_query_err_sql_state = " "
89.local original_query_err_msg = " "
90.local original_response = ""
91.
92.
93.local query_save_queries=""
94.local query_save_details=""
95.
96.-- Things that will be saved with profiling data
97.local client_query_id=0
98.local prof_query_text=""
99.local prof_query_id=0
100.local prof_session=""
101.local prof_session_time=""
102.
103.local res = { }
104.local fields = { }
105.
```

```
106.
107.
108.--[[
109.This function saves the result set that was return
  from the server in order to
110.answer the original client's query.
111.
112.In the end, the resultsett is ready to be send back
  to the client, when the
113.proxy decides it.
114.]]--
115.function save_resultset()
116.end
117.
118.
119.
120.--[[
121.Overide the proxy's functions:
122.  connect_server()
123.  read_query()
124.  read_query_result()
125.]]--
126.
127.function connect_server()
128.  print("\n\n=====")
129.  print("Proxy Version: " ..
  proxy.PROXY_VERSION .." server ndx: "
130.    .. proxy.connection.backend_ndx .. " ,
  STATE:" ..
131.    state .. " , connect_server() \n")
132.end
133.
134.
135.
136.function read_query( packet )
137.
138.  if state==-2 then
139.    --[[
140.    in order to turn ON the profiling we make the
  following assumption:
141.    The mysql command line client sends a query in
  order to ask the
142.    server version. So, we then inject "SET
  profiling=1"
143.    ]]--
```

```
144.    print (prof_session .. " , " .. state .. " ,
  C>S , rq , "
145.      .. proxy.COM_QUERY .. " , " ..
  string.sub(packet, 2))
146.    proxy.queries:append(1, packet )
147.    proxy.queries:append(2,
  string.char(proxy.COM_QUERY) .. "sET profiling=1" )
148.    -- TODO any need to use the following ???
149.    state=-1
150.    return proxy.PROXY_SEND_QUERY
151.  elseif state==0 then
152.    -- query is not COM_QUERY then don't use
  profiling. Simply:
153.    -- send query to server and send response back
  to client
154.    if( packet:byte() ~= proxy.COM_QUERY) then
155.      return proxy.PROXY_SEND_QUERY
156.    end
157.
158.    client_query_id = client_query_id+1
159.    prof_query_text = string.sub(packet, 2)
160.    prof_session =
  proxy.connection.server.thread_id
161.    prof_session_time = os.date("%Y/%m/%d %H:%M:%S
  ")
162.    print("-----")
163.    print(prof_session .. " , " .. state .. " , C>S
  , rq , QUERY ARRIVED , "
164.      .. prof_session_time .. " , ID:" ..
  client_query_id .. " , TYPE:"
165.      .. packet:byte() .. " , " ..
  prof_query_text )
166.
167.    proxy.queries:append(1, packet )
168.    proxy.queries:append(2,
  string.char(proxy.COM_QUERY)
169.      .. "sELECT MAX(QUERY_ID) FROM
  information_schema.PROFILING" )
170.
171.    state=1
172.    return proxy.PROXY_SEND_QUERY
173.  elseif state==2 then
174.    print(prof_session .. " , " .. state ..
175.      " , rq , save profiling data , " ..
  prof_query_id)
176.
```

```
177.    query_save_queries =
178.      "iNSERT INTO profiling.queries(ID, QUERY,
  SESSION, SESSION_TIME) VALUES ("
179.      .. " NULL , "               -- ID : NULL
  because of AUTO_INC
180.      .. "\"" .. prof_query_text .. "\" , "  --
  QUERY
181.      .. prof_session .. " , "        -- SESSION
182.      .. "\'" .. prof_session_time .. "\' )"   --
  SESSION_TIME
183.    proxy.queries:append(3,
  string.char(proxy.COM_QUERY) .. query_save_queries )
184.    -- print( query_save_queries ) -- debug
185.
186.    query_save_details = "iNSERT INTO
  profiling.details "
187.      .. " (ID, SEQ, STATE, DURATION) "
188.             --       ID,         SEQ, STATE,
  DURATION
189.      .. "SELECT LAST_INSERT_ID(), SEQ, STATE,
  DURATION "
190.      .. "FROM information_schema.PROFILING "
191.      .. "WHERE QUERY_ID = " .. prof_query_id
192.    proxy.queries:append(4,
  string.char(proxy.COM_QUERY) .. query_save_details )
193.    -- print( query_save_details ) -- debug
194.
195.    state=3
196.    return proxy.PROXY_SEND_QUERY
197.  else
198.    print(prof_session .. " , " .. "rq !!! oops
  " .. state)
199.  end
200.end
201.
202.
203.
204.function read_query_result( inj )
205.
206.  if state == -1 then
207.    --[[ we make the assumption (which is currently
  correct) that
208.    the order of processing the received queries is
  the same as the order
209.    of their ids. This plays important role when we
  choose to change the
```

```
210.    state (this must happen only after we processed
  all the resultsets) ]]--
211.    if(inj.id == 1) then
212.      print(prof_session .. " , " .. state .. " ,
  S>C , rqr , "
213.      .. inj.id  .. " send resultset")
214.
215.    elseif(inj.id == 2) then
216.      print(prof_session .. " , " .. state .. " ,
  S>C , rqr , "
217.      .. inj.id  .. " ignore injection (if
  exists)")
218.      state = 0
219.      return proxy.PROXY_IGNORE_RESULT
220.    end
221.
222.  elseif state == 1 then
223.
224.    -- the response for the client
225.    if(inj.id == 1) then
226.      --print (original_response)
227.      if(inj.resultset.rows == nil) then
228.        -- TODO DONE???
229.        -- catch error packets
230.        print(prof_session .. " , " .. state ..
231.        " , S>C , rqr , " .. inj.id  ..
232.        " client's response (save and don't FWD),
  NO resultset-simple OK packet"
233.        .. inj.resultset.affected_rows)
234.        res=-1
235.        original_response = inj.resultset.raw
236.        print(inj.resultset.raw)
237.        original_query_OK_query_status =
  inj.resultset.query_status
238.
239.        -- catch packets with mysql error
240.        if(inj.resultset.query_status == -1) then
241.          print("there was an error in the query
  ");
242.        end
243.      else
244.        print(prof_session .. " , " .. state ..
245.        " , S>C , rqr , " .. inj.id  ..
246.        " client's response (save and don't FWD) ,
  resultset: ")
247.        -- save the original query resultset
```

```
248.        -- clean the variables
249.        res = { }
250.        fields = { }
251.        -- save rows in res
252.        for row in inj.resultset.rows do
253.              res[#res + 1] = row
254.        end
255.        -- save fields in fields
256.        for n = 1, #inj.resultset.fields do
257.              fields[#fields + 1] = {
258.                    type =
  inj.resultset.fields[n].type,
259.                    name =
  inj.resultset.fields[n].name,
260.              }
261.        end
262.        -- print result set , just debugging help
263.        if(num_cols == 0) then
264.          print(" 0 cols ")
265.        end
266.        if(num_cols > 255) then
267.          print(" !!! >255 cols")
268.        end
269.        if( (num_cols > 0) and (num_cols < 255) )
  then
270.          dump_query_result(inj)
271.        end
272.      end
273.      -- drop the original query resultset
274.      return proxy.PROXY_IGNORE_RESULT
275.
276.    -- the response for the proxy's injected query
277.    elseif(inj.id == 2)then
278.      print(prof_session .. " , " .. state ..
279.      " , S>C , rqr , " .. inj.id  ..
280.      " \"sel max()\" resultset: ignore inj ,
  profiling resultset:")
281.
282.      for row in inj.resultset.rows do
283.          -- TODO [CHECK]
284.          prof_query_id = row[1]
285.          print("         " .. row[1] ..
286.          " , PROF_QUERY_ID: " .. prof_query_id )
287.      end
288.      -- now proxy is ready to save the profiling
  data in the `profiling` db.
```

```
289.      state=2
290.      read_query()
291.      return proxy.PROXY_IGNORE_RESULT
292.    end
293.  elseif state==3 then
294.    if(inj.id == 1) then
295.      print(prof_session .. " , !!! ??? " ..
  state ..
296.        " , S>P , rqr , " .. inj.id )
297.    elseif(inj.id == 2) then
298.      print(prof_session .. " , !!! ??? " ..
  state ..
299.        " , S>P , rqr , " .. inj.id )
300.    elseif(inj.id == 3) then
301.      print(prof_session .. " , " .. state .. " ,
  S>P , rqr , "
302.        .. inj.id .. " \"queries\" resultset:
  ignore inj " )
303.
304.--==============
305.      if inj.resultset.type ==
  proxy.MYSQLD_PACKET_OK then
306.        print("MYSQLD_PACKET_OK")
307.        if inj.resultset.affected_rows then
308.          print(inj.resultset.affected_rows .. "  "
  .. inj.resultset.insert_id)
309.        end
310.      end
311.--==============
312.
313.      return proxy.PROXY_IGNORE_RESULT
314.    elseif(inj.id == 4) then
315.      print(prof_session .. " , " .. state .. " ,
  S>P , rqr , "
316.        .. inj.id .. " \"details\" resultset:
  ignore inj" )
317.
318.      state=4
319.      print(prof_session .. " , " .. state .. " ,
  P>C")
320.
321.      -- send original resultset to the client
322.      if(res == -1)then
323.        proxy.response.raw = original_response
324.        print(proxy.response.raw)
```

```
325.        proxy.response.type =
  proxy.MYSQLD_PACKET_OK
326.        if(original_query_OK_query_status == -1)
  then
327.          proxy.response.type =
  proxy.MYSQLD_PACKET_ERR
328.        end
329.--[[
330.        proxy.response.type =
  proxy.MYSQLD_PACKET_OK
331.        proxy.response.affected_rows =
  original_query_OK_affected_rows
332.        proxy.response.warning_count =
  original_query_OK_warning_count
333.        proxy.response.query_status =
  original_query_OK_query_status
334.        proxy.response.insert_id =
  original_query_OK_insert_id
335.        if(original_query_OK_query_status == -1)
  then
336.          proxy.response.type =
  proxy.MYSQLD_PACKET_ERR
337.      --    proxy.response.err.code =
  original_query_err_code
338.      --    proxy.response.err.sql_state =
  original_query_err_sql_state
339.          proxy.response.errmsg = "yahoo!" --
  original_query_err_msg --"yahoo!"
340.        end
341.]]--
342.      else
343.        proxy.response = {
344.                type = proxy.MYSQLD_PACKET_OK,
345.                resultset = {
346.                        rows = res
347.                }
348.        }
349.        proxy.response.resultset.fields = fields
350.      end
351.
352.      state=0
353.      return proxy.PROXY_SEND_RESULT
354.    end
355.  else
356.    print(prof_session .. " , " .. "rqr !!! oops
  " .. state)
```

```
357.  end
358.
359.end
360.
361.
362.--[[
363.   the following code is Copyright (C) 2007 MySQL
  AB
364.   retieved in 29/May/2008 from
365.http://svn.mysql.com/svnpublic/mysql-
  proxy/trunk/examples/tutorial-packets.lua
366.
367.   used under the GNU General Public License terms
368.]]--
369.
370.--- dump the result-set to stdout
371.--
372.-- @param inj "packet.injection"
373.local function dump_query_result( inj )
374.  local field_count = 0
375.  local fields = inj.resultset.fields
376.
377.  while fields[field_count] do
378.    local field = fields[field_count]
379.    print("| | field[" .. field_count .. "] =
  { type = " ..
380.            field.type .. ", name = " .. field.name
  .. " }" )
381.    field_count = field_count + 1
382.  end
383.
384.  local row_count = 0
385.  for row in inj.resultset.rows do
386.    local cols = {}
387.    local o
388.    for i = 1, field_count do
389.      if not o then
390.        o = ""
391.      else
392.        o = o .. ", "
393.      end
394.      if not row[i] then
395.        o = o .. "(nul)"
396.      else
397.        o = o .. row[i]
398.      end
```

```
399.    end
400.    print("| | row["..row_count.."] = { " .. o .. "
  }")
401.    row_count = row_count + 1
402.  end
403.end
```

11. Παράρτημα: MySQL server patches

Τα 2 patches διαφέρουν ως προς το πλήθος των νέων σημείων ελέγχου του profiling API τα οποία εισάγουν. Στο σενάριο B εισάγονται 4 σημεία ελέγχου ενώ στο σενάριο C εισάγονται τα 4 σημεία ελέγχου όπως και στο σενάριο B και άλλα 3 σημεία ελέγχου επιπλέον.

11.1. Σενάριο B

Το patch βρίσκεται στο αρχείο flavor_b.patch:

```
1. +++ sql_select.cc2008-06-10 22:48:51.000000000 +0300
2. @@ -938,7 +939,7 @@
3.    sort_by_table= get_sort_by_table(order,
  group_list, select_lex->leaf_tables);
4.
5.    /* Calculate how to do the join */
6. -  thd_proc_info(thd, "statistics");
7. +/*  thd_proc_info(thd, "statistics"); */
8.    if (make_join_statistics(this, select_lex-
  >leaf_tables, conds, &keyuse) ||
9.        thd->is_fatal_error)
10.   {
11.@@ -2453,11 +2454,15 @@
12.   SARGABLE_PARAM *sargables= 0;
13.   JOIN_TAB *stat_vector[MAX_TABLES+1];
14.   DBUG_ENTER("make_join_statistics");
15.+  THD *thd= join->thd;/*VBench*/
16.+  thd_proc_info(thd, "statistics: init");
  /*VBench*/
17.
18.   table_count=join->tables;
19.   stat=(JOIN_TAB*) join->thd-
  >calloc(sizeof(JOIN_TAB)*table_count);
20.   stat_ref=(JOIN_TAB**) join->thd-
  >alloc(sizeof(JOIN_TAB*)*MAX_TABLES);
21.   table_vector=(TABLE**) join->thd-
  >alloc(sizeof(TABLE*)*(table_count*2));
22.+
23.+
24.   if (!stat || !stat_ref || !table_vector)
```

```
25.     DBUG_RETURN(1);                          // Eom /*
  purecov: inspected */
26.
27.@@ -2610,7 +2615,7 @@
28.     else
29.       found_const_table_map|= s->table->map;
30.   }
31.-
32.+     thd_proc_info(thd, "statistics:
  const_tables"); /*VBench*/
33.   /* loop until no more const tables are found */
34.   int ref_changed;
35.   do
36.@@ -2748,6 +2753,7 @@
37.     }
38.   } while (join->const_table_map & found_ref &&
  ref_changed);
39.
40.+  thd_proc_info(thd, "statistics: index_stats");
  /*VBench*/
41.   /*
42.     Update info on indexes that can be used for
  search lookups as
43.     reading const tables may has added new sargable
  predicates.
44.@@ -2770,6 +2776,7 @@
45.
46.   /* Calc how many (possible) matched records in
  each table */
47.
48.+       thd_proc_info(thd, "statistics:
  count_rows"); /*VBench*/
49.   for (s=stat ; s < stat_end ; s++)
50.   {
51.     if (s->type == JT_SYSTEM || s->type ==
  JT_CONST)
52.@@ -2848,8 +2855,9 @@
53.   join->table= join->all_tables=table_vector;
54.   join->const_tables=const_count;
55.   join-
  >found_const_table_map=found_const_table_map;
56.-
57.-  /* Find an optimal join order of the non-constant
  tables. */
58.+
```

```
59.+  thd_proc_info(thd, "statistics: join_order");
  /*VBench*/
60.+      /* Find an optimal join order of the non-
  constant tables. */
61.   if (join->const_tables != join->tables)
62.   {
63.     optimize_keyuse(join, keyuse_array);
64.@@ -4839,6 +4847,8 @@
65.
66.   DBUG_ENTER("greedy_search");
67.
68.+      // THD *thd= join->thd;/*VBench*/
69.+      // thd_proc_info(thd, "statistics:
  greedy_search"); /*VBench*/
70.   /* number of tables that remain to be optimized
  */
71.   size_remain= my_count_bits(remaining_tables);
72.
73.@@ -5163,6 +5173,7 @@
74. {
75.   DBUG_ENTER("find_best");
76.   THD *thd= join->thd;
77.+       //thd_proc_info(thd, "statistics:
  find_best"); /*VBench*/
78.   if (thd->killed)
79.     DBUG_RETURN(TRUE);
80.   if (!rest_tables)
```

11.2. Σενάριο C

Το patch βρίσκεται στο αρχείο flavor_c.patch:

```
1. --- sql_select__.cc 2008-04-08 14:23:02.000000000
  +0300
2. +++ sql_select.cc2008-06-10 20:33:41.000000000 +0300
3. @@ -759,7 +759,8 @@
4.      DBUG_RETURN(0);
5.    optimized= 1;
6.
7. -  thd_proc_info(thd, "optimizing");
8. +  thd_proc_info(thd, "optimizing: init"); /*VBench*/
9. +  /*thd_proc_info(thd, "optimizing");*/
10.   row_limit= ((select_distinct || order ||
  group_list) ? HA_POS_ERROR :
11.             unit->select_limit_cnt);
```

```
12.   /* select_limit is used to decide if we are
  likely to scan the whole table */
13.@@ -817,7 +818,7 @@
14.     if (arena)
15.       thd->restore_active_arena(arena, &backup);
16.   }
17.-
18.+  thd_proc_info(thd, "optimizing: conds");
  /*VBench*/
19.   conds= optimize_cond(this, conds, join_list,
  &cond_value);
20.   if (thd->is_error())
21.   {
22.@@ -870,7 +871,7 @@
23.     }
24.   }
25. #endif
26.-
27.+  thd_proc_info(thd, "optimizing: sums");
  /*VBench*/
28.   /* Optimize count(*), min() and max() */
29.   if (tables_list && tmp_table_param.sum_func_count
  && ! group_list)
30.   {
31.@@ -938,7 +939,7 @@
32.   sort_by_table= get_sort_by_table(order,
  group_list, select_lex->leaf_tables);
33.
34.   /* Calculate how to do the join */
35.-  thd_proc_info(thd, "statistics");
36.+/*  thd_proc_info(thd, "statistics"); */
37.   if (make_join_statistics(this, select_lex-
  >leaf_tables, conds, &keyuse) ||
38.       thd->is_fatal_error)
39.   {
40.@@ -2453,11 +2454,15 @@
41.   SARGABLE_PARAM *sargables= 0;
42.   JOIN_TAB *stat_vector[MAX_TABLES+1];
43.   DBUG_ENTER("make_join_statistics");
44.+  THD *thd= join->thd;/*VBench*/
45.+  thd_proc_info(thd, "statistics: init");
  /*VBench*/
46.
47.   table_count=join->tables;
48.   stat=(JOIN_TAB*) join->thd-
  >calloc(sizeof(JOIN_TAB)*table_count);
```

```
49.   stat_ref=(JOIN_TAB**) join->thd-
  >alloc(sizeof(JOIN_TAB*)*MAX_TABLES);
50.   table_vector=(TABLE**) join->thd-
  >alloc(sizeof(TABLE*)*(table_count*2));
51.+
52.+
53.   if (!stat || !stat_ref || !table_vector)
54.     DBUG_RETURN(1);                       // Eom /*
  purecov: inspected */
55.
56.@@ -2610,7 +2615,7 @@
57.     else
58.       found_const_table_map|= s->table->map;
59.   }
60.-
61.+     thd_proc_info(thd, "statistics: const
  tables"); /*VBench*/
62.   /* loop until no more const tables are found */
63.   int ref_changed;
64.   do
65.@@ -2748,6 +2753,7 @@
66.     }
67.   } while (join->const_table_map & found_ref &&
  ref_changed);
68.
69.+  thd_proc_info(thd, "statistics: index stats");
  /*VBench*/
70.   /*
71.     Update info on indexes that can be used for
  search lookups as
72.     reading const tables may has added new sargable
  predicates.
73.@@ -2770,6 +2776,7 @@
74.
75.   /* Calc how many (possible) matched records in
  each table */
76.
77.+       thd_proc_info(thd, "statistics: count
  rows"); /*VBench*/
78.   for (s=stat ; s < stat_end ; s++)
79.   {
80.     if (s->type == JT_SYSTEM || s->type ==
  JT_CONST)
81.@@ -2848,8 +2855,9 @@
82.   join->table= join->all_tables=table_vector;
83.   join->const_tables=const_count;
```

```
84.   join-
  >found_const_table_map=found_const_table_map;
85.-
86.-  /* Find an optimal join order of the non-constant
  tables. */
87.+
88.+  thd_proc_info(thd, "statistics: join order");
  /*VBench*/
89.+     /* Find an optimal join order of the non-
  constant tables. */
90.   if (join->const_tables != join->tables)
91.   {
92.     optimize_keyuse(join, keyuse_array);
93.@@ -4839,6 +4847,8 @@
94.
95.   DBUG_ENTER("greedy_search");
96.
97.+     // THD *thd= join->thd;/*VBench*/
98.+     // thd_proc_info(thd, "statistics:
  find_best"); /*VBench*/
99.   /* number of tables that remain to be optimized
  */
100.   size_remain= my_count_bits(remaining_tables);
101.
102.@@ -5163,6 +5173,7 @@
103. {
104.   DBUG_ENTER("find_best");
105.   THD *thd= join->thd;
106.+     //thd_proc_info(thd, "statistics:
  find_best"); /*VBench*/
107.   if (thd->killed)
108.     DBUG_RETURN(TRUE);
109.   if (!rest_tables)
```

12. Εκτέλεση του framework

Η εκτέλεση του framework περιλαμβάνει την διαδοχική εκτέλεση των 3 σεναρίων του MySQL server, για ένα πλήθος φορών στο κάθε σενάριο. Για την αυτόματη εκτέλεση των σεναρίων χρησιμοποιείται το ακόλουθο shell script (αρχείο benchmark.sh).

```
1. #!/bin/bash
2.
3. #================================================
   ==========================#
4. #   Copyright (C) 2008 Vangelis Katsikaros
5. #
6. #   This program is free software; you can
   redistribute it and/or modify
7. #   it under the terms of the GNU General Public
   License as published by
8. #   the Free Software Foundation; version 2 of the
   License.
9. #
10.#   This program is distributed in the hope that it
   will be useful,
11.#   but WITHOUT ANY WARRANTY; without even the
   implied warranty of
12.#   MERCHANTABILITY or FITNESS FOR A PARTICULAR
   PURPOSE.  See the
13.#   GNU General Public License for more details.
14.#
15.#   You should have received a copy of the GNU
   General Public License
16.#   along with this program; if not, write to the
   Free Software
17.#   Foundation, Inc., 59 Temple Place, Suite 330,
   Boston, MA  02111-1307  USA
18.#
19.#                                   ---------
20.#
21.# version 0.1 - 07/13/2008
22.#
23.#================================================
   ==========================#
24.
25.
```

```
26.#================================================
  ==========================#
27.#
28.# CHANGE the following according to your environment
  !!!
29.#
30.# mysql_workdir=/home/vag/test_mysql6
31.# mysql_version=5.1.25-rc
32.# mysql_flavor=_a
33.# mysql_port=3311
34.#
35.# Change this manually in the code
36.# TODO change this.
37.# mysql_password=5-1-25a , b , c
38.#
39.#================================================
  ==========================#
40.
41.
42.
43.
44.#================================================
  ==========================#
45.#                              perform the benchmark
  #
46.#================================================
  ==========================#
47.do_benchmark(){
48.for i in `seq 1 1`;
49.do
50.  # we need this for running OSDB
51.  export
  LD_LIBRARY_PATH=$mysql_workdir/test$mysql_version$mys
  ql_flavor/lib/mysql/
52.
53.  echo "==== Create profiling db"
54.  $mysql_workdir/test$mysql_version$mysql_flavor/bin
  /mysql --no-defaults  --user=root -p$mysql_pass
  --host=127.0.0.1 --port=$mysql_port <
  $mysql_workdir/framework/create_prof_db.sql
55.
56.  echo "==== Run OSDB benchmark $i"
57.       echo `date +%Y%m%d_%H%M` "Run $i" >>
  benchmark.out
58.       cd $mysql_workdir
59.       cd osdb/osdb-$mysql_version$mysql_flavor
```

```
60.       ./bin/osdb-my --datadir
  $mysql_workdir/osdb/osdb-$mysql_version$mysql_flavor/
  bin/data/ --dbhost 127.0.0.1 --dbport 4040 --dbuser
  root --dbpassword $mysql_pass
61.
62.
63.       echo "## Copy dataset:"
64.  cd $mysql_workdir/framework/
65.  cp $mysql_workdir/test$mysql_version$mysql_flavor/
  var/profiling/details.CSV .
66.  chmod o+wrx details.CSV
67.       echo "## Process dataset:"
68.  perl $mysql_workdir/framework/get_stats.pl >>
  stats.out
69.       echo "## Xip dataset:"
70.  zip dataset$mysql_version$mysql_flavor.$i.zip
  details.CSV
71.       echo "## Remove unzipped copy:"
72.       rm details.CSV
73.done
74.}
75.
76.
77.
78.#================================================
  ============================#
79.#                                start/stop mysqld
  #
80.#================================================
  ============================#
81.start_mysql_server(){
82.  echo "==== start mysqld"
83.  su mysql
  --command="$mysql_workdir/test$mysql_version$mysql_fl
  avor/bin/mysqld_safe --no-defaults --port=$mysql_port
  --default_storage_engine=innodb
  --basedir=$mysql_workdir/test$mysql_version$mysql_fla
  vor/ & "
84.  sleep 5
85.}
86.
87.stop_mysql_server(){
88.  echo "==== stop mysqld"
89.  su mysql --command="mysqladmin -p$mysql_pass
  --port=$mysql_port -uroot stop --host=127.0.0.1
  shutdown"
```

```
90.  sleep 10
91.}
92.
93.#================================================
  ==========================#
94.#                            start/stop mysql-proxy
  #
95.#================================================
  ==========================#
96.
97.
98.stop_mysql_proxy(){
99.  echo "==== stop mysql-proxy"
100.  # this is a bit brutal
101.  # TODO change to something better
102.  killall mysql-proxy
103.  sleep 8
104.}
105.
106.
107.
108.#================================================
  ==========================#
109.#                            putting it all together
  #
110.#================================================
  ==========================#
111.
112.
113.# TODO change the manual password editing
114.
115.
116.
117.export mysql_workdir=/home/vag/test_mysql6
118.export mysql_port=3311
119.export mysql_version=5.1.25-rc
120.
121.echo "-------------------------------"
122.echo "mysql-proxy should be running!!!"
123.echo "-------------------------------"
124.echo ""
125.#start_mysql_proxy
126.
127.export mysql_flavor=_a
128.export mysql_pass=5-1-25a
```

```
129.echo "============= Benchmarking:
  $mysql_version$mysql_flavor" >> stats.out
130.start_mysql_server
131.do_benchmark
132.stop_mysql_server
133.
134.
135.export mysql_flavor=_b
136.export mysql_pass=5-1-25b
137.echo "============= Benchmarking:
  $mysql_version$mysql_flavor" >> stats.out
138.#start_mysql_server
139.#do_benchmark
140.#stop_mysql_server
141.
142.
143.
144.export mysql_flavor=_c
145.export mysql_pass=5-1-25c
146.echo "============= Benchmarking:
  $mysql_version$mysql_flavor" >> stats.out
147.#start_mysql_server
148.#do_benchmark
149.#stop_mysql_server
150.
151.
152.su stop_mysql_proxy
```