KLEE: A Framework for Distributed top-k Query Algorithms

Sebastian Michel
Peter Triantafillou
Gerhard Weikum

VLDB 2005

Αντικείμενο της εργασίας

- Η εργασία αναφέρεται στο πρόβλημα των top-k queries που αφορούν δεδομένα τα οποία είναι κατανεμημένα σε πολλούς κόμβους
- Για τον υπολογισμό του κόστους συνυπολογίζονται
  - ο λανθάνων χρόνος του δικτύου,
  - η κατανάλωση εύρους ζώνης και
  - ο φόρτος κάθε κόμβου.
- Παρουσιάζεται ο KLEE ένας νέος αλγόριθμος για κατανεμημένες top-k επερωτήσεις, σχεδιασμένος για υψηλή απόδοση και ευελιξία.

KLEE (1)

- Κέρδος σε αποδοτικότητα εάν επιτρέπουμε μικρά λάθη στην «ποιότητα» των αποτελεσμάτων.
- Ο κόμβος αρχικοποίησης της επερώτησης έχει τη δυνατότητα για trade-off
  - ποιότητα των αποτελεσμάτων vs. απόδοσης
  - βημάτων εκτέλεσης vs. απόδοσης του εύρους ζώνης του δικτύου
  - Υλοποιήθηκε ο αλγόριθμος καθώς και άλλοι της ίδιας κατηγορίας και εφαρμόστηκαν σε πραγματικά και συνθετικά δεδομένα ώστε να γίνει αξιολόγηση του αλγόριθμου.

KLEE (2)

- Τα αποτελέσματα δείχνουν ότι ο KLEE επιτυγχάνει σημαντικά κέρδη σε απόδοση σε σχέση
  - με το εύρος ζώνης,
  - το χρόνο απόκρισης στις επερωτήσεις και
  - λιγότερο φόρτο στους κόμβους
- Όλα με μικρό κόστος στην ακρίβεια και σε άλλα ποιοτικά μέτρα (εκτίμησης της ποιότητας).

Top-K (1)

- αναζήτηση ομοιότητας σε δεδομένα πολυμέσων
- ταξινομημένη ανάκτηση σε έγγραφα κειμένου και σε ημι-δομημένα έγγραφα
- network and stream monitoring,
- προτάσεις για παρόμοια είδη και αναζήτηση σε καταλόγους ηλεκτρονικού εμπορίου και
- γενικά σε ταξινόμηση αποτελεσμάτων σε SQL-style επερωτήσεις σε δομημένες πηγές δεδομένων.

Top-K (2)

- Όσον αφορά την απόδοση, οι πιο επιτυχημένες προσεγγίσεις βασίζονται σε αλγορίθμους κατωφλίου(TA).
- Η περιοχή αυτή έχει κατανοηθεί σε μεγάλο βαθμό όσον αφορά δεδομένα σε ένα κόμβο αλλά είναι σε πρώιμο στάδιο για κατανεμημένα συστήματα όπως π.χ. για peer-to-peer.
- Μία εξαίρετη εφαρμογή θα ήταν η υλοποίηση μιας κατανεμημένης μηχανής αναζήτησης.

Υποθέσεις

- Θεωρούμε ότι οι index list είναι κατανεμημένες στους κόμβους
- Διακρίνουμε δυο περιπτώσεις για το κόστος επικοινωνίας:
  - Λανθάνων χρόνος από τους γύρους μηνυμάτων
  - Κατανάλωση εύρους ζώνης από την ανταλλαγή δεδομένων μεταξύ των κόμβων
- Τέλος έχουμε σειριακή και όχι τυχαία προσπέλαση στις index lists ώστε να περιοριστούν τα τοπικά κόστη επεξεργασίας στους κόμβους

Καθορισμός Προβλήματος (1)

- Κατανεμημένο σύστημα N κόμβους P_i, i=1...N συνδεδεμένοι π.χ. με έναν πίνακα κατακερματισμού
- Data items
  - Documents (web)
  - Structured (περιγραφείς ταινιών)
- Κάθε data item είναι συνδεδεμένο με ένα σύνολο από
  - Descriptors
  - Text terms
  - Attribute values

Καθορισμός Προβλήματος (2)

- Η inverted index list ενός descriptor είναι η λίστα των data items στα οποία ο descriptor εμφανίζεται ταξινομημένη σε φθίνουσα σειρά των *scores*.
- Αυτές οι index lists είναι η granularity κατανομή του κατανεμημένου συστήματος
- Κάθε κόμβος έχει αποθηκευμένη μια index list $I_j(t)$ για έναν όρο t.
- Αποτελείται από ζευγάρια *(docID,score)* όπου το

  $score \; \varepsilon(0,1]$

  και δείχνει το πόσο σημαντικό είναι το έγγραφο *docID* για τον όρο t.

Καθορισμός Προβλήματος (3)

- Ένα ερώτημα $q(T, k)$ αποτελείται από το μη κενό σύνολο $T=\{t_1, t_2, \ldots t_t\}$ και έναν ακέραιο k.
- Έστω m οι κόμβοι που περιέχουν τις πιο σχετικές index lists με $m \leq t$.
- P_{init} είναι ο κόμβος που αρχικοποιεί την ερώτηση
- *TotalScore* είναι μια μονοτονική συνάρτηση των επιμέρους *score*. Χρησιμοποιούμε την άθροιση.
- Στόχος μας είναι να επινοήσουμε αποδοτικές μεθόδους ώστε η P_{init} να προσπελαύνει τις m λίστες και να δημιουργεί το αποτέλεσμα για τα top-k έγγραφα.
- Το αποτέλεσμα είναι η διατεταγμένη λίστα *(docID, TotalScore)*

Προσέγγιση Λύσης (1)

- Συγκέντρωση των m λιστών στον αρχικό κόμβο
- Εκτέλεση μιας κεντρική TA-style μεθόδου.
- Για ένα peer-to-peer σύστημα είναι μη αποδεκτή γιατί γίνεται σπατάλη του εύρους ζώνης κατά την μεταφορά των λιστών στο P_{init} .

Προσέγγιση Λύσης (2)

- Εκτέλεση ενός ΤΑ στον P_{init} και κάθε φορά να προσπελαύνουμε ένα entry από τον κόμβο που είναι αποθηκευμένος
- Και αυτό είναι μη αποδεκτό καθώς έχουμε πολλά μικρά μηνύματα στο δίκτυο και χρειαζόμαστε τόσους γύρους μηνυμάτων όσο και το μέγιστο μέγεθος μεταξύ των λιστών.

Προσέγγιση Λύσης (3)

- [CW04]: ένας αποδοτικός κατανεμημένος top-k αλγόριθμος σε σχέση με το δίκτυο θα πρέπει να τερματίζει σε ένα μικρό, fixed, αριθμό βημάτων
- Ιδανικά
  - έχουμε σταθερό αριθμό βημάτων (αποδεκτός λανθάνων χρόνος)
  - απαιτεί από τους συνεργαζόμενους κόμβους να στέλνουν το δυνατόν λιγότερη (μικρή κατανάλωση εύρους ζώνης)
- Σε κάθε βήμα ο P_{init} μαζεύει πληροφορίες από τις index lists των κόμβων και προσπαθεί με έξυπνο τρόπο να καταλάβει εάν έχει την απαραίτητη πληροφορία ώστε να εκτιμήσει το αποτέλεσμα με υψηλή ποιότητα και να τερματίσει την διαδικασία

Προσέγγιση Λύσης (3)

- Υπάρχουν δύο προβλήματα
  - missing scores: τα id των document περιλαμβάνονται στις αναφορές κάποιων κόμβων και όχι σε άλλων. Αυτό κάνει πολύπλοκο τον υπολογισμό του *TotalScore*
  - Αποτέλεσμα: δεν μπορούμε να κάνουμε prune νωρίς υποψήφια top-k ή να παράγουμε μια καλή εκτίμηση του top-k.
  - missing documents: ο συντονιστής δεν έχει πλήρη εικόνα των document. Είναι δύσκολο για τον συντονιστή να μάθει για έγγραφα το οποία είναι χαμηλά στις λίστες για κάθε όρο αλλά συνολικά το άθροισμα να το κάνει ένα καλό υποψήφιο για το top-k.

TA-sorted (1)

- Ο KLEE βασίζεται στην παραλλαγή TA-sorted
- Επεξεργάζεται τις entries (*docID,score)* στις σχετικές λίστες με φθίνουσα σειρά για τα *score*
- Χρησιμοποιεί round robin και
- Κάνει μόνο σειριακή αναζήτηση (sequential access) στις λίστες.
- Ο TA-sorted κρατάει μια ουρά από υποψήφια για top-k και τα μέχρι στιγμής αποτελέσματα στη μνήμη.

TA-sorted (2)

- Ο αλγόριθμος κρατάει ανά πάσα στιγμή μαζί με τα document d το *worstScore* και το *bestScore.*
- *worstScore*: είναι το άθροισμα των *score* για κάθε λίστα που έχει εμφανιστεί το d
- *bestScore:* το *worstScore* προσαυξημένο με τις τιμές που έχουν τα τελευταία documents στις λίστες στις οποίες δεν υπάρχει το d (high(i) για την i λίστα) και είναι άνω όρια για τις τιμές που δεν έχουμε ακόμα επισκεφθεί.
- current top-k: είναι το σύνολο με τα k documents που έχουν το καλύτερο *worstScore*

TA-sorted (3)

- *topKscore* είναι το ελάχιστο μεταξύ των *k* documents
- Εάν για ένα υποψήφιο ισχύει *bestScore*<*topKscore* τότε το d παύει να είναι υποψήφιο
- Ο αλγόριθμος τερματίζει όταν η λίστα υποψηφίων είναι άδεια
- [TES04]: μπορούμε να αντικαταστήσουμε τα high(i) με ποσότητες από τις ουρές των λιστών. Ένα document παύει να είναι υποψήφιο όταν

$$P\left[worstScore(d) + \sum_i S(i) > topKscore\right] > \varepsilon$$

όπου S(i) είναι τυχαίες μεταβλητές για τα άγνωστα *score*

Προηγούμενες Εργασίες (1)

- Ο πρώτος TA-style κατανεμημένος αλγόριθμος παρουσιάστηκε στο [BGM02,MGB04]
- Αφορά σε top-k επερωτήσεις για δεδομένα στο διαδίκτυο για recommendation services (e.g. Restaurants, street finders)
- Η προσέγγιση χρησιμοποιεί έναν υβριδικό αλγόριθμο που επιτρέπει sorted και τυχαία προσπέλαση αλλά προσπαθεί να αποφύγει τον δεύτερο τύπο.
- Στο [CH02] χρησιμοποιείται κάτι παρόμοιο
- Καμία από αυτές τις εργασίες δεν είναι σχετικές με την συγκεκριμένη εργασία γιατί έχουμε αόριστο αριθμό γύρων μηνυμάτων

Προηγούμενες Εργασίες (2)

- Ο [Su03] είναι για δυο κόμβους αλλά δεν λέει πως γενικεύεται σε περισσότερους. Η γενική ιδέα είναι οι 2 συνεργάτες να επικοινωνούν άμεσα και όχι μέσω ενός συντονιστή.
- Ο [Ba05] προτείνει έναν αλγόριθμο για βελτιστοποίηση του κόστους επικοινωνίας για P2P συστήματα. Όμως δίνει έμφαση σε συγκεκριμένες τοπολογίες δικτύων και συγκεκριμένα σε μια υπέρ-κυβική.
- Ο πιο κοντινός στην φιλοσοφία του paper είναι ο TPUT που προτάθηκε από το [CW04]

Προηγούμενες Εργασίες (3)

- Ο TPUT έχει τρία βήματα
  - Παίρνει τα top-k (*docID,score)* από κάθε συνεργαζόμενο κόμβο και υπολογίζει τα *topKscore* βάζοντας μηδέν σε όποια *score* λείπουν.
  - Ζητάει από κάθε κόμβο να το επιστρέψουν τις entries όπου ισχύει

$$score > \frac{topKscore}{m}$$

και υπολογίζει μια καλύτερη προσέγγιση του top-k και αποκλείει υποψήφια τον οποίον το *bestScore<topkScore*

  - Υπολογίζει τα *score* που του λείπουν ζητώντας από τους κόμβους να κάνουν random access.

KLEE

- Η προτεινόμενη προσέγγιση βασίζεται στο να έχουμε έναν συντονιστή για κάθε επερώτηση και μια ομάδα από κόμβους
- Ο συντονιστής είναι ο P_{init} ενώ η ομάδα είναι οι κόμβοι που έχουν τις απαραίτητες λίστες
- Ο αλγόριθμος λειτουργεί με φάσεις επικοινωνίας
- Σε κάθε φάση ο συντονιστής ζητά και λαμβάνει ένα τμήμα τις λίστας που έχει κάθε κόμβος ώστε να τρέξει έναν αλγόριθμο για top-k βασισμένο στα δεδομένα που έχει λάβει.

HistogramBlooms Structure (1)

- Κάθε κόμβος διατηρεί ένα σύνολο από στατιστικά μετά-δεδομένα για να περιγράψει την λίστα του
- Συγκεκριμένα κρατείτε πληροφορία σε μορφή ιστογράμματος για να περιγραφεί η κατανομή του *score* στην λίστα. Το *score* $\varepsilon(0,1]$
- Θεωρούμε ότι τα ιστογράμματα έχουν ίσο πλάτος και είναι χωρισμένα σε n κελιά. Κάθε κελί είναι υπεύθυνο για το 1/n του εύρους του *score*

HistogramBlooms Structure (2)

- Για κάθε κελί ο κόμβος διατηρεί την εξής πληροφορία:
  - Την άνω και κάτω τιμή , ub[i] και lb[i], που καλύπτει το κελί
  - freq[i] που είναι ο αριθμός των document που πέφτουν στο κελί
  - avg[i], που είναι το μέσο score των document στο κελί
  - filter[i] που αναπαριστά τα document που βρίσκονται στο κελί. Για την υλοποίησή του χρησιμοποιούμε Bloom filters.

Χρήση HistogramBlooms Structure (1)

- Στην πρώτη φάση κάθε κόμβος δίνει στον συντονιστή την τοπική top-k λίστα και ένα μέρος του Bloom filter του.
- missing scores: ο συντονιστής βρίσκει μέσω του Bloom filter σε ποιο κελί ανήκει το docID, έστω c, και το αντικαθιστά με το avg[c].
- missing document: αφού έχει λυθεί το προηγούμενο, τότε υπολογίζει μια πρώτη εκτίμηση του *topKscore*.
- Κάθε κόμβος στέλνει στον συντονιστή μία λίστα με τα υποψήφια του τα οποία έχουν

$$score > \frac{topKscore}{m}$$

Χρήση HistogramBlooms Structure (2)

- Τώρα ο συντονιστής μπορεί να υπολογίσει μια υψηλότερου επιπέδου top-k λύση
- Διαισθητικά, με αυτόν τον τρόπο χρησιμοποιώντας τα HistogramBlooms ο συντονιστής συγκεντρώνει αρκετή πληροφορία σε βάθος λιστών ώστε να μην χρειάζεται να σπαταλήσουμε μεγάλο εύρος ζώνης

Candidate List Filters Matrix Structure (1)

- Επιστρέφοντας στο πρόβλημα των missing document παίρνοντας τις μέσες τιμές προκύπτουν δύο προβλήματα:
  - Εάν λείπουν πολλά document τότε θα έχουμε μια προσέγγιση χαμηλής ποιότητας.
  - Εάν το m είναι μεγάλο τότε θα έχουμε μικρό topKscore/m και θα επιστραφούν πολλά υποψήφια και θα σπαταλήσουμε το εύρος ζώνης.
- Για αυτό μερικές φορές εκτελούμε ένα ακόμα βήμα στο οποίο κάνουμε σμίκρυνση της λίστας των υποψηφίων

Candidate List Filters Matrix Structure (2)

- Η γενική ιδέα είναι να βρούμε τα documents που βρίσκονται σε πιο πολλές από τις λίστες και να στείλουμε μόνο αυτά.
- Αυτά έχουν μεγαλύτερη πιθανότητα να έχουν *totalScore>topKscore*
- Σε αυτή τη φάση κάθε κόμβος βρίσκει τα περιεχόμενα της λίστας των υποψηφίων
- Στη συνέχεια φτιάχνει ένα διάνυσμα bitmap τέτοιο ώστε σε κάθε bit μέσω μιας hash function να έχει αντιστοιχηθεί το κατάλληλο document. Αυτό λέγεται Candidate List Filter ή CLF.

Candidate List Filters Matrix Structure (3)

- Ο συντονιστής από την πρώτη φάση γνωρίζει για κάθε κόμβο πόσα document έχουν

$$score > \frac{topKscore}{m}$$

και στέλνει σε κάθε κόμβο τον μέγιστο αυτόν των αριθμών, έστω b ώστε όλα τα CLF να δημιουργηθούν με το ίδιο μέγεθος b.

- Τελικά ο συντονιστής λαμβάνει από κάθε κόμβο τα CLF και δημιουργεί ένα συγκεντρωτικό πίνακα μεγέθους m×b
- Η γραμμή i αντιστοιχεί στον κόμβο i

Χρήση CLF Matrix

- Ο CLF Matrix από κατασκευής του δείχνει σε κάθε στήλη πόσα document είναι κοινά υποψήφια σε κάθε λίστα
- Έτσι επιλέγουμε ένα κατώφλι για την εμφάνιση των document, το οποίο να είναι στην πλειοψηφία των document, και ζητάμε μόνο αυτά
- Στην όλη διαδικασία πρέπει να εξετάσουμε τα εξής:
  - Να συγκεντρώσουμε την απαραίτητη πληροφορία με χαμηλό κόστος σε εύρος ζώνης
  - Να αποφύγουμε μεγάλο φιλτράρισμα των υποψηφίων για να έχουμε μια καλή ποιότητα δεδομένων και
  - Να μπορούμε να προβλέψουμε εάν χρειάζεται το CLF προτού το φτιάξουμε και το χρησιμοποιήσουμε

Προετοιμασία κόμβων

- Κάθε κόμβος δημιουργεί τα HistogramBlooms του, δεδομένης της index list. Αυτό περιλαμβάνει τόσο την δημιουργία των δεδομένων που σχετίζονται με το ιστόγραμμα όσο και τα filter του κάθε κόμβου
- Για τα filter, δημιουργούμε ένα σύνολο *cell-docId-set,* για κάθε κόμβο και μέσω αυτού τα Bloom filter
- Όλοι οι κόμβοι χρησιμοποιούν τις ίδιες hush function
- Αναμένουμε να έχουμε διαφορετικό μέγεθος κελιών για κάθε κόμβο
- Επειδή αυτό είναι χρονοβόρο, μπορούμε να τα υπολογίσουμε αρχικά και στην συνέχεια να τα αποθηκεύσουμε τοπικά

KLEE

- Ο αλγόριθμος εκτελείται σε τέσσερα βήματα:
  1. Εξερεύνηση: στο βήμα αυτό ο συντονιστής επικοινωνεί με τους m κόμβους με σκοπό να υπολογίσει μια καλή εκτίμηση του *topKscore*, το οποίο με τη σειρά του «δημιουργεί» τις λίστες των υποψηφίων
  2. Βελτιστοποίηση: το βήμα αυτό εκτελείται τοπικά στον συντονιστή ώστε να αποφασίσει εάν χρειάζεται ή όχι η εμπλοκή των CLF

KLEE

3. Αποκοπή υποψηφίων: στο βήμα αυτό, το οποίο εκτελείτε ανάλογα του αποτελέσματος του βήματος 2, έχουμε τον επαναϋπολογισμό των υποψηφίων τα οποία είναι τα documents με ικανό αριθμό 1 στα αντίστοιχα bits στον CLF Matrix
4. Ανάκτηση υποψηφίων: είναι το τελικό βήμα στο οποίο ο συντονιστής παίρνει τις λίστες των υποψηφίων και υπολογίζει το τελικό αποτέλεσμα

- Να σημειωθεί ότι το δεύτερο βήμα γίνεται trade-off μεταξύ του εύρους ζώνης και του αριθμού των φάσεων επικοινωνίας. Γίνεται μια πρόβλεψη για το επικείμενο κέρδος και γίνεται η απόφαση με βάση την μετρική που χρησιμοποιούμε.

Εξερεύνηση (1)

- Στο πρώτο βήμα ο συντονιστής στέλνει ένα request με βάση την αρχική επερώτηση.
- Οι κόμβοι απαντούν ως εξής:
  - Στέλνουν την top-k λίστα
  - Για κάθε κελί από αυτά που ανήκουν στα “high-end”, δηλαδή αυτά που έχουν τα documents με τα πιο μεγάλα score στέλνουν το ιστόγραμμα(ub[i], lb[i], freq[i], avg[i] and filter[i]) i=1…c
  - Για τα υπόλοιπα n-c το avg[i] και το freg[i]

Εξερεύνηση (2)

- Ο συντονιστής υπολογίζει την top-k λίστα ως εξής:
  - Όταν το $docID_i$ λείπει για από κάποια λίστα I_j, τότε ψάξε στα filter[r],r=1…c εάν υπάρχει το $docID_i$. Η αναζήτηση σταματάει όταν βρεθεί το document ή όταν εξαντληθούν όλα τα φίλτρα.
  - Εάν βρεθεί, έστω στο filter[k], τότε χρησιμοποιούμε το avg[k] για να συμπληρώσουμε την τιμή που λείπει
  - Διαφορετικά χρησιμοποιούμε τον μέσο όρο με βάρη από τα freq[i], avg[i] i=c+1…n για να υπολογίσουμε την τιμή
  - Η διαδικασία εκτελείται για όλα το docID που λείπουν και για όλους τους κόμβους.

Εξερεύνηση (3)

- Αφού συμπληρώσουμε όλα τα score που λείπουν τότε υπολογίζουμε μια εκτίμηση τις top-k λίστας και το *topKscore*.
- Αυτόματα έχουμε και τις λίστες των υποψηφίων που είναι όσα έχουν

$$score > \frac{topKscore}{m}$$

Βελτιστοποίηση (1)

- Είναι το δεύτερο βήμα του αλγόριθμου και εκτελείται τοπικά στον συντονιστή
- Ο ρόλος του είναι να υπολογίσει τα αναμενόμενα οφέλη στο εύρος ζώνης εάν εμπλέξουμε την φάση μείωσης των υποψηφίων
- Έστω d το μέσο μήκος των υποψηφίων λιστών των κόμβων
- Η πιθανότητα ένα bit ενός κόμβου να είναι 1 είναι $P_i = 1/lf$ όπου lf είναι ο load factor του Bloom filter του κόμβου i
- lf =b/d όπου b το CLF του κόμβου

Βελτιστοποίηση (2)

- Η πιθανότητα μία στήλη να έχει R bit 1 δίνεται από την διωνυμική κατανομή:

$$\Pr = \sum_{i=R}^{m} \binom{m}{i} \times \left(\frac{1}{lf}\right)^{i} \times \left(\frac{lf-1}{lf}\right)^{m-i}$$

- Το κόστος σε εύρος ζώνης χωρίς CLF είναι:

$$C = m \times b$$

- Το κόστος σε εύρος ζώνης με CLF είναι τα άθροισμα να στείλουμε τα CLF, C_1 , και τα τελικά ζευγάρια *(docId, score),* C_2

Βελτιστοποίηση (3)

- Για το C_1 :

$$C = \frac{m \times b}{8}$$

- Για το C_2 :

$$C = \mathrm{Pr} \times \mathrm{b} \times \mathrm{m}$$

- Αφού P_r η πιθανότητα το bit να ανήκει σε στήλη με R bits 1
- b το μήκος του CLF κάθε κόμβου και
- m το σύνολο των κόμβων

Βελτιστοποίηση (4)

- Βλέπουμε ότι τα κόστη στις δύο περιπτώσεις διαφέρουν κατά παράγοντα P_r
- Άρα το P_r καθορίζει το εάν θα έχουμε 3 ή 4 βήματα
- Τα πραγματικά κόστη πρέπει να πολλαπλασιαστούν με τον μέσο αριθμό των bytes που χρειάζονται για κάθε ζευγάρι *(docId, score)*
- Τέλος πρέπει να προσθέσουμε και το C_1

Αποκοπή υποψηφίων (1)

- Αυτό το βήμα εκτελείται εάν κριθεί απαραίτητο από το προηγούμενο βήμα
- Αφορά την κατασκευή και χρήση των CLF καθώς και την βελτίωση των αποτελεσμάτων και την αντιμετώπιση του missing documents
- Εκτελείται σε 5 φάσεις:

  1.Ο συντονιστής επαναπροσδιορίζει της λίστες υποψηφίων ως τα documents τα οποία δεν έχει στείλει ο κόμβος στον συντονιστή και έχουν *score* που είναι μεγαλύτερο από το ελάχιστο *score* του κελιού με τιμή *topKscore/m*

Αποκοπή υποψηφίων (2)

2.Ο συντονιστής υπολογίζει το μέγεθος τις λίστας υποψηφίων για κάθε κόμβο, βασιζόμενος στις πληροφορίες από το πρώτο βήμα, και το μέγιστο τους το *max_size_candidate_list*. Στη συνέχεια

*α.*Ο συντονιστής στέλνει σε κάθε κόμβο την λίστα υποψηφίων και το *max_size_candidate_list.*

β. Κάθε κόμβος στέλνει:

i.το CLF. Για τον υπολογισμό χρησιμοποιεί μια hash function και μέγεθος bitmap *b=lf* max_size_candidate_list* με *lf* αρκετά μεγάλο ώστε να έχουμε μικρή πιθανότητα για false positive

ii. Τα αληθινά *score* για τα document στο top-k

Αποκοπή υποψηφίων (3)

3.Ο συντονιστής κατασκευάζει τον CLF Matrix όπως αναφέρθηκε προηγουμένως

4. Επισημαίνονται οι ενδιαφέρουσες στήλες (R bits 1)

5. Τέλος ο συντονιστής υπολογίζει το σύνολο τον υποψηφίων που αποτελείται από τα documents τα οποία είναι υποσύνολο της αρχικής λίστας υποψηφίων και τα *docID's* κάνουν hash στις ενδιαφέρουσες στήλες του CLF για τον κόμβο i

Πιθανό πρόβλημα: η αντιστοίχηση 2 document ίδιο bit (false positive) και αποστολή περισσότερης πληροφορίας στον συντονιστή

Ανάκτηση υποψηφίων

- Είναι το τελικό βήμα
- Ο συντονιστής ζητάει από τους κόμβους να του στείλουν τα υποψήφια *docId's* όπως καθορίστηκαν στο βήμα 1 ή στο 3
- Ο συντονιστής υπολογίζει το τελικό αποτέλεσμα βασιζόμενος στις λίστες που έλαβε

Παράμετροι του KLEE (1)

- Οι κύριοι παράμετροι που χαρακτηρίζουν την λειτουργικότητα του KLEE είναι:
  - Ο αριθμός c, των κελιών για τα οποία στέλνονται φίλτρα κατά την πρώτη φάση και
  - Ο αριθμός R των bits που χρειάζονται ώστε να θεωρηθεί ενδιαφέρουσα η στήλη του CLF στο τρίτο βήμα
- Άλλες παράμετροι:
  - Ο load factor για τα Bloom filters
  - Ο αριθμός των hash function
- Αυτές επηρεάζουν την πιθανότητα να έχουμε false positive και την κρατούν κάτω από ένα κατώφλι δεδομένου ενός πλήθους από entries

Παράμετροι του KLEE (2)

- Μια καλή επιλογή του c εξαρτάται από την αντισυμμετρικότητα της κατανομής των *score*
- Εφαρμόζεται μια τεχνική που περιορίζει το λάθος για την πρόβλεψη του *score* που γίνεται παίρνοντας ένα μέρος του ιστογράμματος
- Ο ορισμός του R μπορεί να είναι επιρρεπής σε λάθη
- Αντί απλά να αθροίζαμε bit θα μπορούσαμε να εκμεταλλευτούμε τα ιστογράμματα κάθε κόμβου και να πολλαπλασιάσουμε με πληροφορία όπως την μέση τιμή των documents τα οποία δεν έχουν σταλεί στον συντονιστή

Πειραματικά Δεδομένα (1)

- Οι αλγόριθμοι υλοποιήθηκαν σε Java
- Τα δεδομένα ήταν αποθηκευμένα τοπικά σε κάθε κόμβο
- Τα πειράματα έγιναν σε υπολογιστές με 3GHz Pentium επεξεργαστές
- Πραγματικά δεδομένα
  - GOV, IMDB
- Συνθετικά δεδομένα
  - Δημιουργήθηκαν index lists που ακολουθούν τον νόμο του Zipf αλλάζοντας το θ
  - «πειράχτηκαν» η ΒΔ Gov αλλάζοντας τα *score*

Πειραματικά Δεδομένα (2)

- Υλοποιήθηκαν οι αλγόριθμοι
  - DTA
  - TPUT
  - X-TPUT
  - KLEE-3
  - KLEE-4
- Αποδείχτηκε ότι ο KLEE υπερτερεί των υπολοίπων σε όλους τους τομείς

Συμπεράσματα

- Ο KLEE είναι ένας κατανεμημένος αλγόριθμος που εισήγαγε καινοτομίες
  - Είναι ένας ισχυρός αλγόριθμος για κατανεμημένα συστήματα
  - Κέρδος σε αποδοτικότητα εάν επιτρέπουμε μικρά λάθη στην «ποιότητα» των αποτελεσμάτων
  - Είναι εξοπλισμένος με διάφορες ρυθμιστικές παραμέτρους
  - Είναι ευέλικτος
    - Επιτρέπει το trade-off μεταξύ ποιότητας και απόδοσης και μεταξύ εύρους και βημάτων επικοινωνίας

Τέλος!!!

- Ερωτήσεις???