

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΪΚΑ ΜΑΘΗΜΑΤΑ

ανοικτάμαθήματα
opencourses

Ανάκτηση Πληροφορίας

Κατασκευή Ευρετηρίου

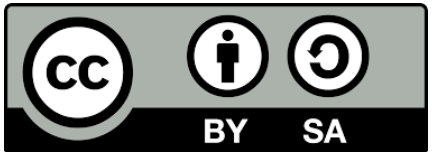
Διδάσκων: Καθ. Ε. Πιτουρά

CC BY SA

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Αδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Introduction to

Information Retrieval

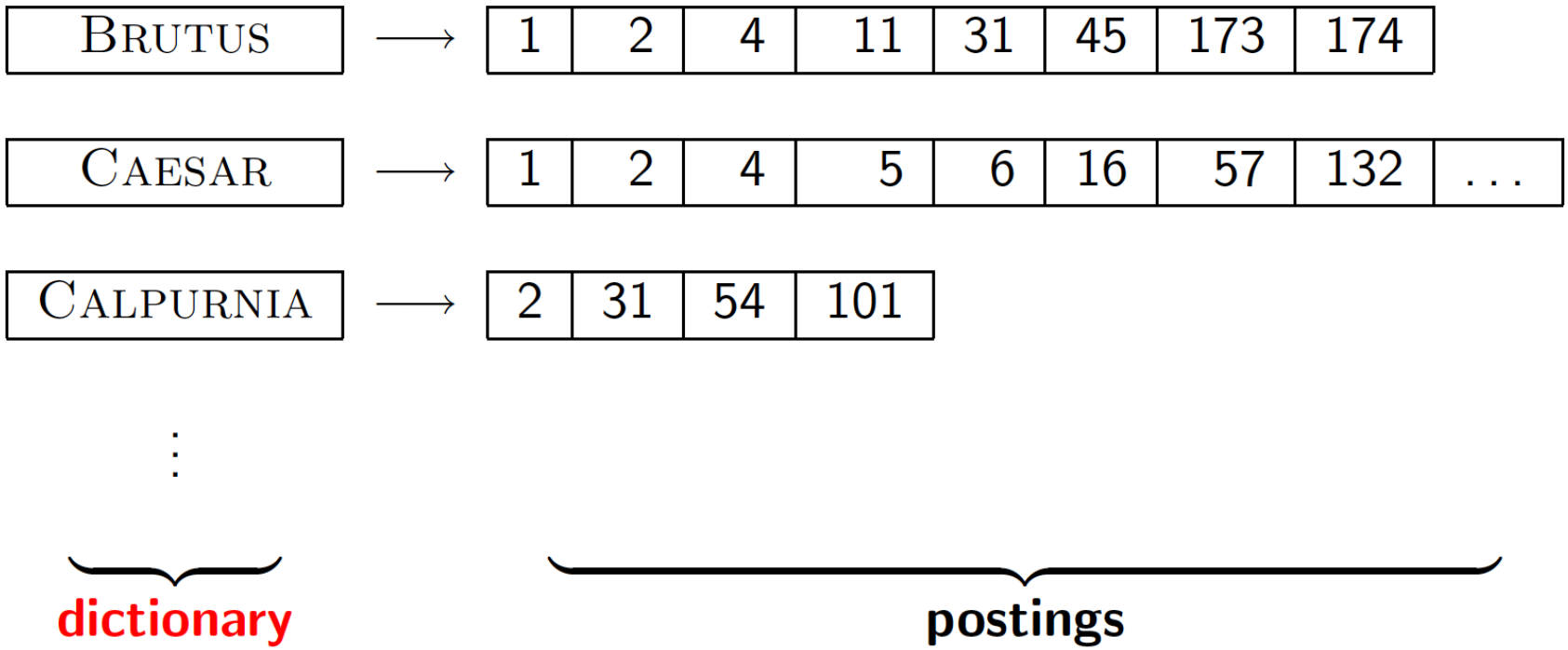
ΠΛΕ70: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Διάλεξη 5: Κατασκευή Ευρετηρίου. Στατιστικά Συλλογής.

ΚΑΤΑΣΚΕΥΗ ΕΥΡΕΤΗΡΙΟΥ

Η βασική δομή: Το αντεστραμμένο ευρετήριο (inverted index)



Λεξικό: οι όροι (term) και η συχνότητα εγγράφων (#εγγράφων της συλλογής που εμφανίζονται).

Λίστες καταχωρίσεων (posting lists) Κάθε καταχώριση (posting) για ένα όρο περιέχει μια διατεταγμένη λίστα με τα έγγραφα (DocID) στα οποία εμφανίζεται ο όρος – συχνά επιπρόσθετα στοιχεία , όπως position, term frequency, κλπ.

Κατασκευή ευρετηρίου

- Πως κατασκευάζουμε το ευρετήριο;
- Ποιες στρατηγικές χρησιμοποιούμε όταν έχουμε περιορισμένη μνήμη;

Βασικά στοιχεία του υλικού

- Πολλές αποφάσεις στην ανάκτηση πληροφορίας βασίζονται στα χαρακτηριστικά του υλικού.
- Ας δούμε μερικά βασικά χαρακτηριστικά.

Βασικά χαρακτηριστικά του υλικού

- Η προσπέλαση δεδομένων στην κύρια μνήμη είναι πολύ *πιο γρήγορη* από την προσπέλαση δεδομένων στο δίσκο (περίπου μια τάξη μεγέθους – δηλαδή x10)
- Disk seeks (χρόνος αναζήτησης): Ενώ τοποθετείται η κεφαλή δεν γίνεται μεταφορά δεδομένων
  - Αρα: Η μεταφορά μεγάλων κομματιών (chunk) δεδομένων από το δίσκο στη μνήμη είναι γρηγορότερη από τη μεταφορά πολλών μικρών
- Η επικοινωνία με το δίσκο (Disk I/O) γίνεται σε σελίδες (block-based): Διαβάζονται και γράφονται ολόκληρα blocks (όχι τμήματά τους). Σχετικός χώρος στη μνήμη – buffer
- Παράλληλα με την επεξεργασία δεδομένων
- Μέγεθος Block: 8KB - 256 KB.

Βασικά χαρακτηριστικά του υλικού

- Οι επεξεργαστές που χρησιμοποιούνται στην ΑΠ διαθέτουν πολλά GB κύριας μνήμης, συχνά δεκάδες από GBs.
- Ο διαθέσιμος χώρος δίσκου είναι πολλές (2–3) τάξεις μεγέθους μεγαλύτερος.
- Η ανοχή στα σφάλματα (Fault tolerance) είναι πολύ ακριβή: φθηνότερο να χρησιμοποιεί κανείς πολλές κανονικές μηχανές παρά μια «μεγάλη».

Η συλλογή RCV1

- Η συλλογή με τα άπαντα του Shakespeare δεν είναι αρκετά μεγάλη για το σκοπό της σημερινής διάλεξης.
- Η συλλογή που θα χρησιμοποιήσουμε δεν είναι στην πραγματικότητα πολύ μεγάλη, αλλά είναι διαθέσιμη στο κοινό.
- Θα χρησιμοποιήσουμε τη συλλογή RCV1.
  - Είναι ένας χρόνος του κυκλώματος ειδήσεων του Reuters (Reuters newswire) (μέρος του 1995 και 1996)
  - 1GB κειμένου.

Ενα έγγραφο της συλλογής Reuters RCV1

REUTERS

You are here: Home > News > Science > Article

Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly Enoug

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

Email This Article | Print This Article | Reprints

[-] Text [+]



SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Ενα έγγραφο της συλλογής Reuters RCV1

Symbol	Statistic	Value
N	documents	800,000
L_{ave}	avg. # tokens per document	200
M	terms	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
T	tokens	100,000,000

Κατασκευή ευρετηρίου

Επεξεργαζόμαστε τα έγγραφα για να βρούμε τις λέξεις - αυτές αποθηκεύονται μαζί με το Document ID.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious



Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Βασικό βήμα: sort

- Αφού έχουμε επεξεργαστεί όλα τα έγγραφα, το αντεστραμμένο ευρετήριο ταξινομείται με βάση τους όρους.

Θα επικεντρωθούμε στο βήμα ταξινόμησης. Πρέπει να ταξινομήσουμε 100M λέξεις.

Στη συνέχεια, για κάθε όρο, ταξινόμηση εγγράφων.

Χρήση termID αντί term.

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Κλιμάκωση της κατασκευής του ευρετηρίου

- Δεν είναι δυνατή η πλήρης κατασκευή του ευρετηρίου στη μνήμη (in-memory)
  - Δεν μπορούμε να φορτώσουμε όλη τη συλλογή στη μνήμη, να την ταξινομήσουμε και να γράψουμε το ευρετήριο πίσω στο δίσκο.
- Πως μπορούμε να κατασκευάσουμε ένα ευρετήριο για μια πολύ μεγάλη συλλογή;
  - Λαμβάνοντας υπ' όψιν τα περιορισμούς και τα χαρακτηριστικά του υλικού. . .

Κατασκευή με βάση την ταξινόμηση

- Καθώς κατασκευάζουμε το ευρετήριο, επεξεργαζόμαστε τα έγγραφα ένα-ένα
- Οι τελικές καταχωρίσεις για κάθε όρο είναι ημιτελείς μέχρι το τέλος.

Μπορούμε να κρατάμε όλο το ευρετήριο στη μνήμη;

- Κάθε εγγραφή καταχώρισης (ακόμα και χωρίς πληροφορία θέσης - non-positional) δηλαδή *(term, doc, freq)* καταλαμβάνει 4+4+4 = 12 bytes και απαιτεί πολύ χώρο για μεγάλες συλλογές
- T = 100,000,000 όροι για το RCV1
  - Αυτή η συλλογή χωράει στη μνήμη, αλλά στην πραγματικότητα πολύ μεγαλύτερες, Π.χ., οι *New York Times παρέχουν ένα ευρετήριο για κύκλωμα ειδήσεων* >150 χρόνια
- Πρέπει να αποθηκεύουμε ενδιάμεσα αποτελέσματα στο δίσκο.

Ταξινόμηση χρησιμοποιώντας το δίσκο σαν «μνήμη»;

- Μπορούμε να χρησιμοποιήσουμε *τον ίδιο αλγόριθμο* κατασκευής για το ευρετήριο αλλά χρησιμοποιώντας δίσκο αντί για μνήμη;

- Οχι: Ταξινόμηση T = 100,000,000 εγγραφών στο δίσκο είναι πολύ αργή – πολλές τυχαίες ανακτήσεις (disk seeks).
- Χρειαζόμαστε έναν αλγόριθμο *εξωτερικής ταξινόμησης (external sorting)*.

Γιατί όχι;

- Διάσχιση του εγγράφου και κατασκευή εγγραφών καταχωρίσεων για ένα έγγραφο τη φορά.

Μετά ταξινόμηση των εγγραφών με βάση τους όρους (και μετά, για κάθε όρο, ταξινόμηση καταχωρίσεων με βάση το έγγραφο).

Αυτή η διαδικασία με τυχαία ανάκτηση στο δίσκο θα ήταν πολύ αργή – ταξινόμηση T=100M εγγραφών.

Αν κάθε σύγκριση χρειάζεται 2 προσπελάσεις στο δίσκο, και για την ταξινόμηση N στοιχείων χρειαζόμαστε N $\log_2 N$ συγκρίσεις, πόσο χρόνο θα χρειαζόμασταν;

BSBI: Αλγόριθμος κατασκευής κατά block (Blocked sort-based Indexing)

1. Χώρισε τη συλλογή σε κομμάτια ίσου μεγέθους.
2. Ταξινόμησε τα ζεύγη termID–docID για κάθε κομμάτι στη μνήμη.
3. Αποθήκευσε τα ενδιάμεσα αποτελέσματα στο δίσκο.
4. Συγχώνευσε τα ενδιάμεσα αποτελέσματα.

BSBI: Αλγόριθμος κατασκευής κατά block (Blocked sort-based Indexing)

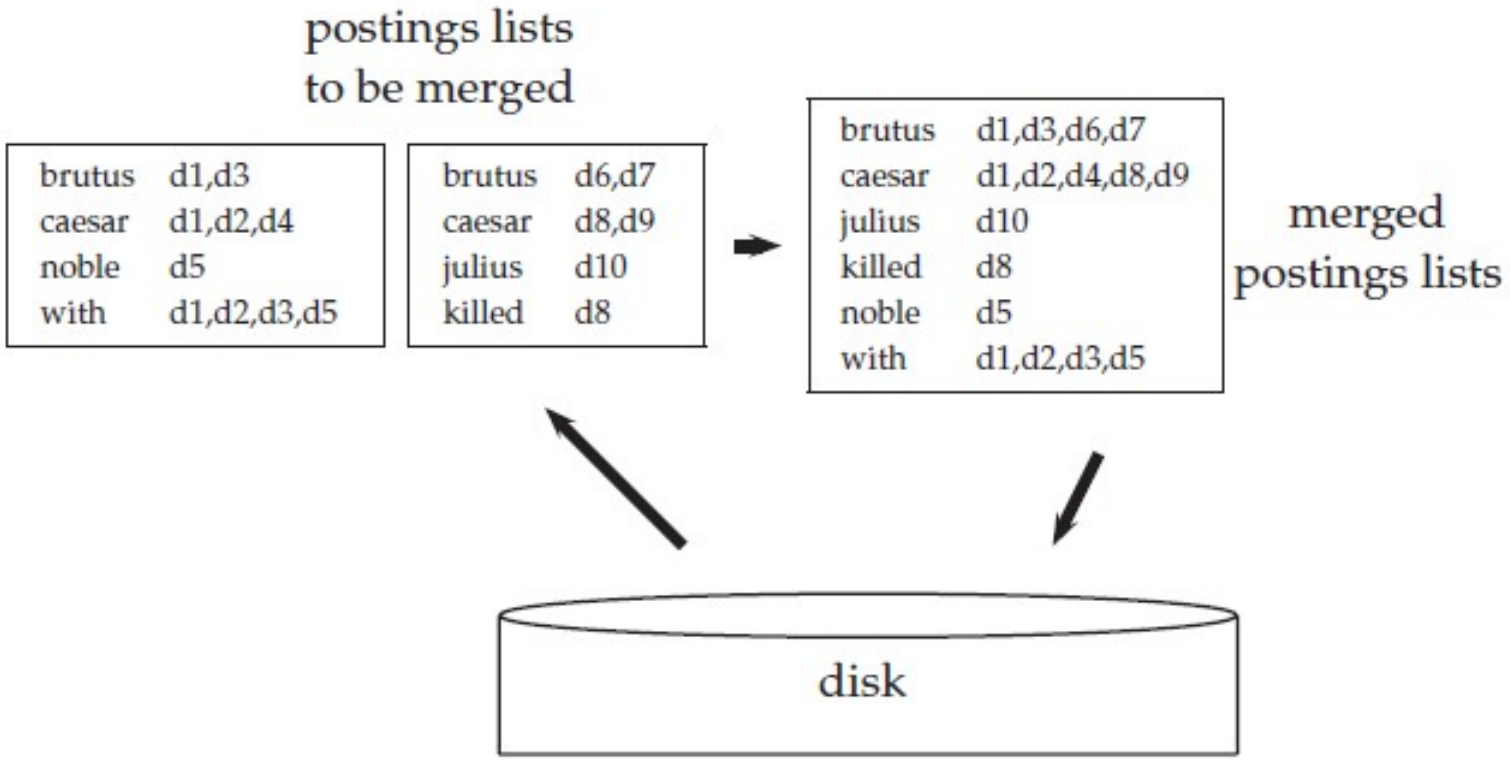
- Εγγραφές 12-byte (4+4+4) *(term, doc, freq).*
- Παράγονται κατά τη διάσχιση των εγγράφων
- Ταξινόμηση 100M τέτοιων 12-byte εγγραφών με βάση τον όρο.
- Ορίζουμε ένα Block ~ 10M τέτοιες εγγραφές
  - Μπορούμε εύκολα να έχουμε κάποια από αυτά στη μνήμη.
  - Αρχικά, 10 τέτοια blocks.
- Βασική ιδέα:
  - Συγκέντρωσε καταχωρίσεις για να γεμίσει ένα block, ταξινόμησε τις καταχωρίσεις σε κάθε block, γράψε το στο δίσκο. (run)
  - Μετά συγχώνευσε τα blocks σε ένα μεγάλο ταξινομημένο block.

Ταξινόμηση 10 blocks των 10M εγγραφών

```
BSBINDEXCONSTRUCTION()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4      block ← PARSENEXTBLOCK()
5      BSBI-INVERT(block)
6      WRITEBLOCKTODISK(block, f_n)
7  MERGEBLOCKS(f_1, ..., f_n; f_merged)
```

Διάβασε ένα-ένα τα έγγραφα – γεμίζοντας ένα block με <term, docid>, ταξινόμησε το block, γράψε το γεμάτο block στο δίσκο.

Παράδειγμα

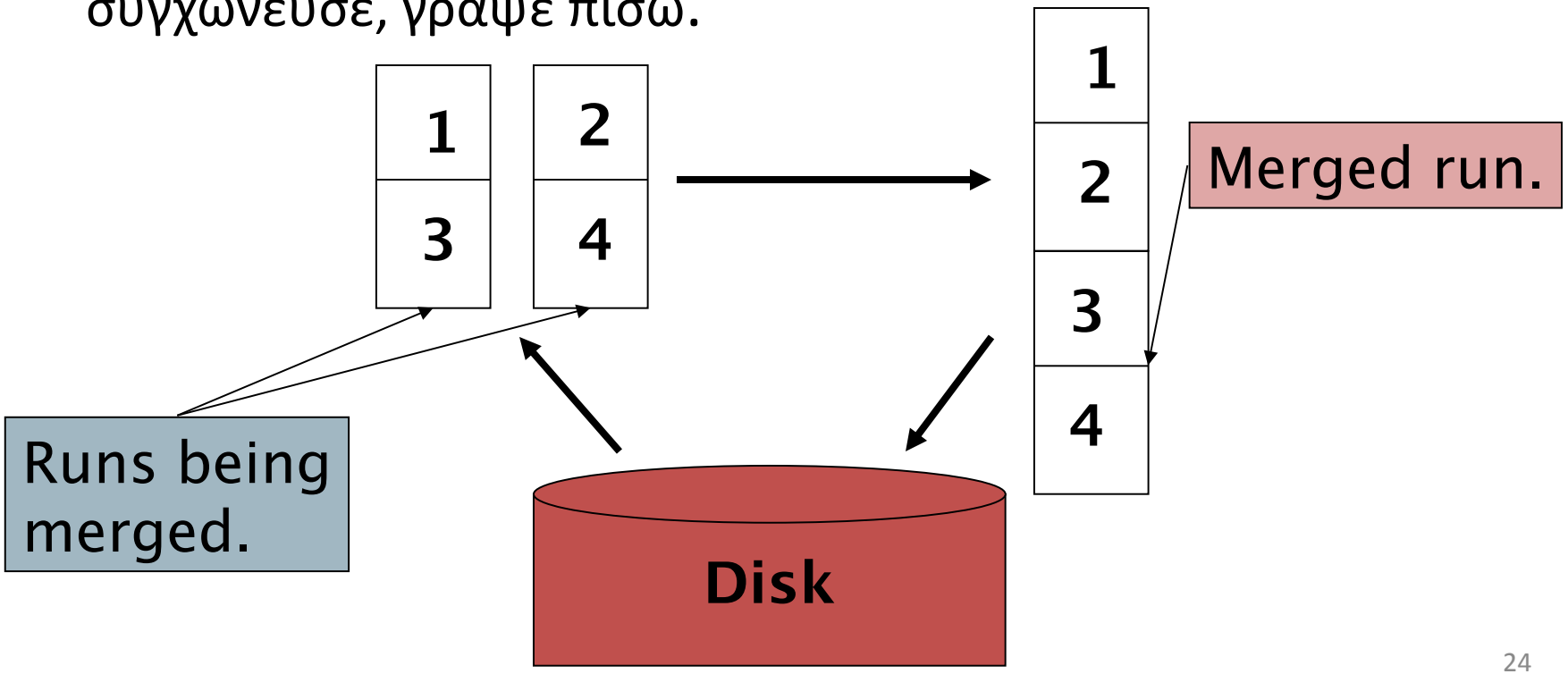

postings lists
to be merged
brutus d1,d3
caesar d1,d2,d4
noble d5
with d1,d2,d3,d5
brutus d6,d7
caesar d8,d9
julius d10
killed d8
brutus d1,d3,d6,d7
caesar d1,d2,d4,d8,d9
julius d10
killed d8
noble d5
with d1,d2,d3,d5
merged
postings lists
disk

Ταξινόμηση 10 blocks των 10M εγγραφών

- Πρώτα, διάβασε κάθε block και ταξινόμησε τις εγγραφές του:
  - Quicksort O(N*$\log_2 N$) αναμενόμενα βήματα
  - Στην περίπτωσή μας, 10M*$\log_2$10M steps
- 10 φορές αυτή η εκτίμηση του χρόνου μας δίνει 10 διατεταγμένα *runs* των 10M εγγραφών το καθένα.
- Ο απλός τρόπος χρειάζεται 2 αντίγραφα των δεδομένων στο δίσκο
  - Αλλά μπορεί να βελτιωθεί.

Πως θα γίνει η συγχώνευση των runs?

- Δυαδική συγχώνευση, μια δεντρική δομή με $\log_2 10 = 4$ επίπεδα.
- Σε κάθε επίπεδο, διάβασε στη μνήμη runs σε blocks των 10M, συγχώνευσε, γράψε πίσω.



Πως θα γίνει η συγχώνευση των runs?

- Πιο αποδοτικά με μια multi-way συγχώνευση, όπου διαβάζουμε από όλα τα blocks ταυτόχρονα.
- Υπό την προϋπόθεση ότι διαβάζουμε στη μνήμη αρκετά μεγάλα κομμάτια κάθε block και μετά γράφουμε πίσω αρκετά μεγάλα κομμάτια, αλλιώς πάλι πρόβλημα με τις αναζητήσεις στο δίσκο.

BSBI: περίληψη

- Βασική ιδέα:
  - Διάβαζε τα έγγραφα, συγκέντρωσε <term, docid> καταχωρίσεις έως να γεμίσει ένα block, ταξινόμησε τις καταχωρίσεις σε κάθε block, γράψε το στο δίσκο.
  - Μετά συγχώνευσε τα blocks σε ένα μεγάλο διατεταγμένο block.
- Δυαδική συγχώνευση, μια δεντρική δομή με $\log_2 B$ επίπεδα, όπου B ο αριθμός των blocks.

Παρατήρηση: μπορούμε να εργαστούμε με termid αντί για term αν κρατάμε το λεξικό (την απεικόνιση term, termid) στη μνήμη.

Χρήση αναγνωριστικού όρου (termID)

- *Υπόθεση: κρατάμε το λεξικό στη μνήμη*
- Χρειαζόμαστε το λεξικό (το οποίο μεγαλώνει δυναμικά) για να υλοποιήσουμε την απεικόνιση μεταξύ όρου (term) σε termID.
- Θα μπορούσαμε να εργαστούμε και με term, docID καταχωρίσεις αντί των termID, docID καταχωρίσεων, αλλά τα ενδιάμεσα αρχεία γίνονται πολύ μεγάλα.

SPIMI: Single-pass in-memory indexing (ευρετηριοποίηση ενός περάσματος)

Αν δεν διατηρούμε term-termID απεικονίσεις μεταξύ blocks.

Εναλλακτικός αλγόριθμος: Αποφυγή της ταξινόμησης των όρων.

- Συγκεντρώνουμε τις καταχωρίσεις σε λίστες καταχωρίσεων όπως αυτές εμφανίζονται.
- Κατασκευή ενός πλήρους αντεστραμμένου ευρετηρίου για κάθε block. Χρησιμοποιούμε κατακερματισμό (hash) ώστε οι καταχωρίσεις του ίδιου όρου στον ίδιο κάδο.
- Μετά συγχωνεύουμε τα ξεχωριστά ευρετήρια σε ένα μεγάλο.

SPIMI-Invert

```
SPIMI-Invert(token_stream)
 1  output_file = NewFile()
 2  dictionary = NewHash()
 3  while (free memory available)
 4  do token ← next(token_stream)
 5     if term(token) ∉ dictionary
 6        then postings_list = AddToDictionary(dictionary, term(token))
 7        else postings_list = GetPostingsList(dictionary, term(token))
 8     if full(postings_list)
 9        then postings_list = DoublePostingsList(dictionary, term(token))
10     AddToPostingsList(postings_list, docID(token))
11  sorted_terms ← SortTerms(dictionary)
12  WriteBlockToDisk(sorted_terms, dictionary, output_file)
13  return output_file
```

Χρησιμοποιούμε κατακερματισμό ώστε οι καταχωρίσεις για τον ίδιο όρο να είναι στον ίδιο «κάδο»

- Η συγχώνευση όπως και στο BSBI.

Δυναμικά ευρετήρια

- Μέχρι στιγμής, θεωρήσαμε ότι τα ευρετήρια είναι στατικά.
- Αυτό συμβαίνει σπάνια, στην πραγματικότητα:
  - Νέα έγγραφα εμφανίζονται και πρέπει να ευρετηριοποιηθούν
  - Εγγραφα τροποποιούνται ή διαγράφονται
- Αυτό σημαίνει ότι *πρέπει να ενημερώσουμε τις λίστες καταχωρίσεων*:
  - Αλλαγές στις καταχωρίσεις όρων που είναι ήδη στο λεξικό
  - Προσθήκη νέων όρων στο λεξικό

Μια απλή προσέγγιση

- Διατήρησε ένα «μεγάλο» κεντρικό ευρετήριο.
- Τα νέα έγγραφα σε μικρό «βοηθητικό» ευρετήριο (auxiliary index) (στη μνήμη).
- Ψάξε και στα δύο, συγχώνευσε το αποτέλεσμα.
- Διαγραφές
  - Invalidation bit-vector για τα διαγραμμένα έγγραφα
  - Φιλτράρισμα αποτελεσμάτων ώστε να αποκλειστούν τα διαγραμμένα.
- Περιοδικά, re-index το βοηθητικό στο κυρίως ευρετήριο.

Θέματα

- Συχνές συγχωνεύσεις
- Κακή απόδοση κατά τη διάρκεια της συγχώνευσης
- Πιο αποδοτικό αν κάθε λίστα καταχωρίσεων ήταν αποθηκευμένη σε διαφορετικό αρχείο (τότε, απλώς append), αλλά θα χρειαζόμαστε πολλά αρχεία (μη αποδοτικό για το Λειτουργικό Σύστημα)

- Θα υποθέσουμε ότι όλο το ευρετήριο σε ένα αρχείο.
- Στην πραγματικότητα: Κάτι ανάμεσα (π.χ., πολλές μικρές λίστες καταχώρισης σε ένα αρχείο, διάσπαση πολύ μεγάλων λιστών, κλπ).

Λογαριθμική συγχώνευση

- Διατήρηση μια σειράς από ευρετήρια, το καθένα διπλάσιου μεγέθους από τα προηγούμενο
  - Κάθε στιγμή, υπάρχουν κάποια από αυτά
- Εστω n ο αριθμός των postings στη μνήμη
- Διατηρούμε στο δίσκο ευρετήρια I_0, I_1, …

I_0 μεγέθους 2^0*n, I_1 μεγέθους 2^1*n, I_2 μεγέθους 2^2*n, …

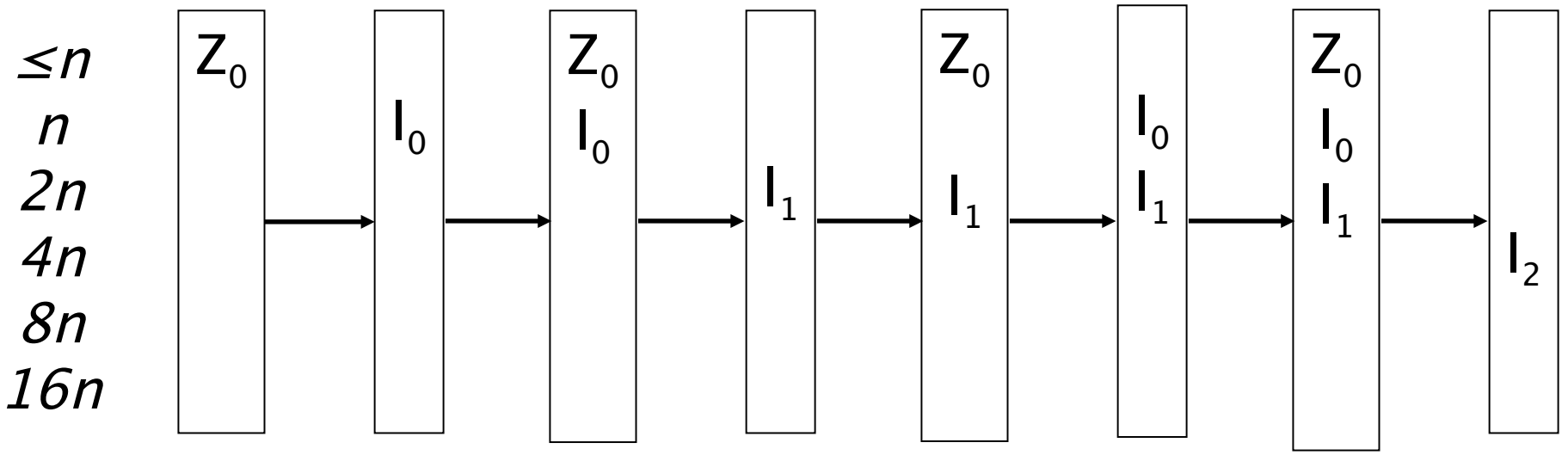
- Ενα βοηθητικό ευρετήριο Z_0 μεγέθους n στη μνήμη

Λογαριθμική συγχώνευση

- Οταν φτάσει το όριο n στη μνήμη, τα 2^0*n postings του Z_0 μεταφέρονται στο δίσκο ως ένα νέο index I_0
- ή αν το I_0 υπάρχει, το Z_0 συγχωνεύεται με το I_0 στο Z_1
- το οποίο αποθηκεύεται ως I_1 (αν δεν υπάρχει ήδη I_1) ή συγχωνεύεται με το I_1 ως Z_2 κοκ.

- Τα ερωτήματα απαντώνται με χρήση του Z_0 στη μνήμη και όσων I_i υπάρχουν στο δίσκο κάθε φορά.

Logarithmic merge in action



```
LMERGEADDTOKEN(indexes, Z_0, token)
 1  Z_0 ← MERGE(Z_0, {token})
 2  if |Z_0| = n
 3     then for i ← 0 to ∞
 4          do if I_i ∈ indexes
 5               then Z_{i+1} ← MERGE(I_i, Z_i)
 6                    (Z_{i+1} is a temporary index on disk.)
 7                    indexes ← indexes − {I_i}
 8               else I_i ← Z_i    (Z_i becomes the permanent index I_i.)
 9                    indexes ← indexes ∪ {I_i}
10                    BREAK
11          Z_0 ← ∅
```

```
LOGARITHMICMERGE()
1  Z_0 ← ∅    (Z_0 is the in-memory index.)
2  indexes ← ∅
3  while true
4  do LMERGEADDTOKEN(indexes, Z_0, GETNEXTTOKEN())
```

Πολυπλοκότητες

Κατασκευή

- Κυρίως και βοηθητικό ευρετήριο: T/n συγχωνεύσεις, σε κάθε μία κοιτάμε όλους τους όρους, άρα πολυπλοκότητα $O(T^2)$.
- Λογαριθμική συγχώνευση: κάθε καταχώριση συγχωνεύεται O(log T)φορές, άρα πολυπλοκότητα O(T log T).

Ερώτημα

- Κυρίως και βοηθητικό ευρετήριο: O(1)
- Λογαριθμική συγχώνευση: κοιτάμε O(log T) ευρετήρια.

Δυναμικά ευρετήρια στις μηχανές αναζήτησης

- Πολύ συχνές αλλαγές
- Συχνά περιοδική *ανακατασκευή του ευρετηρίου από την αρχή*
  - Ενώ κατασκευάζεται το νέο, χρησιμοποιείται το παλιό και όταν η κατασκευή τελειώσει, χρήση του νέου.

Αλλα θέματα

- Λίστες δικαιωμάτων προσπέλασης (Access Control Lists ACLs)
  - Για κάθε χρήστη, μια λίστα καταχωρίσεων με τα έγγραφα που μπορεί να προσπελάσει.

Κατανεμημένη κατασκευή

- Για ευρετήριο κλίμακας web

  Χρήση κατανεμημένου cluster
- Επειδή μια μηχανή είναι επιρρεπής σε αποτυχία
  - Μπορεί απροσδόκητα να γίνει αργή ή να αποτύχει
- Χρησιμοποίηση πολλών μηχανών.

Μερικοί αριθμοί

- The Indexed Web contains at least 1.71 billion pages (Sunday, 16 March, 2014).
- Each year, Google changes its search algorithm around 500–600 times http://moz.com/google-algorithm-change.

Web search engine data centers

- Οι μηχανές αναζήτησης χρησιμοποιούν data centers (Google, Bing, Baidu) κυρίως από commodity μηχανές. *Γιατί; (fault tolerance)*
- Τα κέντρα είναι διάσπαρτα σε όλο τον κόσμο.
- Εκτίμηση: Google ~1 million servers, 3 million processors/cores (Gartner 2007).

http://www.google.com/insidesearch/howsearchworks/thestory/

Θα το δούμε αναλυτικά σε επόμενα μαθήματα
Λίγα «εγκυκλοπαιδικά» για το MapReduce και τη χρήση του στην κατασκευή του ευρετηρίου.

Google index

- index partitioned by document IDs into pieces called shards.
- each shard is replicated onto multiple servers.
- initially, from hard disk drives, now enough servers to keep a copy of the whole index in main memory.

- In June 2010, Caffeine continuously crawl and incrementally update the search index.
- Index separated into several layers, some updated faster than the others.

Μια ματιά στα πολύ μεγάλης κλίμακας ευρετήρια

Παράλληλη κατασκευή

- Maintain a *master* machine directing the indexing job – considered "safe".
- Break up indexing into sets of (parallel) tasks.

- Master machine assigns each task to an idle machine from a pool.

Parallel tasks

- We will use two sets of parallel tasks
  - Parsers
  - Inverters

- Break the input document collection into *splits*
- Each split is a subset of documents (corresponding to blocks in BSBI/SPIMI).

Parsers

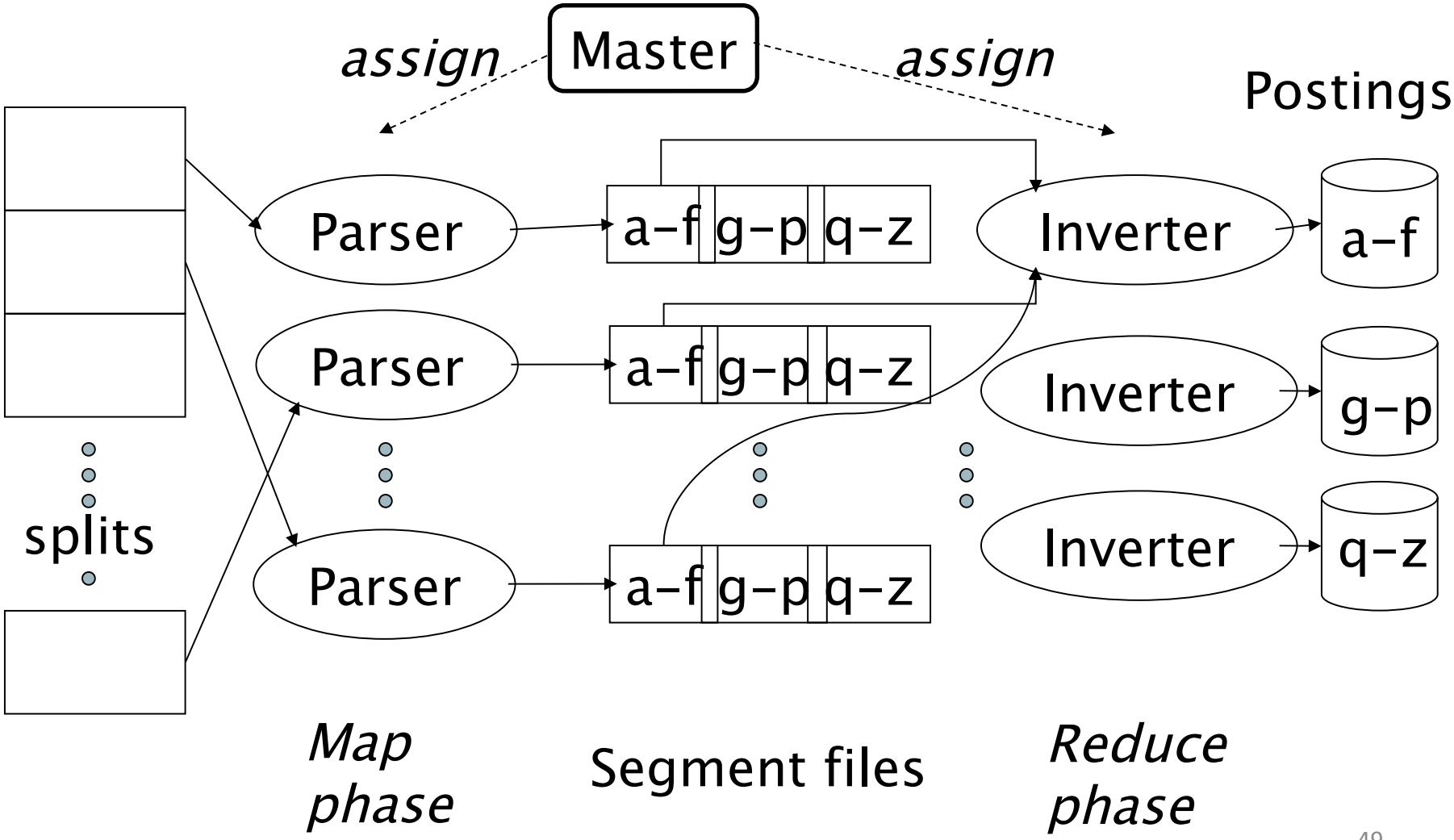
- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into *j* partitions
  - Each partition is for a range of terms' first letters

(e.g., ***a-f, g-p, q-z***) – here *j* = 3.

Inverters

- An inverter collects all (term, doc) pairs (= postings) for one term-partition.
- Sorts and writes to postings lists.

Data flow


assign
Master
assign
Postings
Parser
Parser
Parser
a–f g–p q–z
a–f g–p q–z
a–f g–p q–z
Inverter
Inverter
Inverter
a–f
g–p
q–z
splits
Map phase
Segment files
Reduce phase

MapReduce

- The index construction algorithm we just described is an instance of *MapReduce*.
- MapReduce (Dean and Ghemawat 2004) is a robust and conceptually simple framework for distributed computing without having to write code for the distribution part.
- They describe the Google indexing system (ca. 2002) as consisting of a number of phases, each implemented in MapReduce.

*open source implementation as part of Hadoop**

**http://hadoop.apache.org/*



Example for index construction

Map:

- d1 : C came, C c'ed.
- d2 : C died. →

<C,d1>, <came,d1>, <C,d1>, <c'ed, d1>, <C, d2>, <died,d2>

Reduce:

- (<C,(d1,d2,d1)>, <died,(d2)>, <came,(d1)>, <c'ed,(d1)>) →

(<C,(d1:2,d2:1)>, <died,(d2:1)>, <came,(d1:1)>, <c'ed, (d1:1)>).

Schema for index construction in MapReduce

Schema of map and reduce functions

- map: input → list(k, v) reduce: (k, list(v)) → output.

Instantiation of the schema for index construction

- map: collection → list(termID, docID)
- reduce: (<termID1, list(docID)>, <termID2, list(docID)>, …) → (postings list1, postings list2, …).

MapReduce

- Index construction was just one phase.
- Another phase: transforming a term-partitioned index into a document-partitioned index.
  - *Term-partitioned:* one machine handles a subrange of terms
  - *Document-partitioned:* one machine handles a subrange of documents.
- most search engines use a document-partitioned index … better load balancing, etc.

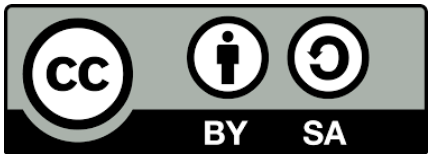
ΤΕΛΟΣ 5ου Μαθήματος

Ερωτήσεις?

Χρησιμοποιήθηκε κάποιο υλικό των:

- ✓ *Pandu Nayak and Prabhakar Raghavan, CS276: Information Retrieval and Web Search (Stanford)*

Τέλος Ενότητας



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Ιωαννίνων**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ενωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημειώματα

Σημείωμα Ιστορικού Εκδόσεων Εργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

Εχουν προηγηθεί οι κάτωθι εκδόσεις:

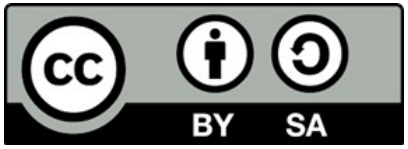
- Εκδοση 1.0 διαθέσιμη εδώ. http://ecourse.uoi.gr/course/view.php?id=1210.

Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Ιωαννίνων, Διδάσκων: Καθ. Ευαγγελία Πιτουρά. «Ανάκτηση Πληροφορίας. Κατασκευή Ευρετηρίου». Εκδοση: 1.0. Ιωάννινα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: http://ecourse.uoi.gr/course/view.php?id=1210.

Σημείωμα Αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Παρόμοια Διανομή, Διεθνής Εκδοση 4.0 [1] ή μεταγενέστερη.



- [1] https://creativecommons.org/licenses/by-sa/4.0/.