1

Κινητός και Διάχυτος Υπολογισμός (Mobile & Pervasive Computing)

Δημήτριος Κατσαρός, Ph.D.

Χειμώνας 2005

Διάλεξη 6η

2

Ιστοσελίδα του μαθήματος

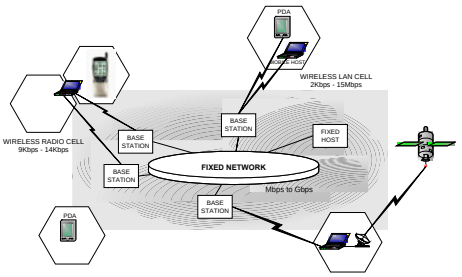
- **http://skyblue.csd.auth.gr/~dimitris/courses/mpc_fall05.htm**
- **http://skyblue.csd.auth.gr/~dimitris/courses/mpc_fall05/**
  - **books/**
  - **papers/**
  - **proj_papers/**
- Θα τοποθετούνται οι διαφάνειες του επόμενου μαθήματος
- Σταδιακά θα τοποθετηθούν και τα research papers που αντιστοιχούν σε κάθε διάλεξη

3

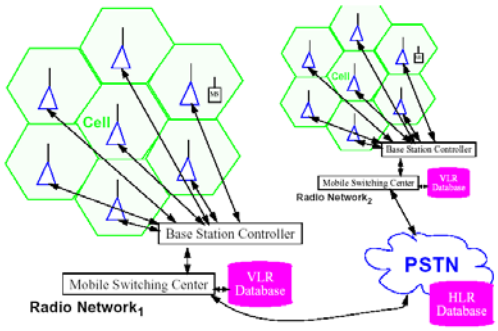
Περιεχόμενα

- **Αρχιτεκτονική δικτύου**
- Συνέπεια της cache (Cache Consistency)
- Πολιτική αντικατάστασης με συνέπεια cache

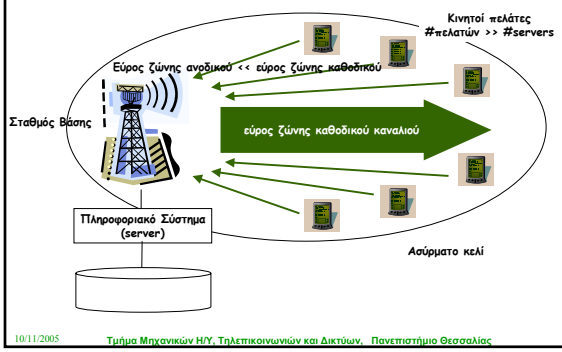
Αρχιτεκτονική κινητού δικτύου



Αρχιτ. Personal Comm. Sys. (PCS)



Μοντέλο εκπομπής



Περιεχόμενα

7

- Αρχιτεκτονική δικτύου
- **Συνέπεια της cache (Cache Consistency)**
- Πολιτική αντικατάστασης με συνέπεια cache

Σχήματα Cache coherency (1/2)

8

- Η γενική μέθοδος των Invalidation Reports
- Σχήματα No-Checking Caching
  1. Broadcasting Timestamps
  2. Amnesic Terminals
  3. Bit-Sequences
- Σχήματα Checking Caching
  1. Simple-checking caching scheme
  2. Simple-grouping caching scheme
  3. Grouping with cold update-set report

Σχήματα Cache coherency (2/2)

9

- Selective cache invalidation
  1. Group-based Cache Invalidation
  2. Hybrid Cache Invalidation
  3. Selective Cache Invalidation

10

Εισαγωγικά (1/3)

- Το caching μπορεί να ελαττώσει τις απαιτήσεις σε εύρος ζώνης στα κινητά δίκτυα
- Αφού χρησιμοποιήσουμε το caching, απαιτείται μια πολιτική ακύρωσης των δεδομένων της cache (cache invalidation strategy) για να εγγυηθεί την εγκυρότητα των δεδομένων της
- Μπορούμε να χρησιμοποιήσουμε μια "Αναφορά Ακύρωσης" (Invalidation Report, IR) για να διατηρήσουμε την εγκυρότητα των δεδομένων του κινητού χρήστη
- Για να εγγυηθούμε την εγκυρότητα, ο server περιοδικά(?) εκπέμπει invalidation reports

11

Εισαγωγικά (2/3)

- Κάθε κινητός πελάτης, εάν είναι ενεργός, ακούει τις αναφορές και ακυρώνει τα σχετικά δεδομένα του
- Όμως, εξαιτίας των περιορισμών σε ενέργεια (μπαταρία), ένας "κινητός" υπολογιστής συχνά λειτουργεί σε doze ή αποσυνδεδεμένο τρόπο λειτουργίας
- Ως αποτέλεσμα αυτού, ο κινητός υπολογιστής μπορεί να "χάσει" μερικές invalidation reports, με συνέπεια να αναγκαστεί να "πετάξει" όλα τα περιεχόμενα της cache του, όταν "ξυπνήσει"

12

Εισαγωγικά (3/3)

- Μπορούμε να κατηγοριοποίησουμε τον server
  - Stateful server
    - Ο server γνωρίζει ποια δεδομένα είναι cached από ποιους πελάτες
  - Stateless Server
    - Ο server δεν γνωρίζει την "κατάσταση" της cache των κινητών πελατών, αλλά ούτε και την κατάσταση του ίδιου του πελάτη, δηλ., εάν είναι αποσυνδεδεμένος ή σε ποια θέση βρίσκεται

13

Σχήμα IR

- Το σχήμα
  - Όταν ο χρήστης κάνει κάποιες αιτήσεις για αντικείμενα, ο κινητός υπολογιστής κρατά, τις αιτήσεις σε μια ουρά
  - Όταν ο κινητός υπολογιστής λάβει μια invalidation report που εκπέμπεται από τον server, θα ακυρώσουν όποια δεδομένα της cache υποδεικνύονται από την invalidation reports
  - Μετά την ακύρωση, ο κινητός υπολογιστής απαντά στις αιτήσεις της ουράς
  - Εάν τα δεδομένα της αίτησης βρίσκονται στην cache, θα προωθηθούν στην εφαρμογή του χρήστη από την cache.
  - Εάν τα δεδομένα της αίτησης δεν βρίσκονται στην cache, ο κινητός υπολογιστής θα κάνει την αίτηση για τα δεδομένα αυτά στον server

14

Κατηγοριοποίηση IR

- Μπορούμε να κατηγοριοποιήσουμε τις IR σύμφωνα με διαφορετικά κριτήρια, ως ακολούθως
  - **Πώς στέλνει ο server τις IR?**
    - Ασύγχρονα (Asynchronous)
      - Ο server εκπέμπει ένα μήνυμα ακύρωσης (invalidation message) για ένα αντικείμενο αμέσως μόλις αλλάξει η τιμή του αντικειμένου
    - Σύγχρονα (Synchronous)

    Όταν οι IR εκπέμπονται περιοδικά
  - **Πώς οργανώνεται η πληροφορία στην IR?**
    - Συμπιεσμένα (Uncompressed)
      - Οι αναφορές περιέχουν πληροφορία για κάθε αντικείμενο ξεχωριστά
    - Συμπιεσμένα
      - Οι αναφορές περιέχουν συνολική πληροφορία για υποσύνολα των αντικειμένων

15

Στόχοι

- Ελάττωση του Netware Transformation Cost
  - Ελάττωση του μεγέθους της IR
  - Βελτιστοποίηση της δομής της IR
  - Να κάνουμε τους πελάτες να μην χάνουν "πολλή πληροφορία", όταν είναι σε doze ή disconnected λειτουργία
- Το Netware transformation cost περιλαμβάνει το IR transformation cost και το data transformation cost.
  - IR transformation cost: ποσότητα της IR που αποστέλλεται στους πελάτες
  - Data transformation cost: Οι ποσότητες των δεδομένων που πρέπει να γίνουν downloaded από τον server, όταν τα επερωτούμενα δεδομένα δεν είναι στην cache
- Ο επόμενος τύπος είναι
  - Stateless, Symmetric, Asynchronous

16

Στρατηγικές NO-Checking Caching

- Ορολογία
  - L: ο server εκπέμπει μια IR κάθε L secs
  - w: το invalidation broadcast window
  - Ti: το τρέχον timestamp
  - T_{lb}: το timestamp της πιο πρόσφατης invalidation report που λήφθηκε από τον πελάτη (MU)
  - o_j: id αντικειμένου
  - t_j: το αντίστοιχο timestamp της πιο πρόσφατης αλλαγής/τροποποίησης του αντικειμένου
  - t_j^c: το timestamp της cache για τον o_j
  - IR: invalidation report

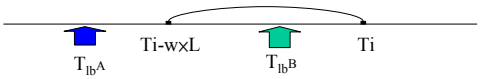
17

Μέθοδος Broadcasting Timestamps

- ΕΠΕΞΕΡΓΑΣΙΑ
  - Ο sever εκπέμπει την IR η οποία περιέχει μια λίστα Ui που ορίζεται ως ακολούθως για τη χρονική στιγμή Ti = iL.
  - Ui = $\{[o_j,t_j] : o_j \in D$ (database) και t_j είναι το timestamp της τελευταίας ενημέρωσης του o_j τέτοιο ώστε $Ti-wxL \le t_j \le Ti\}$
  - Ο MU καταγράφει τα $[o_j,t_j^c]$ όλων των αντικειμένων της cache του, όπου $o_j \in D$ (database) and t_j^c είναι το timestamp της cache του για το o_j
  - Ο MU κρατά επίσης το T_{lb} και μια λίστα Qi που ορίζεται ως ακολούθως:
  - Qi={o_i,: o_j έχει ζητηθεί στο διάστημα [Ti-1,Ti]}
  - Ο MU ακυρώνει τα αντικείμενα στην cache σύμφωνα με την IR
  - Μετά την ακύρωση, ο MU απαντά στις αιτήσεις των εφαρμογών

18

Drop ολόκληρη την cache ή όχι



T_{lbA} : : αγνοούμε όλη την cache

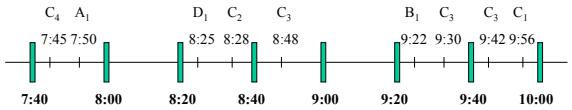
T_{lbB} : : ο MU συγκρίνει τα $[o_j,t_j^c]$ στην cache του με τα $[o_j,t_j]$ στην U_i για να αποφασίσει εάν θα διατηρήσει στην cache του το o_j ή όχι

➢ ΑΛΓΟΡΙΘΜΟΣ BROADCASTING TIMESTAMPS

if ($Ti-T_{lb} > wxL$) {drop the entire cache}
else{
  for every item o_j in the MU cache
  {if there is a pair[o_j,t_j] in Ui {
    if $t_j^c < t_j$ {
      throw o_j out of the cache}
    else {$t_j^c = Ti$}
}}}
for every item $o_j \in Qi$
  {
    if o_j is in the cache
      { use the cache's value to answer the query }
    else
      { go uplink with the query }
    $T_{lb} := Ti$ }

Παράδειγμα (1/5)

Όλα τα ενημερωμένα αντικείμενα



- L = 20 min
- w = 3
- Ti = 10:00

Παράδειγμα (2/5)

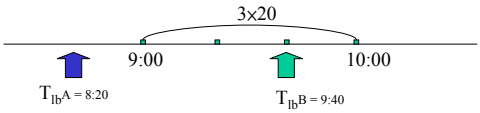
- Ο sever εκπέμπει τα ενημερωμένα αντικείμενα για το διάστημα 9:00 μέχρι 10:00

B_1	C_3	C_1
9:22	9:42	9:56

- Q: τα αντικείμενα που ζητήθηκαν από 9:40 μέχρι 10:00

E_1	C_3

Παράδειγμα (3/5)



$T_{lb}A$: Αρχική MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
8:20	7:50	6:15	6:25	6:35	6:22	6:45	6:55	7:05
	C_1	C_2	C_3	C_4	D_1	D_2	D_3	D_4
	7:15	7:28	7:30	7:40	7:28	7:25	7:35	7:40

Προκύπτουσα MU cache:T_{lb}: 10:00 ⇒ no item στην cache

Παράδειγμα (4/5)

$T_{lb}B$: Αρχική MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
9:40	7:50	6:15	6:25	6:35	9:22	6:45	6:55	7:05
	C_1	C_2	C_3	C_4	D_1	D_2	D_3	D_4
	7:15	8:28	9:30	7:40	8:25	7:25	7:35	7:40

Προκύπτουσα MU cache

T_{lb}	A_1	A_2	A_3	A_4	**B_1**	B_2	B_3	B_4
9:40	7:50	6:15	6:25	6:35	**10:00**	6:45	6:55	7:05
	C_1	C_2	**C_3**	C_4	D_1	D_2	D_3	D_4
	7:15	8:28	**9:30**	7:40	8:25	7:25	7:35	7:40

A,D is not in IR→no change

C_1,C_3 are in IR and $t_j^c < t_j$ → throw C_1, C_3

B_1 : is in IR and $t_j^c >= t_j$ → t_j^c = Ti

Παράδειγμα (5/5)

- Απάντηση αιτήσεων

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4		
10:00	7:50	6:15	6:25	6:35	10:00	6:45	6:55	7:05		
		C_2	**C_3**	C_4	D_1	D_2	D_3	D_4	C_3	**E_1**
		8:28	**9:42**	7:40	8:25	7:25	7:35	7:40	9:42	**6:00**

C_3, E_1 : δεν είναι στην cache, αλλά είναι στην ουρά αιτήσεων

→uplink στον server

T_{lb} → Ti

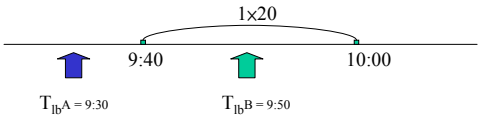
25

Μέθοδος Amnesic Terminals

- ΕΠΕΞΕΡΓΑΣΙΑ
  - Ο sever εκπέμπει μόνο τους προσδιοριστές των αντικειμένων που τροποποιήθηκαν μετά την τελυταία invalidation report, δηλ., w = 1
  - Ui = {o_j :o_j ∈ D (database) και η τελευταία ενημέρωση του o_j συνέβη τη στιγμή t_j τέτοια ώστε $T_{i-1} \le t_j \le T_i$}
  - Ο MU κρατά το T_{lb} και μια λίστα Qi που ορίζεται ως ακολούθως:
  - Qi={o_j:o_j έχει ζητηθεί στο διάστημα [Ti-1,Ti]}
  - Ο MU ακυρώνει τα αντικείμενα στην cache σύμφωνα με την IR
  - Μετά την ακύρωση, ο MU απαντά στις αιτήσεις των εφαρμογών

26

Drop ολόκληρη την cache ή όχι



T_{lbA} : αγνοούμε όλη την cache

T_{lbB} : Εάν ένα cached item αναφέρεται, τότε ο MU το διώχνει από την cache του

27

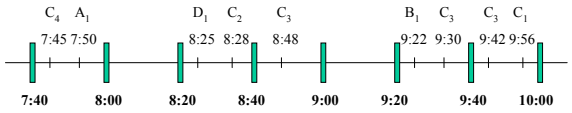
ΑΛΓΟΡΙΘΜΟΣ AMNESIC TERMINALS

```
if (Ti-Tlb > L) {drop the entire cache}
else{
      for every item oj in the MU cache
      {if oj in Ui {
            throw oj out of the cache}
}}
for every item oj ∈ Qi
   {
     if oj is in the cache
       { use the cache's value to answer the query }
     else
       { go uplink with the query }
     Tlb:= Ti }
```

28

Παράδειγμα (1/3)

- Όλα τα ενημερωμένα αντικείμενα

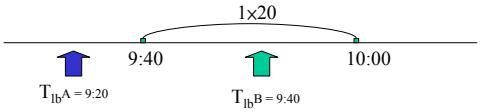


- L = 20 min
- Ti = 10:00
- Ο sever εκπέμπει τα ενημερωμένα αντικείμενα από 9:40 μέχρι 10:00

C_3	C_1

29

Παράδειγμα (2/3)



T_{lbA} : Αρχική MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
9:20	C_1	C_2	C_3	C_4	D_1	D_2	D_3	D_4

Προκύπτουσα MU cache:T_{lb} : 10:00 ⇒ no item στην cache

30

Παράδειγμα (3/3)

T_{lbB} : Αρχική MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
9:40	C_1	C_2	C_3	C_4	D_1	D_2	D_3	D_4

Προκύπτουσα MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
10:00	**C_1**	C_2	**C_3**	C_4	D_1	D_2	D_3	D_4

A, B, D : are not in IR→no change
C_1, C_3 : are in IR → throw C_1, C_3
$T_{lb} \rightarrow T_i$

Σχήμα Bit-Sequences

- Παρατηρήστε ότι
  - Γενικά, υπάρχει ένα tradeoff μεταξύ του μεγέθους και της αποτελεσματικότητας των εκπεμπόμενων αναφορών
  - Στο επόμενο σχήμα, θα αντιμετωπίσουμε το πρόβλημα της βελτιστοποίησης του μεγέθους των εκπεμπόμενων αναφορών

Τεχνικές βελτιστοποίησης

- Ονοματισμός των bit_sequences (bit_sequences naming)
- Συσσώρευση των ενημερώσεων (Update aggregation)
- Ιεραρχική δόμηση των bit-sequences.

Εισαγωγή στις Bit-Sequences (1/2)

- Η ΔΟΜΗ
  - Η IR αποτελείται από ένα σύνολο ακολουθιών bit (bit sequences), κάθε μια από τις οποίες έχει το αντίστοιχο timestamp
  - Κάθε bit αναπαριστά ένα αντικείμενο της database
  - Ένα "1" bit σε μια ακολουθία σημαίνει ότι το αντικείμενο που αναπαρίσταται από το bit αυτό έχει τροποποιηθεί μετά το χρόνο που καθορίζεται από το timestamp της ακολουθίας (sequence)
  - Ένα "0" bit σε μια ακολουθία σημαίνει ότι το αντικείμενο δεν έχει τροποποιηθεί από τη στιγμή του timestamp
  - Το σύνολο των ακολουθιών οργανώνεται περαιτέρω σε μια ιεραρχική δομή, με την B_n στη δομή να έχει N bits, τα οποία αντιστοιχούν στα N αντικείμενα της βάσης

34

Εισαγωγή στις Bit-Sequences (2/2)

- Το πολύ μισά από τα bits στη B_n μπορούν να τεθούν στην τιμή "1" μετά $TS(B_n)$
- Η επόμενη ακολουθία, που συμβολίζεται με B_{n-1}, στη δομή έχει N/2 bits
- Το k-οστό bit στη B_{n-1} αντιστοιχεί στο k-οστό "1" bit στη B_n
- $N/2^2$ bits μπορούν να τεθούν στην τιμή "1" αφότου $TS(B_{n-1})$
- Η ιεραρχική δομή περιέχει B_k ($k=1,\ldots,n,\ 2^n = N$)
- Μια επιπλέον dummy ακολουθία B_0 χρησιμοποιείται, όπου $TS(B_0)$ συμβολίζει το χρόνο μετά τον οποίο κανένα αντικείμενο δεν έχει τροποποιηθεί

35

➢ ΑΛΓΟΡΙΘΜΟΣ BIT-SEQUENCES

if $TS(B_0) \le T_{lb}$

κανένα αντικείμενο της cache δεν ακυρώνεται και ο αλγόριθμος τερματίζεται

if $T_{lb} < TS(B_n)$

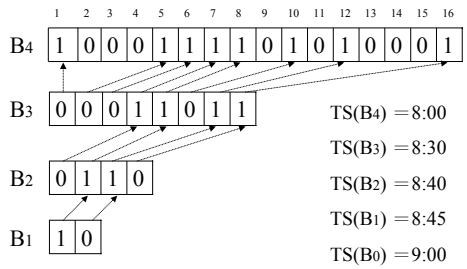
ολόκληρη η cache ακυρώνεται και ο αλγόριθμος τερματίζεται

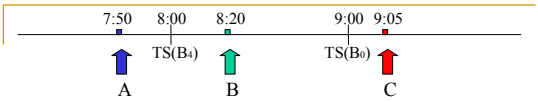
Εντοπίζουμε την bit sequence B_j με timestamp $TS(B_j) \le T_{lb} < TS(B_{j-1})\ \forall\ j\ (1 \le j \le n)$

Ακυρώνουμε όλα τα αντικείμενα που αναπαρίστανται με "1" bits στην B_j

36

Παράδειγμα Bit-Sequences (1/2)





Original cache：1 、2、 4、 7、 11、 15

•Περίπτωση A

$T_{lb} \le TS(B_4)$ ⇒ όλη η cache ακυρώνεται

•Περιπτωση B

$TS(B_4) \le T_{lb} \le TS(B_3)$ (invalidate with B4)

⇒ η προκύπτουσα cache μετά την τροποποίηση

2、4、11、15 .

•Περίπτωση C

$T_{lb} \ge TS(B_0)$ ⇒ καμία αλλαγή

Στρατηγικές Checking Caching

- Παρατηρείστε ότι:
  - Ακόμα και μετά την παρέλευση μεγάλου χρονικού διαστήματος αποσύνδεσης για τον κινητό πελάτη, ίσως μπορούμε να διατηρήσουμε αρκετά (πολλά) αντικείμενα στην cache
  - Πώς;

Σχήμα Simple Checking

- ΕΠΕΞΕΡΓΑΣΙΑ
  - Μετά τη λήψη μιας IR, ο MU πρώτα ακυρώνει την cache του ανάλογα με τα περιεχόμενα της IR
  - Εάν (T_{lb} < Ti -wxL)

    {στέλνει το T_{lb} και τα ids των cached αντικειμένων, τα οποία δεν έχουν ακυρωθεί ακόμα, στον server }

    Αλλιώς { επεξεργασία των queries και τερματισμός }
  - Ο server θα καθορίσει εάν τα αντικείμενα είναι άκυρα με βάση το T_{lb}
  - Ο sever αναφέρει το αποτέλεσμα του ελέγχου
  - Μετά τη λήψη της αναφοράς ελέγχου από τον server, ο πελάτης επεξεργάζεται τις queries

40

Παράδειγμα Simple Checking

$T_{lb}A$ = 8:20

Βήμα1: Invalidate cache items σύμφωνα με IR

Βήμα2: στείλε τα : T_{lb} A_1 A_2 A_3 A_4 B_2 B_3 B_4 C_2 C_4 D_2 D_3 D_4 στο server και ζήτησε τον έλεγχό τους

Βήμα3: Ο sever αναφέρει ότι τα C_2 D_1 είναι άκυρα

Η προκύπτουσα MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
10:00	7:50	6:15	6:25	6:35	9:22	6:45	6:55	7:05
	C_1	C_2	C_3	C_4	D_1	D_2	D_3	D_4
	7:15	8:28	9:30	7:40	8:25	7:25	7:35	7:40

41

Σχήμα Simple-Grouping Checking

- ΕΠΕΞΕΡΓΑΣΙΑ
  - Μετά τη λήψη μιας IR, ο MU πρώτα ακυρώνει την cache του ανάλογα με τα περιεχόμενα της IR
  - Εάν (T_{lb} < Ti - wxL) { στέλνει το T_{lb} και τα group ids στον server } Αλλιώς { επεξεργασία των queries και τερματισμός }
  - Εάν κάποιο αντικείμενο του group έχει ενημερωθεί μετά το T_{lb} { όλο το group ακυρώνεται }
  - Ο sever αναφέρει το αποτέλεσμα του ελέγχου
  - Μετά τη λήψη της αναφοράς ελέγχου από τον server, ο πελάτης επεξεργάζεται τις queries

42

Παράδ. Simple-Grouping Checking

$T_{lb}A$ = 8:20

Βήμα1: Invalidate cache items σύμφωνα με IR

Βήμα2: στείλε τα T_{lb} A B C D στο server και ζήτα έλεγχο

Βήμα3: Ο sever αναφέρει ότι τα groups B C D είναι άκυρα

Η προκύπτουσα MU cache

T_{lb}	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4
10:00	7:50	6:15	6:25	6:35	9:22	6:45	6:55	7:05
	C_1	C_2	C_3	C_4	D_1	D_2	D_3	D_4
	7:15	8:28	9:30	7:40	8:25	7:25	7:35	7:40

Groping with COld update-set REtention (GCORE)

- Παρατηρήστε ότι
  - Επιθυμούμε να αποφύγουμε την ακύρωση ολόκληρου του group.
  - Επιπρόσθετα του grouping, ο server επίσης δυναμικά προσδιορίζει το hot update set που έχει τροποποιηθεί σε ένα group και ***excludes it from the group*** όταν ελέγχει την εγκυρότητα του group

Δομή δεδομένων του GCORE (1/2)

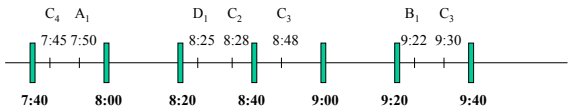
- Ο server διατηρεί για κάθε group την ιστορία τροποποίησης των αντικειμένων για τα τελευταία W διαστήματα εκπομπής (W ≥ w)
- Η group update history αποθηκεύεται στον group_table[ ], ο οποίος περιλαμβάνει
  - Μια λίστα των ids αντικειμένων και το timestamp της πιο πρόσφατης ενημέρωσης
  - Το timestamp της πιο πρόσφατης ενημέρωσης για το group
  - Τον αριθμό των διακριτών αντικειμένων που τροποποιήθηκαν πιο πρόσφατα μεταξύ (Ti-W×L) και (Ti-w×L)

Δομή δεδομένων του GCORE (2/2)


```
struct group_table_entry
  {
      double   time;
      int   total_wW;
      struct   pair   *uplist;
   } group_table[ ];
```

46

Παράδειγμα group_table[ ] (1/2)



- L=20 W=6 w=3
- Τρέχων χρόνος= 9:40
- T_i = 9:40
- group_table[A]：time = 7:50 ;
  total_wW = 1;
  uplist→(A_1,7:50) →NULL;

47

Παράδειγμα group_table[ ] (2/2)

- group_table[B]：time = 9:22;
  total_wW = 0;
  uplist→(B_1,9:22) →NULL;
- group_table[C]：time = 9:30;
  total_wW = 2;
  uplist→(C_3,9:30)→(C_2,8:28)→(C_4,7:45)→NULL;
- group_table[D]：time = 8:25;
  total_wW = 1;
  uplist→(D_1,8:25) →NULL;

48

Ενημέρωση του group_table[ ]

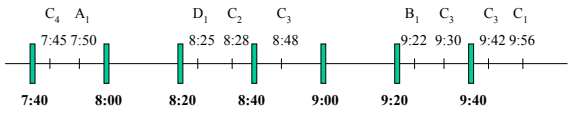
- **Πότε ενημερώνουμε τον group_table?**

1. Για κάθε αλλαγή των αντικειμένων, ο server ενημερώνει τον group_table[]
2. Κάθε φορά που ο server εκπέμπει μια invalidation report, ενημερώνει τον group_table[]

49

Αλλαγές στον group_table (1/4)

➢ **Αλλαγές στον group_table[ ] εξαιτίας τροποποίησης**



- L=20 W=6 w=3
- Τρέχων χρόνος = 9:59
- T_i = 9:40
- group_table[A] : time = 7:50 ;
  total_wW = 1;
  uplist→(A_1,7:50) →NULL

50

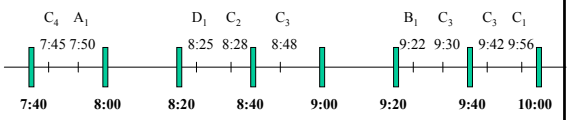
Αλλαγές στον group_table (2/4)

- group_table[B] : time = 9:22;
  total_wW = 0;
  uplist→(B_1,9:22) →NULL;
- group_table[C] : time = 9:56;
  total_wW = 2;
  uplist →(C_1,9:56) →(C_3,9:42)
  →(C_2,8:28) →(C_4,7:45) →NULL;
- group_table[D].time = 8:25 ;
  total_wW = 1;
  uplist→(D_1,8:25) →NULL;

51

Αλλαγές στον group_table (3/4)

➢ **Αλλαγές στον group_table[ ] εξαιτίας νέας εκπομπής**



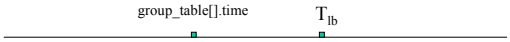
- L=20 W=6 w=3
- Τρέχων χρόνος = 10:00
- Ti = 10:00
- group_table[A] : time = 7:50;
  total_wW = 0;
  uplist→NULL

Αλλαγές στον group_table (4/4)

- group_table[B] : time = 9:22;
  total_wW = 0;
  uplist→(B_1,9:22) →NULL;
- group_table[C] : time = 9:56;
  total_wW = 1;
  uplist →(C_1,9:56) →(C_3,9:42)
  →(C_2,8:28) →NULL;
- group_table[D] : time = 8:25;
  total_wW = 1;
  uplist→(D_1,8:25) →NULL;

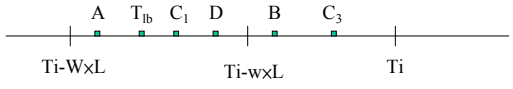
Έλεγχος εγκυρότητας από server

- Για να ελέγξουμε την εγκυρότητα
  - Ο sever ελέγχει την εγκυρότητα ενός group εξετάζοντας το T_{lb} και group_table[ ].total_wW
  - **Περίπτωση Α**



Ο MU παίρνει όλη την πληροφορία τροποποίησης
⇨ Το group είναι έγκυρο

Έλεγχος εγκυρότητας από server

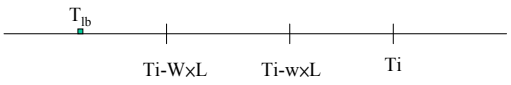


- **Περίπτωση Β**
  group_table[ ].total_wW = = 0
  ⇨ ο MU παίρνει όλη την πληροφορία τροποποίησης
  ⇨ το group είναι έγκυρο
- **Περιπτώσεις C, D**
  group_table[ ].total_wW != 0
  ⇨ ο MU δεν παίρνει όλη την πληροφορία τροποποίησης
  ⇨ το group είναι άκυρο

55

Έλεγχος εγκυρότητας από server

- **Περίπτωση Ε**



Ο sever δεν κράτησε the updated history μετά το T_{lb}
⇨ Όλα τα groups είναι άκυρα

56

➢ΑΛΓΟΡΙΘΜ. ΕΛΕΓΧΟΥ ΕΓΚΥΡΟΤΗΤΑΣ

- After receiving the IR, the MU first invalidates its cache.
- If (T_{lb} < Ti－wxL) { go to check }
  else { process the queries}
- Check：
  if (group_table[ ].time < T_{lb}) { the group is valid ; }
  else if (group _table[].total_wW= = 0 &&
  T_{lb} > Ti －WxL)
  { the group is valid ; }
  else if (group _table[].total_wW != 0 ||
  T_{lb} < Ti－WxL)
  { the group is invalid ; }
- The sever reports the result of checking.
- After receiving the checking report, process the queries

57

Παράδειγμα ελέγχου εγκυρότητας

- T_{lb}＝8:20
- Step1: First, invalidate cache items according to IR.
- Step2: send T_{lb} A B C D to server to ask for checking
- A：group_table[A].time＝7:50 < T_{lb}＝8:20
  ⇨the group is valid
- B：group_table[B].total_wW = = 0
  ⇨the group is valid
- C.D.group_table[B].total_wW != 0
  ⇨the group is invalid
- Step3: The sever reports that C and D are invalid

58

Παράδειγμα ελέγχου εγκυρότητας

- Η προκύπτουσα MU cache

T_{lb}	A_1	A_2	A_3	A_4
10:00	7:50	6:15	6:25	6:35

B_2	B_3	B_4
6:45	6:55	7:05

- Περίπτωση Ε

  T_{lb} = 7:40 ⇒ Όλα τα groups είναι άκυρα

59

Ενεργειακά αποδοτική cache invalidation

Παρατηρήστε ότι:

- Η τεχνική του επιλεκτικού συντονισμού (selective tuning) μπορεί να αξιοποιηθεί για να ελαττώσουμε την κατανάλωση ενέργειας
- Να εκτελέσουμε invalidation στο επίπεδο επερώτησης (query level)

60

Group Cache Invalidation (GCI)

➢ **ΕΠΕΞΕΡΓΑΣΙΑ GCI**

- Στο server, όλη η database διαμερίζεται σε ένα σύνολο διακριτών groups, κάθε group περιέχει τον ίδιο αριθμό αντικειμένων
- Κάθε αντικείμενο έχει ένα επιπρόσθετο χαρακτηριστικό (δηλ., το group id) που σχετίζεται μ' αυτό.
- Κάθε group έχει συσχετισμένο ένα timestamp, το οποίο αντιστοιχεί στο χρόνο της πιο προσφατης τροποποίησης ενός αντικειμένου που ανήκει στο group
- Ο server περιοδικά εκπέμπει broadcasts σταθερού μήκους GIR_i στη χρονική στιγμή T_i
- GIR_i={[gid,TS_{gid}] | gid είναι το group id ενός group αντικειμένων και Ts_{gid} είναι το timestamp της πιο πρόσφατης τροποποίησης του group}

61

Group Cache Invalidation (GCI)

- Στον client, όλα τα αντικείμενα με το ίδιο group id τοποθετούνται μαζί, και όλο το group έχει timestamp Ts^c_{gid} που αναπαριστά την τελευταία γνωστή χρονική στιγμή τροποίησης του group
- Qi είναι μια λίστα των query objects κατά το διάστημα $[T_{i-1}, T_i]$.
- Μπορούμε να αναθέσουμε group ids σε συνεχόμενες τιμές
- Η GIR μπορεί να εκπεμφθεί σε σταθερή σειρά
- Αφού οι clients γνωρίζουν τα groups για τα οποία ενδιαφέρονται, χρειάζεται να είναι σε active mode μόνο όταν εκπέμπονται τα timestamps των groups που τους ενδιαφέρουν

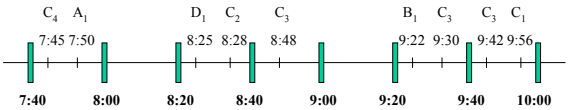
62

Αλγόριθμος GCI

```
for (every pair [gid,TS_gid] in GIRi)  {
    if (client cache objects in group gid){
      if (TS^c_gid < TS_gid){
          remove all objects in the group from the cache}
      else {TS^c_gid ← TS_gid}
    }
}
for every object O ∈ Qi  {
    if O is in the cache
       { use the cache's value to answer the query }
    else
       { go uplink with the query }
 }
```


63

Παράδειγμα GCI (1/3)



- T_{lb}= 8:20
- Ti= 10:00
- GIR

A	B	C	D
7:50	9:22	9:56	8:25

Παράδειγμα GCI (2/3)

- Αρχική cache

T_{lb}	A	A_1	A_2	A_3	A_4	B	B_1	B_2	B_3	B_4
8:20	8:20					8:20				
	C	C_1	C_2	C_3	C_4	D	D_1	D_2	D_3	D_4
	8:20					8:20				

- Προκύπτουσα cache

T_{lb}	A	A_1	A_2	A_3	A_4	**B**	**B_1**	**B_2**	**B_3**	**B_4**
10:00	7:50					**8:20**				
	C	**C_1**	**C_2**	**C_3**	**C_4**	**D**	**D_1**	**D_2**	**D_3**	**D_4**
	8:20					**8:20**				

Παράδειγμα GCI (3/3)

- A : TS^c_A= 8:20 > TS_A= 7:50
  ⇒ TS^c_A← TS_A = 7:50
- B : TS^c_B= 8:20 < TS_B = 9:22
  ⇒ the group is invalid
- C : TS^c_C= 8:20 < TS_C= 9:56
  ⇒ the group is invalid
- D : TS^c_D = 8:20 < TS_D= 8:25
  ⇒ the group is invalid

Χαρακτηριστικά GCI

Παρατηρήστε στο GCI ότι:

- Οικονομία σε χώρο (σταθερό μέγεθος της αναφοράς)
- Ανεξάρτητη του χρόνου αποσύνδεσης του client
- Επιτρέπει επιλεκτικό συντονισμό
- Υπάρχει **εσφαλμένη invalidation**, (δηλ., απορρίπτουμε αντικείμενα τα οποία είναι έγκυρα)

67

Υβριδική Cache Invalidation (HCI)

➢ΕΠΕΞΕΡΓΑΣΙΑ HCI

- Ο server εκπέμπει κάθε L time units ένα ζεύγος από IRs, μια είναι OIR (δηλ., Broadcasting Timestamps) και η άλλη είναι GIR
- OIR είναι μια λίστα από OUi

  OUi={[id,TS_{id}] | id είναι ο identifier του αντικειμένου O στη DB και Ts_{id} είναι το timestamp της τελευταίας τροποποίησης του O, τέτοιας ώστε Ti-wxL ≤ TS_{id} ≤ Ti}
- GIR είναι μια λίστα από GUi

  GUi={[gid,TS_{gid}] | gid είναι ένας group identifier και Ts_{gid} είναι το timestamp της τελευταίας τροποποίησης αντικειμένου του group, τέτοιας ώστε TS_{gid} < Ti-wxL}

68

Υβριδική Cache Invalidation (HCI)

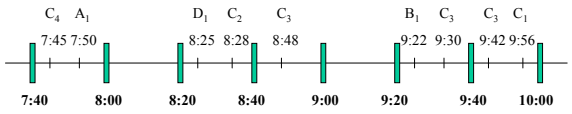
- Διαφορετικά από τον GCI, το valid timestamp για κάθε group δίνεται από το timestamp του αντικειμένου στο group το οποίο είναι κοντινότερα στο T_i-wxL
- Για εκείνους τους clients με μικρό χρόνο αποσύνδεσης, ο έλεγχος της cache γίνεται με την OIR.
- Για εκείνους τους clients που αποσυνδέθηκαν πριν από τη στιγμή Ti-wxL, η OIR και η GIR μπορούν να αξιοποιηθούν ταυτόχρονα, ώστε οι clients να μην πετάξουν ολόκληρη την cache
- Η OIR ελαχιστοποιεί τη false invalidation και η GIR αποφεύγει την invalidation ολόκληρης της cache για μεγάλους χρόνους αποσύνδεσης

69

➢ΑΛΓΟΡΙΘΜΟΣ HCI

```
for (every pair [id, TS_id] ∈ OIR_i) {
    if (object id is in the client cache){
        if (TS^c_id < TS_id) {
            remove object from the cache
        }
        else {TS^c_id ← TS_id}
    }
}
if (T_i − T^c > wL) {
    for (every group gid in the client cache) {
        selectively tune to the pair
            [gid, TS_gid] in GIR_i
        if (TS^c_gid < TS_gid) {
            remove all objects in group
                from the cache
        }
        else {TS^c_gid ← TS_gid}
    }
}
for (every object O ∈ Q_i) {
    if (O is in the cache) {
        use the cache to answer the query
    }
    else {submit request for O}
}
T^c ← T_i
```

Παράδειγμα HCI (1/3)



- T_{lb}= 8:20
- L= 20 min
- w = 3
- Ti = 10:00

Παράδειγμα HCI (2/3)

- OIR

B_1	C_3	C_1
9:22	9:42	9:56

- GIR

A	B	C	D
7:50	7:05	8:48	8:25

Αρχική cache

T_{lb}	A	A_1	A_2	A_3	A_4	B	B_1	B_2	B_3	B_4
8:20	8:20	7:50	6:15	6:25	6:35	8:20	6:22	6:45	6:55	7:05
	C	C_1	C_2	C_3	C_4	D	D_1	D_2	D_3	D_4
	8:20	7:15	7:28	7:30	7:40	8:20	7:28	7:25	7:35	7:40

Παράδειγμα HCI (3/3)

- Προκύπτουσα cache

T_{lb}	A	A_1	A_2	A_3	A_4	B	B_1	B_2	B_3	B_4
10:00	7:50	7:50	6:15	6:25	6:35	7:05	6:22	6:45	6:55	7:05
	C	C_1	C_2	C_3	C_4	D	D_1	D_2	D_3	D_4
	8:20	7:15	7:28	7:30	7:40	8:20	7:28	7:25	7:35	7:40

- A : TS^c_A = 8:20 > TS_A = 7:50
  ➡ $TS^c_A \leftarrow TS_A$ = 7:50
- B : TS^c_B = 8:20 > TS_B = 7:05
  ➡ $TS^c_B \leftarrow TS_B$ = 7:05
- C : TS^c_C = 8:20 < TS_C = 8:48
  ➡ the group is invalid
- D : TS^c_D = 8:20 < TS_D = 8:25
  ➡ the group is invalid

73

Selective Cache Invalidation (SCI)

Παρατηρείστε ότι:

- Για να ελαχιστοποίησουμε την κατανάλωση ενέργειας, θα πρέπει να εστιάζουμε μόνο στα αντικείμενα που ενδιαφέρουν την query
- Δεν επιθυμούμε να ακούμε ολόκληρη την OIR

74

Selective Cache Invalidation (SCI)

➢ ΕΠΕΞΕΡΓΑΣΙΑ SCI

- Η GIR εκπέμπεται πριν από την OIR.
- Τα entries στην OIR είναι ταξινομημένα και εκπέμπονται με βάση τα groups
- Ένας επιπλέον pointer προστίθεται σε κάθε στοιχείο της GIR, και αυτός ο pointer δείχνει στη "θέση εκκίνησης" των αντικειμένων του group στην OIR
- Στον client, πρώτα συντονίζεται επιλεκτικά στην GIR, και κρατάει τους pointers των groups που τον ενδιαφέρουν στην μνήμη του

75

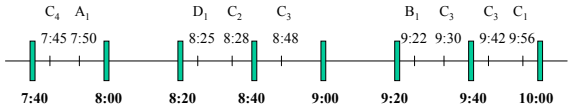
➢ ΑΛΓΟΡΙΘΜΟΣ SCI

```
P ← φ
for (every group in the client cache) {
    selectively tune to the
        triplet [gid, TS_gid, P_gid] in GIR_i
    if TS^c_gid < TS_gid {
        invalidate all objects of group
    }
    else {
        TS^c_gid ← TS_gid
        P ← P ∪ {P_gid}
    }
}
```

```
for (every object O ∈ Q_i) {
    if (O is in the cache) {
        use the cache to answer the query
    }
    else {submit uplink request for O}
}
```

```
while (P ≠ ∅) {
    p ← first(P)
    P ← P − {p}
    switch to doze mode until the position p
        is coming
    while (not encounter partition symbol) {
        download pair [id, TS_id]
        if (id is in client cache) {
            if TS^c_id < TS_id {
                invalidate object
            }
            else {
                TS^c_id ← TS_id
            }
        }
    }
}
```

Παράδειγμα SCI (1/3)



- T_{lb} = 8:20
- L = 20 min
- w = 3
- Ti = 10:00

Παράδειγμα SCI (2/3)

- IR GIR

A	B	C	D
7:50	7:05	8:48	8:25
PA	PB	PC	PD

OIR

B_1		C_1	C_3
9:22		9:56	9:56

Αρχική cache

T_{lb}	A	A_1	A_2	A_3	A_4	B	B_1	B_2	B_3	B_4
8:20	8:20	7:50	6:15	6:25	6:35	8:20	6:22	6:45	6:55	7:05
	C	C_1	C_2	C_3	C_4	D	D_1	D_2	D_3	D_4
	8:20	7:15	7:28	7:30	7:40	8:20	7:28	7:25	7:35	7:40

Παράδειγμα SCI (3/3)

- Προκύπτουσα cache

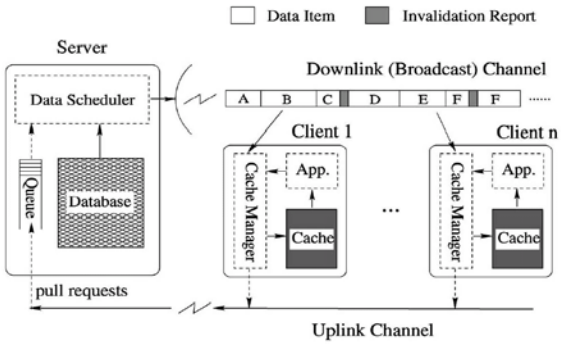
T_{lb}	A	A_1	A_2	A_3	A_4	B	B_1	B_2	B_3	B_4
10:00	7:50	7:50	6:15	6:25	6:35	7:05	6:22	6:45	6:55	7:05
	C	C_1	C_2	C_3	C_4	D	D_1	D_2	D_3	D_4
	8:20	7:15	7:28	7:30	7:40	8:20	7:28	7:25	7:35	7:40

- A : TS^c_A = 8:20 > TS_A = 7:50
  ⇒ TS^c_A ← TS_A = 7:50 , record PA
- B : TS^c_B = 8:20 > TS_B = 7:05
  ⇒ TS^c_B ← TS_B = 7:05, record PB
- C : TS^c_C = 8:20 < TS_C = 8:48
  ⇒ the group is invalid
- D : TS^c_D = 8:20 < TS_D = 8:25
  ⇒ the group is invalid
- PB : TS^c_{B1} < TS_{B1} ⇒ B1 is invalid

Περιεχόμενα

- Αρχιτεκτονική δικτύου
- Συνέπεια της cache (Cache Consistency)
- **Πολιτική αντικατάστασης με συνέπεια cache**

Αρχιτεκτονική συστήματος



Ορολογία (1/2)

- D: the number of data items in the database.
- C: the size of the client cache.
- $\overline{a_i}$: mean access arrival rate of data item i,

$$i = 1, 2, \ldots, D.$$

- $\overline{u_i}$: mean update arrival rate of data item i,

$$i = 1, 2, \ldots, D.$$

- x_i: the ratio of update rate to access rate for data item i, i.e., $x_i = \overline{u_i}/\overline{a_i}$, $i = 1, 2, \ldots, D$.
- p_i: access probability of data item i, $p_i = \overline{a_i}/\sum_{k=1}^{D} \overline{a_k}$ for $i = 1, 2, \ldots, D$.

Ορολογία (2/2)

- l_i: access latency of data item i, $i = 1, 2, \ldots, D$.
- b_i: retrieval delay from the server (i.e., cache miss penalty) for data item i, $i = 1, 2, \ldots, D$.
- s_i: size of data item i, $i = 1, 2, \ldots, D$.
- v: cache validation delay, i.e., access latency of an effective invalidation report.
- d_k: the data item requested in the kth access,[1] $d_k \in \{1, 2, \ldots, D\}$.
- C_k: the set of cached data items after the kth access, $C_k \subseteq \{1, 2, \ldots, D\}$.
- U_k: the set of cached data items that are updated between the kth access and the $(k+1)$th access, $U_k \subseteq C_k$.
- V_k: the set of victims chosen to be replaced in the kth access, $V_k \subseteq (C_{k-1} - U_{k-1})$.

Η πολιτική Min-SAUD

- Ορίζουμε ένα μέτρο αξίας κάθε αντικειμένου ως εξής:

$$gain(i) = \frac{p_i}{s_i}\left(\frac{b_i}{1+x_i} - v\right)$$

- Μετά από k προσπελάσεις, επιθυμούμε να μεγιστοποιήσουμε τη συνολική αξία των αντικειμένων στην cache, δηλ., να αναγνωρίσουμε το βέλτιστο victim set V_k^*: $V_k^* \subseteq C_{k-1} - U_{k-1}$,

$$V_k^* = arg \min_{V_k \subseteq (C_{k-1} - U_{k-1})} \sum_{i \in V_k} gain(i)$$

$$s.t. \quad \sum_{i \in V_k} s_i \geq \sum_{j \in (C_{k-1} - U_{k-1})} s_j + s_{d_k} - C$$

Optimality της Min-SAUD (1/8)

- Θα δείξουμε ότι η Min-SAUD είναι βέλτιστη για το stretch μέτρο
- Υποθέτουμε Independence Reference Model
- Arrival και Updates είναι Poisson κατανεμημένες
- Interarrival times για αιτήσεις και τροποποιήσεις είναι Εκθετικά κατανεμημένες
- Δηλ., οι αντίστοιχες density functions είναι:

$$f(t_i^a) = \overline{a_i} e^{-\overline{a_i} t_i^a}$$

$$g(t_i^u) = \overline{u_i} e^{-\overline{u_i} t_i^u}$$

85

Optimality της Min-SAUD (2/8)

- Το access cost είναι το γινόμενο πιθανότητας προσπέλασης επί stretch (access latency προς service time)
- Αγνοώντας το εύρος ζώνης, χρησιμοποούμε το relative access cost, ορισμένο ως πιθανότητα προσπέλασης επί access latency προς μέγεθος, δηλαδή μετά από k προσπελάσεις, έχουμε:

$$S_k = \sum_{1 \le i \le D} p_i \cdot \frac{l_i}{s_i}$$

86

Optimality της Min-SAUD (3/8)

- Θυμηθείτε ότι ακόμα και εάν έχουμε cache hit, δεν μπορούμε να το δώσουμε στην application, παρά μόνο μετά που θα δούμε την IR που περιέχει πληροφορία για το συγκεκριμένο αντικείμενο.
- Έστω ότι $Pr(U_i)$ είναι η πιθανότητα ότι το αντικείμενο i έχει τροποποιηθεί στο διάστημα από την τρέχουσα στιγμή μέχρι την άφιξη μιας τέτοιας IR μετά την query για το i. Η πρηγούμενη εξίσωση μπορεί να γραφεί:

$$S_k = \sum_{i \in C_k} \frac{p_i \cdot l_i}{s_i} + \sum_{i \notin C_k} \frac{p_i \cdot l_i}{s_i}$$

$$= \sum_{i \in C_k} \frac{p_i \cdot l_i}{s_i} Pr(U_i) + \sum_{i \in C_k} \frac{p_i \cdot l_i}{s_i} (1 - Pr(U_i)) + \sum_{i \notin C_k} \frac{p_i \cdot l_i}{s_i}$$

87

Optimality της Min-SAUD (4/8)

- Η εξίσωση της προηγούμενης διαφάνειας αντιστοιχεί σε τρεις περιπτώσεις, τις ακόλουθες:
  1. a cache hit but an obsolete copy,
  2. a cache hit and an up-to-date copy,
  3. a cache miss.
- Η access latency l_i είναι αντίστοιχα:

$$l_i = \begin{cases} v + b_i & \text{if } i \in C_k \text{ and an obsolete copy;} \\ v & \text{if } i \in C_k \text{ and an up-to-date copy;} \\ b_i & \text{if } i \notin C_k. \end{cases}$$

Optimality της Min-SAUD (5/8)

- Θα παράξουμε το $Pr(U_i)$
- Έστω ότι $Pr(U_i')$ είναι η πιθανότητα τροποποίησης του i από την τρέχουσα στιγμή μέχρι τη στιγμή της επόμενης query για το i.
- Προσεγγίζοντας την $Pr(U_i)$ με την $Pr(U_i')$, έχουμε:

$$Pr(U_i) \doteq Pr(U_i') = Pr(t_i^u < t_i^a) = \int_{t_i^a=0}^{\infty} \int_{t_i^u=0}^{t_i^a} f(t_i^a) g(t_i^u) dt_i^u dt_i^a$$

$$= \frac{\overline{u_i}}{\overline{u_i} + \overline{a_i}}.$$

- Συνδυάζοντας τις εξισώσεις για το S_k, l_i kai $Pr(U_i)$, έχουμε:

Optimality της Min-SAUD (6/8)

$$S_k = \sum_{i \in C_k} \left(\frac{p_i(v + b_i)}{s_i} \cdot \frac{\overline{u_i}}{\overline{u_i} + \overline{a_i}} \right) + \sum_{i \in C_k} \left(\frac{p_i \cdot v}{s_i} \cdot \frac{\overline{a_i}}{\overline{u_i} + \overline{a_i}} \right) +$$

$$\sum_{i \notin C_k} \frac{p_i \cdot b_i}{s_i} = \sum_{i \in C_k} \frac{p_i(v + \frac{\overline{u_i} b_i}{\overline{u_i} + \overline{a_i}})}{s_i} + \sum_{i \notin C_k} \frac{p_i \cdot b_i}{s_i}.$$

- Με βάση την παραπάνω εξίσωση μπορούμε να δείξουμε ότι:

Theorem 1. *The replacement policy* Min-SAUD *gives better access cost, in terms of stretch, than any other replacement policy.*

Optimality της Min-SAUD (7/8)

- Θα αποδείξουμε την optimality της Min-SAUD, εάν δείξουμε ότι το κόστος S_k είναι πάντα το μικρότερο, εάν χρησιμοποιούμε την πολιτική Min-SAUD. Με επαγωγή
- Έστω ότι S_w είναι το βέλτιστο κόστος για κάποιο k=w για μια άλλη πολιτική
- Έστω ότι V_{w+1} είναι το victim set για να κάνουμε χώρο για το d_{w+1}. Επομένως, έχουμε ότι

  $C_{w+1}=C_w-U_w \cup \{d_{w+1}\}-V_{w+1}$
- Συνεπώς:

Optimality της Min-SAUD (8/9)

$$
\begin{aligned}
S_{w+1} &= \sum_{i \in C_{w+1}} \frac{p_i \left(v + \frac{\overline{u_i} b_i}{\overline{u_i} + \overline{a_i}} \right)}{s_i} + \sum_{i \notin C_{w+1}} \frac{p_i \cdot b_i}{s_i} \\
&= S_w + \sum_{i \in U_w} \left(\frac{p_i}{s_i} \left(\frac{b_i}{1 + x_i} - v \right) \right) - \frac{p_{d_{w+1}}}{s_{d_{w+1}}} \left(\frac{b_{d_{w+1}}}{1 + x_{d_{w+1}}} - v \right) \\
&\quad + \sum_{i \in V_{w+1}} \left(\frac{p_i}{s_i} \left(\frac{b_i}{1 + x_i} - v \right) \right) \\
&= B + \sum_{i \in V_{w+1}} \frac{p_i}{s_i} \left(\frac{b_i}{1 + x_i} - v \right),
\end{aligned}
$$

Optimality της Min-SAUD (9/9)

- Όπου το B είναι ίσο με:

$$
B = S_w + \sum_{i \in U_w} \left(\frac{p_i}{s_i} \left(\frac{b_i}{1 + x_i} - v \right) \right) - \frac{p_{d_{w+1}}}{s_{d_{w+1}}} \left(\frac{b_{d_{w+1}}}{1 + x_{d_{w+1}}} - v \right)
$$

- Αφού το B δεν ελέγχεται από την πολιτική αντικατάστασης, συμπεραίνουμε ότι το ελάχιστο access cost επιτυγχάνεται όταν διώχνονται τα αντικείμενα με το μικρότερο :

$$
\sum_{i \in V_{w+1}} \frac{p_i}{s_i} \left(\frac{b_i}{1 + x_i} - v \right)
$$