LRU-K PAGE REPLACEMENT ALGORITHM

E0 261

Jayant Haritsa

Computer Science and Automation

Indian Institute of Science

ABOUT PAPER

- Authors:
  - Elizabeth J. O'Neil
  - Patrick E. O'Neill
  - Gerhard Weikum
- Facts:
  - Presented at ACM SIGMOD Conference May 1993, Washington D.C. and published in its Proceedings, May 1993.
  - Paper has 1235 Citations till date.
  - Implementation in C available on author's website.

LRU

- Concept:
  - When a new buffer page is needed, the buffer pool manager drops the page from buffer that has not been accessed for the longest time.

- Why LRU?
  - Most logical and intuitive
  - Easy to implement
  - Good chance of a question in GATE exam => Increased chances of getting into IISc!

CASE - I

- A multi-user database application, which references randomly chosen customer records through a clustered B-tree indexed key, CUST-ID, to retrieve desired information.
- If pattern of access is: I1, R1, I2, R2, I3, R3, ….
- In traditional LRU, we may drop the index pages, which would then be referenced again, resulting in increased I/O.

CASE - II

- Consider a multi-process database application with good “locality” of shared page reference.
- Suppose, if 5000 buffered pages out of 1 million disk pages get 95% of the references by concurrent processes.
- Sequential scans will replace commonly referenced pages in buffer with pages unlikely to be referenced again, a common complaint in many commercial situations!

PROBLEM

- LRU decides what page to drop from buffer based on too little information, limiting itself to only the time of last reference.
- LRU is unable to differentiate between pages that have relatively frequent references.
- Results in increased I/O.
- In general, bad for database applications!

PRIOR LITERATURE

- Solutions in the literature majorly fall into two categories :

- Page pool tuning :
- DBA constructs page pools, separating different reference patterns into different buffer pools.

- Supported by some commercial database systems, but requires great manual effort!

PRIOR LITERATURE

- Query Plan Analysis :
- Query optimizer should provide hints about the usage pattern of a query plan.
- In multi-user situations, query optimizer plans may overlap in complicated ways.
- There is a need for a more general and global page replacement policy!

TODAY's PAPER

- We'll see LRU-K algorithm!
- It discriminates well between page sets with different levels of reference frequency (e.g., index pages vs. data pages).
- It adapts itself to evolving access patterns.
- It is self-reliant in that it does not need any external hints.
- It is fairly simple and incurs little bookkeeping overhead.

INDEPENDENT REFERENCE MODEL

- The Independent Reference Model is a conceptual model used in the analysis of storage system like disk drives, caches, etc.

- Under this model the references to stored objects are independent random variables.

INDEPENDENT REFERENCE MODEL

- Set N = {1, 2, … n} of disk pages.
- The database system under study makes a succession of references to these pages specified by the reference string: $r_1, r_2, \ldots r_t$ where r_t = page p ($p \varepsilon N$)
- At instant t, we assume that each disk page p has a well defined probability, b_p, to be the next page referenced.

$$Prob(r_{t+1} = p) = b_p \qquad \forall\ p \varepsilon N.$$

- We assume that b_p is independent of t.

INDEPENDENT REFERENCE MODEL

- Each disk page p has an expected reference interarrival time, Ip, the time between successive occurrences of p

- For the reference string in last slide, we have

$$\mathrm{Ip} = \frac{1}{\mathrm{b_P}}$$

- GOAL : The system then attempts to keep in memory buffers only those pages with shortest access interarrival times, or equivalently greatest probability of reference.

BACKWARD K-DISTANCE

Given a reference string known up to time t, $r_1, r_2, \ldots, r_t$, the backward K-distance $b_t(p,K)$ is the distance backward to the K^{th} most recent reference to the page p:

$b_t(p,K) = x$, if r_{t-x} has the value p and there have been exactly K-1 other values i with $t - x < i \leq t$, where $r_i = p$,

$= \infty$, if p does not appear at least K times in $r_1, r_2, \ldots, r_t$

- The victim page (page to be dropped) is the one whose Backward K-distance is the maximum of all pages in buffer.
- Conflict : Incase of >1 pages with $b_t(p,K) = +\infty$, follow classical LRU to choose victim.

EXAMPLE (K=2)

- Consider the reference string :

	7, 5, 6, 2, 9, 1, 8, 7, 6, 8, 1
t :	0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10
x :	∞, ∞,∞,∞, ∞ ,∞ ,∞,7, 6, 4, 5

- And if we have LRU-2,

  - $b_t(p=1,k=2) => 5$
  - $b_t(p=6,k=2) => 6$
  - $b_t(p=9,k=2) => +\infty$

BACKWARD K-DISTANCE

- Ability to make decisions by measurement of an actual interarrival between references, rather than estimating simply by a lower bound of the time back to the most recent reference!

But, we still have problems :

- Early page replacement
- Retained information problem

EARLY PAGE REPLACEMENT

- Four ways in which a pair of references might end up to same disk page :

- (1) Intra-Transaction
- (2) Transaction-Retry ==> correlated reference pairs
- (3) Intra-Process comparatively short span

- (4) Inter-Process ==> comparatively longer span

(1),(2),(3) are called

EARLY PAGE REPLACEMENT

- Interarrival time should be calculated based on non-correlated access pairs!
- Each successive access by the same process within a time-out period (defined) is assumed to be correlated.
- Refer to the approach, which associates correlated references, as the Time-Out Correlation method.
- Refer to the time-out period as the Correlated Reference Period.

RETAINED INFO PROBLEM

- Each time a page p is referenced, it is made buffer resident.
- If our estimate of $b_t(p,K) = +\infty$, we drop it!
  But what if the page is popular, but just not able to meet the backward-k criteria?
- Though the page is frequently referenced, we would have no history about it to recognize this fact!
- We need to maintain history information about any page for some period after its most recent access. We refer to this period as the Retained Information Period.

DATA STRUCTURES : LRU-K

- HIST(p)
- Denotes the history control block of page p
- Contains the times of the K most recent references to page p
- HIST(p,1), HIST(p,2) denotes the time of last and second last references to page p.

- LAST(p)
- Denotes the time of the most recent reference to page p.

- Maintained IN-MEMORY! Why not store in page header?

PSEUDO CODE : LRU-K

```
Procedure to be invoked upon reference to page p at time t:

if p is already in the buffer
then    /* update history information of p */
   if t - LAST(p) > Correlated_Reference_Period
   then    /* a new, uncorrelated reference */
      correl_period_of_refd_page := LAST(p) - HIST(p,1)
      for i := 2 to K do
         HIST(p,i) := HIST(p,i-1)  +
                           correl_period_of_refd_page
      od
      HIST (p,1) := t
      LAST(p) := t
   else    /* a correlated reference */
      LAST(p) := t
   fi
else    /* select replacement victim */
   min := t
   for all pages q in the buffer do
      if t - LAST(q) > Correlated_Reference_Period
                              /* if eligible for replacement */
         and HIST(q,K) < min
                     /* and max Backward K-distance so far */
      then
         victim := q
         min := HIST(q,K)
      fi
   od
   if victim is dirty then
                  write victim back into the database fi
   /* now fetch the referenced page */
   fetch p into the buffer frame previously held by victim
   if HIST(p) does not exist
   then    /* initialize history control block */
      allocate HIST(p)
      for i := 2 to K do HIST(p,i) := 0 od
   else
      for i := 2 to K do HIST(p,i) := HIST(p,i-1) od
   fi
   HIST(p,1) := t
   LAST(p) := t
fi
```

OPTIMALITY OF LRU-K

- Set N = {1, 2, … n} of disk pages.
- Set M = {1, 2, … m} of memory buffers. $1 \leq m \leq n$

- Reference string: $r_1, r_2, \ldots r_t$ where r_t = page p (p ε N)

- Recall that by Independence reference model,

Mean interarrival time : $Ip = \frac{1}{b_P}$

Ao ALGORITHM

- Let Ao denote the buffering algorithm that replaces the buffered page p in memory whose expected value I_p is the maximum, i.e., the page for which b_p is smallest.

- Algorithm Ao is optimal under the independent reference assumption. (Proof in the full version of the paper)

- The best possible approximation of Ao is the LRU-K algorithm. As K increases, LRU-K gives very close results to the optimal algorithm.

TWO-POOL EXPERIMENT

- Pool 1 with N1 pages and Pool 2 with N2 pages
- In this two pool experiment, alternating references are made :

  I1, R1, I2, R2, I3, R3...

- (Recall that this experiment models Case-I in the initial slides!)

B	LRU-1	LRU-2	LRU-3	A_0	B(1)/B(2)
60	0.14	0.291	0.300	0.300	2.3
80	0.18	0.382	0.400	0.400	2.6
100	0.22	0.459	0.495	0.500	3.0
120	0.26	0.496	0.501	0.501	3.3
140	0.29	0.502	0.502	0.502	3.2
160	0.32	0.503	0.503	0.503	2.8
180	0.34	0.504	0.504	0.504	2.5
200	0.37	0.505	0.505	0.505	2.3
250	0.42	0.508	0.508	0.508	2.2
300	0.45	0.510	0.510	0.510	2.0
350	0.48	0.513	0.513	0.513	1.9
400	0.49	0.515	0.515	0.515	1.9
450	0.50	0.517	0.518	0.518	1.8

Table 4.1. Simulation results of the two pool experiment, with disk page pools of $N_1 = 100$ pages and $N_2 = 10{,}000$ pages. The first column shows the buffer size B. The second through fifth columns show the hit ratios of LRU-1, LRU-2, LRU-3, and A_0. The last column shows the equi-effective buffer size ratio B(1)/B(2) of LRU-1 vs. LRU-2.

ZIPFIAN RANDOM ACCESS

- A single pool of pages with skewed random access. We generated references to N pages with a Zipfian distribution of reference frequencies.

B	LRU-1	LRU-2	A_0	B(1)/B(2)
40	0.53	0.61	0.640	2.0
60	0.57	0.65	0.677	2.2
80	0.61	0.67	0.705	2.1
100	0.63	0.68	0.727	1.6
120	0.64	0.71	0.745	1.5
140	0.67	0.72	0.761	1.4
160	0.70	0.74	0.776	1.5
180	0.71	0.73	0.788	1.2
200	0.72	0.76	0.825	1.3
300	0.78	0.80	0.846	1.1
500	0.87	0.87	0.908	1.0

Table 4.2. Simulation results on buffer cache hit ratios for random access with Zipfian 80-20 distribution to a disk page pool of N=1000 pages.

OLTP TRACE

- One-hour page reference trace of the production OLTP system of a large bank.

- This trace contained approximately 470,000 page references to a CODASYL database with a total size of 20 Gigabytes.

B	LRU-1	LRU-2	LFU	B(1)/B(2)
100	0.005	0.07	0.07	4.5
200	0.01	0.15	0.11	3.25
300	0.02	0.20	0.15	3.0
400	0.06	0.23	0.17	2.75
500	0.09	0.24	0.19	2.4
600	0.13	0.25	0.20	2.16
800	0.18	0.28	0.23	1.9
1000	0.22	0.29	0.25	1.6
1200	0.24	0.31	0.27	1.66
1400	0.26	0.33	0.30	1.5
1600	0.29	0.34	0.31	1.5
2000	0.31	0.36	0.33	1.3
3000	0.38	0.40	0.39	1.1
5000	0.46	0.47	0.44	1.05

Table 4.3. Simulation results on buffer cache hit ratios using an OLTP trace.

END LRU-K

E0 261