CHAPTER 22

SWAPPING: POLICIES

OUTLINE

The chapter explains briefly about different replacement polices in paging out pages

- Optimal replacement policy
- First-in first-out
- Random
- Least-frequently-used
- Dirty pages
- Thrashing

CACHE MANAGEMENT

Given that main memory hold subset of all pages, it can be seen as cache for virtual memory pages in the system. The goal of replacement policy is to minimize cache misses

Average memory access time (AMAT) equation:

$$AMAT = T_M + (P_{miss} * T_D)$$

T_M : cost of accessing memory

T_D : cost of accessing disk

P_M : probability of cache misses

OPTIMAL REPLACEMENT

- The optimal replacement is the one that leads to fewest-possible cache misses, replacing pages that will be accessed *furthest in the future*
- Simple but difficult to implement
- Used as criteria to know which policies are close to "ideal"

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		0	compulsory misses
1	Miss		0, 1	
2	Miss		0, 1, 2	
0	Hit		0, 1, 2	
1	Hit		0, 1, 2	
3	Miss	2	0, 1, 3	furthest in the future
0	Hit		0, 1, 3	
3	Hit		0, 1, 3	
1	Hit		0, 1, 3	
2	Miss	3	0, 1, 2	
1	Hit		0, 1, 2	

hit rate = 54.5%

FIRST-IN FIRST-OUT (FIFO)

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		First-in→	0
1	Miss		First-in→	0, 1
2	Miss		First-in→	0, 1, 2
0	Hit		First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2
3	Miss	0	First-in→	1, 2, 3
0	Miss	1	First-in→	2, 3, 0
3	Hit		First-in→	2, 3, 0
1	Miss	2	First-in→	3, 0, 1
2	Miss	3	First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2

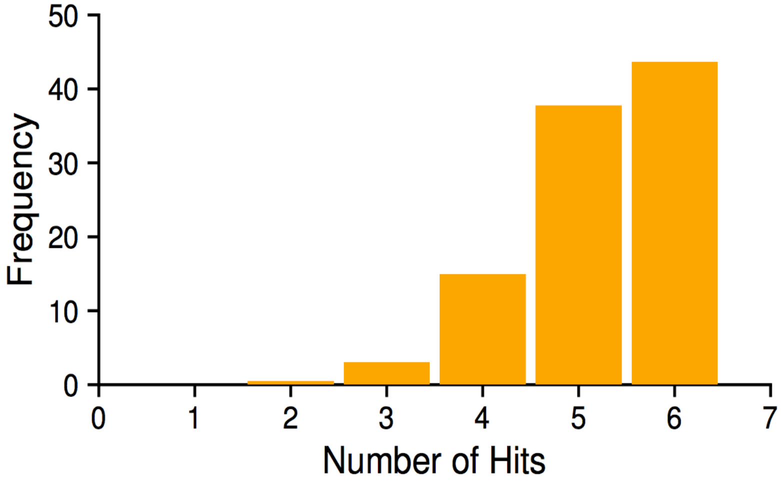
tail

hit rate = 36.4%

- The *first-in page* on the tail of the page queue will be evicted firstly
- Strength: simple to implement
- Weakness:
  - Can't determine the importance of blocks
  - Cache hit rates even get worse when the caches get larger

RANDOM

- Picks a *random* page to replace
- Simple to implement, but depends entirely on luck
- One of its few properties is that it doesn't have corner-case behavior in which hit rate is 0



Random Performance Over 10,000 Trials

USING HISTORY: LEAST RECENTLY USED (LRU)

- The idea: To improve the hit rate based on principle of locality, by looking at *page behavior in the past* to figure out which pages are important, including
  - Frequency: Least-Frequently-Used LFU
  - Recency: Least-Recently-Used (LRU)

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		LRU→	0
1	Miss		LRU→	0, 1
2	Miss		LRU→	0, 1, 2
0	Hit		LRU→	1, 2, 0
1	Hit		LRU→	2, 0, 1
3	Miss	2	LRU→	0, 1, 3
0	Hit		LRU→	1, 3, 0
3	Hit		LRU→	1, 0, 3
1	Hit		LRU→	0, 3, 1
2	Miss	0	LRU→	3, 1, 2
1	Hit		LRU→	3, 2, 1

0,1 have been accessed more recently

WORKLOAD COMPARISON

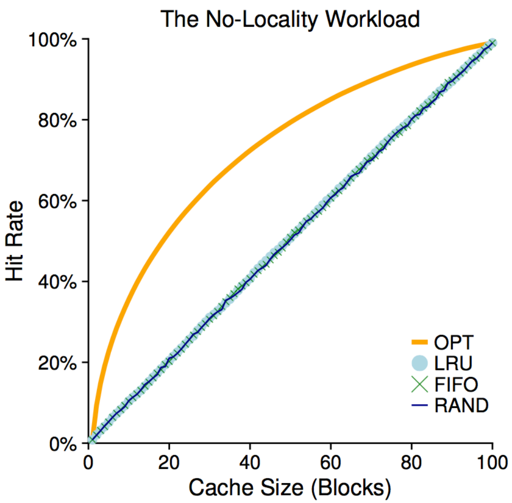


Figure 22.6: **The No-Locality Workload**

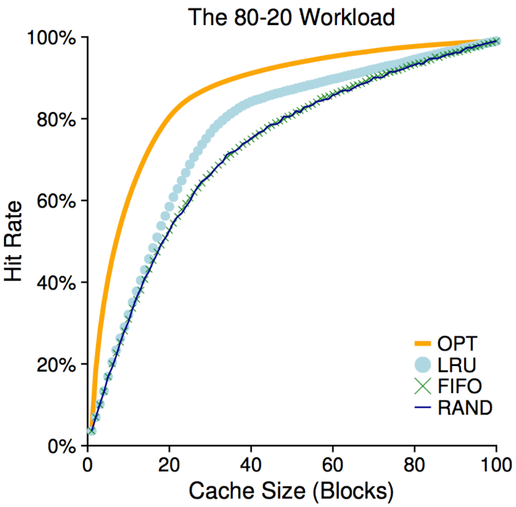


Figure 22.7: **The 80-20 Workload**

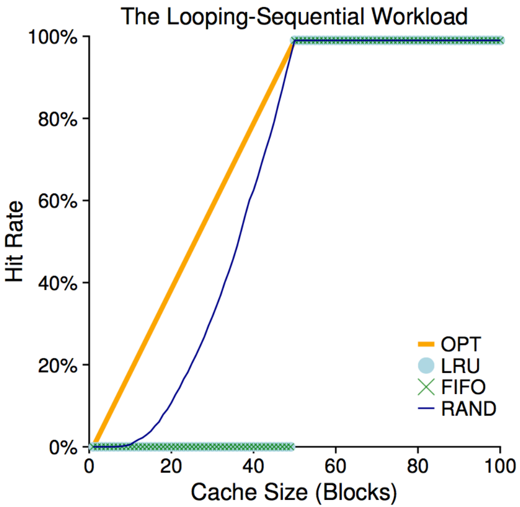


Figure 22.8: **The Looping Workload**

LRU = FIFO = RAND

LRU > FIFO = RAND

RAND > LRU = FIFO

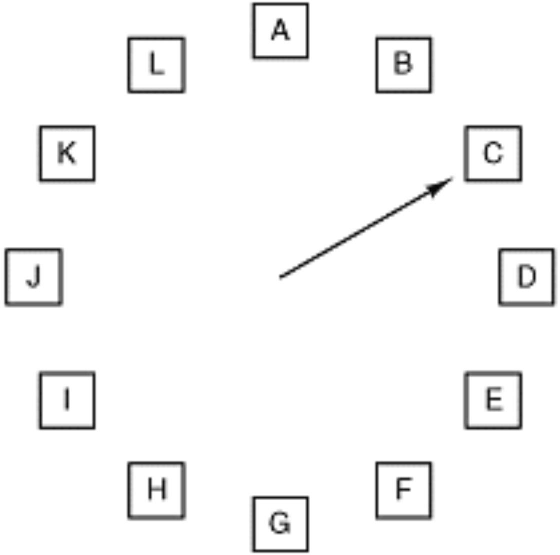
APPROXIMATING LRU

Problem: When the number of pages increases and scanning through the whole pages reduces performance, how can the system find which one is least recently used?

The idea of approximation:

- Use bit (reference bit) is implemented to differentiate which page is recently used and which is not
- 1 use bit per page. *Use bit = 1* when page is referenced (read and write), and 0 when it's not
- Clock algorithm is a simple approach that use bit to approximate LRU

APPROXIMATING LRU



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

- R = 0: Evict the page
- R = 1: Clear R and advance hand

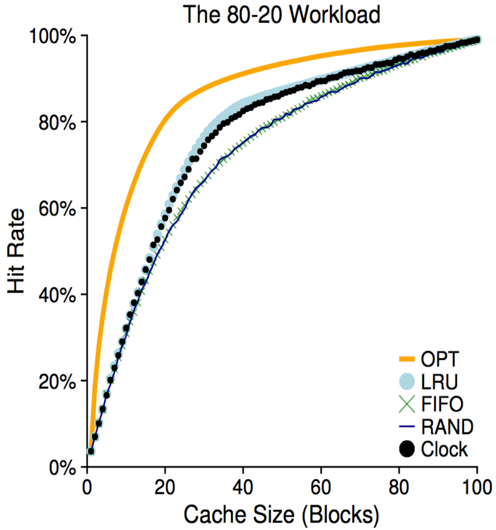


Figure 22.9: **The 80-20 Workload With Clock**

OTHERS TO REPLACEMENT

- Dirty page: page that has been modified in memory, and must be written back to disk to evict

$\Longrightarrow$ In clock algorithm, virtual memory system prefers to evict clean pages (free eviction) over dirty pages (expensive eviction)

- Thrashing happens when memory is oversubscribed, resulting in constant paging of the system

  Admission control approach: to detect run or not to run a set of processes, less but better

  Out-of-memory killer: to kill memory-intensive process

PAGE SELECTION

Page replacement	Page selection
What page to replace	When to bring a page into memory

Page selection policy is to decide when to bring a page into memory

- Demand paging: page is brought to memory when being accesded
- Pre-fetching: fetch page into memory before it is used

Clustering in writing pages out: write out a number of pending pages at one time, rather do it individually

Thank you!!!