Operating System (OS)
CS232

Beyond Physical Memory: Policies

Dr. Muhammad Mobeen Movania

Outlines

- Cache management
- Replacement policies
  - Optimal Replacement Policy
  - FIFO Policy
  - Random
  - LRU
- Experiments on different workloads
- Dirty pages and thrashing
- Summary

Cache Management

- Main memory acts as cache for virtual memory pages
  - We want to minimize cache misses and maximize cache hits
- Calculation metric: AMAT

$$AMAT = T_M + (P_{Miss} \cdot T_D)$$

  - T_M=cost of accessing memory
  - T_D=cost of accessing disk
  - P_{Miss}=probability of a cache miss
- Crux
  - cost of disk access is very high in modern systems that even a tiny miss rate will quickly dominate the overall AMAT of running programs

Optimal Replacement Policy

- A hypothetical policy that uses future to predict the required page
- Impossible to implement but may be used to compare performance of other page replacement policies

Example: Optimal Replacement Policy

- Stream of accessed virtual pages
  - 0,1,2,0,1,3,0,3,1,2,1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
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
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

Figure 22.1: **Tracing The Optimal Policy**

FIFO Replacement Policy

- Pages are placed in a queue when they enter the system
- When a replacement occurs, the page on the head of the queue (the “first-in” page) is evicted
- FIFO is quite simple to implement

Example: FIFO Replacement Policy

- Stream of accessed virtual pages
  - 0,1,2,0,1,3,0,3,1,2,1

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

Figure 22.2: **Tracing The FIFO Policy**

Random Replacement Policy

- Simply picks a random page to replace under memory pressure
- Similar to FIFO, it is simple to implement, but it doesn't really try to be too intelligent in picking which blocks to evict
- Performs better than FIFO

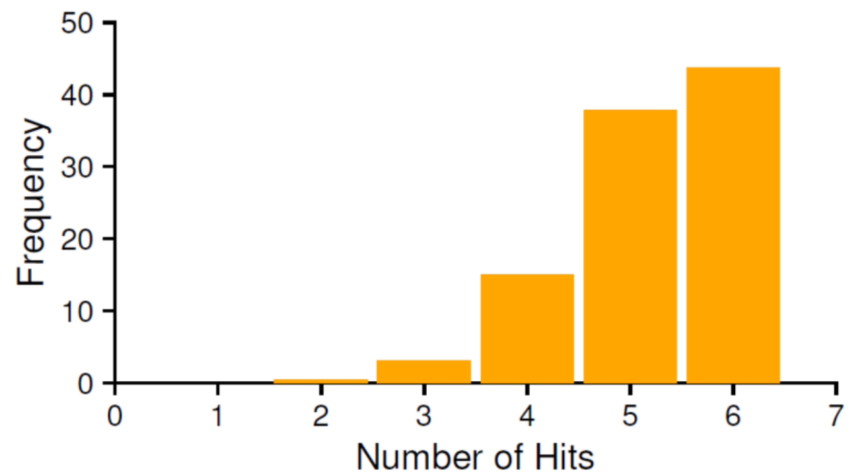

Figure 22.4: **Random Performance Over 10,000 Trials**

Example: Random Replacement Policy

- Stream of accessed virtual pages
  - 0,1,2,0,1,3,0,3,1,2,1

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

Figure 22.3: **Tracing The Random Policy**

LRU Replacement Policy

- Uses historical information (generally termed principle of locality)
  - Can use **frequency**: if a page has been accessed many times, perhaps it should not be replaced as it clearly has some value.
    - **Least-Frequently-Used** (**LFU**) policy replaces the least-frequently used page
  - Can use **recency** of access: the more recently a page has been accessed, perhaps the more likely it will be accessed again.
    - **Least-Recently-Used** (**LRU**) policy replaces the least-recently-used page

Experiments on different workloads

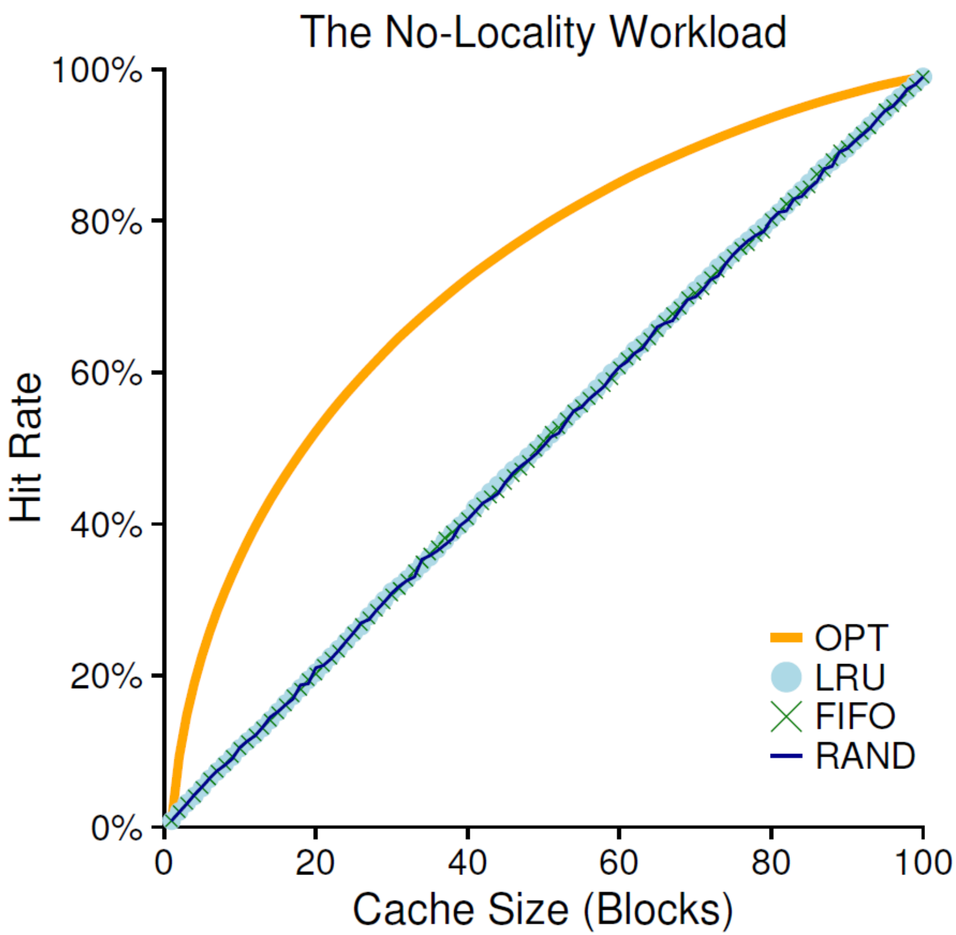

Figure 22.6: **The No-Locality Workload**

Experiments on different workloads

- 80-20 workload uses locality information
  - LRU performs best
  - Random and FIFO perform the same

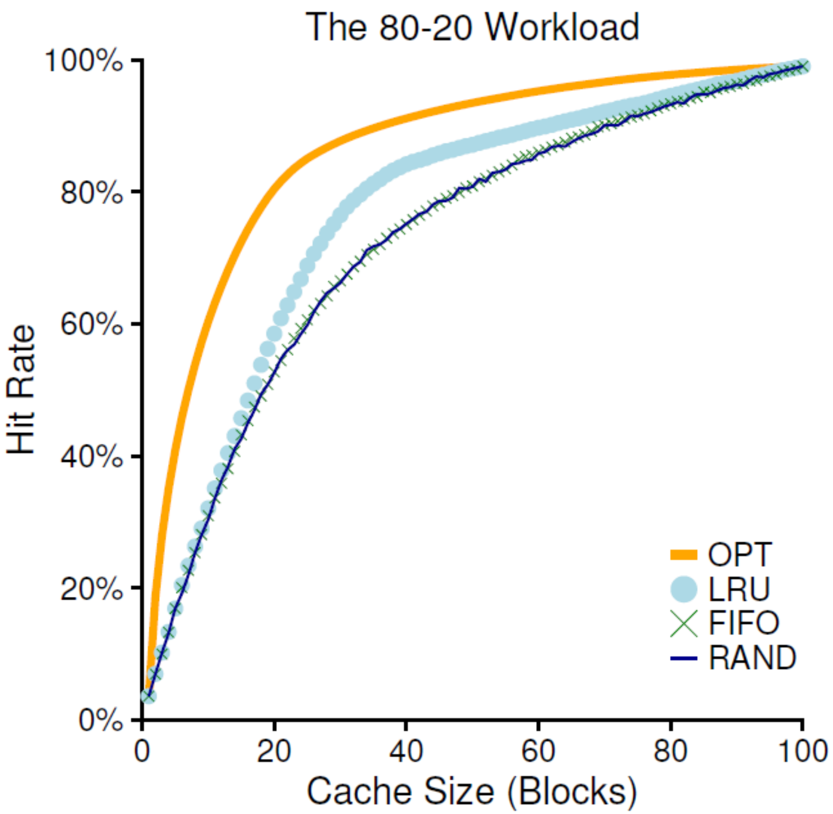

Figure 22.7: **The 80-20 Workload**

Experiments on different workloads

- Looping sequential workload kicks out old pages so they are not available in cache
  - LRU and FIFO performs worst
  - Random performs better

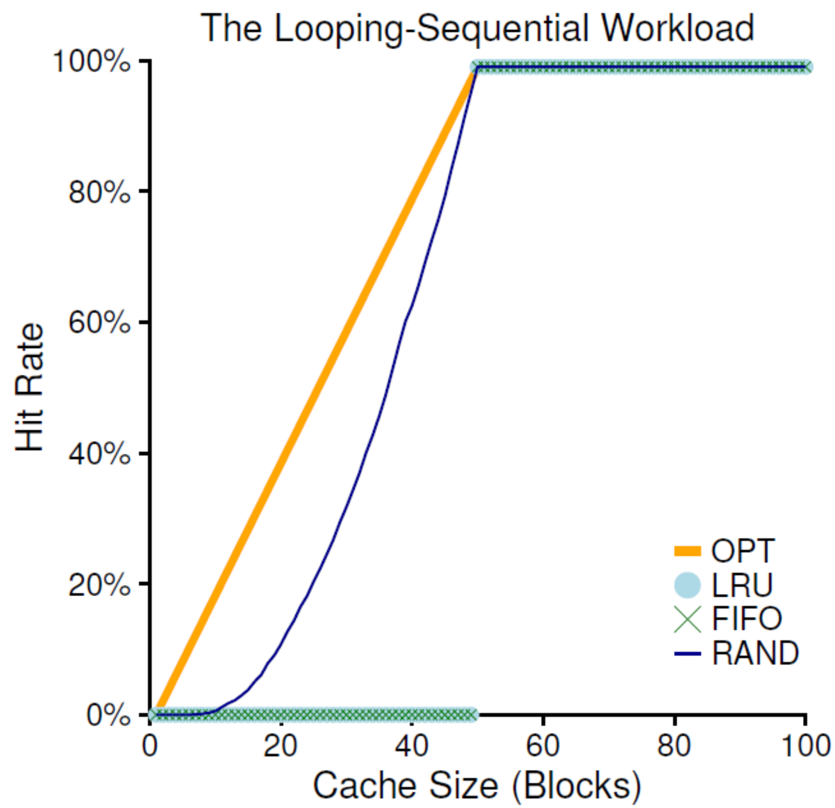

Figure 22.8: **The Looping Workload**

Dirty pages and thrashing

- A page that has been modified is dirty, it must be written back to disk
  - Writing back is costly
- Hardware provides a dirty bit in PTE so that disk writes may be coalesced to improve performance
- **Thrashing:** A condition of constant paging that happens when memory is oversubscribed (processes demanding more memory than available)

Summary

- We have seen the introduction of a number of page-replacement to help decide which page to evict from main memory
- LRU and several variants (like clocks) have been implemented to avoid worst case behavior of LRU
- Paging to disk is expensive, the cost of frequent paging is prohibitive
  - Best solution to excessive paging is to get more memory