



# Cache Basics

Ohad Rodeh



# Caches

---

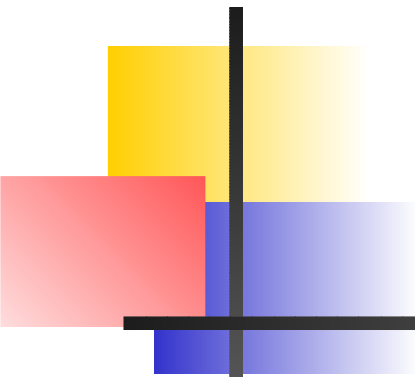
- Processor
  1. Registers
  2. L1, L2, L3
- Memory
  1. NUMA: local memory vs. remote memory
- Disk
  1. Disks today have read caches (8MB)
  2. Some disks also have NVRAM
- Disk arrays have a lot of cache



# Read cache vs. Write cache

---

- Take as an example a file-system client
- The client manages a cache of file-data
- It can:
  1. Not cache anything
  2. Cache read-only data
  3. Cache clean and dirty data



# Non volatile memory (NVRAM)

- Regular RAM connected to a battery
- Can last as long as the battery if electricity shuts down
- NVRAM is used by storage servers to
  1. Improve response times
  2. Safely cache dirty data
- Dirty data cannot be cached for ever
- Once the cache is filled, pages have to be written to disk
- Which is the best page to write?
- How to coalesce pages together to form large extents?
- Better to write a single large extent than many small



# Server cache vs. Storage cache

---

- Example
  1. Database server (Oracle, SQL, DB2)
  2. Connected to a disk
- Both database and disk
  1. Have caches
  2. Perform caching
  3. Cache clean and dirty data
- Where is it best place to cache a particular page?



# Double caching

---

- Ideally, we could use both caches and form one big cache
  1. Push pages down from the database down to the controller
  2. Read them back
- Problem: different systems, different policies
- The Database cache is absorbing most of the workload.
- What kind of caching policy should the disk-array use?
  1. LRU?
  2. LFU?



# To cache or not to cache

- Is caching always a good strategy?
- Should we always cache file data?
- Caching data works well most of the time
- When does it not work?
  - Streaming: playing an mp3 file, playing a movie
- Caching does not help streaming workloads
  1. Need to be able to read large disk extents efficiently
  2. Read-ahead can help



# Least recently used (LRU)

- LRU is a very popular scheme
  1. Simple
  2. Easy to implement
- LRU works well for many workloads
- When does it not work?
  1. Workloads that do not exhibit locality (of course)
  2. Workloads that are frequency based
- For example:
  1. Databases use b-trees
  2. The b-tree access pattern is frequency based
  3. The leaves pollute the cache



# LFU

- *IRM* Independent Reference Model
- Assumes that each page reference is drawn in independent fashion from a fixed distribution over the set of all pages.
- Under IRM, LFU is optimal.
- Problems with LFU:
  - logarithmic implementation complexity in cache size
  - Pays little attention to recent history, does not adapt well
  - Keeps stale pages if their frequency of usage is high

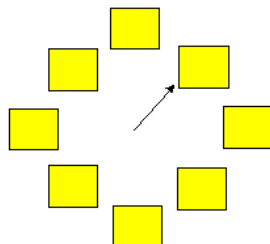


# LRFU

- Least Recently/Frequently Used
- Initially assign a value of  $C(x) = 0$  to every page
- At every time  $t$  update as
  1. if  $x$  is referenced:  $C(x) = 1 + 2^{-\lambda}C(x)$
  2. otherwise:  $C(x) = 2^{-\lambda}C(x)$

# Clock

- Clock is an approximation of LFU
  - Place all cache elements in a circle
  - Each element has a counter
  - Increment the counter on access to an element
- In order to find a victim:
  - Move the clock hand until finding an element whose counter is zero
  - Decrement the counter for each element found on the way





# LRU or LFU?

---

- LRU works best when pages are accessed according to recency
- LFU works best when pages are accessed according to some fixed frequency
- Scan resistance: one time accesses can pollute the cache
- Is there an algorithm that gets the best of both?