Caching principles and paging performance

Professor Hugh C. Lauer
CS-3013 — Operating Systems

(Slides include copyright materials from *Operating Systems: Three Easy Step*, by Remzi and Andrea Arpaci-Dusseau, from *Modern Operating Systems*, by Andrew S. Tanenbaum, 3rd edition, and from other sources)

Fundamental observation

- **Paging allows us to treat physical memory as a *cache* of virtual memory**
- **Therefore, all of the principles and issues of caches apply to paging**
- **… especially paging performance**

Definition — *cache*

- **A small subset of active items held in small, *fast* storage while most of the items remain in much larger, *slower* storage**

- **Includes mechanisms for**
  - Recognizing what items are in the cache and for accessing them quickly
  - Bringing things into cache and throwing them out again

Note on caches and caching

- This topic is not adequately covered in Tanenbaum or most other Operating System textbooks
- Silbershatz, Galvin, & Gagne discuss paging performance somewhat
- It is treated extensively in Computer Architecture textbooks

Caching is a fundamental, cross-cutting concept with broad application to all areas of computing science and to many areas of system design, embedded systems, etc.

Paging — *two* examples of caches

- **Paged Virtual Memory**
  - Very large, mostly stored on (slow) disk
  - Small *working set* in (fast) RAM during execution

- **Page tables**
  - Very large, mostly stored in (slow) RAM
  - Small *working set* stored in (fast) TLB registers

Caching is ubiquitous in computing

- **Transaction processing**
  - Keep records of today's departures in RAM or local storage while records of future flights are on remote database
- **Program execution**
  - Keep the bytes near the current program counter in on-chip memory while rest of program is in RAM
- **File management**
  - Keep disk maps of open files in RAM while retaining maps of all files on disk
- **Game design**
  - Keep nearby environment in cache of each character
- **…**

Caching issues

This is a very important list!

- When to put something in the cache
- What to throw out to create cache space for new items
- How to keep cached item and stored item in sync after one or the other is updated
- How to keep multiple caches in sync across processors or machines
- Size of cache needed to be effective
- Size of cache items for efficiency
- …

General observation on *caching*

- **We create caches because**
  - There is not enough fast memory to hold everything we need
  - Memory that *is* large enough is too slow

- **Performance metric for all caches is *EAT***
  - *Effective Access Time*

- **Goal is to make overall performance close to cache memory performance**
  - By taking advantage of *locality* — temporal and spatial
  - By burying a small number of accesses to slow memory under many, many accesses to fast memory

Definition – Effective Access Time (EAT)

- **The *average access time* to memory items, where some items are cached in fast storage and other items are not cached…**

- **…weighted by *p*, the *fault rate***
  - $0 \le p < 1$

Computer architecture textbooks use *Average Access Time* = (*cache access time*) + p * (*miss penalty*)

- ***EAT* = (*1-p*) * (*cache access time*) +**
  ***p* * (*non-cache access time*)**

Goal of Caching

- **To take advantage of locality to achieve *__nearly__* the same performance of the *fast* memory when most data is in *slow* memory**

- **I.e., solve *EAT* equation for *p***

$$(1-p)*(cache\_time) + p*(non\_cache\_time) < (1+x)*(cache\_time)$$

EAT

***x* is "acceptable" performance penalty**

$$p < x * \frac{cache\_time}{(non\_cache\_time - cache\_time)}$$

Goal of Caching (continued)

- Select *size of cache* and *size of cache items* so that *p* is low enough to meet acceptable performance goal

- Usually requires simulation of a suite of benchmarks

Application to Demand Paging

- ***Page Fault Rate (p)***

  $0 \leq p < 1.0$ (measured in average number of faults / reference)

- ***Page Fault Overhead***

  = *fault service time* + *read page time* + *restart process time*

  - *Fault service time* ~ 0.1–10 μsec
  - *Restart process time* ~ 0.1–10–100 μsec
  - *Read page time* ~ 8-20 *milli*seconds!

- **Dominated by time to read page in from disk!**

Demand Paging Performance (continued)

- ***Effective Access Time*** **(*EAT*)**

  = (*1-p*) * (*memory access time*) +
  p * (*page fault overhead*)

- **Want *EAT* to degrade no more than, say, 10% from true *memory access time***
  - i.e., *EAT* < (*1 + 10%*) * *memory access time*

Performance Example

- ***Memory access time*** **= 100 nanosec = 10^{-7}**
- ***Page fault overhead*** **= 25 millisec = 0.025**
- ***Page fault rate*** **= 1/1000 = 10^{-3}**

- **$EAT = (1-p) * 10^{-7} + p * (0.025)$**

  $= (0.999) * 10^{-7} + 10^{-3} * 0.025$

  $\cong$ 25 microseconds per reference!

- **I.e.,**
  - *250 * memory access time*!

Performance Goal

- **To achieve less than 10% degradation**

  $$(1-p) * 10^{-7} + p * (0.025) < 1.1 * 10^{-7}$$

  i.e.,

  $$p < (0.1 * 10^{-7}) / (0.025 - 10^{-7})$$

  $$\cong 0.0000004$$

- ***I.e.,***

  1 fault in 2,500,000 accesses!

Working Set Size

- **Assume average swap time of 25 millisec.**

- **For *memory access time* = 100 nanoseconds**
  - Require < 1 page fault per 2,500,000 accesses
- **For *memory access time* = 1 microsecond**
  - Require < 1 page fault per 250,000 accesses
- **For *memory access time* = 10 microseconds**
  - Require < 1 page fault per 25,000 accesses

Object Lesson

- **Working sets *must* get larger in proportion to memory speed!**
    - Disk speed ~ constant (nearly)

- **I.e., faster computers *need* larger physical memories to exploit the speed!**

Class Discussion

1. What is first thing to do when the PC you bought last year becomes too slow?
2. What else might help?

3. Can we do the same analysis on TLB performance?

TLB fault performance

- **Assumptions**
  - *m = memory access time* = 100 nsec
  - *t* = TLB *load time from memory* = 300 nsec = 3 * *m*
- **Goal is < 5% penalty for TLB misses**
  - I.e., *EAT* < 1.05 * *m*
- **How low does TLB fault rate need to be?**

TLB fault performance

- **Assumptions**
  - *m = memory access time* = 100 nsec
  - *t* = TLB *load time from memory* = 300 nsec = 3 * *m*
- **Goal is < 5% penalty for TLB misses**
  - I.e., *EAT* < 1.05 * *m*

- $EAT = (1-p) * m + p * t < 1.05 * m \Rightarrow$

  $p < (0.05 * m) / (t - m)$

  $= 0.05 * m / (2 * m)$

  $= 0.025$

- **I.e., TLB fault rate should be < 1 per 40 accesses!**

TLB fault performance (continued)

- **Q: How large should TLB be so that TLB faults are not onerous, in these circumstances?**

- **A: Somewhat less than 40 entries**
  - Assuming a reasonable degree of locality!

Note: pathological cases may prevent goal altogether!

What if Software Loaded TLB?

- E.g., with hashed or inverted page tables?
- Assume TLB load time is 100 * *m*
- *Work out on white board*

$$p < x * \frac{cache\_time}{(non\_cache\_time - cache\_time)}$$

Summary of this Topic

- **A quantitative way of estimating how large the *cache* needs to be to avoid excessive thrashing, where**
  - *Cache* = Working set in physical memory
  - *Cache* = TLB size in hardware

- **Applicable to all forms of *caching***

General Observation on *Caching*

- **We create caches because**
  - There is not enough fast memory to hold everything we need
  - Memory that *is* large enough is too slow

- **Performance metric for all caches is EAT**
  - *Effective Access Time*

- **Goal is to make overall performance close to cache memory performance**
  - By taking advantage of *locality* — temporal and spatial
  - By burying a small number of accesses to slow memory under many, many accesses to fast memory

Cache Applications

- ***Physical memory:*** **cache of** ***virtual memory***
  - I.e., RAM over disk
- ***TLB:*** **cache of** ***page table entries***
  - I.e., Registers over RAM
- **Processor** ***L2 cache:*** **over RAM**
  - I.e., nanosecond memory over 10’s of nanoseconds
- **Processor** ***L1 cache:*** **over L2 cache**
  - I.e., picosecond registers over nanosecond memory
- **…**

Cache Applications (continued)

- **Recently accessed blocks of a file**
  - I.e., RAM over disk blocks
- **Today's airline flights**
  - I.e., local disk over remote disk

Questions?