Prefetching and Caching

- "Application Controlled File Caching" USENIX'94, OSDI'94, TOCS'96: Cao, Felten, Li
- "Informed Multi-Process Prefetching and Caching" Sigmetrics'97: Tomkins, Patterson, Gibson

Caching and Prefetching

- Target: improve cache hit ratio
- Two-level policies
  - *allocation*: kernel allocates physical pages to individual applications
  - *replacement*: each application suggests which page to be replaced
- Better *local* and *global* performance

Two-Level Placement

Basic Idea

- LRU not best for all cases
- New: application-tailored cache replacement policy
- Intelligent (and not harmful) decisions
- Processes should not hurt each-other
- Allocation/replacement should work across processes
- Basic principles
  1. Ordinary processes (i.e., unwilling or unable to predict their own access should perform *at least* as well as with LRU
  2. Ensure that a malicious process cannot hurt the performance of other processes
  3. Decisions should be made such that performance improves

First-Try Policy

- Same as global LRU
- When a replacement is necessary, the block at end of the LRU list is suggested as a candidate and its owner process is asked to give up a block
- Problem: if the owner process overrules the kernel, the candidate block still stays at the end of the LRU list and on the next miss the same process will be asked to give up a block
- Example

Swapping

- First swap, then replace
- Process is no longer hurt by its choice
- Global hit ratio is the same as or better than it would be under global LRU
- Example

Place-Holders

- Idea: prevent foolish processes from acquiring extra cache
- Place-holder refers to a page and records which block would have occupied that page under the global LRU page
- A process may hurt itself by a foolish decision, but does not hurt anyone else.
- Example

Putting it all together

- If cache hit, then the block is moved to the head of global LRU list (if there is a placeholder pointing to the block, it is deleted)
- If cache miss, then
  - if there is a place-holder for the block b pointing to block b_1, then b_1 is replaced and its page is given to b.
  - if there is NO place-holder for b, then the kernel finds the block at the end of the LRU list and the kernel consults the process which block to replace, it swaps it with the LRU one, etc...
- It never asks a process to replace a block for another process more often than global LRU
- When a process makes a bad decision, only that process is punished and the state is restored (i.e., place-holders help to recover from errors)

Other issues...

- Shared files? have a single process acting as manager, else let the kernel decide
- Cache size?
- Access patterns?
- Heuristics?
  - use suffix: .ps are accessed sequentially, .o are accessed in certain sequence
  - lseeks (from compilers)
  - hints
- Replace the one farthest in the future...
- Implementation

- Key: applications that disclose patterns
  - hints (API)
  - compilers
- Other issues...
  - manage prefetching and caching within each stream of hinted accesses
  - allocation problem of dividing disk and cache resources among many access streams
  - prefetch depth?
  - cost-benefit framework (TIPTOE, Tomkins et. al.)
  - access pattern
  - not necessarily regular
  - how many times do you prefetch the same missing block?

- (want to avoid thrashing)
- balance I/O workload across disks
- identify "constrained" disks
  - prefetch when a disk does idle
  - issue multiple fetches to benefit from disk reordering (cscan)
  - proximity in hints is a good indicator of disk proximity
- But, distributions is also an issue...

- Simulation study
- Singe and multiprocess (up to two) results
- Improvements
- Lessons
  - Access rates must be related to cache reuse
  - Unconstrained disks are not a problem: prefetching is not as critical there
  - Constrained disks, caching is very important (be careful what to evict from cache)
  - Eviction decisions impact locality of re-fetched data
  - Deeper disk queues result in lower average disk service times
  - Over-aggressive prefetching can result in eviction of prefetched but unread data.

- Main contribution
  - adapt prefetch aggressiveness according to disk load
  - use estimates of benefit