# Disk I/O-Intensive Online Servers

**CS 256/456**

**Dept. of Computer Science, University of Rochester**



## Motivation

- Performance of most CPU-bound workloads has exceeded what is needed
  - throughput of a Web server when all data is in memory?
- Server performance when the data size far exceeds the available memory
  - caching is effective in this case.
  - throughput of a Web server when all data resides on disk?



## Targeted Application Characteristics

- High-concurrent online servers
- Read-only request processing
- Contain significant sequential access pattern
- Examples:
  - Web/ftp servers
  - index searching

    Fine: list of Web pages matching "Fine"

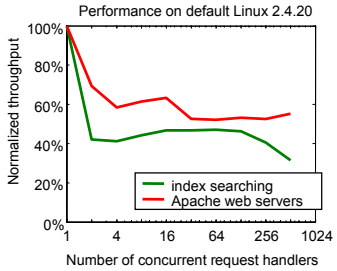
    Arts: list of Web pages matching "Arts"

    ... ... ... ...
  - database applications (with complex table joins)



## Problem Description

- Request handler often reads data in chunks because:
  - application semantics
  - limited buffer space
  - memory-mapped I/O
- The problem:
  - frequent I/O switching (disk seeks) under concurrent workload



## Improving the I/O Efficiency

- Anticipatory scheduling [Iyer & Druschel, SOSP 2001]
  - at the completion of an I/O request, the disk scheduler will wait a bit (despite there is other work to do), in anticipation that a new request with strong locality will be issued.
  - there is a timeout associated with this wait, and the disk scheduler would go ahead to schedule another request if no such new request appears before timeout.
- Anticipatory scheduling is ineffective when each individual process performs non-sequential I/O (e.g., interleaving I/O).
- Another way to reduce the I/O switching frequency
  - more aggressive I/O prefetching/read-ahead



## Aggressive Prefetching

- Linux 2.4 read-ahead
  - for sequential access stream, the prefetching size starts at 3 pages and grows at a semi-exponential fashion until it reaches 32 pages
  - 3, 7, 13, 25, 32, 32, 32, 32 pages, … …
- Pitfalls of over-aggressive prefetching
  - kernel-level prefetching may retrieve unneeded data
    - magnified by aggressive prefetching
  - increasing memory contention
    - magnified by high server concurrency
- Must balance I/O efficiency with these pitfalls



## Increased Memory Contention

- Prefetching-incurred page thrashing
  - aggressive prefetching creates higher memory contention
  - magnified by high execution concurrency in online servers
  - at some point, a prefetched page may be evicted before being accessed
- page thrashing
  - waste on fetching pages repeatedly
  - holes in prefetch streams that require small-granularity I/O



## Managing Prefetching Memory

- Prefetch memory
  - memory pages that were prefetched but not yet accessed
- Which page should we evict when there is memory pressure?
  - access history/frequency-based policies (e.g., LRU or LFU) make no sense since no pages in the pool have even been accessed
  - LRU according to access history on prefetching streams instead of on pages.