More on Server System Management

CS 256/456

Dept. of Computer Science, University of Rochester

Server System Management

- Multi-processing vs. multi-threading
  - overhead vs. fault isolation
- User threads
  - blocking problem for servers using user threads
- Event-driven servers
  - all user-level management, no synchronization overhead
- Problem with high concurrency
  - more synchronization overhead, more context switches, poor system call scalability
- Server overload
  - if requests have to be abandoned, abandon them ASAP

Isolation in Server Systems

- Isolation of request execution in
  - resource accounting
  - fault protection
  - resource provisioning
  - system configuration
- Challenges
  - what is the existing OS principal for resource accounting and fault isolation?
  - challenge #1: process/thread does not complete encapsulate a request execution
  - challenge #2: lack of mechanisms for isolation in resource provisioning, fault protection, and system configuration

Request-granularity Resource Accounting

- Problems:
  - request processing over multiple thread/process
  - thread/process pooling
  - resource accounting for interrupt handlers
- Request execution encapsulation in OS:
  - Resource containers [Banga et al., OSDI 1999]
  - Magpie [Barham et al., OSDI 2004]

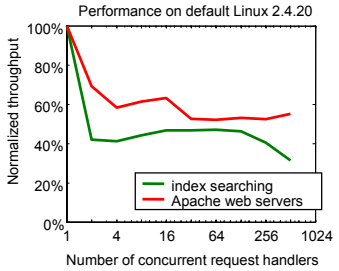
Isolation Using Virtual Machines

- Virtual machines (Vmware, Xen) allow
  - strong fault isolation
  - flexible resource provisioning
  - customized system configuration
- Issue
  - excessive overhead when there are many virtual machines
- Coarse-grain isolation in service hosting centers
- Light-weight virtual machines
  - Denali [Witaker et al., OSDI 2002]

Background for Data-intensive Servers

- 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 not very effective in this case.
  - throughput of a Web server when all data resides on disk?

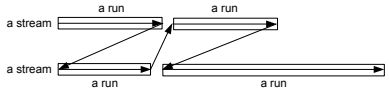
Problem Description



- The problem:
  - frequent I/O switching (disk seeks) under concurrent workload

Improving the I/O Efficiency

- Anticipatory scheduling [Iyer & Druschel, SOSP 2001]
  - when an I/O request completes, the 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 with strong locality appears before timeout.
- Anticipatory scheduling is ineffective when each individual process performs interleaving I/O.



Aggressive Prefetching

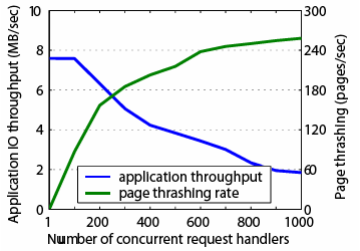
- Aggressive prefetching: another way to reduce the I/O switching frequency
- Pitfalls of over-aggressive prefetching
  - kernel-level prefetching may retrieve unneeded data
    - magnified by aggressive prefetching
  - increasing memory contention
    - magnified by high server concurrency
- Must balance I/O efficiency with these pitfalls
- Linux 2.4 read-ahead for sequential access stream
  - 3, 7, 13, 25, 32, 32, 32, 32 pages, … …

Competitive Prefetching

- The problem:
  - we do not know exactly how much data is needed by the application ahead of time.
  - balance the efficiency of large-granularity I/O and the overhead of retrieving unneeded data
- Competitive prefetching [Li et al., OASIS 2004]
  - when the prefetching size is equal to the amount of data that can be transferred within a single seek/rotation time, the total disk consumption is at most twice that of the optimal offline strategy
  - provides a worst-case performance bound
  - competitive prefetching size in practice
    - average seek time 6.3ms; average rotation delay 3ms; average transfer rate 53.7MB/sec

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



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 [Li and Shen, FAST2005]
    1. Pages whose owner request handler has exited
    2. Last page from the longest prefetch stream
    3. Last page from the least recently accessed prefetch stream