

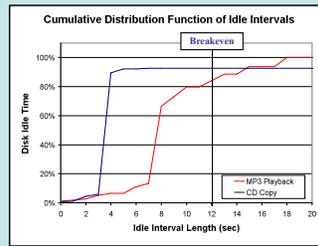
Energy Efficient Prefetching and Caching

Athanasios E. Papathanasiou and Michael L. Scott
Department of Computer Science
University of Rochester

Prefetching and caching are standard practice in modern file systems. They serve to improve performance--to increase throughput and decrease latency--by eliminating as many I/O requests as possible, and by spreading the requests that remain as smoothly as possible over time, resulting in relatively short intervals of inactivity. This strategy ignores the goal of energy efficiency so important to mobile systems, and in fact can frustrate that goal. Magnetic disks, network interfaces, and similar devices provide low power modes that save energy only when idle intervals are relatively long. A smooth access pattern can eliminate opportunities to save energy even during such light workloads as MPEG and MP3 playback. In contrast a bursty access pattern can improve energy efficiency in several important cases without significantly affecting performance, if implemented carefully.

Motivation

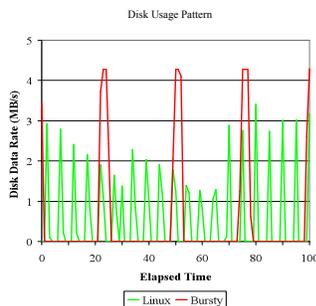
- Power Efficient Devices:
 - ✓ Save energy by exploiting idle time
 - ✓ Require long idle intervals
 - ✓ Remain in low power mode for a minimum period: **Breakeven** point
- Modern operating systems:
 - ✓ Maximize throughput
 - ✓ Minimize latency
- **What about energy?**
 - ✓ Idle times too short to exploit for savings



- File System behavior examples:
 - ✓ MP3 Playback (300 seconds)
 - Disk idle time: 291 seconds
 - 66% shorter than 8 seconds
 - ✓ CD copy (1359 seconds)
 - Disk idle time: 1191 seconds
 - 92% shorter than 5 seconds
- Intuitively increased system memory
 - ✓ Leads to **reduced** disk energy consumption
 - **But: 8-fold** memory increase
 - Practically **no** energy savings

New OS Design Goal: Increase Burstiness

- Maximize Energy Efficiency
 - ✓ Maximize idle interval length to allow a power state transition
 - ✓ Operate at max disk bandwidth when disk is active
 - ✓ Decrease number of transitions



- Design Guidelines
 - ✓ Maximize idle phases
 - Aggressive, speculative prefetching
 - Bursty periodic update
 - ✓ Coordinate I/O across applications
 - ✓ Preactivate disk in anticipation of next use

Prototype

- Epoch-Based Extensions to Linux Memory Management System
- Two Phases per Epoch
 - ✓ Request Generation Phase
 - Estimate memory size for prefetching
 - Predict and prefetch
 - ✓ Idle Phase
 - Estimate time to next request
 - Power disk down if possible
 - Schedule preactivation
- Deciding what to prefetch
 - ✓ Sequential Accesses
 - Detect pattern & rate -- Prefetch accordingly
 - ✓ Random or multiple-file accesses
 - Improve prefetching accuracy through hints
 - Prefetch very **speculatively**
- Estimating Memory for Prefetching
 - ✓ Extend LRU with **Prefetch Cache**
 - ✓ First miss determines Prefetch Cache size
 - **Compulsory Miss**: No change
 - **Prefetch Miss**: Increase by constant
 - **Eviction Miss**: Decrease by number of useful pages evicted in favor of prefetching
 - ✓ **Eviction Cache**: Stores eviction history
- Coordinating across applications
 - ✓ **Prefetch Thread**
 - Prefetching requests for all applications
- Bursty Periodic Update
 - ✓ Application discloses reliability constraints

Experimental Evaluation

