VM and I/O

IO-Lite: A Unified I/O Buffering and Caching System
Vivek S. Pai, Peter Druschel, Willy Zwaenepoel

Software Prefetching and Caching for TLBs
Kavita Bala, M. Frans Kaashoek, William E. Weihl
General themes

• CPU, network bandwidth increasing rapidly
• Main memory, IPC unable to keep up
  – trend towards microkernels increase number of IPC transactions
General themes

- CPU, network bandwidth increasing rapidly
- Main memory, IPC unable to keep up
  - trend towards microkernels increase number of IPC transactions

One remedy is to increase speed/bandwidth of IPC data (data moving between processes)
fbufs

- Attempts to increase bandwidth within network subsystem
- In a nutshell: provides immutable buffers shared among processes of subsystem
- Implemented using shared memory and page remapping in a specialized OS: the x-kernel
fbuf, details

- Incoming “packet data units” passed to higher protocols in fbufs
- PDUs are assembled into “application data units” by use of an aggregation ADT
fbufs, details

• fbuf interface does not support writes after producer fills buffer (PDU)
  – fbufs can be reused after consumer is finished; leads to *sequential* use of fbufs
  – applications shouldn’t have to modify data anyway
fbufs, details

• fbuf interface does not support writes after producer fills buffer (PDU)
  – fbufs can be reused after consumer is finished; leads to sequential use of fbufs
  – applications shouldn’t have to modify data anyway
  – LIMITATION, especially in a more general system
Enter IO-Lite

• Take fbufs, but make them
  – more general, accessible to the filesystem in addition to the network subsystem
  – more versatile, usable on standard OSes (not just x-kernel)

• Solves a more general problem: rapidly increasing CPUs (not just network bandwidth)
Before comparing them to fbufs...

- Problems in the “old way” of doing things
  - redundant data copying
  - redundant copies of data lying around
  - no special optimizations between subsystems
IO-Lite at a high level

- IO-Lite must provide system-wide buffers to prevent multiple copies
  - UNIX allocates filesystem buffer cache from different pool of kernel memory than, say, network buffers and application-level buffers
Access Control Lists

• Processes must be granted permission to view buffers
  – each buffer pool has an ACL for this purpose
  – for each buffer space, list of processes granted permission to access it
Consequence of ACLs

• Producer must know data path to consumer
  – gets slightly tricky with incoming network packets
  – must use *early demultiplexing* (mentioned as a common enough technique)
Buffers:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLs:</td>
<td>P1, P2</td>
<td>P1, P3, P4</td>
<td>P4</td>
</tr>
</tbody>
</table>

P1: file system
P2: CGI
P3: web server
P4: TCP/IP
Buffers: | 1 | 2 | 3
---|---|---|---
ACLs: | P1, P2 | P1, P3, P4 | P4

P1: file system
P2: CGI
P3: web server
P4: TCP/IP
Pipelining

- Abstractly represents good modularity
- Conceptually data *moves* through pipeline from producer to consumer
- IO-Lite comes close to implementing this in practice
  - when the path is known ahead of time, context switches are the biggest overheads in pipeline
immutable --> mutable

• Data in an OS must be manipulated in various ways
  – network protocols (same as fbufs)
  – modifying cached files (i.e., to send to various clients via a network/writing checksums)

• IO-Lite must support *concurrent* buffer use among sharing processes
immutable --> mutable

IO-Lite

fbufs
immutable --> mutable

File Cache

Buffer 1

Buffer
Aggregate
(in user process)
immutable --> mutable
immutable --> mutable
Consequences of mutable bufs

• Whole buffers are rewritten
  – same as if there was no IO-Lite -- same penalty as a data copy

• Bits and pieces of files are rewritten
  – what this system was designed for -- ADT handles modified sections nicely

• Too many bits and pieces are rewritten
  – IO-Lite uses mmap to make it contiguous -- usually results in a kernel memory copy
Evicting I/O pages

• LRU policy on unreferenced bufs (if one exists)
• Otherwise, LRU on referenced bufs
  – since bufs can have multiple references, might require multiple write-backs to disk
• Tradeoff between size of I/O cache and size of VM pages
  – greater than 50% replaced pages are IO-Lite, evict one to reduce the number
The bad news

- Applications must be modified to use special IO-Lite read/write calls
- Both applications at either end of a UNIX pipe must use library to gain benefits of IO-Lite’s IPC
The good news

• Many applications can take further advantage of IPC
  – computing packet checksums only once
The good news

• Many applications can take further advantage of IPC
  – computing packet checksums only once

<generation #, addr> --> I/O buf data
Flash-Lite

• Flash web server modified to use IO-Lite
• HTTP
  – up to 43% faster than Flash
  – up to 137% faster than Apache
• Persistent HTTP (less TCP overhead)
  – up to 90% network saturation
• Dynamic pages have advantage because of IPC between server and CGI program
HTTP/PHTTP

Figure 3: HTTP

Figure 4: Persistent HTTP
PHTTP with CGI

Figure 6: P-HTTP/FastCGI
Something else fbufs can’t do

- Non-network applications
- Fewer memory copies across IPC
On to prefetching/caching…

• Once again, CPU speeds far exceed main memory speeds

• Tradeoff
  – prefetch too early --> less cache space
  – cache too long --> less room for prefetching

• Try to strike a balance
Let’s focus on the TLB

- Microkernel modularity pays a price: more TLB misses
- Solution in software -- no hardware mods
- Handles only kernel misses -- 50% of total
user addr space
next level of page tables

user page tables

kernel data structs

Unmapped Physical Memory

Mapped Kernel VM

Mapped User VM

user addr space
next level of page tables

user page tables

kernel data structs

user addr space
Prefetching

- Prefetch on IPC path
  - concurrency in separate domains increases misses
  - fetch L2 mappings to process stack, code, and data segments

- Generic trap handles misses first time, caches them in flat PTLB for future hash lookups
Caching

• Goal: avoid cascaded misses in page table
  – entries evicted from TLB are cached in STLB
  – adds 4-cycle overhead to most misses in general trap handler

• When using STLB, don’t prefetch L3
  – usually evicts useful cached entries

• In fact, using both caching + prefetching only improves performance if have a lot of IPCs, such as in servers
Performance -- PTLB

Kernel TLB misses (thousands)

- L1K, Mach
- L1K, Mach+PTLB
- L2, Mach
- L2, Mach+PTLB
- L3, Mach
- L3, Mach+PTLB

Kernel TLB penalties (seconds)

- Mach
- Prefetching Overhead
- Mach+PTLB

mpeg play  jpeg play  video play  IOzone  ouster-out  mab

mpeg play  jpeg play  video play  IOzone  ouster-out  mab
Performance -- overall
Performance -- overall

BUT NO OVERALL GRAPH GIVEN FOR NUMBER OF PENALTIES
Amdahl’s Law in action

- Overall performance only marginally better

<table>
<thead>
<tr>
<th>Application</th>
<th>Mach</th>
<th>PTLB+ STLBB</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpeg_play</td>
<td>124.6</td>
<td>18.4</td>
<td>1.09%</td>
</tr>
<tr>
<td>jpeg_play</td>
<td>13.0</td>
<td>1.6</td>
<td>0.27%</td>
</tr>
<tr>
<td>video_play</td>
<td>108.0</td>
<td>11.4</td>
<td>3.04%</td>
</tr>
<tr>
<td>IOzone</td>
<td>3.4</td>
<td>0.6</td>
<td>0.99%</td>
</tr>
<tr>
<td>ousterhout</td>
<td>24.0</td>
<td>3.4</td>
<td>1.65%</td>
</tr>
<tr>
<td>mab</td>
<td>52.0</td>
<td>13.6</td>
<td>0.25%</td>
</tr>
</tbody>
</table>
Summary

- Bridging the gap between memory speeds and CPU is worthwhile
- Microkernels have fallen out of favor
  - but could come back
  - relatively slow memory is still a problem
- Sharing resources between processes without placing too many restrictions on the data is a good approach