A Framework for Virtual Device Driver Development & Virtual Device-Based Performance Modeling

Zachary H. Jones

Ph.D. Defense – Nov. 12, 2010

Committee

R. M. Geist (Chair), J. M. Westall, B. C. Dean, and H. C. Grossman

Acknowledgements

This research has been supported by

- IBM Faculty Awards
- IBM Ph.D. Fellowship
- NSF Grant Award 0722313

Thesis Statement

We will provide solutions to problems in operating system virtualization that have been motivated by major projects in CPSC 822.

Motivation

- CPSC 822, Operating System Design: A Case Study
  - Second level, graduate OS course
  - Modify schedulers for performance
  - Build new kernels
  - Write drivers for real devices
- Limitations:
  - Dedicated hardware required (usually crashed)
  - Limited enrollment
  - Waiting list, every semester
- Standard evaluation (5 yrs. out): the most valuable course of educational career

Virtualization?

- A large part of the course could be virtualized (VMWare, XEN, KVM)
- Oops! Two important projects resist this:
  - Write a driver for non-trivial graphics card (interrupts, DMA, buffer handling, memory mapping)
  - Design a new disk scheduler that outperforms default Linux schedulers
- How do we manage & deploy a virtualized OS lab?
  - Ensure accurate performance measurements
- We need to simplify the process of providing unique non-trivial graphics card each semester.

The Framework

Card Gen

VPCI | VPT | V822L

Kernel Probes & Iprobe | KVM

Linux Kernel

Kernel Probes

- Debug mechanism to monitor events in a system.
  - Probe almost any instruction in the kernel
  - Pre-handler runs, execute probed instruction, post-handler runs
- Three flavors
  - kprobe
  - jprobe
  - kretprobe
- Caveat: Probes cannot be attached to inline functions or functions declared with ___*kprobes*

- kprobe utility lacks ability to dynamically replace any kernel function...

Intercept Probe (iprobe)

- Use jprobe to check function parameters
- If caller flagged as accessing VPCI device, intercept
  - Temporarily replace probed instruction with *nop*
  - Kprobe post-handler modifies the IP to address of replacement function.

Virtual PCI Framework

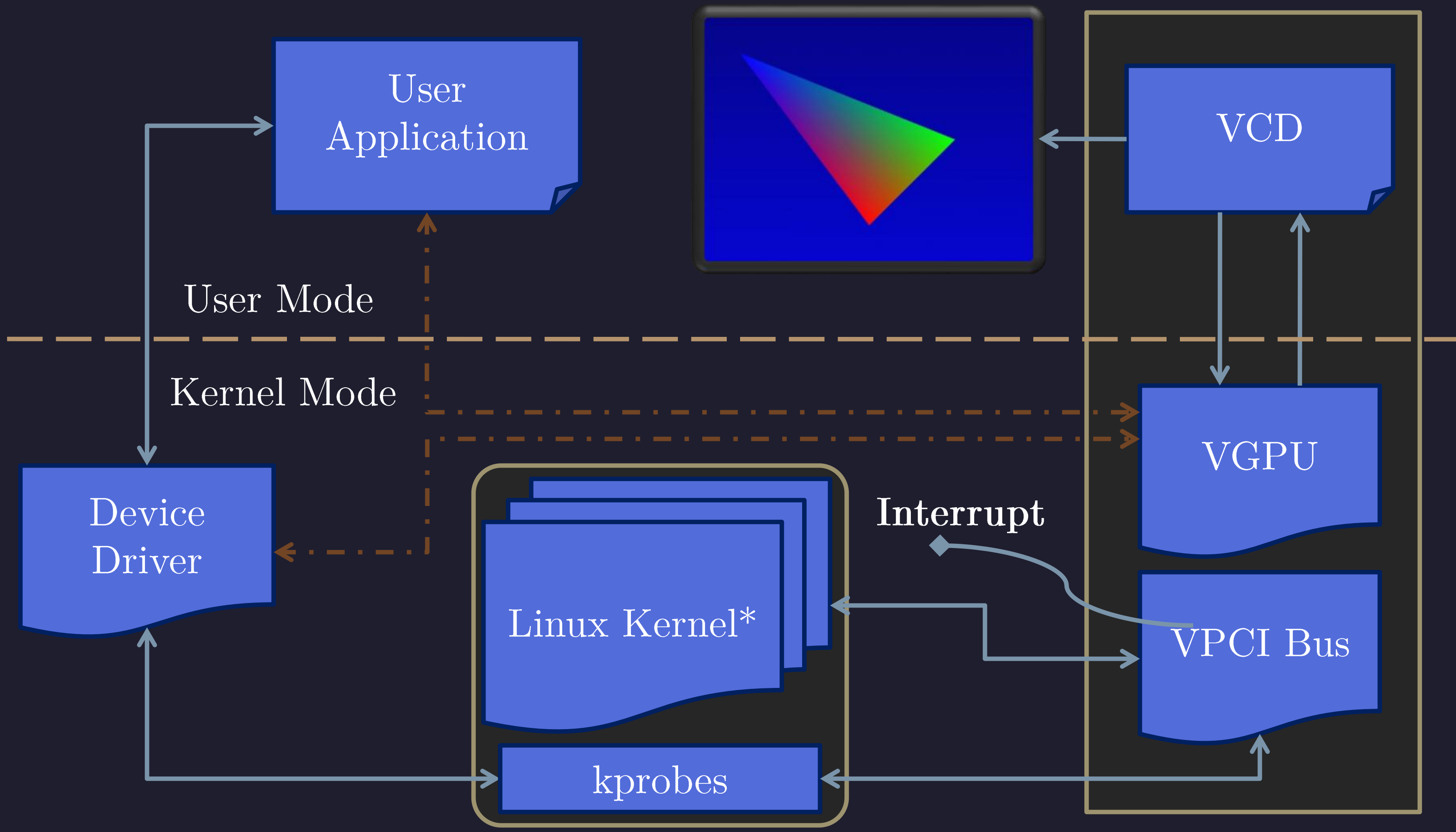
Device Driver Development

- Device driver for non-trivial graphics card
  - Formerly 3Dlabs Permedia2V
- Virtualization provides only a simple VGA controller
  - Hardware accelerated graphics solutions virtualized at API level rather than architecture level
  - Experimental support for importing devices to virtual machines starting to emerge
- Still leaves one more problem
  - Virtualization of one architecture is inadequate...graduate students have been known to communicate with one another

Design Goals for Virtual PCI Framework

- Support sophisticated components:
  - Requires scheduling
  - Requires memory mapping
  - Generates Interrupts
  - Provides DMA
- Require no special function calls.
- No modifications to existing Linux kernel
- Virtual architecture that can be easily reconfigured each semester

Virtual PCI Framework



VPCI Bus: DD Interface

- Intercept PCI functions
  - DD does not talk to real PCI bus when accessing a virtual device
- Intercept *dma_alloc_consistent()*
  - Capture buffer addresses
- Intercept remap_pfn_range()
  - Capture user mapped address to device register base

VPCI Bus: Memory Management

- Added VPCI Page Fault to the page fault handler (*do_page_fault()*)
- When device driver requests the memory to be mapped with write permissions
  - VPCIB intercepts
  - sets the memory to read-only; flush TLB
  - activates the fault handler
- When the device driver/user code writes to the memory
  - Page fault exception
  - *do_page_fault()* hands control to the VPCI fault handler.
  - VPCI fault handler restores write permission, flush TLB, wakes up the VCD, and returns execution to spot of the fault

VPCI Bus: Interrupts

- When a DMA buffer is finished, an IRQ must be raised so the DD interrupt handler is notified
- How do you throw a hardware IRQ from software?
  - Intel x86 instruction *int* is for generating software interrupts
  - x86 interrupt mapping starts at 32
  - x86_64 interrupt mapping starts at 48

VGPU

- Allocates a kernel page
- Registers the device with VPCI Bus
  - Gives memory address, IRQ number, device IDs for use with Iprobes
- In response to fault handler
  - Wake up daemon
- In response to VCD
  - When done processing registers, write protect the register page, flush TLB, sleep
  - Generate interrupt if DMA is running, sleep until device driver interrupt handler finishes

VCD

Map register page from VGPU
Initialize registers
Use ioctl() to sleep

Forever {

 if (register values changed) take action
  e.g. start graphics mode, draw to the screen, initiate DMA

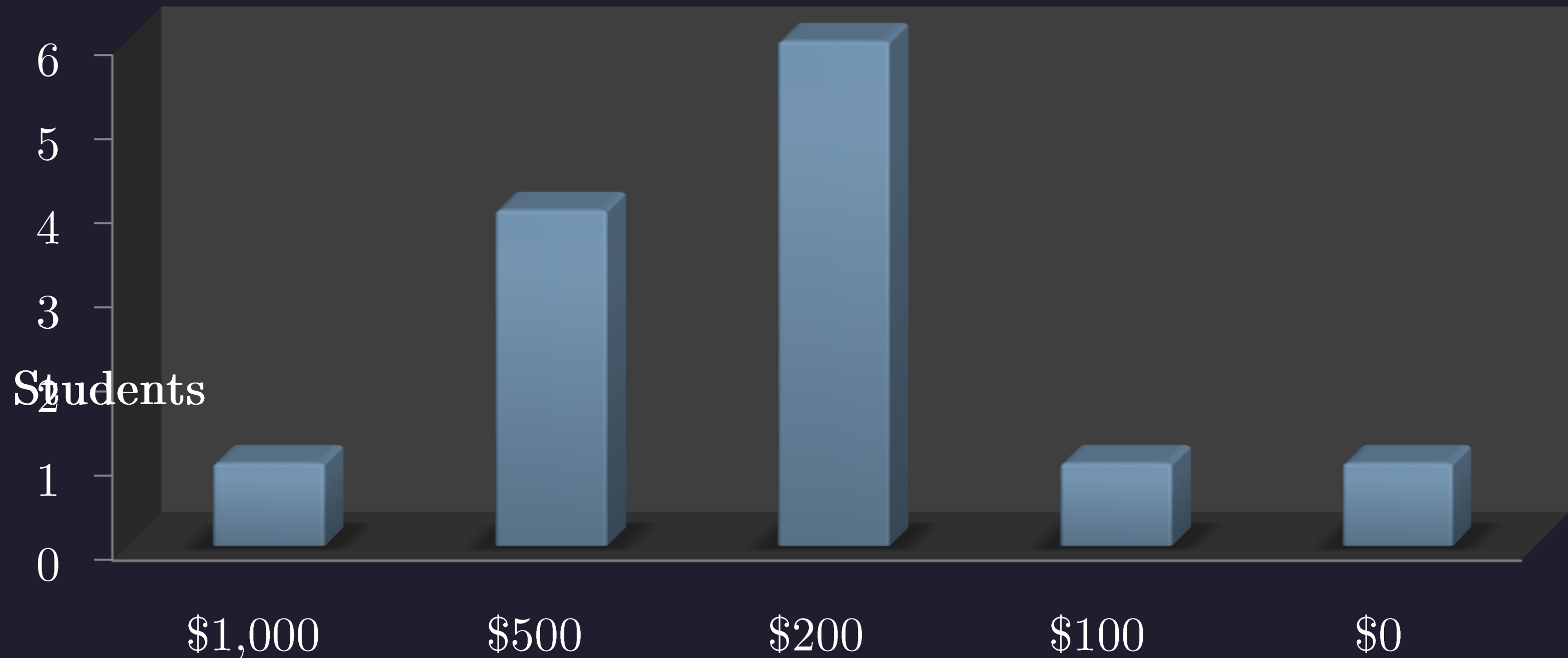
 Use ioctl() to write protect the page and sleep
}

Status

- Successfully deployed in CPSC 822 for 4 semesters

- Kernel Modifications:
  - Most are avoided by kernel probes
  - Remove *__kprobes* from *do_page_fault()*
- Devices:
  - Can write binary compatible device drivers for use with real hardware
- Allow for parallel development of DD and HW

Transparency to Students

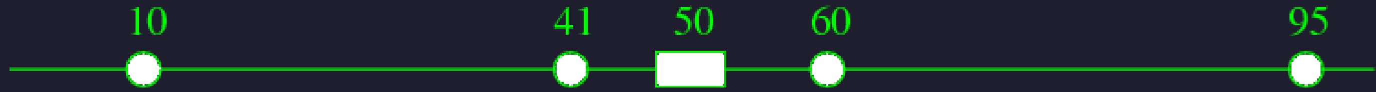


Virtual Performance Throttles

Disk Scheduling

- Heavily-loaded system: non-empty queue of pending disk requests likely
  - Schedule in which order?
- Algorithms studied for at least 4 decades!
- Increasing importance:
  - 20 years ago: CPU speed in µs, disk speed in ms
  - Today: CPU speed in ns, disk speed still in ms
  - Disks are performance bottlenecks
- Algorithms not constrained to be *work-conserving*

Example



- Order of arrival is 95, 10, 60, 41 (r/w head at 50)
- Travel time constant per unit distance

FCFS	Service	Wait	Response
95	45	0	45
10	85	45	130
60	50	130	180
41	19	180	199
mean	49.75	88.75	138.50

Example

- Is greedy (shortest-access time first) better?

SATF	Service	Wait	Response
41	9	0	9
60	19	9	28
95	35	28	63
10	85	63	148
mean	37.00	25.00	62.00

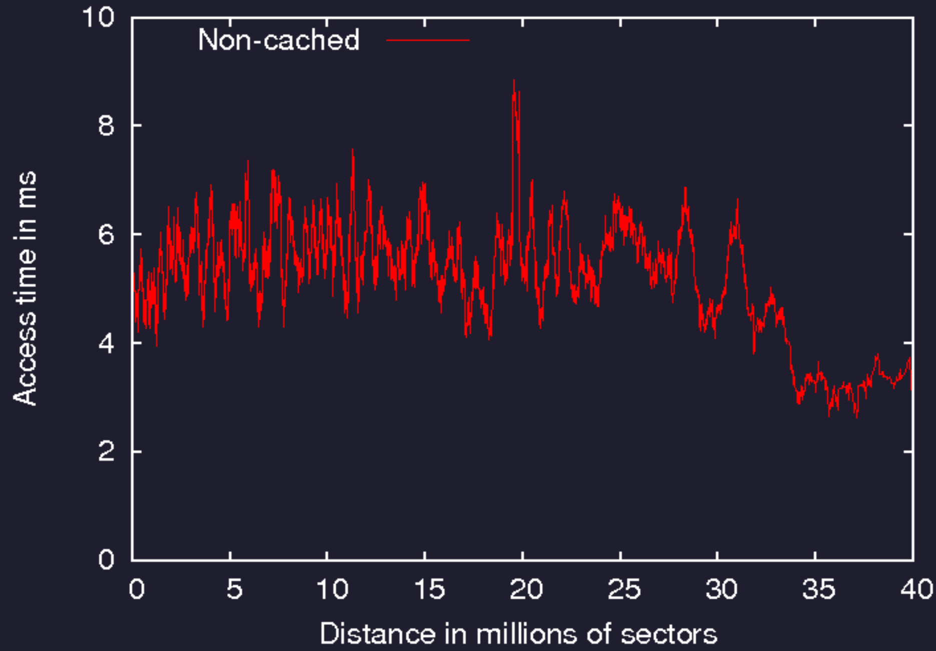
Example

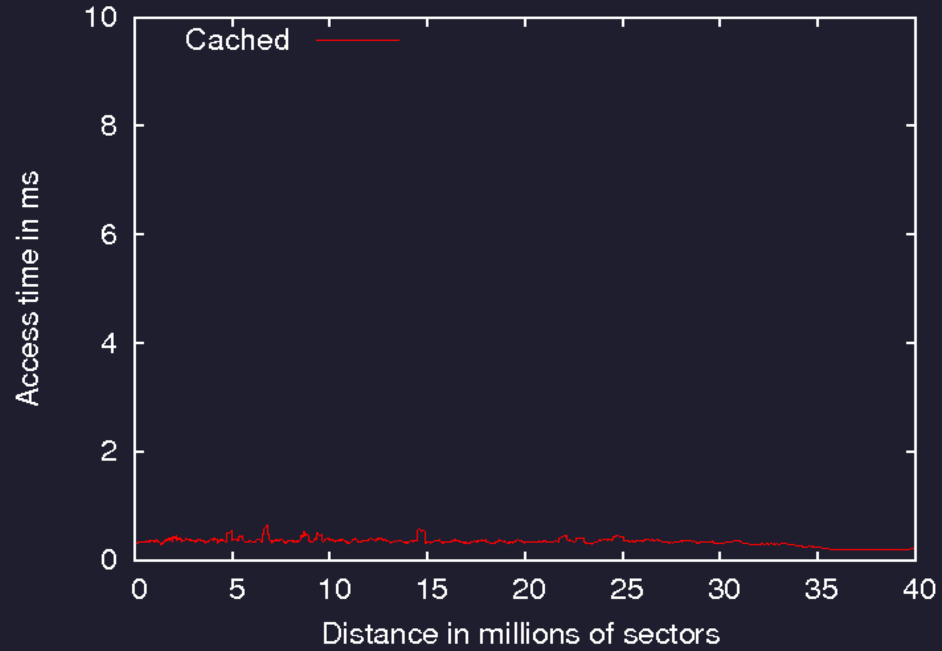
- But is it optimal?

OPT	Service	Wait	Response
60	10	0	10
41	19	10	29
10	31	29	60
95	85	60	145
mean	36.25	24.75	61.00

Disk Scheduling Development

- Design a new disk scheduler that outperforms default Linux schedulers
- Development can be carried out easily on virtual machines.





Virtual-to-Physical Abstraction

VPT Design Goals

- Provide a method for predicting the performance of disk scheduling algorithms on real machines using only their performance on virtual machines
- Provide a new, high-performance, disk scheduling algorithm as a case study

VPT Implementation

- iprobe & jprobe in SCSI path to force virtual service times to be proportional to real ones.
  - Linear Seek Model: $X_r = R_r/2 + S_r\,(d_r\,/D_r)$
- Force virtual service time kX_r, where k is constant
- Observed virtual service time is X_v
- jprobe: calculate kX_r
- iprobe: delay request completion $kX_r - X_v$

More Implementation

Dynamic Scaling

- Up: More than 1% of requests complete after target time
  - k to small → increase k
- Down: Less than 0.1% of requests complete after target time
  - k to large → decrease k
- Ability to lock k for measurement runs
- System reports current k
  - Rule 1: accuracy
  - Rule 2: simulation run-time

Cache Mode

- Shadow the real cache
- On a cache hit adjust service time to match target drive cache service time

Case Study

- Test viability of VPTs
- Predict the performance of disk scheduling algorithms, one of which is new
- Compare the performance of each algorithm on a VM employing a VPT versus a real machine

Schedulers

- Shipped with Linux
  - No-op
  - Anticipatory (*Defunct*)
  - *Deadline*
  - *Completely Fair Queuing*
- Proposed
  - *Cache-Aware Table Scheduling* (*CATS*)

CATS

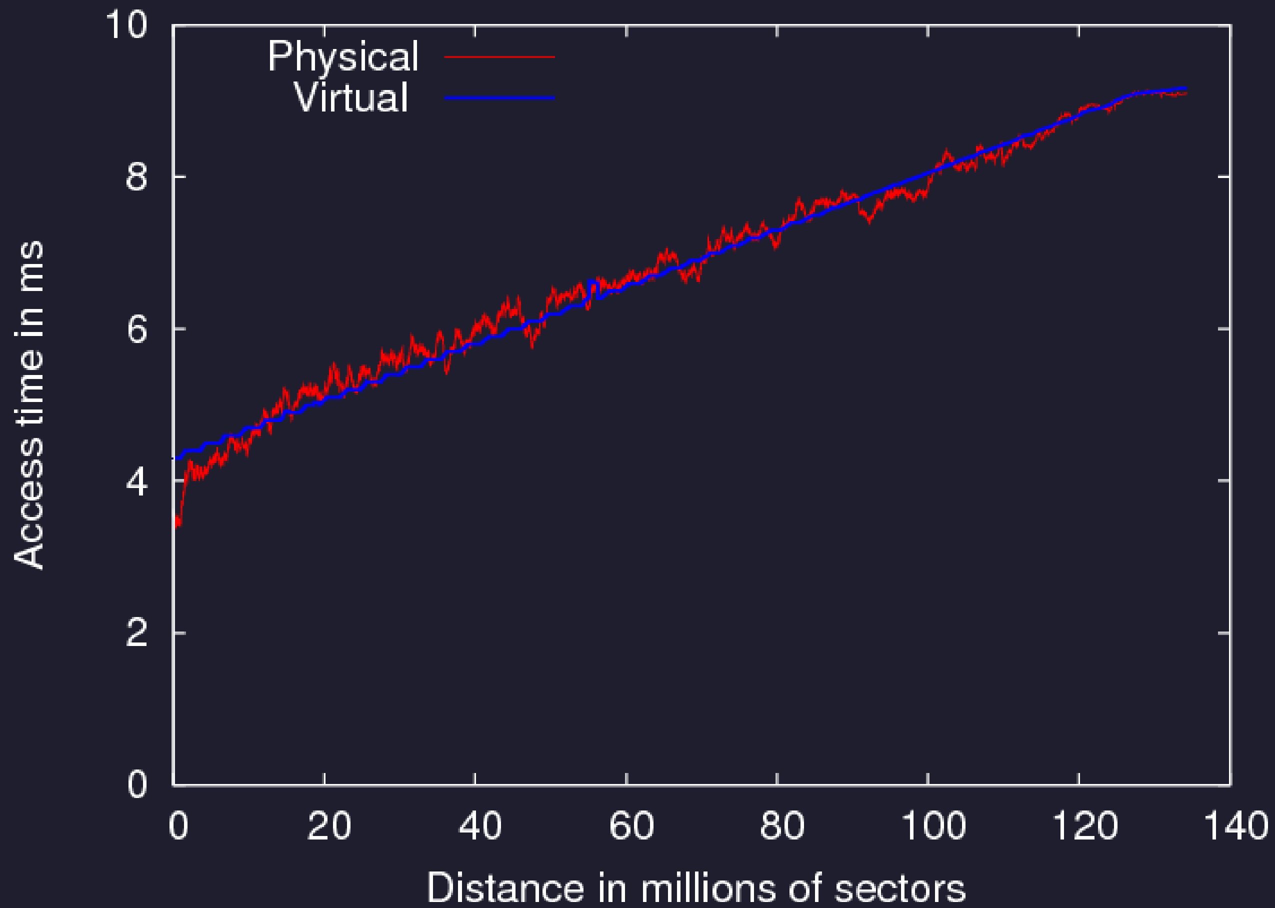
- Separate reads and writes; reads have priority
- Writes use CSCAN with request coalescing
- Writes served in bursts (MAX/MIN WRITEDELAY)
- Reads use ***algorithm T*** with request coalescing:
  - For any collection of n requests, find optimal (minimum response time) completion sequence in worst-case $O(n^2)$ time
  - Serve first request from optimal list
  - Re-compute optimal list, if new arrivals
- Out-wait deceptive idleness (*5 ms*)

CATS

- Cache model: number of segments, sectors per segment, pre-fetch size (sectors)
- Cache model assumptions: fully associative, FIFO replacement, wrap-around within segments
- Affects on scheduling:
  - Maintain shadow cache within scheduler
  - On each dispatch, check entire queue for predicted cache hit
  - If predicted hit, schedule immediately

Platform

- Real:
  - Linux 2.6.30
  - Dual Intel Xeon 2.80GHz CPUs
  - Western Digital IDE system drive
  - Dual 73.4 GB Seagate Cheetah 15.4K SCSI drives
  - Dual Adaptec 39320A Ultra320 SCSI controllers
  - Tests restricted to single SCSI drive
- Virtual:
  - KVM-based, virtual Linux 2.6.30
  - Hosted on IBM 8853AC1 dual 2.83GHz Xeon blade
  - Virtual 73.4 GB SCSI disk
  - Virtual disk on NetApp FAS960c, access NFS
- Cache Model
  - 64 segments
  - 221 sectors per segment
  - 64-sector pre-fetch



Platform Drives

Workload

- Barford & Crovella: (SURGE) tool
- 64 processes, each executes:

```
forever{
    generate a file count, n, from Pareto(α1 ,k1 );
    repeat(n times){
        select file from L files using Zipf(L);
        while(file not read){
            read one page;
            generate t from Pareto(α2 ,k2 );
            sleep t ms;
        }
    }
}
```

Results

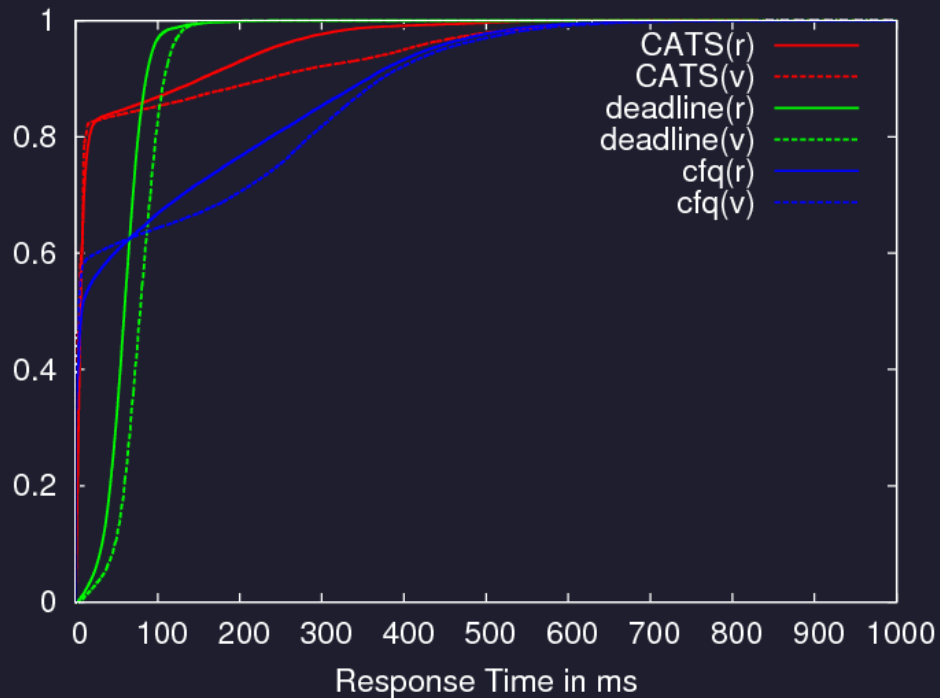
- 64 processes, 50,000 requests, O_DIRECT
  - Service and response times in ms
  - Throughput in sectors/ms

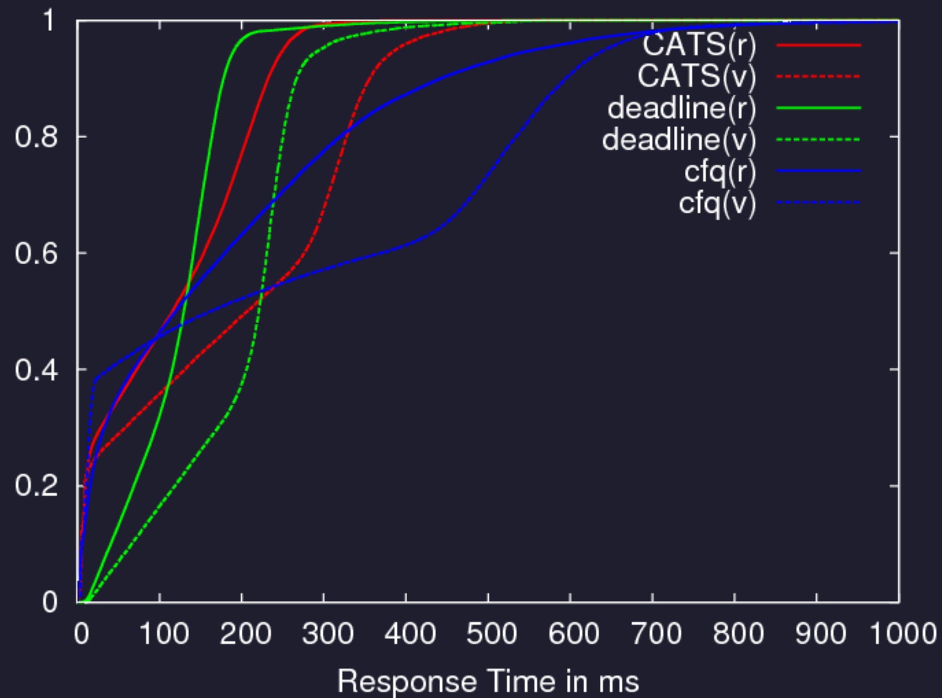
	Physical			VPT (k=8)		
	CATS	dline	cfq	CATS	dline	cfq
Mean Service	1.96	2.71	1.39	2.58	3.24	2.36
Mean Response	37.35	59.87	124.70	53.79	78.27	117.13
Throughput	8.19	6.08	2.19	6.15	5.06	3.38

Results

- 64 processes, 50,000 requests, non-O_DIRECT
  - Service and response times in ms
  - Throughput in sectors/ms

	Physical			VPT (k=8)		
	CATS	dline	cfq	CATS	dline	cfq
Mean Service	6.53	7.41	7.80	7.15	7.60	8.57
Mean Response	114.91	121.87	179.17	189.198	198.33	258.75
Throughput	12.00	12.04	9.08	11.44	11.68	8.82





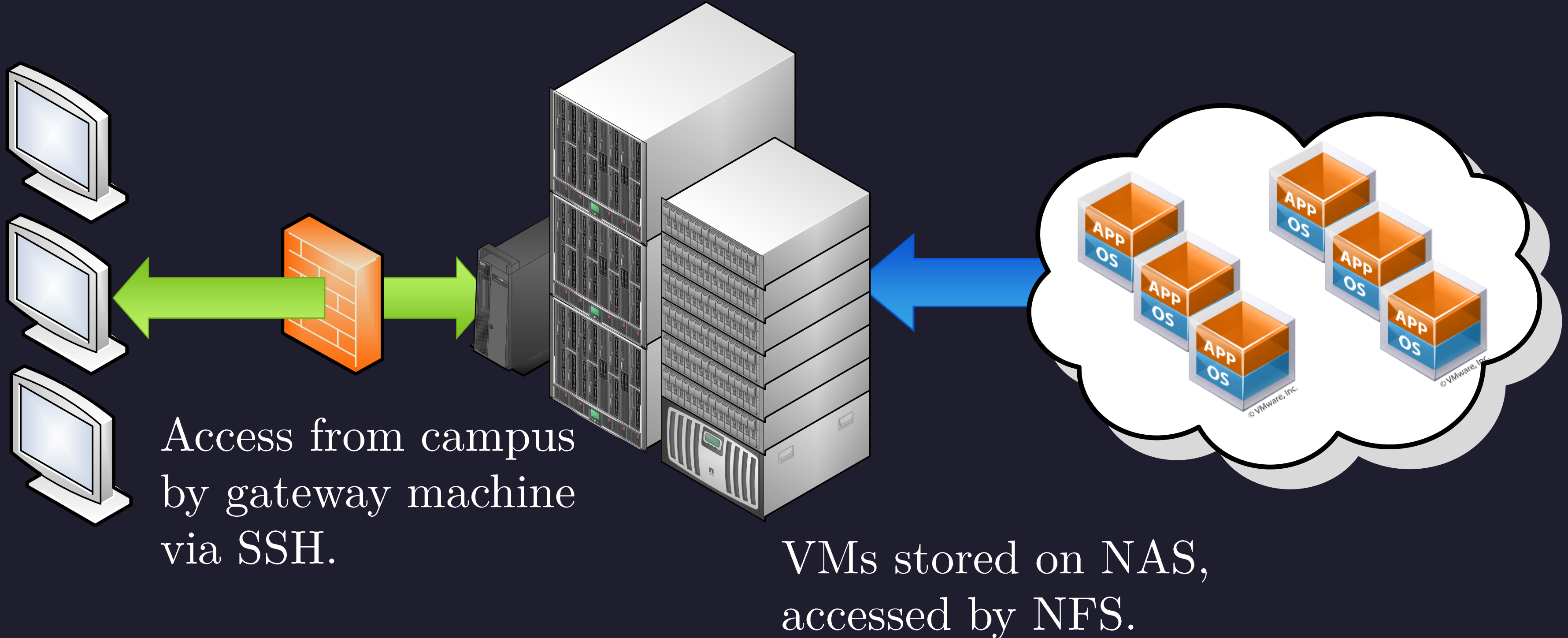
Response Distributions

Distribution plots for O_DIRECT and non-O_DIRECT workloads

Virtual 822 Lab

Lab Environment

VMs run on 6 dual-Xeon blades.
Part of 42-blade eServer BladeCenter.



Custom scripts used for managing and accessing lab.

Lab Environment

- Timing
  - Paravirtualized clock
  - When enabled guest kernel will receive system time updates from the hypervisor
- Performance
  - Cost of virtualization on CPU bound tasks.
  - Using the right CPU pinning $<1\%$ penalty.

Status

- New method for predicting (real) disk scheduler performance using solely a VM
- Method uses new iprobe to force virtual service times to match simple service model
- New disk scheduler (CATS) provided as case study
- Absolute performance predictions not yet accurate, but relative predictions are quite accurate
- Fair criticism: just using virtual Linux as elaborate simulator
  - True, but good results with almost zero programming effort!

Virtual Card Generation

Conclusions

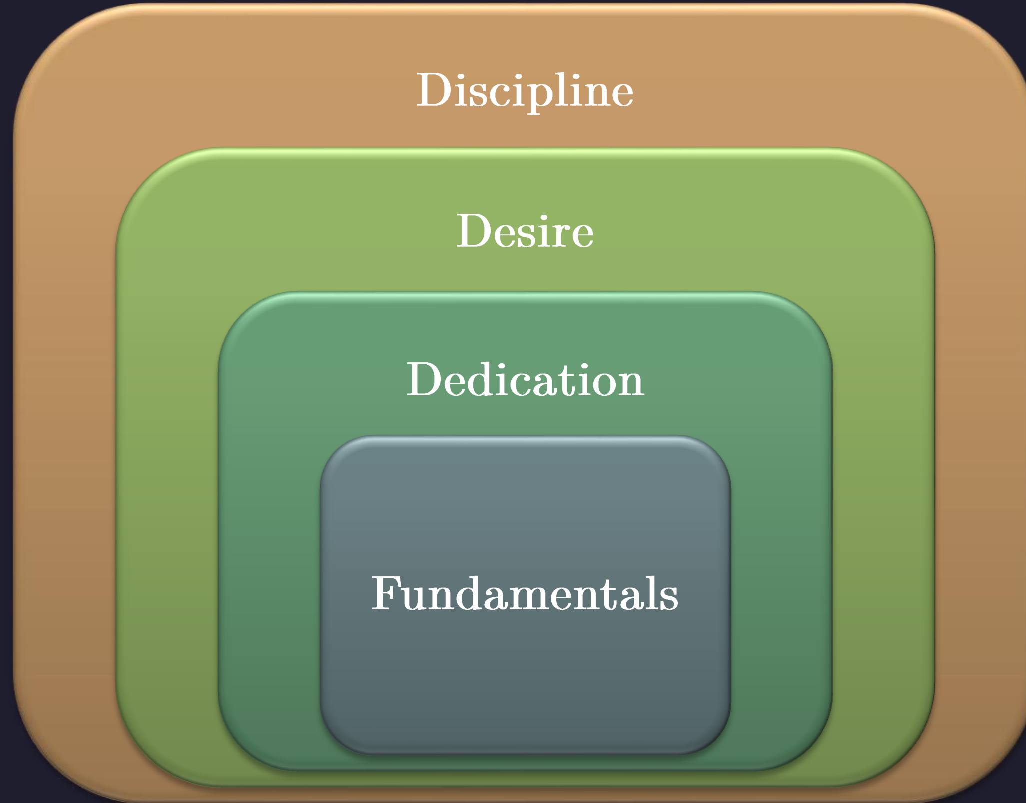
- Virtual PCI Architecture
- Virtual Performance Throttles
  - CATS
- Virtual 822 Lab
  - Virtual Card Generation

Future Work

- Virtualize CUDA devices
  - Card sharing or merging

- Wish list
  - VPCI Architecture
    - More devices
  - VPTs
    - Use for network I/O
    - Model the decay of SSDs
  - Virtual Card Generation
    - More tunable parameters

Solid Foundation



Measuring Success

"For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mold what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable."

– Edsger W. Dijkstra,
EWD 709: My hopes of computing science

Questions & Comments