Instructor Notes -119 Noland

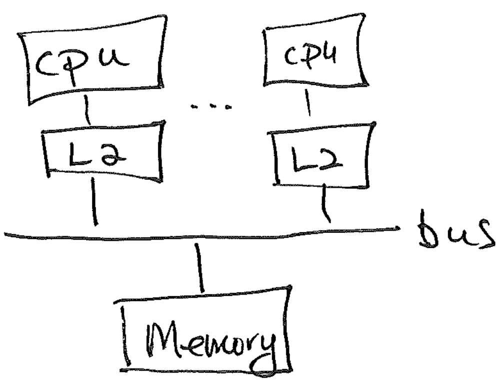
Disco: Running Commodity Operating Systems on Scalable Multiprocessors
Bugnion, Devine, and Rosenblum
SOSP 1997

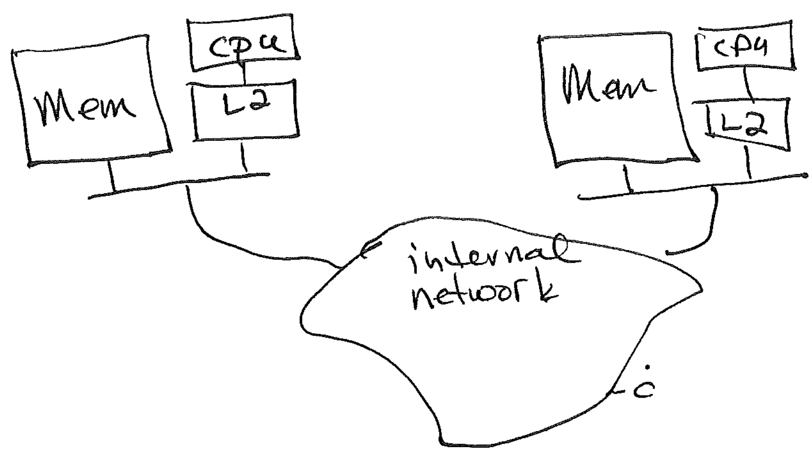
1. What was the goal of Disco?

Run commodity OS on NUMA

2. What are the pros and cons of a CC-NUMA architecture (vs SMP UMA)? What are some issues with building or modifying an OS for CC-NUMA?

cache-coherent Non-Uniform Memory Architecture





+ same performance to all memory from everywhere

- hard to make hw scalable - hit limits of bus

- crazy perf

+ scalable hw

(correctness the same)

---

0) Write from scratch

Difficulties:

1) Take existing OS + port: huge, ugly, difficult to understand

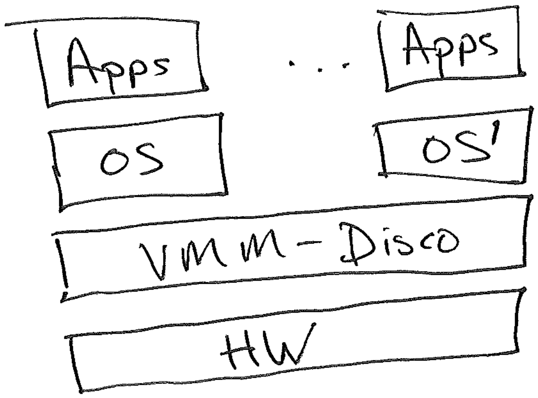
-need to allocate OS data structures across NUMA (hard enough for SMPs)

locks

⇒ VMM!

3. What are the advantages of Virtual Machines?

add new layer



- hide tough issues from OS
- portability layer
- smaller, easier to understand + trust
- run different OSes concurrently (almost unmodified)

4. At a high level, what are some of the challenges of Virtual Machine?

1) Overhead - cost of virtualizing

a) Time:

app
OS
disco
hw

vmm acts as emulator
- most instr can just run, but priv. instr. + TLB inst must be trapped + emulated

b) Space:

OS | OS
disco

- multiple copies wastes memory
- OS code + file cache

2) Resource management

App
OS
VMM

- lose information about what is being used
a) cpu - idle thread?
b) memory - free list

3) Sharing problems
- most OS require exclusive access to disk

a c

5. How does Disco virtualize the MIPS R10000 CPU? What happens on a system call w/o and w/ Disco? ~~Why are 3~~ modes for user, supervisor , and kernel key?

b

- most: direct execution
  - track each VM that must be run, set real registers to VCPU regs, jmp to VPC (similar to OS + process table entries)
  - hard? priv. instr.

MIPS detail:

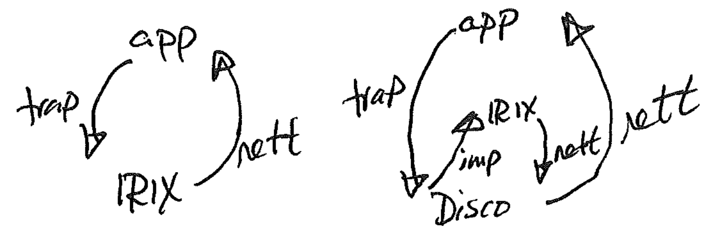
user

supervisor — access all segments of mem, but not priv. instr, or phys mem

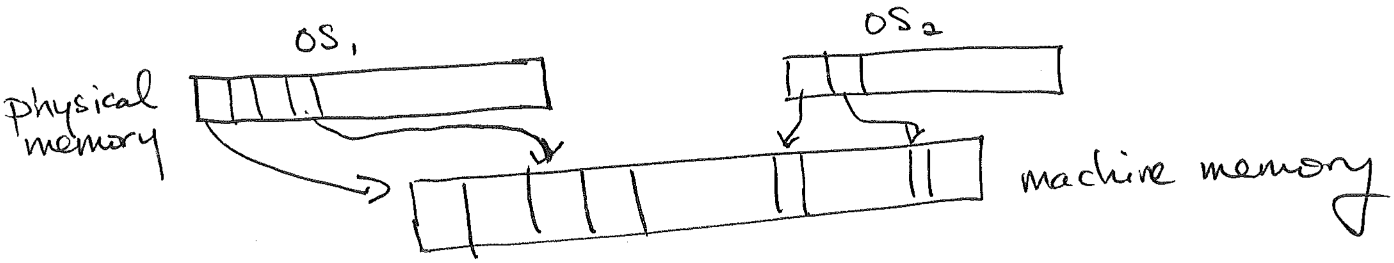
kernel

usual	Disco
app	app
—	OS - IRIX
IRIX	DISCO

OS can still access all of memory + manage apps



6. How does Disco virtualize memory? What will be held in the TLB? What happens on a TLB miss with w/o and w/ Disco? What data structure does Disco add?



v.a. → p.a. → m.a.

~~Do thru TLB:~~

normal

vpn	off

↓

p.pn	offset

use pagetables

could trap on every ~~ld~~ + st,

old

vpn	ppn

DISCO

vpn	mpn

TLB miss?

normal

user

TLB miss

return

IRIX - walks pg tables

lds TLB

user

TLB miss

v.a.? DISCO

IRIX - walk pg tables
- ld TLB <vpn, ppn>

priv

looks up ppn → mpn in pmap for each VM

replace <vpn, mpn> in TLB

ret

7. What complexity was caused by IRIX living in kseg0? What was Disco's solution?

KSEG0 - unmapped physical memory

→ can't interpose via TLB

- all OS memory references ~~would be~~ ...

→ Relink IRIX to use different segment

8. Why are TLB misses more significant with Disco? What is Disco's solution?

1) more costly

2) more of them

a) OS is using TLB too

b) must flush TLB between VCPU switches; Why?

TLB

ASID	vpn	mpn

difficult/ costly to virtualize ASID across all VMs, so instead flush

⇒ Add 2nd-level TLB in SW to do quick replacements - no need to walk pt in IRIX

(modify prev. diagram)

9. What are Disco's goals that are specific to NUMA? When should a page be ~~replicated~~ mig? ~~migrated~~ rep? How does Disco perform ~~replication~~ mig and ~~migration~~ rep? What should happen if a page is heavily write-shared?

- Handle performance of NUMA (not correctness)
  - cache misses from CPU handled in local memory

Key: Use VA → PA → MA to handle issues

- Replicate?
  - Read sharing ~~(OS text)~~
- Migrate?
  - Activity on new CPU (e.g. DISCO scheduler moved VM) w/ affinity, tries not to, but...

How?

- modify old TLB appropriately
- migration: invalidate! old mpn entries
- replication: downgrade to read-only

If heavily-write-shared?

Don't migrate or replicate

10. Why are large memory footprints a concern for Disco? Why does sharing occur across VMs? Why is copy-on-write useful?

- Multiple OSes now running (text-code, data); encourage use of NFS server for shared file access

- OSes share same code; NFS shares file data across client + server

Figure 3.

- intercept DMA from block X
- if already in memory, just record PA→MA
- record copy-on-write to track shared data efficiently

Figure 4.

① send becomes additional mapping (emulate device)

② copy becomes add. mapping

11. Running a completely unmodified commodity OS on Disco is tricky. What changes did Disco make to IRIX to improve performance?

(1) Some priv. ops are very simple (reading regs)

→ replace w/ ld/st to mem. addresses

(2) zero'ed pages:

- ~~both~~ OS does when allocates new page to process (must for privacy)

- Disco must across VMs

- avoid double work

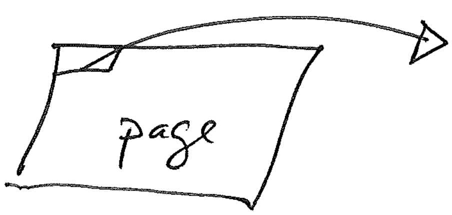
(3) Pages on free list - Disco should know about

(4) cpu is idle

- Disco detects low power

(5) bcopy → remap in NFS client

(6) mbuf structure



12. As shown in Figure 5, how much time overhead does DISCO impose for a uniprocessor workload? Why does some of the original kernel time decrease?

- more for compute-bound w/ TLB misses
  - pmake uses many system services

- zeroing pages
- 2nd-level TLB

13. What does Figure 6 show? Does Disco do a decent job sharing buffer cache space across VMs? Of sharing IRIX text? IRIX data?

yes yes no

- pmake workload
- virtual footprint vs. machine memory

14. What does Figure 7 show? Where can you find an evaluation of Disco's replication and migration policies?

8-processor cc-NUMA

IRIX · Lots of time in kernel & memlock scalability limit

1VM · just add DISCO overhead

2VM → 8VM actually improves

because DISCO does not have bad lock

turned off migration + replication

Fig. 8.

- much less time accessing remote mem

15. Conclusions?

Strengths

+ Power of layering/indirection

+ small monitor hides tough NUMA + parallel issues

+ Really works - useful in many settings (not just cc-NUMA)

Problems:

- True virtualization tricky

- Handling info loss thru stack