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?

- Same performance to all memory from everywhere
- Hard to make hw scalable - hit limits of bus
- Crazy perf
- Scalable hw

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?

- 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: \[
   \frac{\text{app}}{\text{OS}} \quad \frac{\text{OS}}{\text{disco}} \quad \frac{\text{hw}}{\text{vmm acts as emulator}}
   \]
   - most instr can just run,
   - but priv. instr. + TLB inst
   must be trapped + emulated

   b) Space:
   \[
   \frac{\text{OS}}{\text{OS}} \quad \frac{\text{OS}}{\text{disco}}
   \]
   - multiple copies wastes memory
   - OS code + file cache

2) Resource management
   a) CPU - idle thread?
   b) Memory - free list

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

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

MIPS detail:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Access</th>
<th>Memory Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>all segments of mem</td>
<td>not priv. instr. or phys mem</td>
</tr>
<tr>
<td>Supervisor</td>
<td>-access</td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OS can still access all of memory + manage apps.

Diagrams:

- Usual:
  - app
  - IRIX

- Disco:
  - app
  - OS-IRIX
  - DISCO

Traps:

- Trap: IRIX
- Trap: Disco
- Trap: IRIX

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?

![Diagram of memory hierarchy]

V.A. → P.A. → M.A.

**TLB**

<table>
<thead>
<tr>
<th>Acceptable()</th>
<th>User VPn, Offset</th>
<th>use page tables</th>
<th>could trap on every add, sub, \mbox{ok}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disco ()</td>
<td>VPn, MPn ()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TLB miss?**

<table>
<thead>
<tr>
<th>Acceptable()</th>
<th>User VPn, Offset</th>
<th>use page tables</th>
<th>could trap on every add, sub, \mbox{ok}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disco ()</td>
<td>VPn, MPn ()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IRIX** walks pg tables

1st TLB

\(\) IRIX - walks pg tables

\(\) I's TLB

\(\) TLB mis

\(\) IRIX - walks pg tables

\(\) I.d TLB \(\langle\)VPn, PPN\(\rangle\)

\(\) priv ?

\(\) looks up PPN \(\rightarrow\) MPN in \(\text{Pmap}\) for each VM

\(\) replace \(\langle\)VPn, MPn\(\rangle\) in TLB
7. What complexity was caused by IRIX living in kseg0? What was Disco’s solution?

- KSEGD - 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?

<table>
<thead>
<tr>
<th>TLB</th>
<th>ASID</th>
<th>VPN</th>
<th>MPN</th>
</tr>
</thead>
</table>

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 IPLX
(modify prev. diagram)
9. What are Disco's goals that are specific to NUMA? When should a page be replicated? migrated? How does Disco perform replication and migration? What should happen if a page is heavily write-shared?

- Handle performance of NUMA (not correctness)
- cache misses from CPU handled in local
  Key: Use VA → PA → MA to handle issues in memory

  - Replicate?
    - Read sharing
  - Migrate?
    - Activity on new CPU (e.g., scheduler moved VM)

  How?
  - modify old TLB appropriately
  - migration: invalidate old mpu 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; 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 DMA
- Record copy-on-write to track shared data efficiently

Figure 4.
1. send becomes additional mapping (device)
2. copy becomes additional 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'd 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
- `ps` 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

1M - lots of time in kernel & memlock 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 into less than stack