“Disco: A bad idea from the 70’s, and it’s back!”
Mendel Rosenblum (tongue in cheek)

1 Preliminaries

1.1 Review

1.2 Outline

1.3 Preview

• This week: Disco and Exokernel. One lesson: “If at first you don’t succeed, try try again”

2 Overview

• Disco motivation

– Background: Stanford FLASH
  * SMP
  * NUMA
    • performance (e.g., memory locality)
    • scale (bottlenecks; NUMA enables larger scale than SMP)
    • fault tolerance (isolation; NUMA enables larger scale and less tight coupling than SMP – some need for/hope for isolation)

– Want: convert a large SMP into a collection of virtual machines that run concurrently
– Existing commercial OSs don’t do well on FLASH (thanks to NUMAness)
  * Hard to modify them
    · Traditional view: “Software flexible, HW inflexible”
    · Reality: processor architectures respun every 3-5 years, OS architectures respun never
      → Reality: Software inflexible
    · Why? (One theory: HW has narrow interface; OS interface gets broader and broader)
  * The VMs will hopefully share resources better than the big OS can
    · locality – VM small ← can be hand-tuned “like a parallel program
    · bottlenecks – hierarchical mgmt – Guest OS/VM
    · isolation – OS runs in different address spaces “distributed system”
  * Mix OSs, especially commercial ones with specialized ones
  * Fault containment
  * Economics: if you are a commercial operating system vendor, where spend time: adding features to commodity OS that sells millions of copies per year or porting and maintaining 50M-line OS to run on a few hundred high-end machines?

• Basic goals
  – Support commodity OS with no modification
    * Disco: 13000 lines of code
    * v. Exokernel goal: absolute max performance

• Basic solution
  – Disco
    * Virtualize the hardware
    * Guest OS’s have no idea that multiplexing is happening
    * Core challenge: virtualize hardware without breaking existing OS
  – (v. Exokernel)
    * Export the hardware
    * LibOS’s are active accomplices in virtualizing hardware
    * Core challenge: balance cooperation v. protection for max performance

• Challenges for VM’s
  – Overhead of virtualizing hardware
    * Emulate key instructions (IO, interrupts, memory management, …)
    * Space overhead: many copies of OS, binaries, etc in memory
  – Difficult to manage resources w/o OS participation
    * When is cpu idle?
      · Disco guest OS gives hint for idle loop (give up a bit of transparency…)
    * When is memory free?
  – Lack of sharing across VM’s
* Interprocess communication on IBM VM: Connect virtual card punch of VM1 to
virtual card reader of VM2.
  - Lack of ccNUMA support

3 Virtualizing a machine

• Most instructions execute at hardware speed

• Privileged instructions?
  - Trap to VM; VM emulates
  - Example: What is pseudo-code for “handle process in guest OS TLB miss (for SW filled
TLB)?”

• Exceptions and interrupts?

• “Virtualizable machine”

4 Admin

• Project

• hamming paper

5 Key subsystems

5.1 Virtual memory

• GuestOS sees only “virtual physical addresses”
  - When GuestOS tries to update TLB, VM remaps virtual PA to real PA
  - GuestApp only sees virtual addresses – it cannot detect the “lie”
  - GuestOS sees GuestApp virtual addresses and virtual physical addresses – it cannot
detect the “lie”
  - QUESTION: Why does Disco relink the OS?
  - QUESTION: What does disco need internally to do this? How do you handle TLB
miss? How do you take a page away from a VM? How do you migrate a page within a
VM? How do you give a page to a VM? Why does Disco not virtualize ASIDs?
  - Disco intercepts disk requests for blocks already cached and gives the requesting VM a
read-only mapping to the page (if the request is a multiple of the page size). This leads
to transparent sharing of root disk.
  - But how do we do this trick when OSes are sharing mutale file systems via NFS? A
virtual network device with unlimited MTU. Change IRIX mbuf implementation to not
write buffer memory and change bcopy to monitor’s remap
  - OS tells monitor when a page will not be reclaimed (break abstraction)

• Virtual Memory on ccNUMA (pre-disco)
- Shared virtual address space across nodes – where to locate data?
- Disjoint virtual address space within a node – replicate identical data (e.g., “/bin”)

- Transparent replication and migration for shared virtual pages on different machine nodes
- Transparent sharing of machine pages that are virtually different on same machine node
- **QUESTION**: Given pmap, how share a page between VMs? How replicate a (read only) page within a VM on multiple processors?

### 5.2 File system

- Virtually distinct disks

- **Optimization**: Identical data on virtually distinct disks
  - Copy-on-write sharing
  - Read-only disk file system sharing
  - Write sharing via NFS

- **Optimization**: Copy-on-write + rollback → useful for debugging, test, etc.

### 5.3 Devices

- Option 1: Trap all programmed I/O and emulate

- Option 2: Add fake Disco-aware device drivers

- Ranges from transparent to not-so-transparent

- Ranges from slow to fast
5.4 Fast communication

- VM370: Virtual card writer to virtual card reader
- Disco: Fake virtual subnet
  - Use TCP/NFS to connect virtual machines
  - Zero copying by sharing underlying VMM buffers

5.5 Resource allocation

- Simple time-sharing scheduler among VM’s
  - Guest OS hints for idle loop
- ccNUMA-aware management (e.g., affinity scheduling)
  - How to do gang scheduling?
  - How to do real time applications?

6 Disco evaluation

- More complete than the original UNIX paper...they measure compilation of GNU chess
- What are they trying to prove? What should they be trying to prove (what are key questions about VMM approach?) Taking their experiments at face value, should you be convinced?
- Discussion
  - IRIX memlock was a total disaster and no OS has a problem that big today
  - NUMA scalability experiment compares to optimal (UMA)

7 Questions

- Research question: Is Disco simple enough to allow formal verification of correctness?
  - Can you add “performance hints” interface that does not hurt correctness argument?
- What lessons (if any) for structuring OS’s?
- How much complexity belongs in “virtual machine” and how much in OS? As we add more to virtual machine, can we simplify OS?
- What is the right layering? (What layering does Disco end up using?)
- What is the right HAL interface?
8 Evaluation: Disco v. Exo v. Micro

- Extensibility
  - Microkernel: experiment with new subsystems
  - Exokernel: some of the most interesting apps are specialized appliances, but how much protection do you need?
  - Disco: specialized OS for novel machines

- Concurrent personality
  - Microkernel: by hopefully sharing some underlying subsystems
  - Exokernel: concurrent secure sharing of low level resource by different personalities
  - Exokernel: the most interesting personalities (apps) do not necessarily have to be concurrent
  - Disco: excellent support for concurrent personality but how many VMM subsystems can you share?

- Modularity
  - Microkernel: easier to foster good s/w engineering discipline
  - Exokernel: ouch
  - Disco: OS/HAL, no more modular than that

- Distributed system support
  - Microkernel: moving subsystems over the net
  - Exokernel: N/A
  - Disco
    - Aggressive copy-on-write is cool
    - The opposite—distributed system support enables the communication of VMs
    - It’s a strange idea to turn an expensive SMP into a dumb cheap cluster

- Security
  - Microkernel: smaller trust base, hopefully
  - Exokernel
    - Exokernel itself is smaller so this is good
    - But unclear with all the code injection and tight interactions with apps
    - But maybe protection doesn’t matter that much for specialized appliances
  - Disco
    - Nice opportunity of fault containment of VMs on SMPs

- Portability
  - Microkernel: subsystems are easier to port
  - Exokernel: ouch
  - Disco:
    - HAL in modern OS makes this easy
    - Small tweaks of OS can make it more efficient