Disco: Running Commodity Operating Systems on Scalable Multiprocessors

Edouard Bugnion, Scott Devine, Kinskuk Govil and Mendel Rosenblum

Stanford University

Presented by :

Long Zhang

Overiew

- Background
  - What are we doing here?
- A Return to Virtual Machine Montiors
  - What does Disco Do?
- Disco: a return to VMMs
  - How does Disco do it?
- Experiment Result
  - Does Disco really do it?
  - How well does Disco DO it?

Background

- Multiprocessor in the market (1990s)
  - Innovative Hardware
- Provide system software for innovative hardware
- Resource-intensive Modification of OS (hard and time waste for size, etc)
- Make a Virtual Machine Monitor (software) between OS and Hardware to resolve the Problem [Old Idea]

Two opposite Way for System Software

- Address these challenges in the operating system: OS-Intensive
  - Hive , Hurricane, Cellular-IRIX, etc
  - innovative, single system image
  - But large effort.
- Hard-partition machine into independent failure units: OS-light
  - Sun Enterprise10000 machine
  - Partial single system image
  - Cannot dynamically adapt the partitioning

One Compromise Way between OS-intensive & OS-light

- Disco was introduced to allow trading off between the costs of performance and development cost.

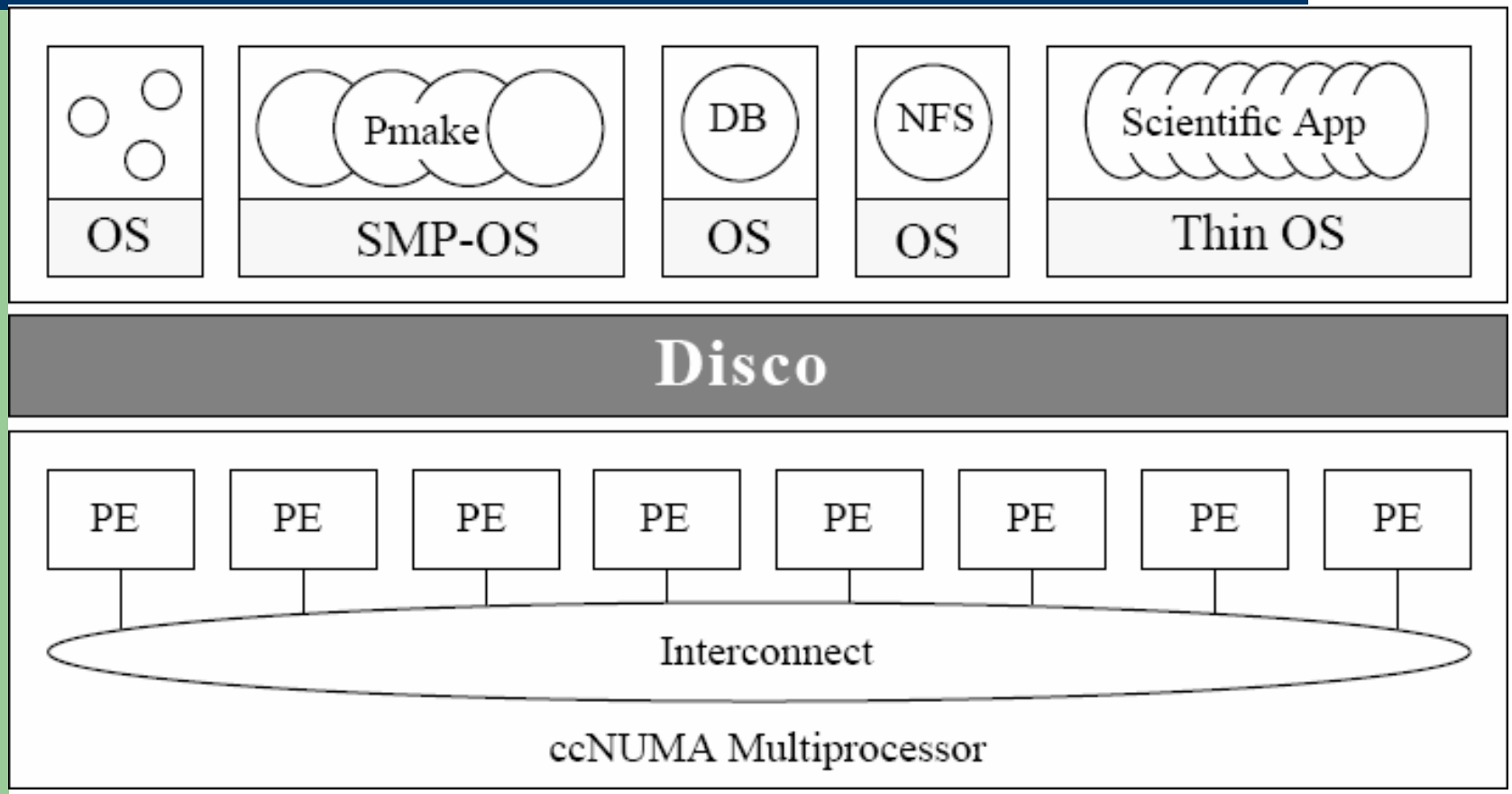
Target of Disco

- Run operating systems efficiently on Innovative hardware: Scalable Shared-Memory Machines
- Small implementation effort with no major changes to the OS
- VMM: Insert an additional layer of software between the hardware and the operating system.
- VM: fault containment between OS and machine.
- Three challenges
  - Overhead
  - Resource Management
  - Communication & Sharing

Disco Outline

- Disco Architecture
- Disco Implementation
  1. Virtual CPUs
  2. Virtual Physical Memory
  3. NUMA Men management
  4. Copy-On-Write Disks.
  5. Virtual I/O devices
  6. Virtual Network interfaces
- Running Commodity Operating Systems
- Experience
- Related Work

Disco Architecture



Disco's Interface

- Disco runs multiple independent virtual machines simultaneously on the same hardware.
  - *Processors*: Emulates MIPS R10000
  - *Physical Memory*: An abstraction of main memory residing in a contiguous physical address space starting at address zero.
    - Dynamic page migration and replication
  - *I/O Devices*: Each virtual machine is created with a specified set of I/O devices

Virtual CPUs

- Schedules virtual machine/CPU as task
- Sets the real machines' registers to the virtual CPU's. Jump to the current PC of the virtual CPU, Direct execution on the real CPU
- Challenge: Detection and fast emulation of operations that cannot be safely exported to the virtual machine (privileged instructions or physical memory)
  - TLB modification
  - Direct access to physical memory and I/O devices.
- Controlled access to memory & Instruction

Virtual Physical Memory 1

- Address translation & maintains a physical-to-machine address mapping.
- Virtual machines use *physical addresses*
- Disco map *physical addresses* to *machine addresses* (FLASH's 40 bit addresses)
- Software reloaded translation-lookaside buffer (TLB) of the MIPS processor (Hardware reload or Software reload?)

Virtual Physical Memory 2

- OS want to update the TLB, Disco emulates this operation, compute the corrected TLB entry & insert it.
- Pmap ->Quick compute TLB
- Each VM has an associated pmap in the monitor
- *pmap* also has a backmaps pointing to its virtual address to help invalidate mappings in the TLB

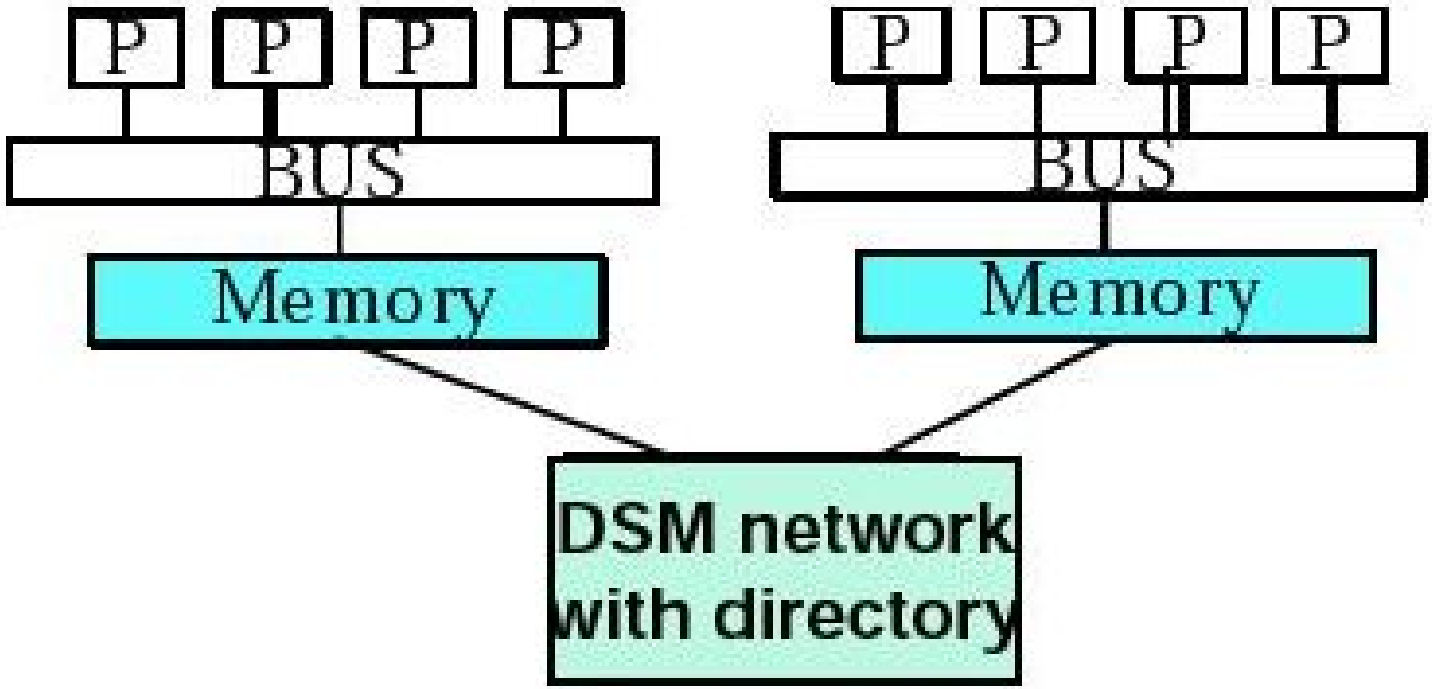
Virtual Physical Memory 3

- MIPS has a tagged TLB, called address space identifier (ASID).
- ASIDs are not virtualized, so TLB must be flushed on VM context switches
- TLB missing: 2nd level software TLB, idea like cache?

NUMA Memory Management

- NUMA: memory access time depends on the memory location relative to a processor. local memory faster than non-local memory (SGI)
- CCNUMA: all NUMA computers sold to the market use special-purpose hardware to maintain cache coherence (Non CCNUMA system is Complex in programing)
- Cache misses should be satisfied from local memory rather (fast) than remote memory (slow)
- Dynamic page migration and page replication system

NUMA Architecture-from Wiki



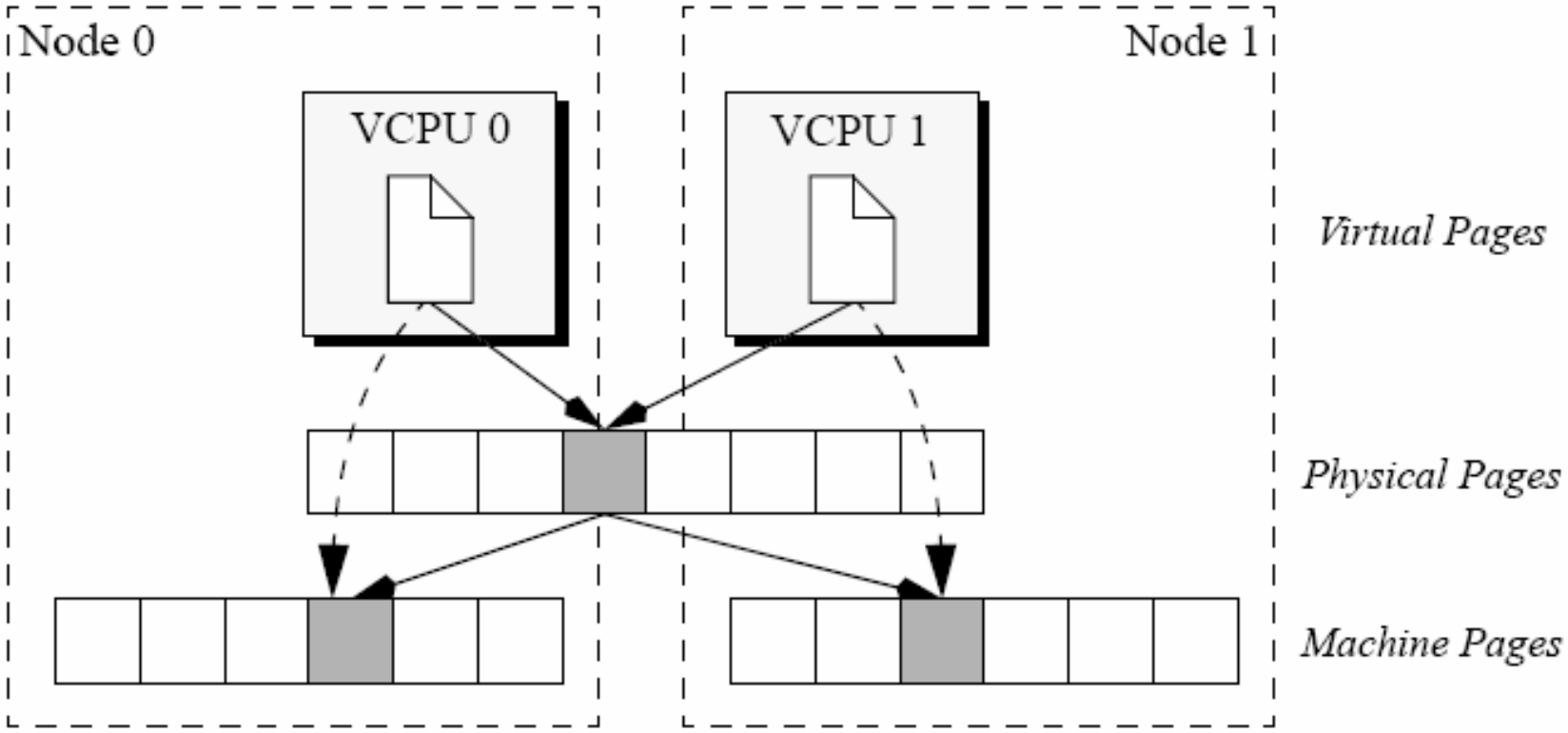
NUMA Memory Management

- Read and read-shared pages are migrated to all nodes that frequently access them
- Write-shared are not, since maintaining consistency requires remote access anyway
- Migration and replacement policy is driven by cache-miss-counting facility provided by the FLASH hardware

Transparent Page Replication

- two different virtual processors of the same virtual machine logically read-share the same physical page, but each virtual processor accesses a local copy.

Transparent Page Replication



Page migration

- To migrate a page, the monitor transparently changes the physical-to-machine mapping.
- *memmap* contains an entry for each real machine memory page. It also contains the list of pmap entries that refer to a certain machine address.
  - Used during TLB shootdown.

Major Data Structures of Disco

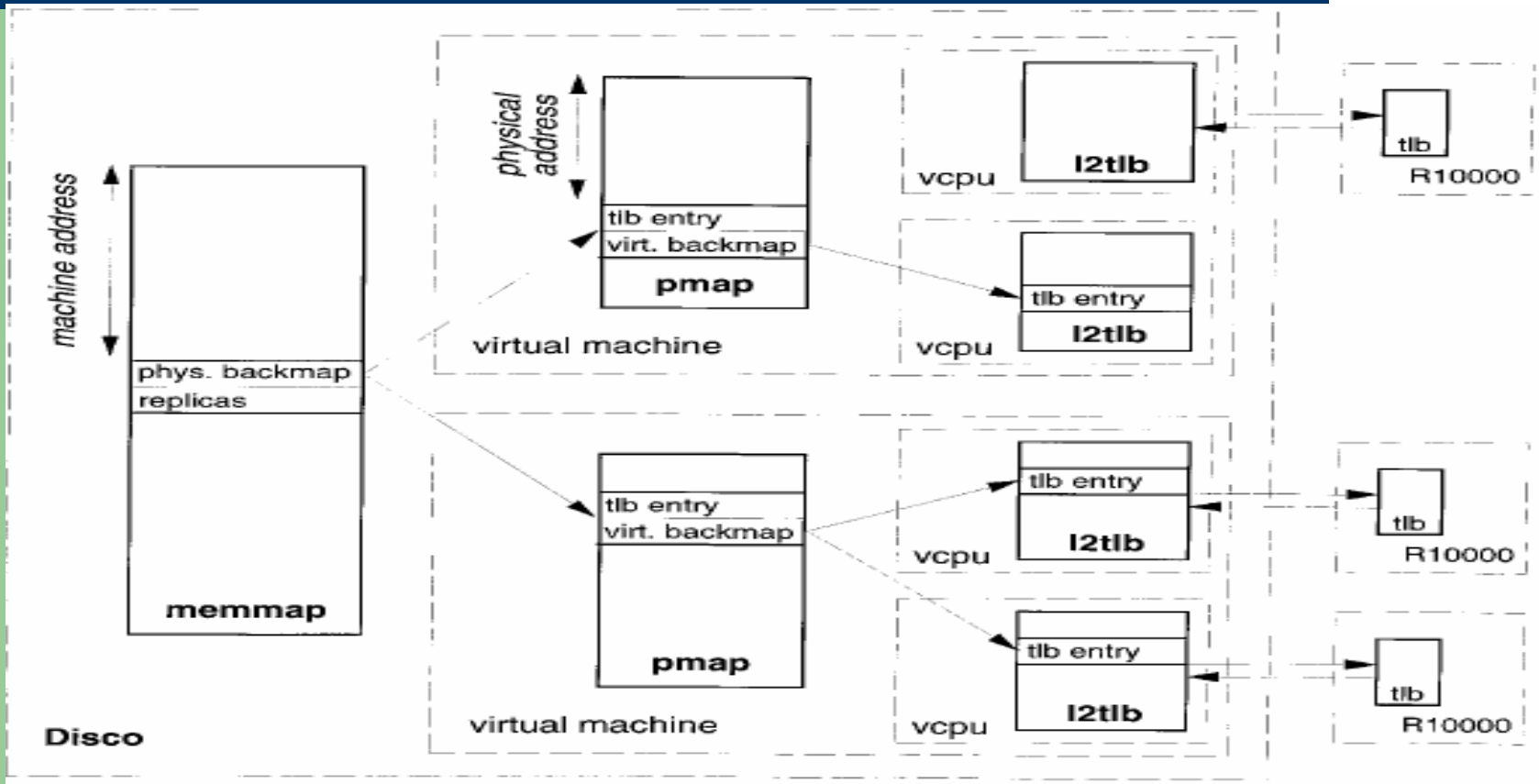


Fig. 3. Major data structures of Disco.

Virtual I/O Devices

- Disco intercepts all device accesses from the virtual machine and forwards them to the physical devices (*may cause extra overhead*)
- Each Disco device defines a monitor call used by the device driver to pass all command arguments in a single trap
- Disco intercept DMA requests to translate the physical addresses into machine addresses.

Copy-on-Write Disks

- Disco intercepts every disk request that DMAs data into memory
- Attempts to modify a shared page will result in a copy-on-write fault handled internally by the monitor.

Shared Machine Memory

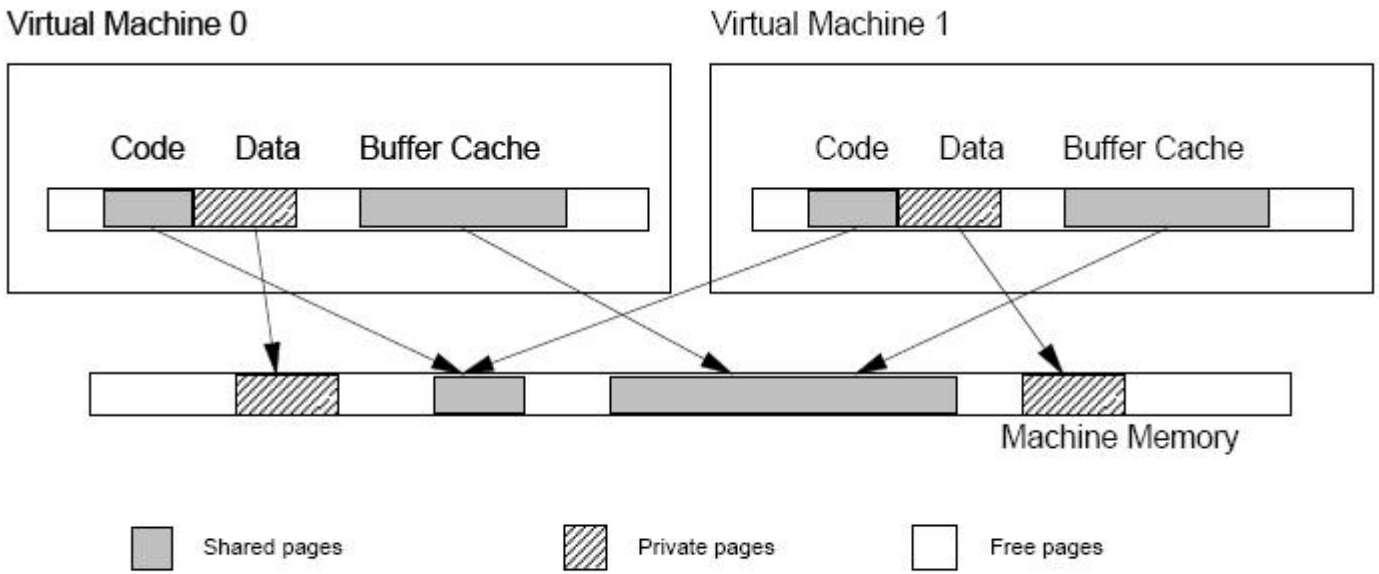


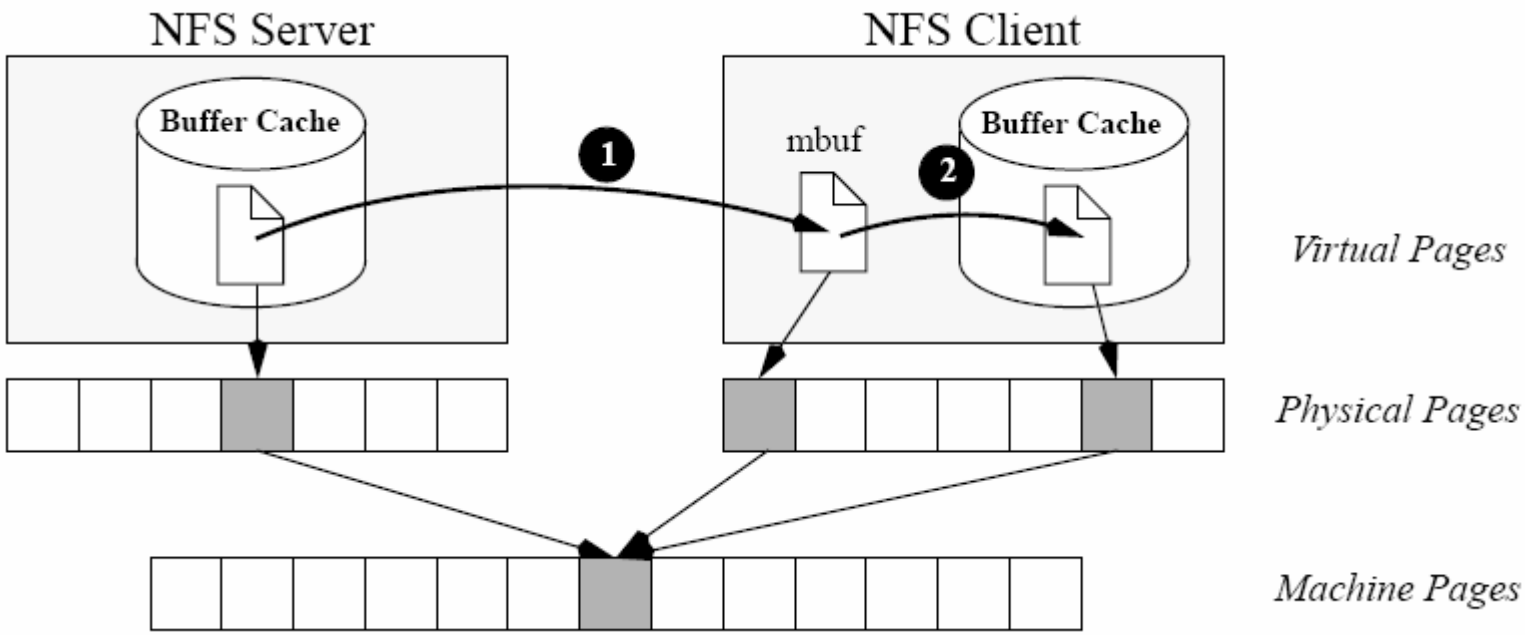
Fig. 4. Memory Sharing in Disco

Isolation of disk write

- To preserve the isolation of the virtual machines, disk writes must be kept private to the virtual machine that issues them.
- Disco logs the modified sectors so that the copy-on-write disk is never actually modified.

Virtual Network Interface

- Use standard distributed protocols to communicate.
- Communication between virtual machines by accessing data in shared cache
- Avoid duplication of data
- Use sharing whenever possible


NFS Server
NFS Client
Buffer Cache
mbuf
Buffer Cache
1
2
Virtual Pages
Physical Pages
Machine Pages

Running Commodity Operating Systems

- Minor changes to kernel code and data segment (MIPS architecture)
- Disco uses IRIX's original device drivers
- Patched the HAL of IRIX to convert frequently used privileged instructions to use non-trapping load and store instructions to a special page of address space that contains these registers.
- Added code to the HAL to pass hints to the monitor for better decision
- IRIX

SPLASHOS

- Thin specialized library OS, supported directly by Disco (no need for virtual memory subsystem)

Experimental result

- No Flash Hardware-> Simulation on SimOS
- Four Workloads
- Execution Overheads
- Memory Overheads
- Scalability
- Dynamic Page Migration and Replication
- Single Hardware-> SGI Origen200

Related Work and Conclusion

- System Software for Scalable Shared Memory Machines
- Virtual Machine Monitors
- Other System Software Structuring Techniques
- CC-NUMA Memory Management

My point to Disco

- Target Disco for a machine that doesn't physically exist, e.g. FLUSH, is a little bit silly.
- Supporting for VMs based on multiple ISAs.
- Scheduling among different VMs.
- In order to reduce the memory overhead, Disco transparently share major data structures such as the program code and the file system buffer cache. But is may be meaningless if OSs using different format of execution binaries.
- Overall, a interesting paper.

Weakness

- Virtual physical memory is done in Disco by catching TLB misses. However, this only applies to the MIPS architecture. Nearly all other architectures use hardware loaded TLB. The paper doesn't propose a way to provide virtual physical memory without the ability to catch TLB misses, which is often true. It seems that there isn't a simple solution to this. So it may be a considerable problem for implementing virtual machines efficiently on other architectures

Discussion Areas

- Modification to hardware and operating systems
- Scheduling on machine CPUs
- Implementation details
- Comparison with other works
- Overall Performance

Modification to Hardware and OS

- Disco required the OS to be modified slightly to be run on it. Does this remain true even with the new hardware assisted virtualization on chipsets?
- In order to improve performance, guest OS needs to be changed. Can current intel-vt solve this problem?

Modification to Hardware and OS

- It seems that the hardware on which Disco runs has very little variation in peripheral devices—that is, there is ***one*** type of Ethernet card for which Disco has a driver, there is ***one*** type of disk drive, and so on. This makes the virtual machine easy to implement. In reality, the vast majority of the source code in Linux today is drivers for the huge number of different types of devices. Doesn't this argue that Disco's approach to building a VMM is going to result in a VMM that contains almost as much code as an OS, which is exactly what the authors were trying to avoid?

Modification to Hardware and OS

- How many code in both disco and OS should be added/modified? I think it should be more than directly modifying the OS.
- How can the techniques be modified (or perhaps without modification?) to be utilized on more recent multi-core processors?

Modification to Hardware and OS

- In the paper, the authors run the IRIX operating system on Disco, and say that most of the changes were made in the hardware abstraction level, and could have been performed as easily with a binary translation. Would running a modern OS that required many more changes to the HAL still be as easy as they claim? Would supporting different OS's require many more changes in Disco itself? This could discount one of their main points, which is that they obtain modest performance that is proportional to the amount of development effort.

Scheduling and fairness

- What kind of steps are taken to ensure fairness between the OSs that are running simultaneously get fair access to all the resources - memory, I/O time and CPU time? Or are they expected to be co-operative?
- In deciding which VM gets the processor, it seems that its not going to make a optimal decision in that. How would you make it more optimal than the current one ?

Scheduling and fairness

- How to schedules VMs on the real CPUs?
  In order to improve the data locality, is one VM affiliated to one CPU? But in this way, the mapping between VM and CPU is static.
  And how many virtual CPU should be exposed to a VM?
- The paper has not mentioned how it schedule virtual CPU on multiprocessor, I want to ask whether it can create deadlock.

Implementation Details

- In network interface virtualization ,disco use copy-on write to reduce memory copies. However, how do deal with the memory alignment problem, if two operating systems using different alignment?

Overall Performance

- Do you think DISCO is a good trade-off between scalability and system performance?
- As the author mentioned that FLASH is not available at that time, is it available now?

Overall Performance

- I'm curious about DISCO's extensibility. How easy do you think is it to introduce a new machine with a new operating system into the existing Disco setup that already contains a set of operating systems?

Overall Performance

- The paper says that eight virtual machines can run some workloads 1.7 faster than on a commercial symmetric multiprocessor OS, which confuses me so much. Consider the overload made by the schedule of Virtrual CPU and others, whether the process can speed up?