

Cellular Disco

K. Govil, D. Teodosiu, Y. Huang, M. Rosenblum
Symposium on Operating System Principles 1999

CS 6210, Advanced Operating Systems

Presented by Scott McManus

Cellular Disco Overview

- Motivation & Background
- Architecture
- Virtualization
- Resource Management
- Fault Recovery
- Performance
- Conclusion

OS Issues with Shared Memory Systems

- Scalability
 - Centralized resource management won't work
- Fault Tolerance
 - A single fault can cause an entire system to become unreliable or crash, especially with shared memory
 - Faults occur more frequently with so many nodes
- Not all of the hardware features are being used well (i.e., software can't keep up with hardware)
 - Cache Coherent, Non-Uniform Memory Access (CC-NUMA) is not handled well – local versus remote memory access was being treated the same in most OSs.

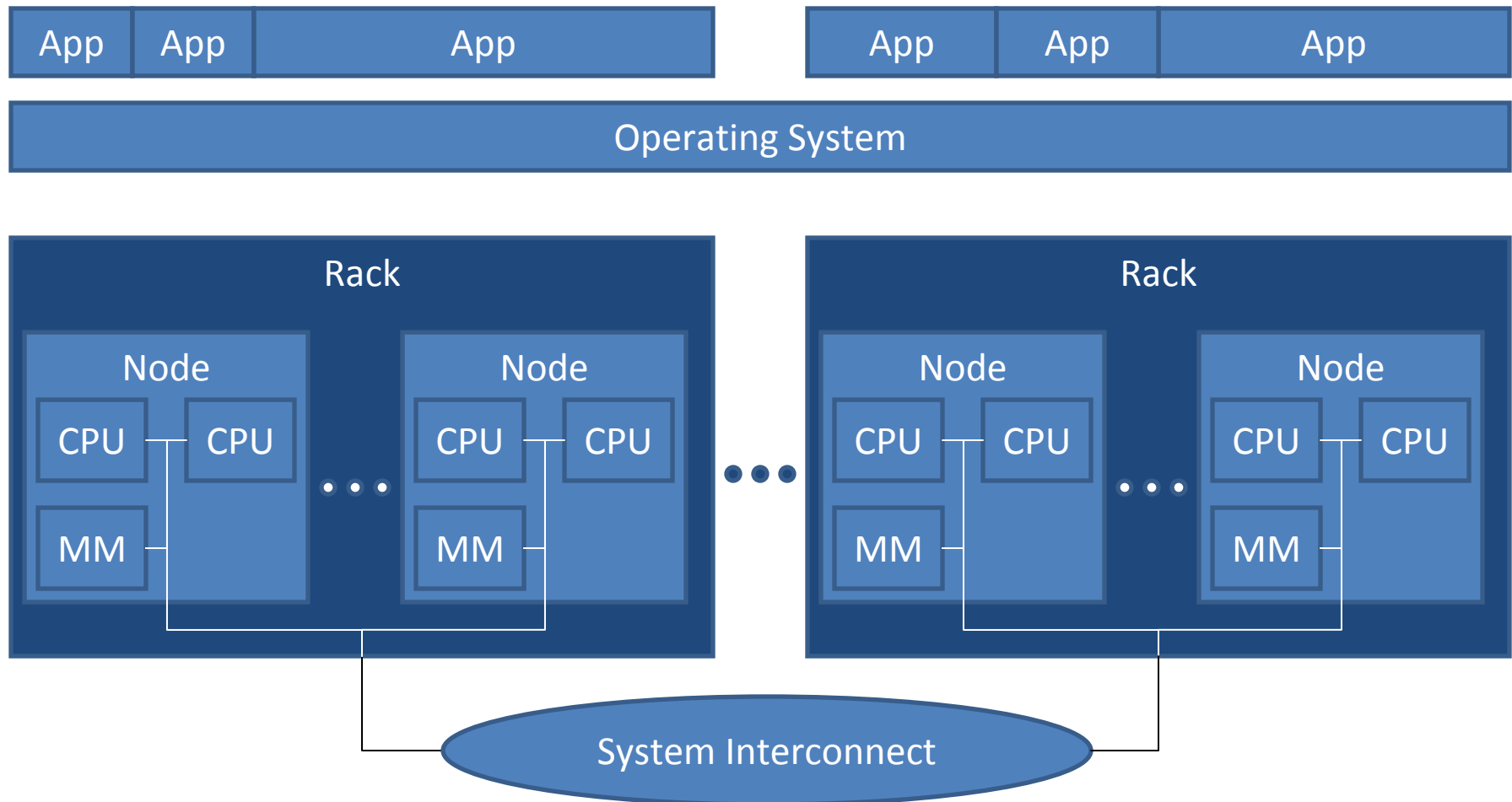
What Groups Could Be Interested

- High Performance Computing
 - Thousands of nodes that work on a single problem
 - Platform in paper (SGI Origin 2000) actually one of top 10 benchmarked machines in 1999 (top500.org)
- Large services – e.g., Google
- Cloud Computing
 - Depends on your definition of “cloud”, but fault management and scalability are certainly issues

Shared Memory Architecture for Paper

- Each node has:
 - 2 CPUs
 - Up to 4 GB main memory
 - Access to low-latency hypercube @ 6.4 Gbps
 - Other nodes can directly access main memory
- Each rack/cabinet has:
 - Some nodes (up to 8)
 - Access to disk (for paging, storage, etc)

Architecture Under Consideration – Origin 2000/Onyx2 (SGI)



What Doesn't Work

- Why not just partition the hardware into smaller clusters?
 - Pro: Fault containment achieved
 - Con: Limits resource sharing
- Why not just write an OS for these systems?
 - Complexity of hardware makes writing software difficult and expensive

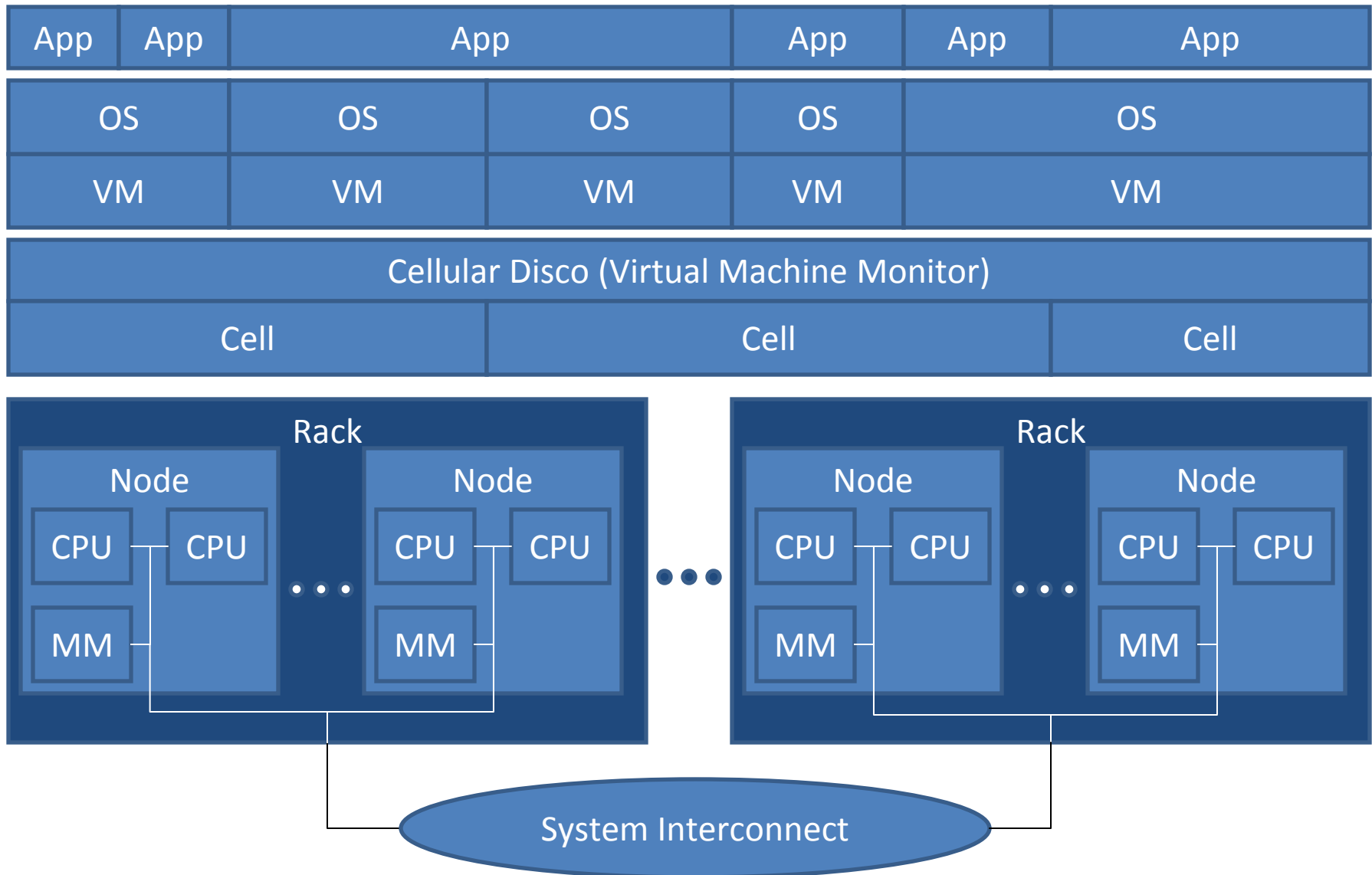
Cellular Disco Precursor - Disco

- Start with Disco
 - Disco is a Virtual Machine Monitor (VMM)
 - Commodity OSs run with low overhead
 - Full virtualization – no changes to OS needed (unlike, say, Xen)

Cellular Disco Approach

- Cellular Disco is a Virtual Machine Monitor (VMM) that handles:
 - Resource management
 - Hardware fault containment
- SMP machines are now turned into a group of virtual clusters or *cells*
- Cells:
 - Run on some number of nodes
 - Isolate hardware faults to the nodes in the cell
 - Contain a complete copy of the VMM for reliability

Cellular Disco Architecture



Hardware Virtualization - Resources

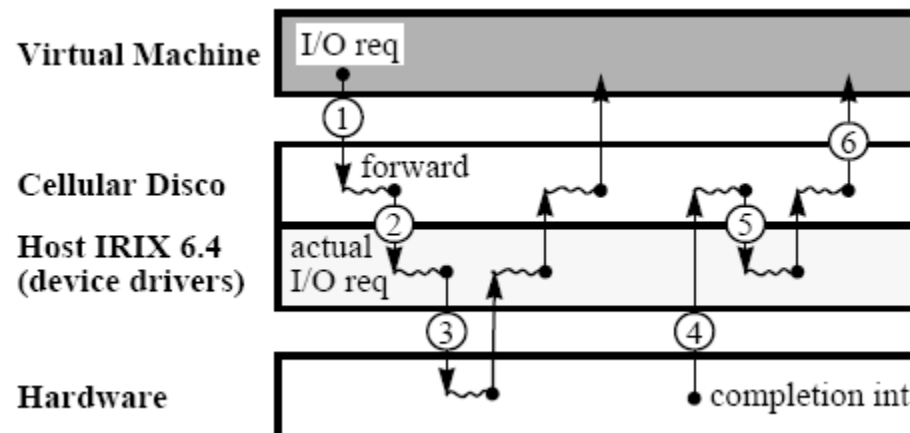
- ***Physical resources*** are visible to a VM.
- ***Machine resources*** are only visible to Cellular Disco
- Cellular Disco allocates the machine resources to the VMs as needed.

Hardware Virtualization – Run Levels

- Rather than intercept and handle all privileged operations, use the MIPS privilege levels
 - User mode: least privileged
 - Supervisor mode: have constrained access to supervisor space
 - Kernel mode: full privileges, including execution of privileged instructions
- Cellular Disco:
 - Only VMM runs in kernel mode
 - OSs in VMs get access to supervisor mode

Hardware Virtualization – I/O

- I/O access is privileged
 - VMM intercepts requests and forwards them or handles them
 - IRIX (original host kernel) is awoken **only** for device driver access



Hardware Virtualization - Memory

- Memory allocation
 - Only the VMM has privilege to allocate memory
 - The *pmap* maps physical to machine memory
 - The *memmap* does the inverse mapping
 - These are needed for replication and fault recovery
- VMM avoids pmap access using a software Translation Lookaside Buffer (TLB) called L2TLB
 - Accessing L2TLB is faster than sending TLB miss exception to the OS

Resource Management - CPU

- Each actual CPU maintains run queue of VCPUs
- Load between actual CPUs will eventually differ
- VCPU migration is used
 - Intranode: migrate between CPUs on a node
 - Internode: migrate between nodes in a cell
 - Intercell: migrate between cells

VCPU Migration Details

- Intranode (same node, different CPU)
 - The real cost is the loss of cache affinity
- Internode (same cell, different node)
 - Frequently accessed structures (L2TLB) and memory copied over

VCPU Migration Details - Intercell

- Intercell (different cells)
 - Avoid faults in old cell by moving all memory structures
 - Want to avoid penalty for moving physical-machine maps (pmap & memmap) often
 - Only migrate when ALL VCPUs in a cell have migrated
 - Eventually migrate data pages (not just mappings)

Resource Management - Scheduling

- Gang scheduling idea:
 - All tasks are scheduled at the same time on a CPU
 - This avoids penalties when one task is waiting on another descheduled task
- Cellular Disco uses gang scheduling for same reason – to avoid problems from spinlock waiting

Resource Management – Gang Scheduling

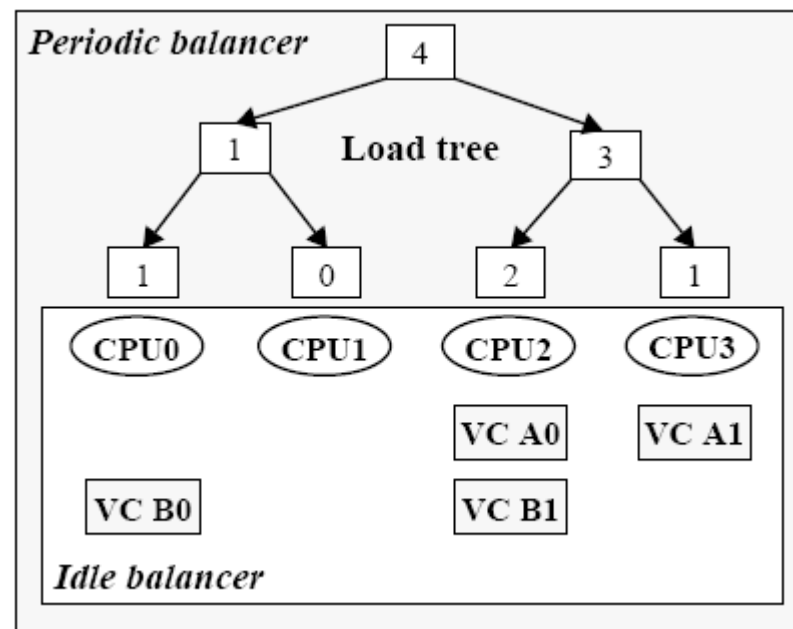
- Cellular Disco always picks the highest-priority gang-runnable VCPU
 - Scheduler sends RPC messages to all processors that have VCPUs corresponding to the VCPU's VM
 - The high-priority VCPUs are scheduled by each physical processor
 - (Think of the mechanism for coscheduling in project 1.)

Resource Management – CPU Idle Balancing

- Idle Balancer
 - Does most of balancing work
 - Run whenever a processor becomes idle
 - Look for VCPUs on same cell to steal
 - Gang scheduling restricts what VCPUs can be run
 - Want to steal VCPUs that can run if they're migrated

CPU Idle Balancing Example

- Figure 8 from paper:

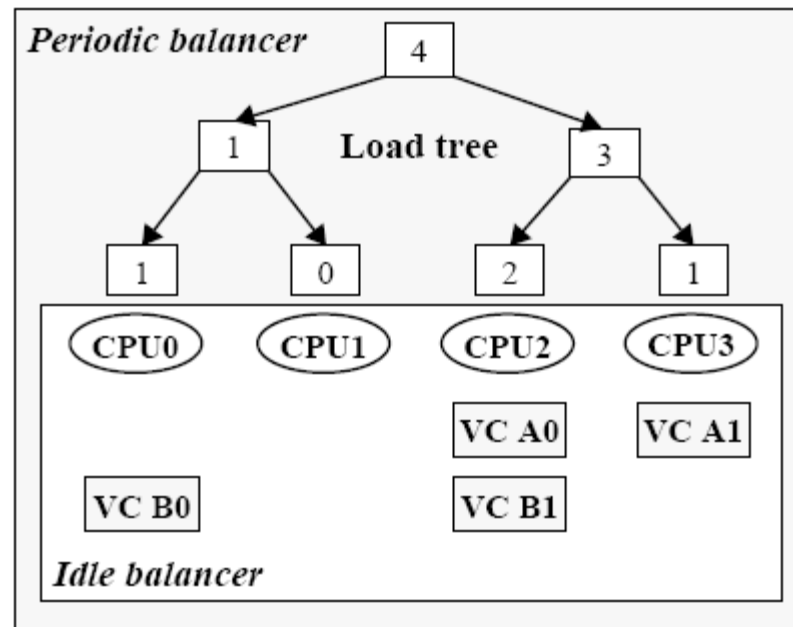


Periodic Balancer

- Idle Balancer only looks at same-cell information, so it's not globally optimal
- Periodic balancer uses global load information to migrate VCPUs
 - Just as in MCS, the tree is distributed across nodes
 - Processor updates tree every 10 ms
 - Tree load is aggregated to find disparities
 - Avoid faults by migrating VCPUs already dependent on the cell that will receive it.

Periodic Balancer - Example

- Figure 8 from paper:



Resource Management - Memory

- Paper focuses on intercell management
- Each cell maintains freelist indexed by home node of pages

Resource Management – Memory Balancing Policies

- Attempt to acquire memory on the same node first
- Want small VMs to avoid remote accesses
 - Larger VMs more likely to handle the cost
- Cells borrow memory only when low threshold of pages met
 - Avoid running out of pages altogether

Memory Balancing Policy

- When cell drops below low-water mark, ask all preferred cells for a chunk of memory.
- Cells will only approve the request if they have a significant amount of memory available.

Resource Management - Paging

- Eventually have to page out to disk
- Three issues:
 - How are actively used pages identified?
 - How should Cellular Disco handle memory pages shared by different VMs?
 - How should Cellular Disco avoid redundant paging between the OS and the VMM?

Resource Management - Paging

- Identifying actively-used pages:
 - The VMM monitors physical pages being used
- Handling shared memory pages:
 - Write sharing information out to disk, too
- Avoiding redundant paging:
 - The VMM monitors writes to disk.
 - If the information has been paged, then the VMM just marks it as already being on disk.

Fault Recovery

- Hardware faults on a single node can bring even big systems to an abrupt halt.
 - If many VMs use a particular node, then all of those VMs are lost.
 - Dependencies on a faulty VM can wreck all computation.
 - Likewise, large VMs can still be halted when they depend on a single node.
 - Even today, the solution is often to reset and reboot.

Fault Recovery

- CPUs must be able to recover from faults to some degree
- Recovery sequence:
 - Agree on set of live nodes (*liveset*)
 - Drop all communications to or from failed cells
 - Terminate all VMs that had dependencies on the failed cells
 - Scan all memory for incoherent cache lines
 - May wait to terminate VM until incoherent data read

Performance

- For the most part, performance is within 10% of non-Cellular Disco performance
 - Couple of outliers @ 20% slower
- Most of the performance loss is due to virtualization overheads

Conclusion

- Cellular Disco gives good scalability, fault recovery, and ccNUMA awareness to justify its overhead