Shared Memory Multiprocessors

Ken Birman

Draws extensively on slides by Ravikant Dintyala

Big picture debate

- How best to exploit hardware parallelism?
  - "Old" model: develop an operating system married to the hardware; use it to run one of the major computational science packages
  - "New" models: seek to offer a more transparent way of exploiting parallelism
- Today's two papers offer distinct perspectives on this topic

Contrasting perspectives

- Disco:
  - Here, the basic idea is to use a new VMM to make the parallel machine look like a very fast cluster
  - Disco runs commodity operating system on it
- Question raised
  - Given that interconnects are so fast, why not just buy a *real* cluster?
  - Disco: focus is on benefits of shared VM

Time warp…

- As it turns out, Disco found a commercially important opportunity
  - But it wasn't exploitation of ccNUMA machines
  - Disco morphed into VMWare, a major product for running Windows on Linux and vice versa
  - Company was ultimately sold for $550M
- …. Proving that research can pay off!

Contrasting perspectives

- Tornado:
  - Here, assumption is that shared memory will be the big attraction to end user
    - But performance can be whacked by contention, false sharing
    - Want "illusion" of sharing but hardware-sensitive implementation
  - They also believe that user is working in an OO paradigm (today would point to languages like Java and C#, or platforms like .net and CORBA)
  - Goal becomes: provide amazingly good support for shared component integration in a world of threads and objects that interact heavily

Bottom line here?

- Key idea: clustered object
  - Looks like a shared object
  - But actually, implemented cleverly with one local object instance per thread…
- Tornado was interesting…
  - … and got some people PhD's and tenure
  - … but it ultimately didn't change the work in any noticeable way
- Why?
  - Is this a judgment on the work? (Very architecture-dependent)
  - Or a comment about the nature of "majority" OS platforms (Linux, Windows, perhaps QNX)?

Trends when work was done

- A period when multiprocessors were
  - Fairly tightly coupled, with memory coherence
  - Viewed as a possible cost/performance winner for server applications
- And cluster interconnects were still fairly slow
- Research focused on several kinds of concerns:
  - Higher memory latencies; TLB management is critical
  - Large write sharing costs on many platforms
  - Large secondary caches needed to mask disk delays
  - NUMA h/w, which suffers from false sharing of cache lines
  - Contention for shared objects
  - Large system sizes

OS Issues for multiprocessors

- Efficient sharing
- Scalability
- Flexibility (keep pace with new hardware innovations)
- Reliability

Ideas

- Statically partition the machine and run multiple, independent OS's that export a partial single-system image (Map locality and independence in the applications to their servicing - localization aware scheduling and caching/replication hiding NUMA)
- Partition the resources into cells that coordinate to manage the hardware resources efficiently and export a single system image
- Handle resource management in a separate wrapper between the hardware and OS
- Design a flexible object oriented framework that can be optimized in an incremental fashion

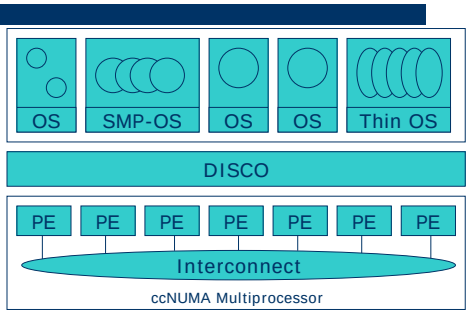
Virtual Machine Monitor

- Additional layer between hardware and operating system
- Provides a hardware interface to the OS, manages the actual hardware
- Can run multiple copies of the operating system
- Fault containment – os and hardware

Virtual Machine Monitor

- Additional layer between hardware and operating system
- Provides a hardware interface to the OS, manages the actual hardware
- Can run multiple copies of the operating system
- Fault containment – os and hardware
- Overhead, Uninformed resource management, Communication and sharing between virtual machines?

DISCO



Interface

- Processors – MIPS R10000 processor (kernel pages in unmapped segments)
- Physical Memory – contiguous physical address space starting at address zero (non NUMA aware)
- I/O Devices – virtual disks (private/shared), virtual networking (each virtual machine is assigned a distinct link level address on an internal virtual subnet managed by DISCO; communication with outside world, DISCO acts as a gateway), other devices have appropriate device drivers

Implementation

- Virtual CPU
- Virtual Physical Memory
- Virtual I/O Devices
- Virtual Disks
- Virtual Network Interface

All in 13000 lines of code

Major Data Structures

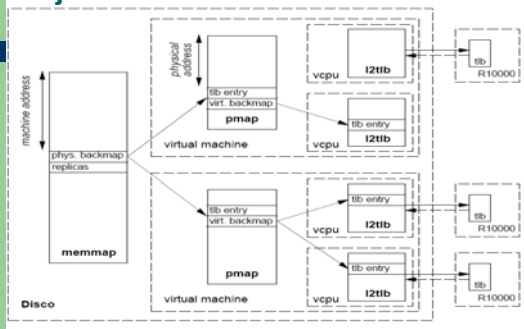


Fig. 3. Major Data Structures of Disco

Virtual CPU

- Virtual processors time-shared across the physical processors (under "data locality" constraints)
- Each Virtual CPU has a "process table entry" + privileged registers + TLB contents
- DISCO runs in kernel mode, the host OS in supervisor mode, others run in user mode
- Operations that cannot be issued in supervisor mode are emulated (on trap – update the privileged registers of the virtual processor and jump to the virtual machine's trap vector)

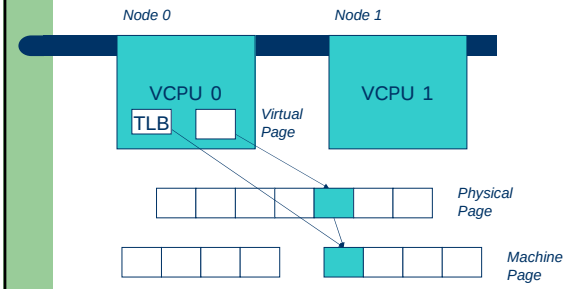
Virtual Physical Memory

- Mapping from physical address (virtual machine physical) to machine address maintained in pmap
- Processor TLB contains the virtual-to-machine mapping
- Kernel pages – relink the operating system code and data into mapped region.
- Recent TLB history saved in a second-level software cache
- Tagged TLB not used

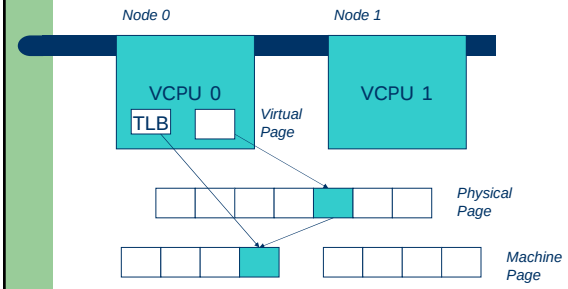
NUMA Memory Management

- Migrate/replicate pages to maintain locality between virtual CPU and it's memory
- Uses hardware support for detecting "hot pages"
  - Pages heavily used by one node are migrated to that node
  - Pages that are read-shared are replicated to the nodes most heavily accessing them
  - Pages that are write-shared are not moved
  - Number of moves of a page limited
- Maintains an "inverted page table" analogue (memmap) to maintain consistent TLB, pmap entries after replication/migration

Page Migration

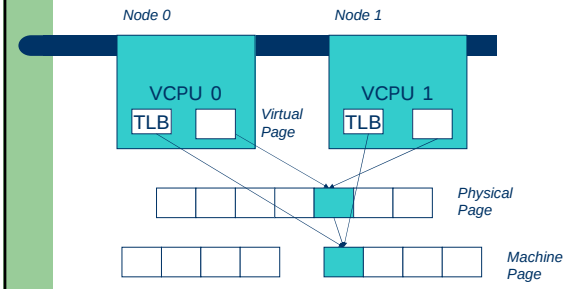


Page Migration

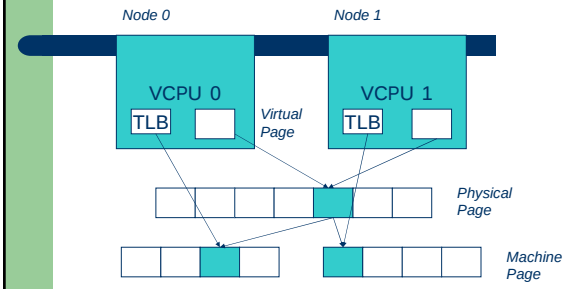


memmap, pmap and tlb entries updated

Page Migration



Page Migration



memmap, pmap and tlb entries updated

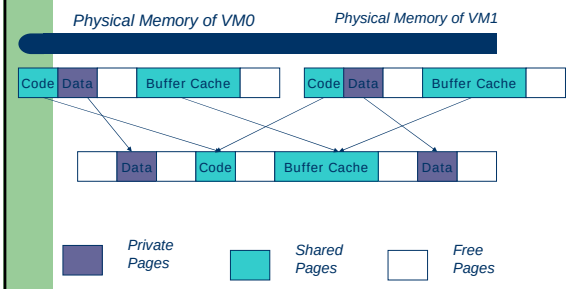
Virtual I/O Devices

- Each DISCO device defines a monitor call used to pass all command arguments in a single trap
- Special device drivers added into the OS
- DMA maps intercepted and translated from physical addresses to machine addresses
- Virtual network devices emulated using (copy-on-write) shared memory

Virtual Disks

- Virtual disk, machine memory relation is similar to buffer aggregates and shared memory in IOLite
- The machine memory is like a cache (disk requests serviced from machine memory whenever possible)
- Two B-Trees are maintained per virtual disk, one keeps track of the mapping between disk addresses and machine addresses, the other keeps track of the updates made to the virtual disk by the virtual processor
- Propose to log the updates in a disk partition (actual implementation handles non persistent virtual disks in the above manner and persistent disk writes routed to the physical disk)

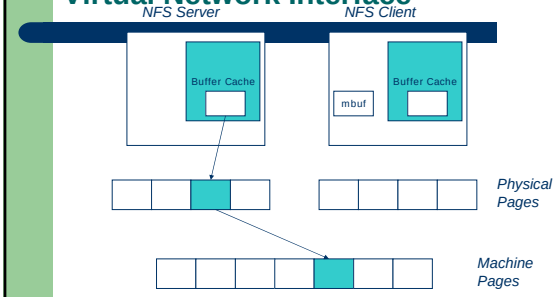
Virtual Disks



Virtual Network Interface

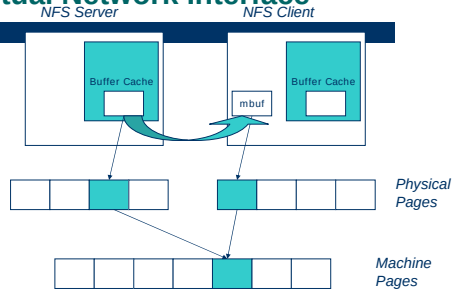
- Messages transferred between virtual machines mapped read only into both the sending and receiving virtual machine's physical address spaces
- Updated device drivers maintain data alignment
- Cross layer optimizations

Virtual Network Interface



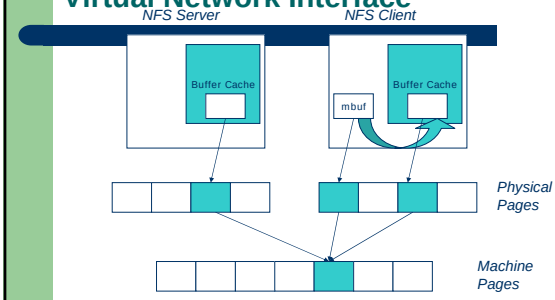
Read request from client

Virtual Network Interface



Data page remapped from source's machine address space to the destination's

Virtual Network Interface



Data page from driver's mbuf remapped to the clients buffer cache

Running Commodity OS

- Modified the Hardware Abstraction Level (HAL) of IRIX to reduce the overhead of virtualization and improve resource use
- Relocate the kernel to use the mapped supervisor segment in place of the unmapped segment
- Access to privileged registers – convert frequently used privileged instructions to use non trapping load and store instructions to a special page of the address space that contains these registers

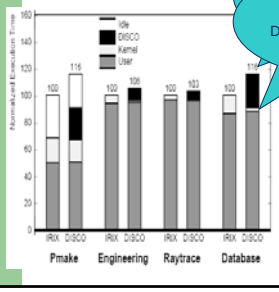
Running Commodity OS

- Update device drivers
- Add code to HAL to pass hints to the monitor, giving it higher level knowledge of resource utilization (eg: a page has been put on the OS free page list without chance of reclamation)
- Update mbuf management to prevent freelist linking using the first word of the pages and NFS implementation to avoid copying

Results – Virtualization Overhead

16% overhead due to

Decrease in kernel overhead since DISCO handles some of the work



chess
gcc
Engineering – concurrent simulation of part of FLASH MAGIC chip

- Raytrace – renders the "car" model from SPLASH-2 suite
- Database – decision support workload

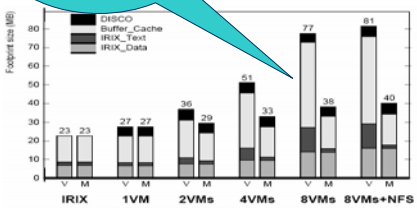
Results – Overhead breakdown of Pmake workload

Operating System Service	% of System Time (IRIX)	Avg Time per Invocation (IRIX)	Slowdown on Disco	Relative Execution Time on Disco				
				Kernel Execution	TLB Write Emulation	Other Privileged Instructions	Monitor Calls & Page Faults	Kernel TLB Faults
DEMAND_ZERO	30%	21 µs	1.42	0.43	0.21	0.16	0.47	0.16
QUICK_FAULT	10%	5 µs	3.17	1.27	0.80	0.56	0.00	0.53
open	9%	42 µs	1.63	1.16	0.08	0.06	0.02	0.30
UTLB_MISS	7%	0.035 µs	1.35	0.07	1.22	0.05	0.00	0.02
write	6%	12 µs	2.14	1.01	0.24	0.21	0.31	0.17
read	6%	23 µs	1.53	1.10	0.13	0.09	0.01	0.20
execve	6%	437 µs	1.60	0.97	0.03	0.05	0.17	0.40

Common path to enter and leave the kernel for all page faults, system calls and interrupts includes many privileged instructions that must be individually emulated

Re ry Overheads

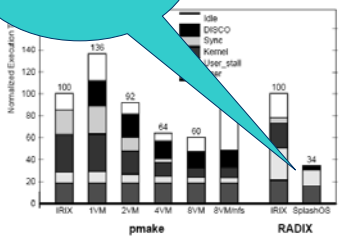
Increase in memory footprint since each virtual machine has associated kernel data structures that cannot be shared



Workload consists of eight different copies of basic Pmake workload. Each Pmake instance uses different data, rest is identical

Re d Scalability

Synchronization overhead decreases Lesser communication misses and lesser time spent in the kernel

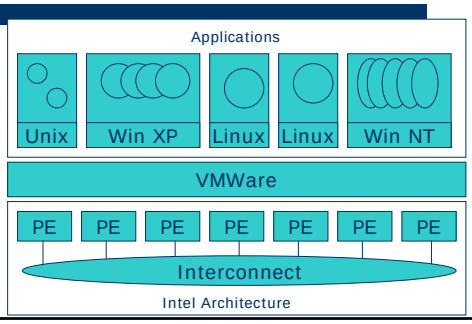


Radix – sorts 4 million integers

Results – On Real Hardware

Breakdown	Pmake			Engineering		
	IRIX (sec.)	Disco (sec.)	Ratio	IRIX (sec.)	Disco (sec.)	Ratio
User	11.3	11.7	1.03	65.2	69.7	1.07
Kernel	5.9	9.6	1.62	0.2	0.2	1.00
Idle	13.1	11.4	0.87	0	0	
Total	30.3	32.7	1.08	65.4	69.9	1.07

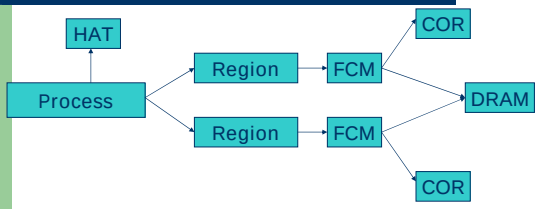
VMWare: DISCO turned into a product



Tornado

- Object oriented design – every virtual and physical resource represented as an object
- Independent resources mapped to independent objects
- Clustered objects – support partitioning of contended objects across processors
- Protected Procedure Call – preserves locality and concurrency of IPC
- Fine grained locking (locking internal to objects)
- Semi-automatic garbage collection

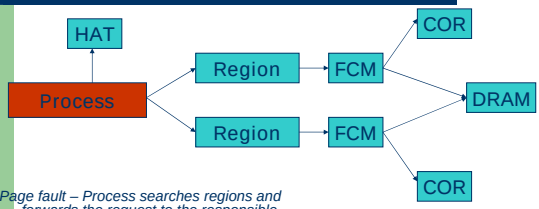
OO Design



Current Structure

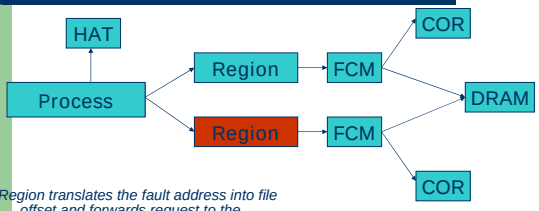
Key: HAT – "hardware address translation. FCM – File cache manager. COR – clustered object representative

OO Design



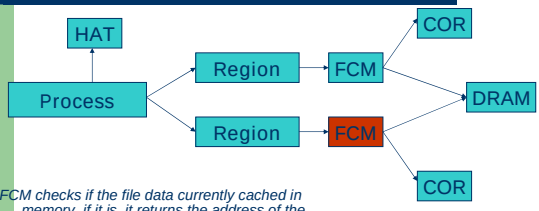
Page fault – Process searches regions and forwards the request to the responsible region

OO Design



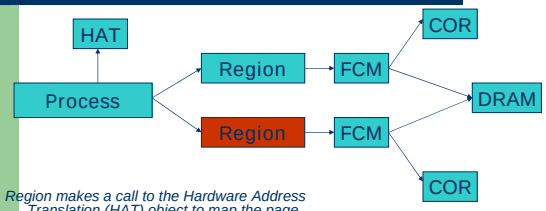
Region translates the fault address into file offset and forwards request to the corresponding File Cache Manager

OO Design



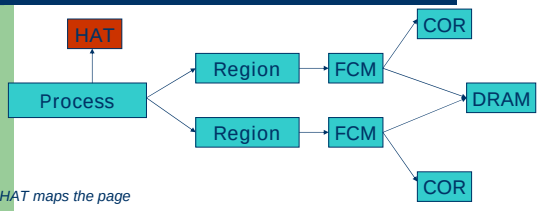
FCM checks if the file data currently cached in memory, if it is, it returns the address of the corresponding physical page frame to the region

OO Design



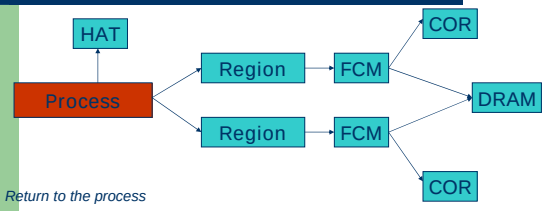
Region makes a call to the Hardware Address Translation (HAT) object to map the page and returns

OO Design



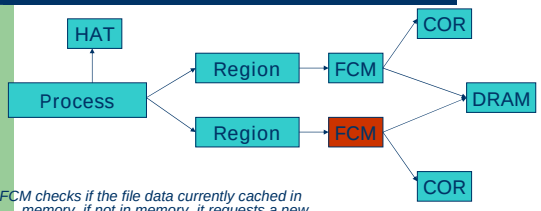
HAT maps the page

OO Design



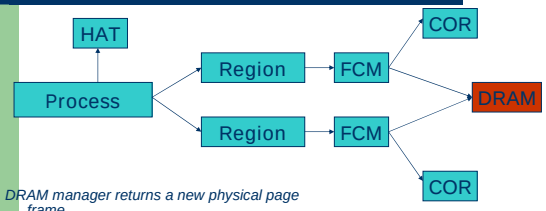
Return to the process

OO Design – miss case



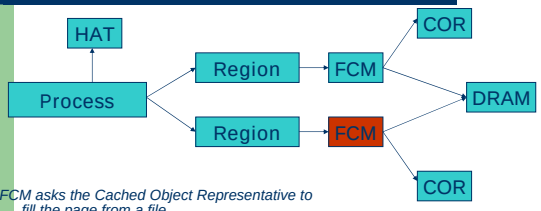
FCM checks if the file data currently cached in memory, if not in memory, it requests a new physical frame from the DRAM manager

OO Design



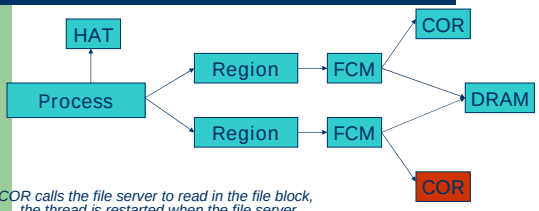
DRAM manager returns a new physical page frame

OO Design



FCM asks the Cached Object Representative to fill the page from a file

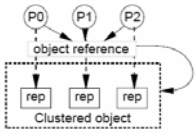
OO Design



COR calls the file server to read in the file block, the thread is restarted when the file server returns with the required data

Handling Shared Objects – Clustered Object

- A combination of multiple objects that presents the view of a single object to any client
- Each component object represents the collective whole for some set of clients – representative
- All client accesses reference the appropriate local representative
- Representatives coordinate (through shared memory/PPC) and maintain a consistent sate of the object



Key: PPC: Protected procedure call

Clustered Object - Benefits

- Replication or partitioning of data structures and locks
- Encapsulation
- Internal optimization (on demand creation of representatives)
- Hot Swapping – dynamically reload a current optimal implementation of the clustered object

Clustered Object example - Process

- Mostly read only
- Replicated on each processor the process has threads running
- Other processors have reps for redirecting
- Modifications like changes to the priority done through broadcast
- Modifications like the region changes updated on demand as they are referenced

Replication - Tradeoffs

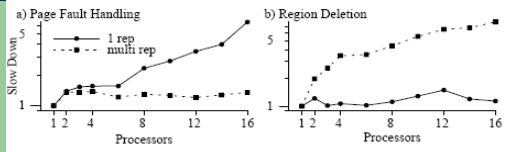
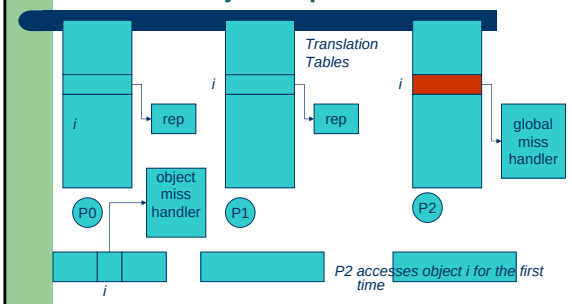


Figure 5: *Comparison of the performance of a Process clustered object with one rep vs. a Process clustered object with n reps (one per processor) for (a) in-core page fault handling and (b) Region destruction*

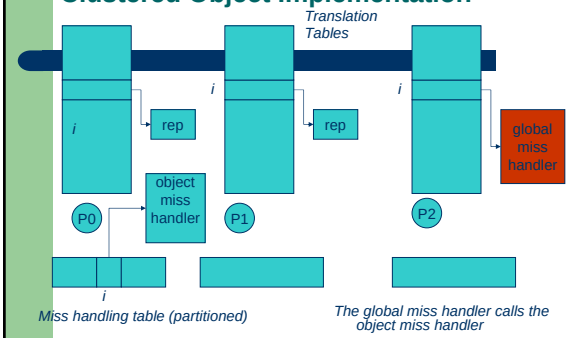
Clustered Object Implementation

- Per processor translation table
- Representatives created on demand
- Translation table entries point to a global miss handler by default
- Global miss handler has references to the processor containing the object miss handler (object miss handlers partitioned across processors)
- Object miss handler handles the miss by updating the translation table entry to a (new/existing) rep
- Miss handling ~ 150 instructions
- Translation table entries discarded if table gets full

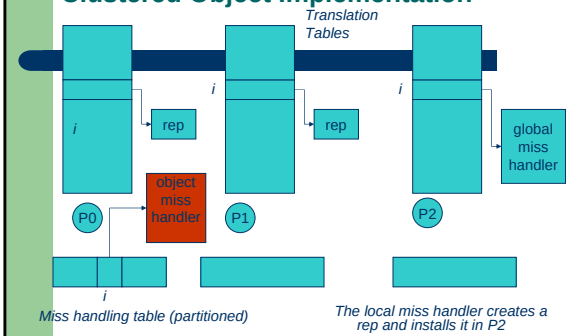
Clustered Object Implementation



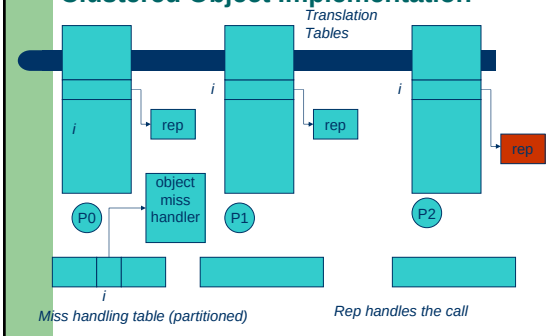
Clustered Object Implementation



Clustered Object Implementation



Clustered Object Implementation



Dynamic Memory Allocation

- Provide a separate per-processor pool for small blocks that are intended to be accessed strictly locally
- Per-processor pools
- Cluster pools of free memory based on NUMA locality

Synchronization

- Locking
  - all locks encapsulated within individual objects
- Existence guarantees
  - garbage collection

Garbage Collection

- Phase 1
  - remove persistent references
- Phase 2
  - uni-processor - keep track of number of temporary references to the object
  - multi-processor – circulate a token among the processors that access this clustered object, a processor passes the token when it completes the uni-processor phase-2
- Phase 3
  - destroy the representatives, release the memory and free the object entry

Protected Procedure Call (PPC)

- Servers are passive objects, just consisting of an address space
- Client process crosses directly into the server's address space when making a call
- Similar to unix trap to kernel

PPC Properties

- Client requests are always handled on their local processor
- Clients and servers share the processor in a manner similar to handoff scheduling
- There are as many threads in the server as client requests
- Client retains its state (no argument passing)

PPC Implementation

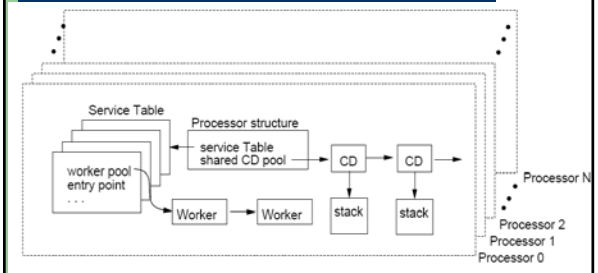


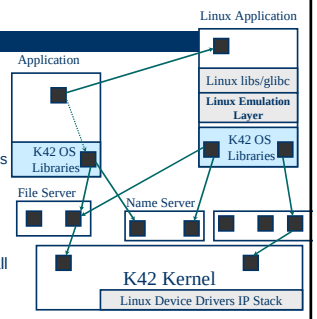
Figure 1: Per-processor PPC data structures

Results - Microbenchmarks

- Effected by false sharing of cache lines
- Overhead is around 50% when tested with 4-way set associative cache
- Does well for both multi-programmed and multi-threaded applications

K42

- Most OS functionality implemented in user-level library
  - thread library
  - allows OS services to be customized for applications with specialized needs
  - also avoids interactions with kernel and reduces space/time overhead in kernel
- Object-oriented design at all levels



Fair Sharing

- Resource management to address fairness (how to attain fairness and still achieve high throughput?)
- Logical entities (eg users) are entitled to certain shares of resources, processes are grouped into these logical entities; logical entities can share/revoke their entitlements

Conclusion

- DISCO – VM layer, not a full scale OS
  - OS researchers who set out to “do good” for the commercial world, by preserving existing value
  - Ultimately a home run (but not in way intended!)
- Tornado – object oriented, flexible and extensible OS; resource management and sharing through clustered objects and PPC
  - But complex – a whole new OS architecture
  - And ultimately not accepted by commercial users