Modern systems: multicore issues

By Paul Grubbs

Portions of this talk were taken from Deniz Altinbuken’s talk on Disco in 2009:
What papers will we be discussing?


High-level context

General-purpose operating systems must run efficiently on many different architectures.

Multiprocessing

Non-uniform memory access (NUMA)

(Cache coherence?)

Commodity, general-purpose OSs are not designed to do this

Rewriting them should be avoided

Multi-Core vs. Multi-Processor

Multi-Core Processor with Shared L2 Cache

Multi-Processor System with Cores that share L2 Cache
Shared Memory Access

Uniform memory access
• Access time to all regions of memory the same
• Access time by all processors the same

Non-uniform memory access
• Different processors access different regions of memory at different speeds
What is the problem being considered?

Multiprocessing requires extensive OS rewrites

NUMA is hard, more rewrites
What is the authors’ solution to this problem?

A new twist on an old idea: virtual machine monitors (VMM).

Updated VMMs for the multiprocessing era.
Disco vs. exokernels?

Exokernel leaves resource management to applications

Only multiplexes physical resources

Disco virtualizes them

Disco can run commodity OSs with little or no modification

More difficult to run commodity OSs on Exokernels
Disco vs. System/370?

Both are VM monitors

VM/370 maps virtual disks to physical disk partitions

  Disco uses shared copy-on-write disks to decrease storage overhead

Disco supports ccNUMA multiprocessors

  Heavily optimizes for NUMA and shared mem access
**FIGURE 1. Architecture of Disco:** Disco is a virtual machine monitor, a software layer between the hardware and multiple virtual machines that run independent operating systems. This allows multiple copies of a commodity operating system to coexist with specialized “thin” operating systems on the same hardware. The multiprocessor consists of a set of processing elements (PE) connected by a high-performance interconnect. Each processing element contains a number of processors and a portion of the memory of the machine.
Abstractions of hardware

Virtual CPU

Virtualized physical memory

Virtualized I/O devices
Virtual CPUs

No emulation of most instructions: code runs “raw” on hardware CPU

Exception: privileged calls (TLB, device access) must be emulated by Disco

   Disco keeps process table for each vCPU for fast emulation

vCPU scheduler to allow time-sharing on physical CPUs

Compare to Xen paravirtualization?
Virtualized physical memory

Offers uniform memory abstraction to commodity OSs,

  - uses ccNUMA memory of multiprocessor

  Dynamic page migration/replication

a small change to OS: Disco allocates shared memory

  - regions that multiple VMs can access

  - DB w/ shared buffer cache

Drawback: redundant OS/application code

  Solution: Transparent sharing of redundant read-only pages like kernel code
Virtualized I/O devices

No device virtualization really

Add special VMM-specific device drivers to kernel of OS

Pages handled using copy-on-write

  Works well for read-only

  Persistent disks only mounted on one VM

  VMs read other disks using NFS
DISCO: Copy-on-write Disks

Physical Memory of VM0

Code | Data | Buffer Cache

Physical Memory of VM1

Code | Data | Buffer Cache

Data | Code | Buffer Cache | Data

Private Pages | Shared Pages | Free Pages
How do they assess the quality of their solution?

FLASH didn’t exist yet so used an OS simulator

They weren’t able to simulate the machine particularly well

No benchmarks for long-running or complicated processes

Disco’s resource sharing policies were only superficially tested

They focused on four uses cases

Parallel compilation of GNU chess application

Verilog simulation of hardware

Raytracing

Sybase RDBMS
Thoughts/Questions?

Do you prefer Disco’s virtualization approach or hardware multiplexing, e.g. Exokernels? Which do you think is better?

Disco makes support for commodity OSs a first-class goal.

   Is this desirable? Does it lead to suboptimal design decisions?

   In OS research is it necessary to preserve backwards-compatibility?

Does not having a real machine to test on hurt the paper?

What did you really like about this paper?

What did you really not like about this paper?
What is the problem being considered?

Diversity in systems, diversity in cores, diversity in multiprocessor architectures

What is the authors’ solution to this problem?

New OS structure: “multikernel”

How do they assess the quality of their solution?

Various benchmarks for cache coherence, RPC overhead
Three key ideas:

1. Make all inter-core communication explicit.

1. Make OS structure hardware-neutral.

1. View state as replicated instead of shared.
1. Make all inter-core communication explicit.

Inter-core communication uses explicit messages

- Avoids shared memory

Multiprocessors look more and more like networks

- Using messages allows easy pipelining/batching
- Makes interconnect use more efficient

Automated analysis/formal verification

- Calculi for reasoning about concurrency
2. Make OS structure hardware-neutral.

Separate OS structure from physical instantiation: abstraction!

- Only message transport and hardware interfaces are machine-specific

Minimizes code change to OS

Separate IPC protocols from hardware implementation

- Performance/extensibility benefits
3. View state as replicated instead of shared.

Shared state is accessed as a local replica

Shared state consistency through messages

   Consistency reqs tunable using diff protocols

Reduces interconnect traffic and synchronization overhead

   Fault-tolerant to failures in CPUs
Figure 1: The multikernel model.
Thoughts/questions?

Relying on distributed protocols for consistency of shared state

   Good idea/bad idea? Why?

Multikernels do not target support for commodity OS

   Good idea/bad idea? Why?

Is their “system-as-network” model accurate? Should the interconnect be treated like other communication channels?

What did you really like about this paper?

What did you really not like about this paper?
What connects these two papers?

Multiprocessing! NUMA!