CS 6410
28sep2010
Multi-Processors

Vainstein K.

Multi-Processor: Definition, Kinds

- Defined: shared-memory machine, *N* CPUs, each of which can access remote memory directly via HW. Flynn's classification [1972]: MIMD (multiple instruction streams, multiple data streams)
- UMA, bus (➔ Your laptop) or switched. aka SMP
- NUMA
  - NC-NUMA: no local caches
  - ccNUMA: with local caches, which are kept coherent

    ➔ What we'll be talking about today

Source: Tanenbaum, "Structured Computer Organization", ed 4

First VMMs: When and Why

- 3rd-generation OSes: multiprogramming, "grid computing" (timesharing), spooling. 1965-80
- notables: OS/360 family, MULTICS, CTSS
- OSes divided into *privileged software nucleus* (modern term: kernel), and everything else
- nucleus's "ABI" + nonprivileged instructions = *extended machine*, what the rest of the OS ran on
- once extended machine "covered over" bare metal, programs targeting bare metal couldn't run

Source: Robert P. Goldberg, "Survey of Virtual Machine Research," IEEE Computer Magazine, 7(6), pp. 34-45, June 1974.

Disco: Running Commodity Operating Systems on Scalable Multiprocessors, Bugnion et al. (1997)

- Edouard Bugnion: Stanford MS (PhD dropout), VMware '98-'05, founder Nuova '05, bought by Cisco '08, VP/CTO of BU
- Scott Devine: Cornell BS (go Big Red!), Stanford MS (PhD dropout), still with VMware
- Mendel Rosenblum: Professor at Stanford
- VMware: $36.1B market cap, 146.2 P/E
- Source: LinkedIn, VMware website

Key Ideas/Takeaways

- parallel HW here now, OSes not yet (why? hard)
- VMs good solution whenever SW trails HW
- Disco is a *VM monitor*: dynamically reallocate resources (CPU cycles, memory) between VMs
- key concern: data locality (remove NUMA-ness)
- require minor/no modifications to guest OSes
- prefer cross-hatching to shading

Design Structure

- layer between OSes, and the multiple processors of the multi-processor machine
- global policy specifies how to allocate all resources
- virtualize: CPU, memory, I/O (upcoming slides)
- benefit: can support specialized OSes (e.g. with FS cache policy designed for RDBMSes)

CPU Virtualization

- direct execution, intercept: TLB updates, DMA
- Disco can move VCPUs around physical CPUs
- on VCPU switch, save state of switched-out VCPU:

```
typedef struct {
      Register_t data[];
      Register_t PC;
      Register_t ...
} VirtualCPU_t;
```

Memory Virtualization

- terminology
  - virtual: what apps think they have
  - physical: what OS thinks it has
  - machine: what is really there (on 1+ NUMA nodes)
- intercept virtual->physical map from OS, replace with virtual->machine TLB entry
- flush TLB on VCPU switch, expensive; 2nd-level STLB (inside VM?) ameliorates cost

Memory Management

- replicate read-only pages (e.g. text)
- migrate pages to where they're accessed more
  - how decisions informed: hardware provides cache miss counter
- VM-aware apps can share memory (e.g. FS buffer cache)

I/O Virtualization

- if page requested in machine memory, map DMA block to the page
- copy-on-write, writes private to VM
- log modified sectors
- inter-VM communication via messages; message pages mapped to sender+receiver

What is SimOS? (also used by Gamsa team)

- simulator, models MIPS-family multiprocessors
- simulates HW components: CPU, caches, memory buses, HDs, ethernet, consoles, ...
- dynamic control of speed-detail tradeoff
- simulations fully deterministic
- has access to symbol table in simulated machine
- Tcl annotations control what is logged (cf. DTrace)
- source: "Using the SimOS machine simulator to study complex computer systems", ACM Transactions on Modeling and Computer Simulation (TOMACS) Vol 7, Iss 1 (Jan 1997)

Performance

- ran on: SimOS emulation of FLASH (ccNUMA)
- trap emulation is expensive; most VM overhead is here
- virtualization overhead, slowdown ≤ 16%
- faster to run *N* VMs with IRIX in uniprocessor mode, than IRIX on bare metal in *N*-processor mode
- memory locality (due to management) gives 33% speedup, for representative workload

Where This Paper Excels

- does not assume prior knowledge of domain
- graphs help visualize software organization, flow of control
- pragmatic outlook

VM vs Exokernel vs SPIN

- (obvious) VMs don't require OS rewrite, viable
- who controls the resources?
- where is the policy? (*beside* the mechanism...)
- how do we specialize an OS module? (e.g. FS buffer cache)
- how do we get fault-tolerance? process isolation?
- is an STLB needed? if so, why?

Tornado: maximizing locality and concurrency in a shared memory multiprocessor operating system, Gamsa et al. (1999)

- Ben Gamsa: U Toronto PhD '99, programmer
- Orran Krieger: U Toronto PhD '94, IBM researcher '96-'07, programmer at VMware
- Jonathan Appavoo: U Toronto PhD, IBM researcher, Asst Professor at U Boston
- Michael Stumm: Professor at U Toronto
- Tornado: licensed to IBM in '98, open-sourced by IBM, no activity after '02-'04
- Source: LinkedIn

Key Ideas/Takeaways

- every virtual, physical resource is an object
- key concerns: data locality + independence
- want to minimize *false sharing* in (large) cache lines

Design Structure

- localize (their term: "object-oriented") a process's PCB to processor running said process
- multiple implementations of OS objects (our term: "stub vs full implementation")
- desired future elaboration: swap implementations at runtime
- aim: minimize global state

Innovation: Clustered Objects

- clustered OS object composed of 1+ component objects ("representatives", or "reps")
- component objects reachable via single object reference, which redirects to the right rep
- reps kept consistent via shared memory, or PPCs
- complexity (actual location, consistency protocol) hidden within clustered object
- each rep can be locked independently
- created on 1st access, with object's miss handler

Innovation: Protected Procedure Call

- in their words: "call from a client object to a server object acts like a clustered object call *[undefined]* that crosses from the protection domain *[undefined]* of the client to that of the server, and then back on completion"
- cross-process, cross-processor
- implementation claimed to improve locality and concurrency

Innovation: Semi-Automatic GC

- unknown what makes it "semi-automatic"
- *temporary* references are thread-private
- *persistent* references are shared

1. remove persistent references
2. remove temporary references (wait for sys calls which could have accessed this reference, to complete)
3. remove clustered object itself

Performance (top of pages[n-3])

- ran on: [cc]NUMAchine, and SimOS emulating...??
- thread creation/destruction, page fault handling, fstat: no slowdown with either *1..N* threads in 1 process ("mt"), or *1..N* processes of 1 thread each ("mp")
- commercial OSes of the day degrade logarithmically, in the "mt" case; also no slowdown in the "mp" case

Reasons for Skepticism

- a full-featured OS would have many more objects (in kind, and number). Would clustered objects scale? (Concern about overhead)
- if commercial OSes are just as good with N processes, why not simply write multi-process apps for those?
- no macrobenchmarks (entire apps); unsure that they really put Tornado "through its paces"

Where This Paper Falls Short

- nonstandard terminology (e.g. "hardware address translation [HAT]" == TLB)
- not copy-edited completely (e.g. "Platforms on which micro-benchmarks ***where*** run", "system ***insures*** that all temporary references have been eliminated")
- incomplete definitions (e.g. PPC, above); perhaps complete definitions given in group's prior publications?

Comparison of the Papers

	Bugnion (Disco)	Gamsa (Tornado)
style problems	***none***	***not enough high-level***
clarity of exposition	***good***	***below average***
comprehensive performance analysis	***yes***	***not really***
innovation	***moderate***	***pretty high***
prototype: engineering challenge	***interoperability with mature products***	***algorithm, design***

Towards Transparent and Efficient Software Distributed Shared Memory, Scales and Gharachorloo. 16th SOSP, 1997.

- software DSM (distributed shared memory)
  - rewrite LOADs/STOREs to access remote nodes
  - let uniprocessor-targeted binary run on a cluster
- problem: support complete ISA, incl. atomic ops; emulate wonted memory consistency model
- problem: extend OS services across many nodes
- cache coherence, via directory-based invalidation protocol (what ccNUMAs do in HW)
- code modification, doable at link-/load-time

Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors, Verghese et al. 8th ASPLOS, Oct 1998.

- SMP servers have become the "new mainframes"
- don't let users hog: CPU, memory, I/O bandwidth
- software performance unit (SPU): mini-machine
- guarantee minimum level of resources
- policy specifies whether to "loan" idle cycles
- 1+ CPUs given to an SPU; fractions via timeslicing
- implemented by: augmenting [hacking up] kernel

Discussion