Saving Power with OS and VM Scheduling

Low-Power Computing
Carnegie Mellon University
David Andersen

OS Tasks

- OSes handle many periodic tasks
  - Applications blocked on select(), sleep(), usleep(), ...
  - Networking: Timeouts (TCP, arp, etc);
  - Network polling
    - At high packet rates, interrupts are expensive - poll device and get all packets that arrived in last few ms

Interrupts

- Are (kind of) expensive
  - They wake the CPU
  - They cause a context switch
    - Which causes some cache and TLB misses and takes time to save/restore register state, etc.

Packet handling

- Packet arrives in card
- Card interrupts CPU
- CPU pulls packet from card inspects packet
- CPU sends packet to (other) card
- card interrupts CPU saying "xmit done!"

Example: Routers

- Building a router with a conventional machine (using, e.g., Click or just BSD/Linux)
- Gigabit ethernet can deliver ~1M small (125 byte) packets per second. If interrupt handler takes a few hundred cycles...
  - FreeBSD 3.3, 500Mhz PIII Xeon: Interrupt overhead 4.36 µs.

Interrupt overhead

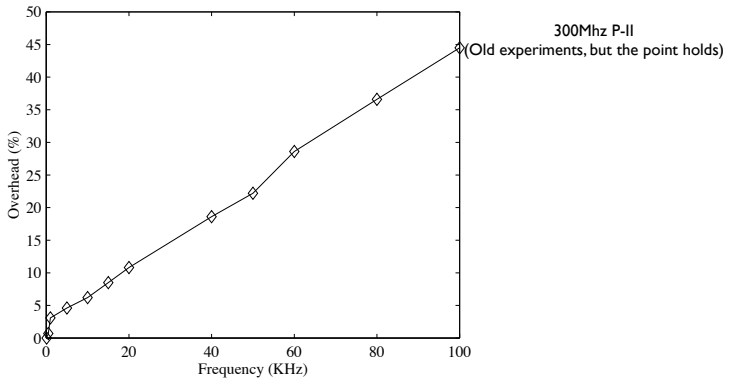

Figure 3. Base interrupt overhead

Source: Aron & Druschel

Conventional solution

- Set HZ=1000
- Deal with it.
- May want higher rates, though: 1000Hz -> sometimes adds 1ms of delay to packets. Matters, e.g., for traffic shaping.
  - Ex: Emulab Dummynet nodes use HZ=10,000 if you've asked for very fine-grained delay shaping

Two alternatives

- SoftTimers approach
  - Check timer Q when system is interrupted for other reasons - every interrupt handler, system call entry, etc. Even page faults, TLB misses on software TLB architectures.
  - Cache/TLB/etc. already disturbed at some "trigger" states; others (SYSENTER) don't mess things up as much
  - Read clock (CPU register, etc.)
  - Compare with head of timer queue
  - Can be done quick-like-bunny.

When SoftTimers?

- Very good for busy, I/O and system-call intensive servers and workloads
  - Lots of context switches, etc., already going on, so just make use of what's already happening
- Not so useful on an idle system - might wait forever!

Tickless Kernels

- Depends on old-but-new hardware: Programmable timers. RTC, HPET, etc.
- Instead of
  - Every 1ms, check queue
- Check queue, find next expiry
  - Set timer to fire in *expiry* time units
- Is this good or bad? Depends on cost of scheduling a timer interrupt.

Challenges

- These are scheduling challenges: How do we meet {deadlines, performance targets, etc.} while being efficient
  - Timer expiration
  - Work units inside VMs
- In all of them:
  - How do we *know* what deadlines are?
  - How do we schedule to meet them?

Themes

- Make scheduling explicit (we've seen this before) instead of poking people and saying "do you want to be awake yet? huh?"
- Expose enough options for people to tell you what they really really want, not just what the hardware can provide - e.g., 800Mhz P-state; helps when you're combining multiple VMs
- When you're already awake, do as much other work as possible

VirtPower

- Consolidation + Hardware Scaling
- Run (whatever) power management inside VM
  - Use this as a signal to the management system for how much the VM needs