Power Management in the Linux Kernel

Tate Hornbeck, Peter Hokanson

7 April 2011
Intel Open Source Technology Center

Venkatesh Pallipadi
- Senior Staff Software Engineer
- 2001 - Joined Intel
- Processor and platform power management
- 2010 - Moved to Google

Suresh Siddha
- Senior Staff Software Engineer
- 2001 - Joined Intel
- Multicore, process scheduling, system scalability
Vaidyanathan Srinivasan
- Energy optimizations for Linux servers
- Contributed to Linux power aware scheduler

Gautham R Shenoy
- Worked on cpufreq, idle system power management, idle load balancer
Ottawa Linux Symposium (OLS)

- Source of most of our papers
- One of three major grassroots Linux and Open Source conferences in the world
- Held since 1999
Intel's lesswatts.org

- Intel-sponsored project to improve Linux power consumption

- Major projects:
  - PowerTOP
  - Tickless idle
  - Linux ACPI
  - Linux Battery Life Toolkit (BLTK)
  - Power management on Intel graphics chipsets
Why optimize Linux for energy?

- Linux is being used in a wide variety of energy-critical applications:
  - Android phones and tablets
  - Datacenters:
    - Utility Costs
    - Thermal management
  - Embedded Linux
  - Laptops
Overview of Technologies

Non-Idle power management:
- CPU Frequency Scaling
- Process Scheduling on multicore systems

Idle-time power management:
- CPU Idle time allocation
- Removing timer interrupts
- Interrupt migration

Accurate energy-based measurements
- PowerTOP
- Battery Life Toolkit
CPU Frequency Scaling

- Kernel subsystem cpufreq
  - Common interface to CPU frequency stepping
  - Supports multiple platforms and methods (ACPI, BIOS)
  - Standard frequency governors

- CPU Governors:
  - performance - highest frequency
  - powersave - lowest frequency
  - userspace - proxy: exports data through sysfs interface for user-mode control
  - ondemand
  - conservative

The Ondemand Governor

- Modern CPUs can frequency-step in about 10us
- Userspace tools lack sufficient response time

For each CPU:
Every X milliseconds:
Get utilization since last check
if( utilization > UP_THRESHOLD)
    increase frequency to MAX
Every Y milliseconds:Get utilization since last check
if( utilization < DOWN_THRESHOLD)
    decrease frequency 20%

- Configurable thresholds and sample rates
- Conservative differs only in frequency rise time

Energy Aware Scheduling on multicore systems

Traditional Schedulers:
- Spread tasks across logical CPUs
- Unaware of underlying multicore and NUMA topology

Energy Aware Scheduling Goals:
- Consolidate all activity on fewest number of CPUs
- Remaining CPUs can enter idle states
- Understand the underlying topology
- Trade-off between throughput and energy efficiency

Two sockets with 4 cores each

Frequency and Voltage can only be controlled at socket level

Figure 1: Two socket quad-core system

Bad Througput
Good Energy Usage

Figure 2: Good task distribution from power perspective

Good Througput
Bad Energy Usage

Figure 3: Bad task distribution from power perspective
CFS Scheduler by Igno Molnar

- Time ordered red-black tree
- Nanosecond granularity accounting
- Each task has *virtual runtime*
  - represents amount of time the task has executed
  - the smaller a task's *virtual runtime*, the more the task needs to be scheduled
  - left node is in most need of CPU time
- choosing task is O(1)
- reinserting takes O(logN)

sched_mc_power_savings
Bias to keep workload on single package
Idle-State Power: cpuidle

- Idle States
  - Tradeoff between idling power and amount of state a processor saves and the enter/exit latency

- Kernel subsystem cpuidle
  - Generic idle management framework
  - Clean interface for processor hardware to make use of different idle levels
  - Create abstraction between idle-governors and idle-drivers
  - Make informed decision about which idle state to enter
    - kernel has most information about current running applications and idle state characteristics

V. Pallipadi, A. Belay: "cpuidle - Do nothing, efficiently...", The Linux Symposium 2007
**cpuidle Architecture**

1. Just before going idle, governor selects best idle state
2. cpuidle invokes the entry point for that particular state in the cpuidle driver

---

Contains information about each individual idle state

```
struct cpuidle_state {
    char name[CPUIDLE_NAME_LEN];
    void *driver_data;

    unsigned int flags;
    unsigned int exit_latency; /* in US */
    unsigned int power_usage; /* in mW */
    unsigned int target_residency; /* in US */

    unsigned int usage;
    unsigned int time; /* in US */

    int (*enter)(struct cpuidle_device *dev,
                 struct cpuidle_state *state);

    struct kobject kobj;
};
```
**cpuidle Governors**

**Ladder Governor**
- Simple, step-wise approach
- Works well with periodic tick based kernels

**Menu Governor**
- Analyzes different parameters and current context
  - dyntick expected sleep time
  - previous C-state residency
  - per cpu load average
  - number of processes waiting for I/O on current cpu
- Jumps to lowest possible idle state that does not significantly affect performance
- Aims at getting maximum possible power advantage with little impact on performance
Tickless Idle: The Timer Interrupt

- Traditional systems use a periodic interrupt 'tick'
  - update the system clock
  - provide a timer interface for scheduling
  - previously at 100Hz to 1000Hz
  - tick requires wakeup from idle state
- Tickless kernel
  - requires cross-platform clock source interface: hrtimer
  - skip over ticks before next required event
  - currently only affects idle state

Tickless Idle: Cross-platform hrtimers

- Moving to a tickless kernel requires high resolution timing
- Linux is cross-platform, so changes must be portable
- Original architecture

Tickless Idle: Removing the Timer

- Measurements show more idle time:

<table>
<thead>
<tr>
<th></th>
<th># interrupts</th>
<th>#events</th>
<th>Avg CPU idle residency (uS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With ticks</td>
<td>2002</td>
<td>59.59</td>
<td>651</td>
</tr>
<tr>
<td>Tickless</td>
<td>118</td>
<td>60.60</td>
<td>10161</td>
</tr>
</tbody>
</table>

- Some platform-dependent code required
- Small delay in load-balancing incurred initially
  - Due to delay in running load balancer
  - Fixed by designating one idle CPU as load balancer
- Does not prevent inefficient user-space polling

Tickless Idle APIs

The tickless patch adds two flags to timers:

- **__round_jiffies()**
  - Rounds jiffy timeout values to the nearest second
  - Prevents multiple wakeups from staggered interrupts
  - Used where precise timeout not required

- **deferabletimers**
  - Timer interrupts that are unnecessary when idle
  - Interrupt deferred until processor is active
  - Used for ondemand cpufreq governor
  - __next_timer_interrupt() skips over deferrable timers
  - Flag added to last bit of 2-byte aligned address

Interrupt Routing/Balancing

- HW (nic, hdd, etc.) use interrupts to inform OS of events
- Default interrupt routing either:
  - Routes all interrupts to logical CPU0 (1st core of 1st socket)
    - Could spend a disproportional amount of time processing interrupts under heavy loads
    - Other CPU0 tasks could starve/perform poorly
  - Broadcasts to all CPUs
    - Non-ideal if some CPUs are idle
    - Causes unneeded wakeups

**irqbalance daemon**

- Userspace solution by Intel
- Included in most distros
- Understands processor topology and cache domains
- Classifies irqs based on performance sensitivity
  - Network most important
- Power Save Mode
  - All irqs assigned to first logical CPU
  - Consolidates irqs and distributes at lower rate
  - When system activity increases:
    - Increases distribution rate
    - Begins to distribute among different CPUs
Accurate Energy Measurement

Motivation

- Not all problems are solved in kernel space
- Userspace applications need to be written to take advantage of these optimized APIs
- Need a way to pinpoint misbehaving applications
- Measuring should not change system execution significantly

Tools

- PowerTOP
- Battery Life Tool Kit
PowerTOP

- PowerTOP can measure processor states and wakeups
- Finds causes of CPU wakeups by thread
  - Kernel
  - User-level
- Help Linux developers test applications
- Provide system tuning suggestions to achieve low power consumption
- Helped expose inefficiencies in Firefox and other applications
PowerTOP

PowerTOP version 1.8  (c) 2007 Intel Corporation

Cn          Avg residency  P-states (frequencies)
C0 (cpu running)  ( 3.3%)    1.61 GHz  0.0%
C1          0.0ms ( 0.0%)    1.60 GHz  0.0%
C2          5.5ms (10.6%)    1200 Mhz  0.0%
C3          8.4ms (86.1%)    800 Mhz   100.0%

Wakeups-from-idle per second : 121.8   interval: 15.0s

Power usage (ACPI estimate): 14.1W (1.1 hours)

Top causes for wakeups:

16.4% (18.9)  firefox-bin : futex wait (hrtimer wakeup)
15.6% (17.9)  <interrupt> : uhci_hcd:usb4, ohci1394, HDA Intel, iwl4965
13.0% (14.9)  xchat : schedule_timeout (process_timeout)
  9.6% (11.0)  S20powernowd : queue_delayed_work_on (delayed_work_timer_fn)
  7.6% ( 8.7)  <interrupt> : extra timer interrupt
  7.2% ( 8.3)  Xorg : do_setitimer (it_real_fn)
  4.8% ( 5.5)  gnome-terminal : schedule_timeout (process_timeout)
  3.8% ( 4.3)  psi : schedule_timeout (process_timeout)
  2.9% ( 3.3)  trackerd : schedule_timeout (process_timeout)
  2.7% ( 3.1)  <interrupt> : acpi
  2.6% ( 2.9)  <interrupt> : PS/2 keyboard/mouse/touchpad
  1.7% ( 1.9)  <kernel core> : queue_delayed_work_on (delayed_work_timer_fn)
  1.5% ( 1.7)  Xorg : schedule_timeout (process_timeout)
  1.5% ( 1.2)  wpa_supplicant : schedule_timeout (process_timeout)
  0.9% ( 1.0)  trackerd : do_nanosleep (hrtimer_wakeup)
  0.9% ( 1.0)  nm-applet : schedule_timeout (process_timeout)
  0.6% ( 0.7)  <kernel core> : neigh_table_init_no_netlink (neigh_periodic_timer)
  0.5% ( 0.6)  <kernel core> : dst_run_gc (dst_run_gc)
  0.5% ( 0.5)  <kernel module> : neigh_table_init_no_netlink (neigh_periodic_timer)
  0.5% ( 0.5)  NetworkManager : schedule_timeout (process_timeout)

Suggestion: increase the VM dirty writeback time from 5.00 to 15 seconds with:
   echo 1500 > /proc/sys/vm/dirty_writeback_centisecs
This wakes the disk up less frequently for background VM activity

Q - Quit   R - Refresh   W - Increase Writeback time
Battery Life Toolkit (BLTK)

- Power consumption difficult to measure reliably and easily
  - External power supplies can effect power modes
  - Modern batteries build in sufficient telemetry
  - Runs battery to exhaustion
- Package provides software infrastructure for launching workloads for power performance measurements
- No additional hardware required
- Repeatably measures battery life with precise workloads
  - Web browsing, DVD playback, office application
  - Full scripting ability to simulate input

Outstanding Issues

- Task consolidation/load balancing relies on accurate CPU load calculation
  - CPU load is sampled periodically
  - Short lived tasks (e.g. daemons) could not be factored into calculation
  - Tasks that are factored into load may not be CPU intensive, instead I/O bound

- Consider example workload:
  
  ```
  make -j 2
  compiling kernel w/ 2 threads
  2 socket dual core machine
  CPU0/1 and CPU2/3 are core siblings
  sched_mc_power_savings = 1
  ```
Figure 6: ebizzy with sched_mc_power_savings=0

Figure 8: make -j2 with sched_mc_power_savings=0

Figure 7: ebizzy with sched_mc_power_savings=1

Figure 9: make -j2 with sched_mc_power_savings=1
Workload
  - Large number of short lived jobs
  - High rate of process creation/destruction
  - Mix of IO operations (not CPU bound)
  - No imbalance detected so continue to wakeup idle CPUs

2 core case
  - Utilization high enough to raise ondemand frequency

4 core case
  - Both threads jumped between all 4 processors
    - Scheduler could never get accurate CPU load
  - CPU utilization not high enough to trigger ondemand

Table 4: ‘make -j2’ with varying number of cpus
How to keep Userspace Quiet

- Do not poll periodically for status change
  - Use some sort of event notification
  - hal (hardware abstraction layer) daemon used to poll for media changes
    - New SATA supports Asynchronous Notification
- Use intelligent mechanisms to minimize periodic timers
  - Group them and expire at same instance
  - Use provided API to help
    - g_timeout_add_seconds() consolidates glib timers to the nearest second
- Use tools like PowerTOP and BLTK
Conclusions

- Linux power management has become a serious focus in the past 10 years
- Several techniques have seriously improved efficiency:
  - Frequency stepping with the ondemand governor
  - Tickless kernels to improve idle time
  - Energy-aware multicore scheduler improvements
  - Improvements to the userspace software stack
- Work is in progress on several other improvements:
  - Virtualization
  - Power-aware scheduling of short-lived tasks
References