Getting maximum mileage out of tickless

Len Brown

For

Suresh Siddha, Venkatesh Pallipadi, Arjan van de Ven

Open Source Technology Center

Kernel with periodic ticks

- Periodic timer tick in the pre tickless kernels
  - HZ determines the frequency
  - Ticks irrespective of the processor state (idle Vs busy)
- Periodic tick used for
  - Per CPU timer mgmt
  - Task time slice mgmt
  - Periodic SMP load balancing
- Periodic wakeup's are bad, especially during idle
  - Prevents processor from going into deep sleep states
  - Affects battery life
  - Limits Linux guests in a virtualized environment

Tickless idle kernel

- Removes the periodic tick during idle
- Restructuring the timekeeping code laid down the foundation
  - Generic clockevents and clocksources
  - High resolution timers
- Per CPU periodic timer tick
  - Managed by per CPU clockevent device
    - LAPIC/HPET/PIT/RTC
- Tickless kernel
  - No periodic tick during idle
  - Programs clockevent device based on next future event

Tickless base data

	# interrupts	#events	Avg CPU idle residency (uS)
With ticks	2002	59.59	651
Tickless	118	60.60	10161

System activity during idle with and without periodic ticks, with HZ=1000*

Is this the best we can do? Can we further reduce number of events and interrupts?

* Intel® Core™ 2 Duo based system (2 CPU cores)

Keeping kernel quiet

- Avoiding staggered timers
  - Most of the timers can be grouped
  - Kernel API rounding the timeout value to nearest second
    - round_jiffies()
- Deferrable kernel timers
  - Defer the expiry of non critical timers during idle
  - Normal timer expiry during busy
  - Ondemand governor uses this
  - init_timer_deferrable() API instead of init_timer()
  - Can we extend this concept for user timers as well?

Tickless Data with ondemand changes

	# interrupts	#events	Avg CPU idle residency (uS)
Ondemand	118	60.60	10161
Ondemand + deferrable timer	89	17.17	20312

System activity during idle with and without deferrable timer usage in ondemand

Platform timer event sources

- Tickless depends on per CPU timer event source
- On x86, Local APIC timer is mostly dependable
  - Stops working in low power processor states
    - ACPI C2, C3 states
    - Mostly laptops have these low power capabilities
- Broadcast timer as a workaround for LAPIC stoppage issue
  - Timer shared across pool of processors
  - Wake's up the processors in the pool through IPI
  - Uses platform timers like PIT/HPET

PIT Vs HPET

- PIT/8254
  - Frequency of 1193182Hz
  - Maximum timeout of 27462uS
- HPET
  - Frequency of timer varies from platform to platform
  - Typically max timeout is much greater than PIT
    - System under test has 14318179Hz, max timeout of > 3 seconds
  - Reduces number of interrupts to manage timers
- Not all platforms enable HPET
- And not all platforms advertise presence of HPET
  - Resulting in using PIT as broadcast timer for most of laptops
- Kernel patches for force detection and enabling HPET
  - Using pci quirks

Tickless Data with force detection of HPET

	# interrupts	#events	Avg CPU idle residency (uS)
PIT	89	17.17	20312
HPET	32	15.15	56451

System activity during idle with PIT Vs HPET

HPET modes

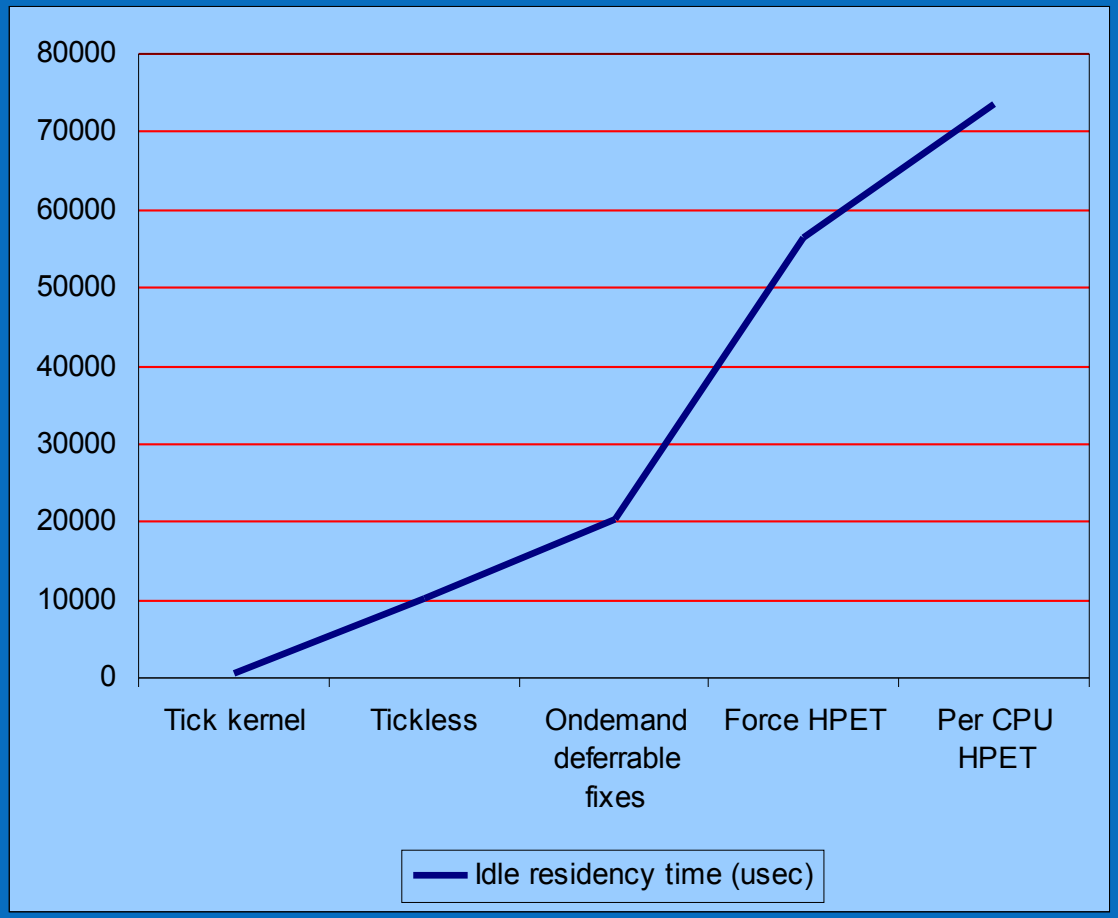
- Legacy replacement mode
  - Channel's 0 and 1 generating IRQ's 0 and 8
  - Appears like legacy PIT and RTC
  - Other channels routed independently
- Standard interrupt delivery mode
  - Different channels routed independently
  - Can be used a per CPU timer
    - If # channels >= # of logical processors
    - Two channels required on a dual core laptop
- HPET as per CPU timer
  - Nice workaround for LAPIC stoppage issue
  - Eliminates the need for broadcasting timer interrupts
  - Patches to be posted to lkml

Tickless Data with per CPU HPET channels

	# interrupts	#events	Avg CPU idle residency (uS)
global HPET	32	15.15	56451
percpu HPET	22	15.15	73514

System activity during idle with global Vs percpu HPET channels

Idle residency time with kernel enhancements we have talked so far!



Keeping user space quiet

- User space plays a prominent role
  - Tickless kernel is just the foundation stone!
  - User space need to quiet down during idle
    - Enabling long and deep idle processor/platform sleep
- Dave Jones OLS 2006 talk
  - Why user space sucks?
- Powertop (**http://www.linuxpowertop.org/**)
  - Showed that user space still stucks!
  - Tool identifies biggest offenders in the kernel and user space
- Group timers to minimize scattered wakeups
  - Glib timer API g_timeout_add_seconds()
- Polling is evil
  - Use event notification where ever possible
  - For ex: 'hal' daemon using SATA AN feature
    - Avoids polling for identifying media change event

Idle process load balancing

- Process load balancing
  - For fairness and improving system throughput
  - Periodic timer tick kicks SMP process load balancing
    - Pull mechanism on each processor
    - Load balancing happens more often during idle
- No periodic tick during idle in tickless kernel
- Tickless enables processor to sleep for long durations
  - Results in less frequent idle load balancing
  - Potentially affecting system throughput
- Different proposals to fix the problem
  - Busy processor doing push instead of idle processor doing Pull
    - How often we need to push from a busy guy?
    - Where to push?
      - Difficult to identify the destination in domain topology
  - Retain idle pull mechanism with exponential back off
    - May not respond immediately to changes in load

Tickless Idle process load balancing fixes

- Our solution to the problem
  - Nominate one idle processor as the owner for idle load balancing
    - Load balances on behalf of all the idle processors in the system
    - Owner will have periodic timer ticks enabled
    - All the other idle processors will be in tickless mode
- No owner when the system is completely idle
  - No need for periodic load balancing when everyone is idle
- Intelligent owner selection can minimize power wastage
  - Like nominating an idle core/sibling in an already busy package
  - Not yet done in today's patches

Performance regression with base tickless

Garbage collector	Perf Regression
parallel	6.3%
gencon	7.7%

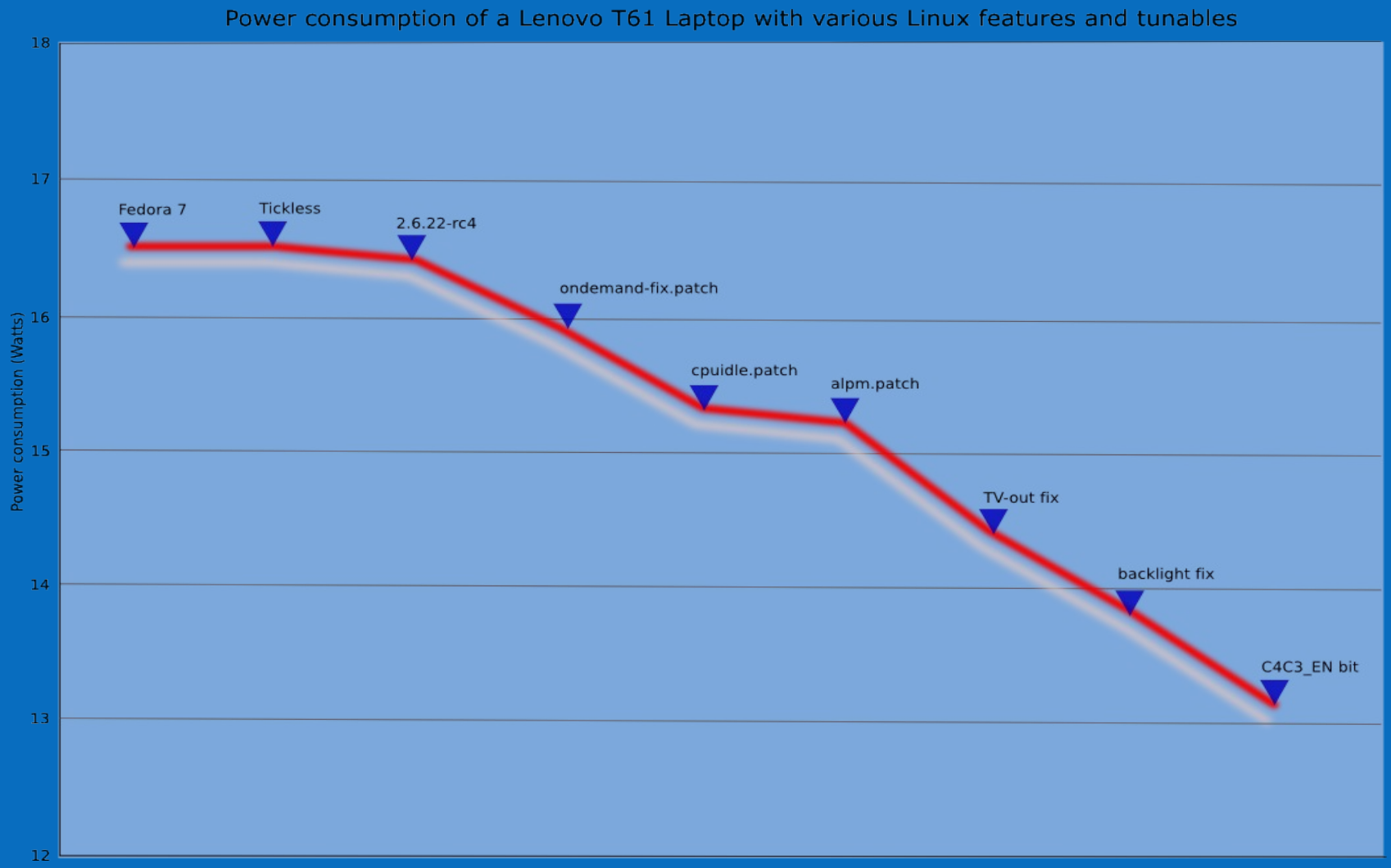
SPECjbb2000 performance regression with base tickless kernel. Idle load balancing fixes recovered this performance regression.*

*System under test is a 8 core system (dual package system with quad core processors). SPECjbb2000 benchmark run with 512MB heap and 8 warehouse configuration.

Conclusions

- Tickless kernel is the first important step
- Kernel features presented in this talk improved idle residency time by ~7 times
- Run ‘powertop’ on your laptops
  - Report top wakeup’s
  - Idle residency time can be increased to ~1 second and higher
- Responsibility lies with both system and user level SW
  - For taking maximum advantage of power savings capabilities in HW
- Next evolutionary step is the complete tickless kernel

Leave you with some cutting edge data*



* Source: http://www.linuxpowertop.org/results.php

Thank You!