Linaro connect
Bangkok 2016

Sched idle

Next irq event prediction

Presented by

Daniel Lezcano

Date

BKK16-203 March 8, 2016

Event

Linaro Connect BKK16

Introduction

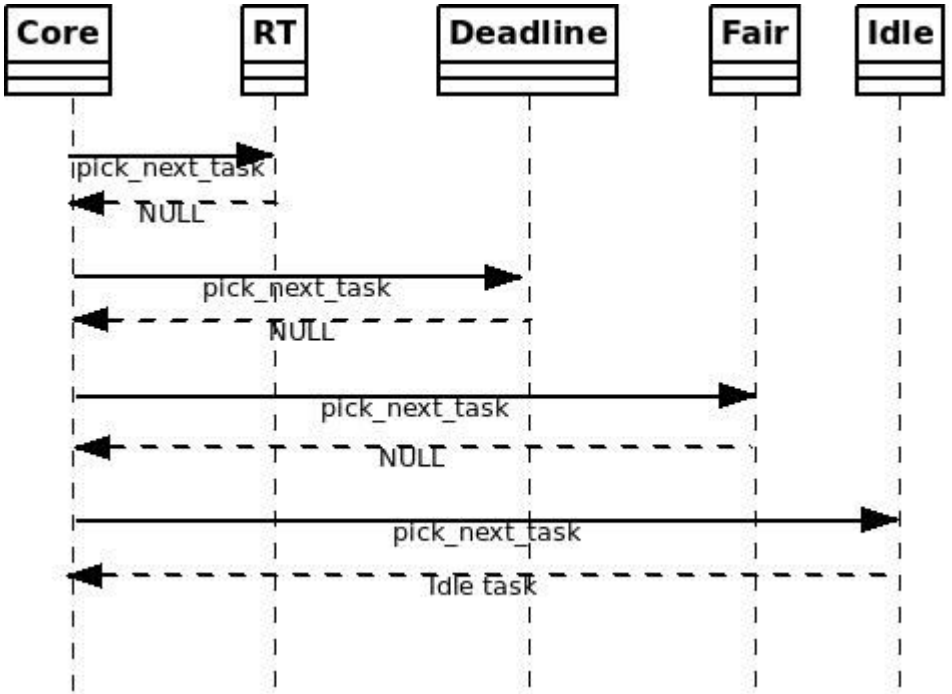
- Idle mechanism
- Limitations
- New approach

Introduction

- Idle does not work if the break even is not reached
  - Consumes more energy
  - Worst performance
- Target residency is the break even
  - https://wiki.linaro.org/WorkingGroups/PowerManagement/Doc/ComputingTargetResidency
- Idle duration prediction is the key

The idle process

- No task for each sched class
- Default task is the idle task



The idle task

- Two infinite loops
  - One to ensure idle routine is called next time
  - One to handle wakeup + idle
- Yield to another task if any
  - As soon as there is a task to be run, the idle task is switched
- Enters / exit the idle routine
  - Idle time measurements

Current approach

- When a CPU wakes up from an idle state
  - idle time measurement

- When a CPU goes idle
  - statistics on idle time
  - Reuse statistics to guess estimate the idle duration

Current approach

- What woke up the CPU ?
  - a timer ? an IO device ? an IPI ?
    - https://wiki.linaro.org/WorkingGroups/PowerManagement/Doc/WakeUpSources
- It is not possible to know. It results to:
  - statistics on timers but they are deterministic
  - statistics on IPI, thus on the scheduler behavior

Current approach

- This wake up sources soup leads to some complex re-adjustments in the statistics
  - empiric
  - platform specific (x86)
  - monolithic
- It is hard to tell if the prediction was correct

Current approach

- Prediction and governor are tied
  - CPUidle is a standalone subsystem
  - No interaction with the scheduler

- Scheduler can’t use the prediction value to anticipate anything
  - Hard to integrate the power subsystems

What do we want ?

- The scheduler to act:
  - as a governor
  - pro actively
  - with knowledge of future events

What do we need ?

- Split prediction and idle state selection
- Increase the prediction accuracy
- Let the scheduler to have access to the prediction

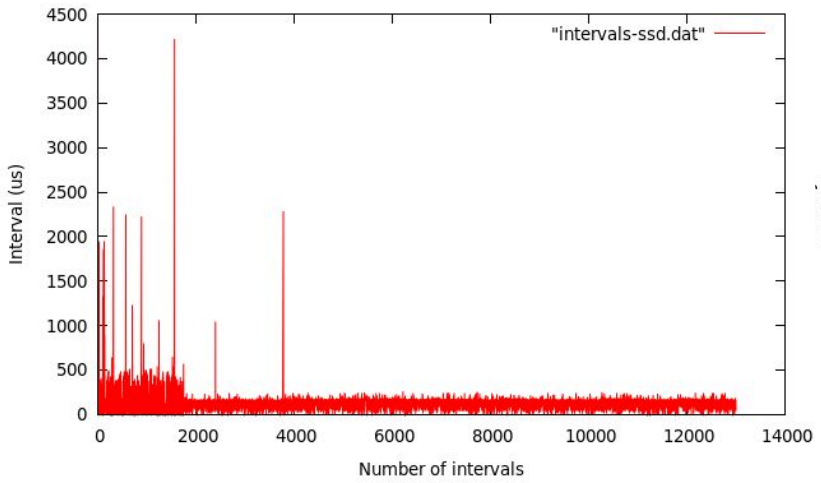
How do we do that ?

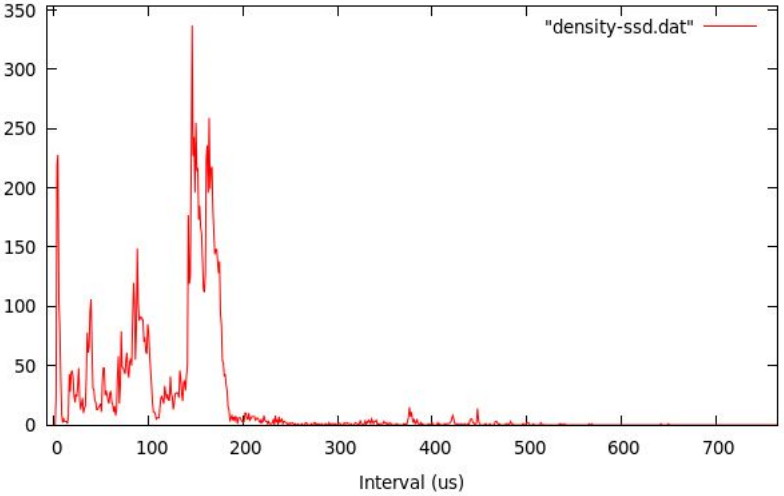
- Track the interrupts independently
  - measure their intervals
  - apply a simple mathematical formula for their behavior
  - guess estimate the next event per interrupt
- Timers and IPI are out of the equation
- Full view of what is happening on the system

Analysis

- A network trafic + console output
  - kernelshark trace-net.dat
- A SSD random read
  - kernelshark trace-ssd.dat

Analysis - SSD

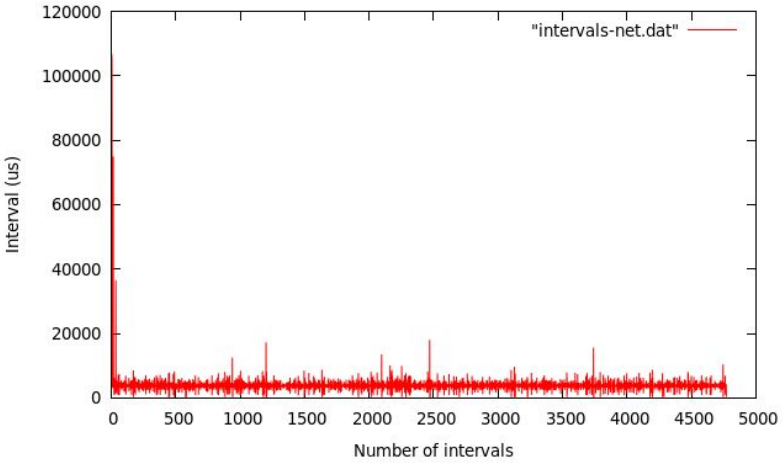


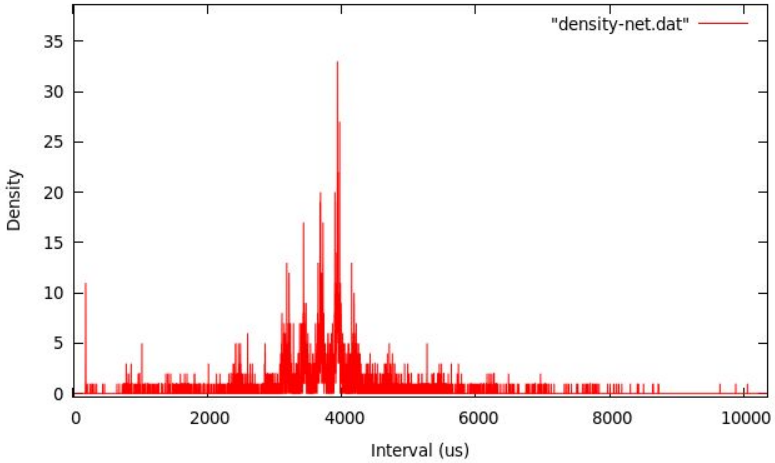


Analysis - SSD

- Beginning of the measurement => writing a big file
- Rest of the measurement => random read
- Write is hard to predict
- Read is stable

Analysis - Network



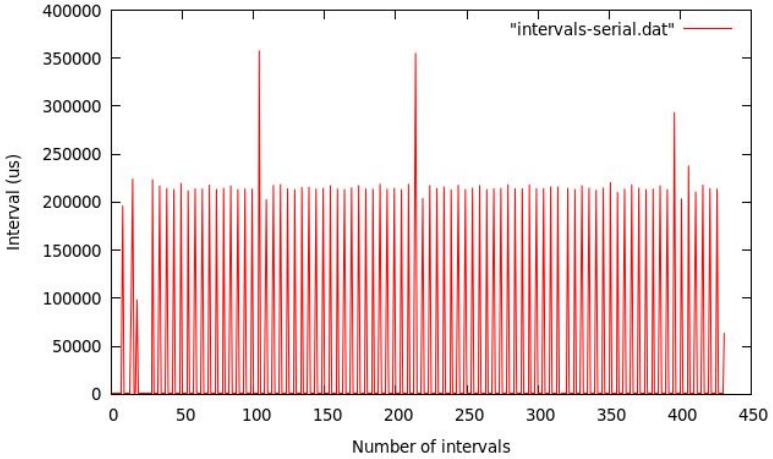


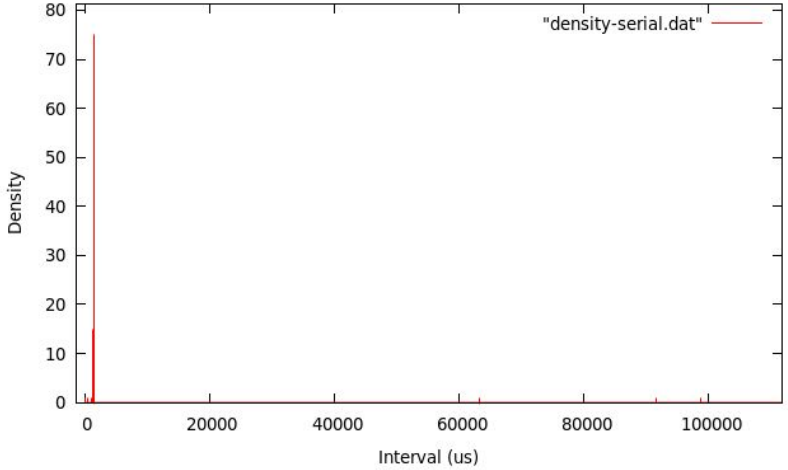
Analysis - Network

- Intervals are stable

- Repeating patterns
  - Four values
  - Looks like gaussian => normal law ?

Analysis - Serial console



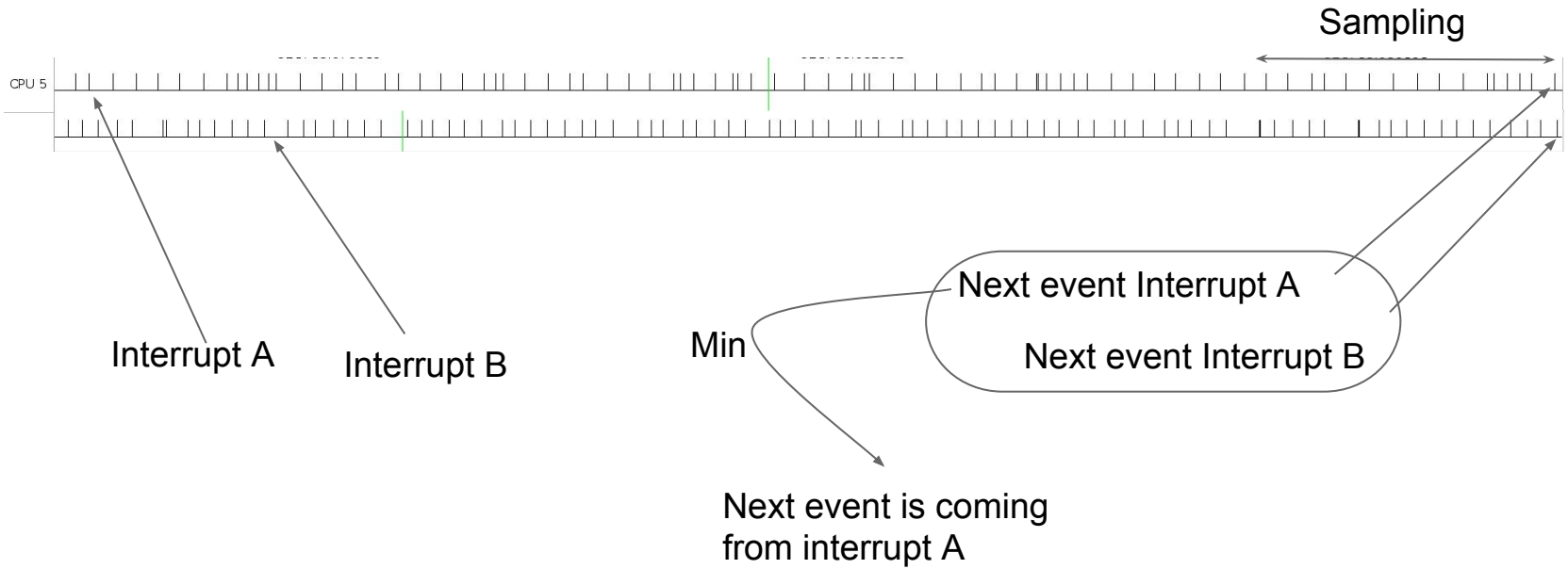


Analysis - serial

- Very stable interrupt

- Repeating pattern easily predictable
  - ~1ms x 5 + ~210ms

Analysis


Sampling
CPU 5
Interrupt A
Interrupt B
Min
Next event Interrupt A
Next event Interrupt B
Next event is coming
from interrupt A

Analysis - Conclusion

- With a correct prediction for each interrupt
  - increase of the accuracy for the next event

- Mixing all interrupts, we can choose the earliest one

Analysis - Conclusion

- Risk ?
  - Cumulative error on each irq prediction
  - Mathematical model vs performance in kernel
  - Different behavior for each irq
- Challenge ?
  - Solve the above
  - Find the right mathematical model
- Approach ?
  - Incremental

IRQ based next event status

- Submitted upstream as RFC
  - https://lwn.net/Articles/670505/

- Positive feedbacks and article in LWN
  - https://lwn.net/Articles/673641/

Scheduler + idle

- Peter Ziljstra :
  - “I think the scheduler simply wants to say: we expect to go idle for X ns, we want a guaranteed wakeup latency of Y ns -- go do your thing.”
  - https://lkml.org/lkml/2013/11/11/353“

Scheduler + idle

- Two new APIs:
  - s64 sched_idle_next_wakeup(void);
    - “[…] we expect to go idle for X ns […]”
  - int sched_idle(s64 duration, unsigned int latency);
    - “[…] go do your thing […]”
  - (Latency API is already there)

Scheduler + idle

- From the scheduler, the information ...
  - When a CPU will wake up ?
  - How long a CPU will be idle ?
- … are now available
- The scheduler has the information to take a decision regarding the energy saving
  - the scheduler will be able to act as a governor

Scheduler + idle

- Task rebalancing
  - scheduler knows if it is worth to migrate a task
    - eg. is the target CPU about to wake up ?

- Scheduler can force latencies

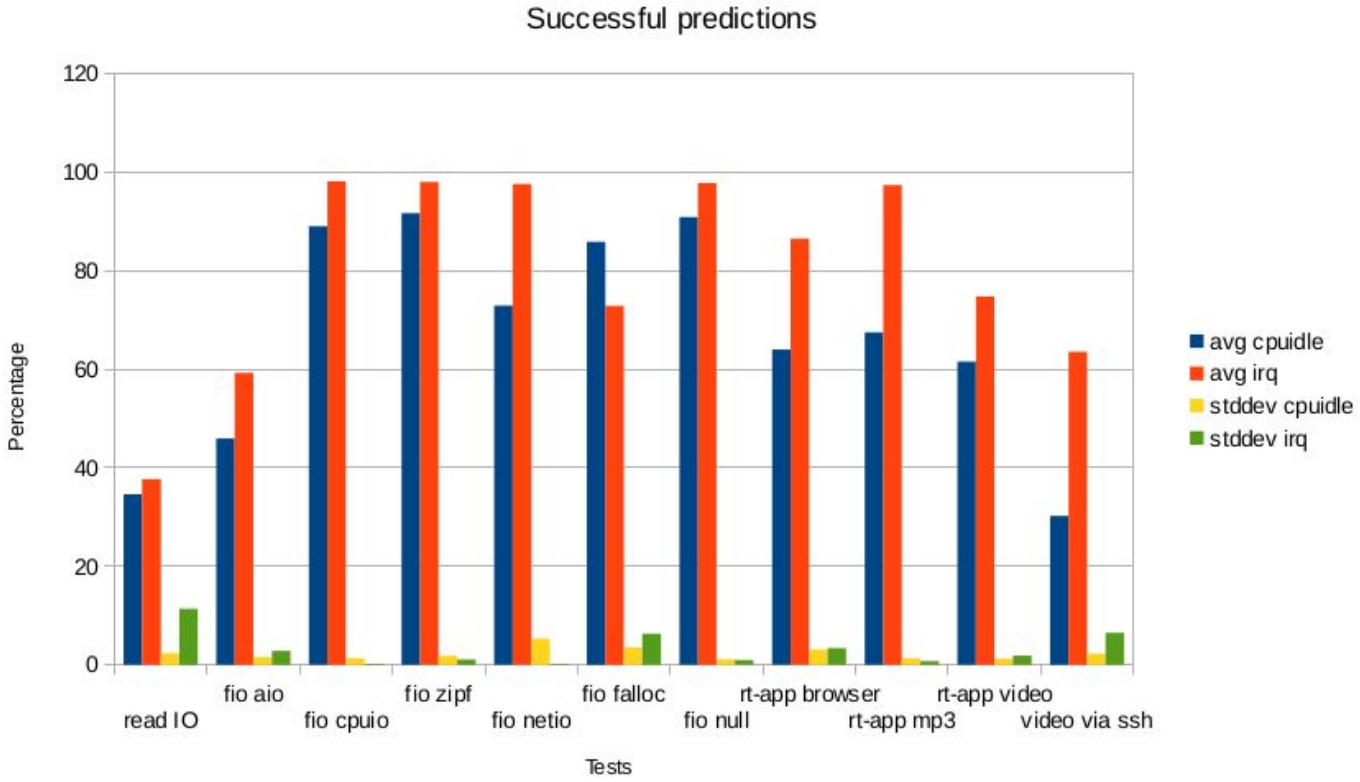
Scheduler + idle

- Topology and idle information
  - Compute target residency at different level:
    - Cpu
    - Cluster
    - “SoC”
  - Predicted idle time is the time intersection between components
    - Eg. idle(cluster0) = idle(cpu0) ∩ idle(cpu1)

Some results

- Simple approach but better results in some cases
- SMP is much better but UP worst

Some results



Next steps

- Instrumentation
- Improve the model
  - Bayesian network (TBS)
- Upstream the first bits
  - Approach is accepted
- Enlarge community (very important)
  - More testing and validation
- Scheduler tuning