Power management

IRQ next event prediction - Where are we?

Daniel Lezcano
Agenda

- Introduction
- Energy / Idle states / Break even
- Finding the sleep duration
- The sources of wake up
- The governor and the heuristics
- Energy consumption and governors
- Changing the approach
- Measuring the events
- Predict the next event
- A dedicated embedded governor
- Comparisons
- Conclusion
Consumption vs idle states
Consumption vs idle states
Consumption vs idle states
Consumption vs idle states

Energy

\[ E(\text{idle}_1) = E(\text{idle}_2) \]

\[ E(\text{idle}_1) < E(\text{idle}_2) \]

Break even

Beginning idle \quad \text{time(\text{idle}_1)} = \text{time(\text{idle}_2)}

Target residency for idle 2

Time

Idle 2
Idle 1
Running
Computing the target residency

- Formula to compute the minimum residency time

\[ \text{time} = \frac{W_{idle2} - W_{idle1}}{P_{idle1} - P_{idle2}} \]

- Demonstration available on the [PMWG wiki page](http://example.com)

- Alternatively, empiric approach presented at [HKG18](http://example.com)
Idle states characteristics

- Idle states must be described accurately

  - Target residencies
    - Usually very approximate values

  - Exit latencies per OPP
    - Only worst case is provided

  - Power at the idle state per OPP
    - These are not available
Choosing the idle state

- Take decision on which idle state to choose
- Based on past events
- Try to predict the future
- Algorithm must be simple
Sleep durations

- Origin of the wake up source
- Statistics on the sleep durations
Problematic

● As we read the sleep duration, the source of wake up can be anything
  ○ How do we sort out this?
  ○ We try to predict the scheduler behavior
  ○ We try to predict the interruption with the noise of the scheduling + timers

● That can work only if there are periodic wakes up
  ○ Specific workload, especially IOs
Experiments with governors

Let’s create dummy governors and compare them to the reference: the menu governor

- Random governor: Randomly choose an idle state
- Modulo governor: Always +1 on the selected state modulo number of states
- Deepest governor: Always choose the deepest idle state
- Shallowest governor: Always choose the shallowest idle state
Jankbench / image list vs governors

- Reference (menu): Count 100, Frame duration 100, Energy 100
- Random: Count 85.1, Frame duration 105.8, Energy 101
- Modulo: Count 76.9, Frame duration 99.9, Energy 98
- Deepest: Count 68.8, Frame duration 148.1, Energy 105
- Shallowest: Count 42.4, Frame duration 137.3, Energy 108
Jankbench / edit text vs governors

![Bar chart showing performance metrics for different governors: Count, Frame duration, Energy. The governors are Reference (menu), Random, Modulo, Deepest, Shallowest. The chart compares their performance across these metrics.](chart.png)
Exoplayer audio vs governors (no frame dropped)

- Reference (menu): 100
- Random: 98.1
- Modulo: 96
- Deepest: 115.1
- Shallowest: 96.4
Exoplayer video vs governors (no frame dropped)
Observations

- Going always for the deepest idle state kill performance and consume more energy
- “Randomly” choosing idle state gives same or better results than the menu governor
- Using the shallowest idle state saves up to 8% of energy with audio and video
- Using the shallowest idle state reduces the frame rendering duration up to 58% with an energy drop of 8%
- What is going on?
What is going on? (Jankbench test1)

menu

wfi
What is going on? (exoplayer ogg)

menu

wfi
What is going on?

● EAS scheduler behaves differently regarding the idle states:
  ○ Race to idle
  ○ Tasks are packed

● The menu governor is doing a lot of mispredictions
Wake up sources

CPU0

CPU1

GIC

MMC

Network

Graphic

Timer

IPI reschedule

drivers

timer_list

hrtimer

scheduler

Linaro connect 2019
Wake up sources

CPU0

CPU1

GIC

Timer

IPI reschedule

MMC

Network

Graphic

drivers

timer_list

hrtimer

scheduler

prediction

prediction
How behave devices?

- SSD
- Network
How behave devices?

- Graphics
- Console
How behaves the idle task rescheduling

![Graph showing idle task rescheduling behavior](image-url)
How behave the timers?

That’s a good question, the answer is “as expected”

We always know the next event for the timer
Observations

- Devices can have periodic interrupt
  - Periodicity in the intervals
  - Periodicity of a group of intervals

- Idle task rescheduling is almost random
  - Based on scheduled work
  - Tasks taking locks
  - Tasks blocked on IO

- Timers give an accurate information for the next wakeup

- Side note: On mobile, interrupts are usually pinned on CPU0
Hypothesis

- Why not predict for each wake up source?
  - Per interrupt
  - Per need_resched duration
  - Make scheduler idle wise
  - Timers are predictable
Wake up sources

CPU1

CPU0

GIC

Timer

IPI reschedule

Prediction

MMC

Network

Graphic

drivers

timer_list

hrtimer

scheduler
Predicting the interrupts from devices

- Store the interrupts <irq,timestamp> when they happen
- At idle time, look at the interrupt history and compute intervals
- Store the interrupt intervals in a log2 array
- Use a fast algorithm based on array suffix
- Use the exponential moving average for similar past events
At runtime
Store the interrupts and timestamp

```c
__handle_irq_event_percpu(desc)
⇒ record_irq_time(desc)
⇒ irq_timings_encode(irq, timestamp)
⇒ irq_timings_store()
```

Per cpu circular buffer

- irq
- timestamp
- 0...15
- 16...63
- U64
At idle time
Discretization of intervals

- High number of different values
- Time events: the higher the interval, the lower the precision
- Group the intervals per range
  - \([0, 2]\) \([2, 4]\) \([4, 8]\) \([8, 16]\) \([16, 32]\) \([2^{31}, \infty]\)
  - An array of 32 values
- Log2 is fast and has dedicated ASM function
Compute intervals on log2 basis

31 / 12345
31 / 12455
31 / 12650
67 / 12870
31 / 23380
32 / 23390
31 / 24502
67 / 25326

log2

100 us
195 us
1122 us

irq31

log2

6
7
10
Tracking signals with EMA

- Each intervals is separately tracked with exponential moving average

- Exponential moving average:
  - Stock value tracking
  - Very fast
  - Tweakable via alphas
Store in EMA array

- 31 / 12345
- 31 / 12455
- 31 / 12650
- 67 / 12870
- 31 / 23380
- 32 / 23390
- 31 / 24502
- 67 / 25326

- 100 us
- 195 us
- 1122 us

- log2
- log2
- log2

- 6
- 7
- 10

- ema
- irq31

- ema irq31

- index=6
Array suffix

- Data structure for full text indices search, data compression algorithm, bibliometrics, combinatorics on words, bioinformatics

- Build an array of suffix of the terms:
  - Eg. banana has the suffixes: banana, anana, nana, ana, na, a

- Per irq tables have suite of numbers between <1, 32> resulting from log2
Store in EMA array

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>12345</td>
</tr>
<tr>
<td>31</td>
<td>12455</td>
</tr>
<tr>
<td>31</td>
<td>12650</td>
</tr>
<tr>
<td>67</td>
<td>12870</td>
</tr>
<tr>
<td>31</td>
<td>23380</td>
</tr>
<tr>
<td>32</td>
<td>23390</td>
</tr>
<tr>
<td>31</td>
<td>24502</td>
</tr>
<tr>
<td>67</td>
<td>25326</td>
</tr>
</tbody>
</table>

100 us
195 us
1122 us

log2
log2
log2

History of the past events
An interrupt is predictable if there is a repetition
  - We need to find the period of this repetition

Experiment showed a max period of 5 for repeating patterns
  - We assume pattern repeating 3 times has a strong period
  - We take the last 3 x 5 = 15 events

Example with MMC:

<table>
<thead>
<tr>
<th>Interval</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1385</td>
<td>212240</td>
<td>1240</td>
<td>1386</td>
<td>1386</td>
<td>1386</td>
<td>214415</td>
<td>1236</td>
<td>1384</td>
<td>1386</td>
<td>1387</td>
<td>214276</td>
<td>1234</td>
<td>1384</td>
</tr>
<tr>
<td>log2</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Max period = 5

Last 3x5 = 15 events
Search with array suffix

- Other example with console

<table>
<thead>
<tr>
<th>Interval</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>110</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>7</th>
<th>4</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>log2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Search with array suffix

<table>
<thead>
<tr>
<th>Interval</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>110</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>7</th>
<th>4</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>log2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>p=5</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>p=4</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>p=3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>p=2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Search with array suffix

Next event index = last pattern length % period
Next event index = 3 % 4 = 3
Search with array suffix

- Other example with console

<table>
<thead>
<tr>
<th>Interval</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>110</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>7</th>
<th>4</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>log2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>p=4</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Next event index = last pattern length % period
Next event index = 3 % 4 = 3

suffix p=4: [2 2 7 2]
Search with array suffix

- Other example with console

<table>
<thead>
<tr>
<th>Interval</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>110</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>112</th>
<th>4</th>
<th>7</th>
<th>4</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>log2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>p=4</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Next event index = last pattern length % period
Next event index = 3 % 4 = 3

Suffix p=4

ema table

ema[2] = 4
Embedded cpuidle governor

- Makes use of the interrupt prediction

- Clearly identifies the source of wake up in the prediction path

- Designed to work with the embedded systems, especially mobile
  - Tweaked for mobile workload (video, audio, benchmarks)
  - Iteratively improved with non-regression testing for existing and defined workloads
  - Avoids to use biased heuristics

- How does it compare with the existing?
Selection latency

![Graph showing selection latency with different CPU frequencies and load types.](image)
Selection latency

- Higher latency on the CPU with the interrupts
  - Usually CPU0

- Other CPUs have a negligible latency

- The higher the interrupts number, the higher the load, the lower the idle duration
  - Do we really care about these latencies?

- Some part of the prediction can be still optimized
  - Suffixes on the fly, unpredictable interrupts discarded from the prediction, etc...
Measurements - Jankbench test1
Measurements - Jankbench test2

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>Frame duration</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference (menu)</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>mbed</td>
<td>127.595884</td>
<td>54.99558694</td>
<td>99.70631424</td>
</tr>
</tbody>
</table>
Measurements - exoplayer (ogg)

Reference (menu) 100
mbed 97.24390244
Measurements - exoplayer (mov)
Conclusion

● Splitting different wake sources signals to predict works
  ○ Despite the simplicity of the actual governor we do better predictions
  ○ Better performances for better energy

● There is still room for more improvements on the mbed governor
  ○ Identified workload (expecting more than 8% energy improvement for ogg/video)
  ○ Identified weaknesses in the prediction (need_resched)
  ○ Scheduler interactions (idle wise)

● Next steps
  ○ Put noisy wakeup sources apart
  ○ Offer an API to drivers to register their next interrupt event
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org