Energy-Efficient CPU Scheduling for Mobile OS

COEN 283 Operating Systems
Term Project Final Report
Team #1

Submitted by:
Zong Wang
Jiwei Wei
Chengcheng Huang

Submitted to:
Prof. Ming-Hwa Wang

12/09/2014

Preface

Modern operating system based mobile devices are more and more powerful as the SoC technology and wireless communication technology rapidly advanced. However, their performances are severely limited by their battery capacity. There are two major solutions to this problem. The first is find a method to enlarge the battery capacity. The second one is to improve the way we consume battery. Since the former one obviously belongs to Chemistry Engineer, we Computer Engineers try to solve this problem with the later one. In this project, we mainly focus on how to improve the tasks scheduling algorithms of operating systems to lower power consumption while keeping the performance of the operating system.

Acknowledgement

We first want to thank Prof. Ming-hwa Wang for his kindly support and guidance. Then, we need to thank those authors of the paper in the bibliography for their helpful supplement materials to our project. Third, we should thank Santa Clara University Library for offering our discussion room and Design Center for project coding and implementation. Finally we want to thank each of our team members for their hard working on the project.

Table of Contents

List of Tables and Figures:

Please refer to the Chapter 9 Appendices.

Abstract

In this project report we propose to add energy estimation module to BSEFQ energy-centric OS scheduling scheme both to make the original BSEFQ algorithm more practical and to enhance the performance of time-sensitive tasks' deadline miss rate.

1. Introduction

Objective

Since most of the energy-aware scheduling algorithms extremely rely on the accuracy of the energy estimation during runtime, our goal is to develop our own energy estimation algorithm to improve the efficiency of energy-centric scheduling algorithm.

What is the Problem

The Energy-based Fair Queuing (EFQ) currently is an effective energy-efficient task scheduling algorithm for non-real-time tasks of Operation System. However, in a realistic mobile operating system, we have to consider meeting the deadline of each time-sensitive task.

Why this project related to the class

COEN 283 is a class mainly talking about the Operating System(OS). An OS includes multi-process (threads) scheduling, CPU utilization and power management, which are also the keywords of our project. Scheduling is one of the major tasks performed by the operating systems. Our project involves more accurate implementation of a scheduling algorithm for mobile systems.

Why other approaches are not good

The shortage of EFQ scheduling algorithm is that for real-time tasks and periodic processes, it may make a lot of misses for the deadline, because it is designed with only considering to have the minimum energy consumption.

Why our approach is better

Our approach improve the accuracy of “warp”. With a more accurate warp, the better energy estimation we have.

Statement of the problem

We need an energy estimation scheme to facilitate the computation of key variables that will be used in various EFQ-type algorithms. More accurate warp parameters can also be obtained to address the time-sensitive deadline missing problem.

Scope of Investigation

We will investigate the relevant energy estimation methods that can be used to estimate the actual energy consumption of a time-sensitive task in mobile devices. The accuracy of such estimation is critical to guarantee stringent time-constraint compliance in both EFQ-type energy-aware scheduling scheme and other various kind of general energy-efficient methods.

2. Theoretical Bases and Literature Review

Definition of the Problem

In most OS-level energy-aware scheduling schemes, the energy consumption of a task is estimated using an energy model and then it can used by the scheduling algorithm to use to allocate the demanded energy resource for the task. For time-sensitive tasks, i.e. real-time tasks or periodic tasks, the deadline will be missed if the estimation is not accurate enough.

Theoretical Background of The Problem

Energy-based fair queuing (EFQ) [4] is a recently-developed energy-centric scheduling algorithm. It is an extended application of the classical fair queuing algorithm in the energy domain. EFQ treats energy as the first-class resource in the system.

EFQ is an implementation of the Generalized Processor Sharing (GPS) [5] in the energy domain. In a real system, the energy is served to tasks along with discrete time quanta whose length is constrained by the context switching overheads. An energy packet is defined as the energy consumption of a task during its time quantum. The starting-energy tag is the normalized energy consumption of a task (with its weight) before it is served the energy packet.

A starting-energy based fair queuing (SEFQ) scheduler simply selects the task with the lowest starting energy tag.

To give time-sensitive tasks priority on consuming their energy shares, a real-time friendly mechanism named warp [1] is combined into SEFQ, and the new algorithm is called borrowed starting-energy based fair queuing (BSEFQ).

The quantization of service time and energy brings inevitable energy allocation error, which is defined as the difference between the actual energy a task is served in the real system and the expected energy it should be served in the GPS model.

Related Research To Solve The Problem

The paper talked about the WARP.

To compute the starting-energy of each task, a benchmark power profile was used to set the various energy packet size.

Advantage/Disadvantage Of Those Research

For the "Warp" (i.e. the BSEFQ algorithm):

*Advantag*e: Warp mechanism can solve to some extent the time-sensitive task issue.

Disadvantages: Warp parameters are predefined based on some assumptions. Inaccurate parameters can lead to either misses of tasks (due to too stringent parameter values) or over-used energy share by those tasks (in case of too loose values). In the later case, energy-based scheduling is weakened by the traditional more priority-based scheduling.

For using benchmark power profile:

Advantages: It can be used to compare the performance of different energy-based scheduling algorithms in simulation environment.

Disadvantages: In real systems, the real-time and interactive tasks don't have accurate power consumption profiling available. An estimation of energy packet sizes must be used to compute the starting-energy tags that will be used by EFQ-type algorithms. Errors in estimation can cause the time-sensitive tasks missing deadlines.

Our Solution To Solve This Problem

Accurate Estimation by building a system-level power model using nonnegative linear regression method that describes the aggregate the power consumption of the processors, the wireless network interface and the display, which are obtained from OS or application, without monitoring the power status of each hardware component.

Our solution is trying to find a balance point between high missing rate with strict warp modification and low missing rate with less energy efficiency.

Where Our Solution Different From Others' Solutions

Our solution will still need to use warp mechanism but its usage is limited to a low level that will not affect the nature of the energy-centric scheduling.

While in Borrowed Starting-Energy based Fair Queuing(BSEFQ), since there is no such estimation (benchmark power profiling is used instead) mechanism, improper setting of warp parameters may change the nature of scheduling to a traditional priority-based one.

Why Our Solution Is Better

Our solution doesn't rely on specific benchmark power profiling to compute the starting-energy tag values. Instead, we use a system-level power model to estimate the expected run-time task energy consumption, which can be used for the computation of starting-energy tag values.

3. Hypothesis

Multiple Hypothesis

We claim our energy estimation algorithm will deliver better performance than the current algorithm proposed in Wei J. et al. 2014[4]. Then, the new warp based on our algorithm will have a more accurate energy-efficient scheduling.

4. Methodology

How to Generate/Collect Input Data

We will run some code of typical tasks and collect the data of run-time energy consumption of those tasks.

To profile the power consumption of the above benchmarks, each basic computational component, including *fft*, *fft_io, stringsearch, cubic, isqrt*, and *rad2deg*, has been executed individually on the BeagelBoard System platform [6] with a single core processor, and the energy consumption has been measured with a power metering system.

The workloads we use are categorized into five types, namely, idle with different brightness levels, audio/video players, file download/upload at different network data rates, and streaming.

How to Solve the Problem

Algorithm Design
1. Build an energy consumption model with nonnegative linear regression method.
2. Get adaption of the model for different tasks and collect the result of energy packet size.
3. Use the energy consumption estimation result to constrain the warp value.

Language Used
Since Linux is implemented in C, we will use C to modify the file *fair.c* under the Linux kernel directory */kernel/sched.*

Tools Used
Use the power metering system to measure the energy consumption of each task.

How to Generate Output

The outputs will be the result produced by our energy estimation algorithm with tasks inputs and their real energy consumption.

How to Test Against Hypothesis

We will compare the results produced by our energy consumption model with the real energy consumption we measured by using the power metering system.

How to Proof Correctness(Required by Dissertation)

The more our results similar to the real energy consumption we measured, the more accurate our method is.

5. Implementation

Code

Simulator Description

This simulator creates a simple model to simulate the scheduling of processes based on BSEFQ with improvement of addition of energy estimation module.

Functions

ProcessBuild

ProcessCall

Input

```
int pid;
int weight;
int starting_time;
int starting_energy_tag;
int finish_energy_tag;
int iPacket;
int *Power;
int warp_value;
int warp_limit;
```

Output

```
MissRate
```

Energy Fair Queuing Source Code

We sent several emails to the author of the paper we researched in proposal presentation in order to get the source code of the EFQ. However, we do not get any reply from the author. Therefore, the source code will not be included here.

Design Document and FlowChart

Design of the Energy Estimation Model

The power consumption is estimated based on the linear regression model [2] below.

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j} \tag{1}$$

β_0 is the intercept, which implies the response when all the variables are set to zero, and β_j, $j = 1, ..., p$ are the variable coefficients (system-specific) to be set by model fitting [7]. In our experimental implementation design, p is set to 19 which includes 17 HPCs (Hardware Performance Counters) for processors, the downlink and uplink data rates for the WNI (Wireless Network Interface), and the brightness level for the display. These 20 variables are used for runtime power estimation. The $x_{i,j}$, $i = 1, 2, ..., n$, $j = 0, 1, ..., p$ are $p+1$ predictors in our model for n test cases.

Note: example HPCs available on ARM 1136 are:

DCACHE_MISS,
CPU_CYCLES,
ITLB_MISS,
TLB_MISS,
CYCLES_DATA_STALL,
INSN_EXECUTED, DTLB_MISS,
DCACHE_ACCESS,
DCACHE_MISS, EXP_EXTERNAL,
DCACHE_ACCESS_ALL,
IFU_IFETCH_MISS,
BR_INST_MISS_PRED,
CYCLES_IFU_MEM_STALL,
LSU_STALL,
PC_CHANGE,
BR_INST_EXECUTED.

The observed power consumption can be then expressed as $y_i = f(y_i) + e_i$, where e_i is an additive noise with zero mean and constant variance. The values of the intercept and each coefficient are automatically adjusted during the model fitting towards a model in which the response can be the best predicted from the predictor variables. To evaluate the goodness of the prediction, in [7] the authors define the sum of the squared deviations of the predicted response, as follows:

$$S(\beta_0,...,\beta_p) = \sum_{i=1}^{n} (y_i - f(y_i))^2 \tag{2}$$

The model fitting process is to find a set of coefficients that minimizes the sum. An example system-level energy estimation formula looks like

$$
\begin{aligned}
Power(W) = 0.7655 + 0.2474 \times \frac{u_i / d - 1316.84}{1349.423} \\
+ 0.0815 \times \frac{w_i / c_0 - 0.000901}{0.00045} \\
+ 0.0606 \times \frac{m_i / c_0 - 0.000513}{0.000365}.
\end{aligned}
\tag{3}
$$

In which, we assume that there are N processes contributing to the HPCs during a monitoring period d . For process i *(i=0..N-1)*, the increments in DCACHE_WB, TLB_MISS, and CPU_CYCLES are defined as w_i, m_i and u_i, respectively.

Once the coefficients are obtained during the initialization of energy estimation model, equation (1) can be used to generate the estimated energy that is to feed into our improved BSEFQ scheduling module.

Flowchart

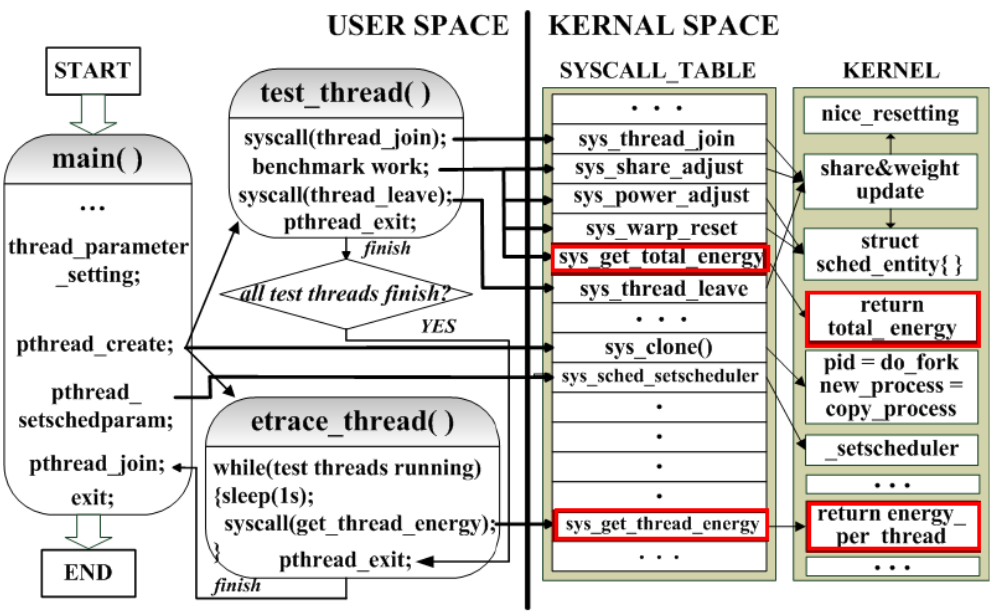


Figure 1. Structure diagram of pthread-based test-bench

The test-bench the authors of [4] proposed uses the energy variables in Figure 1 (the red-color function modules) with value that has already measured before the runtime. In our implementation, we use the runtime output from our energy estimation model to replace the already measured energy variable.

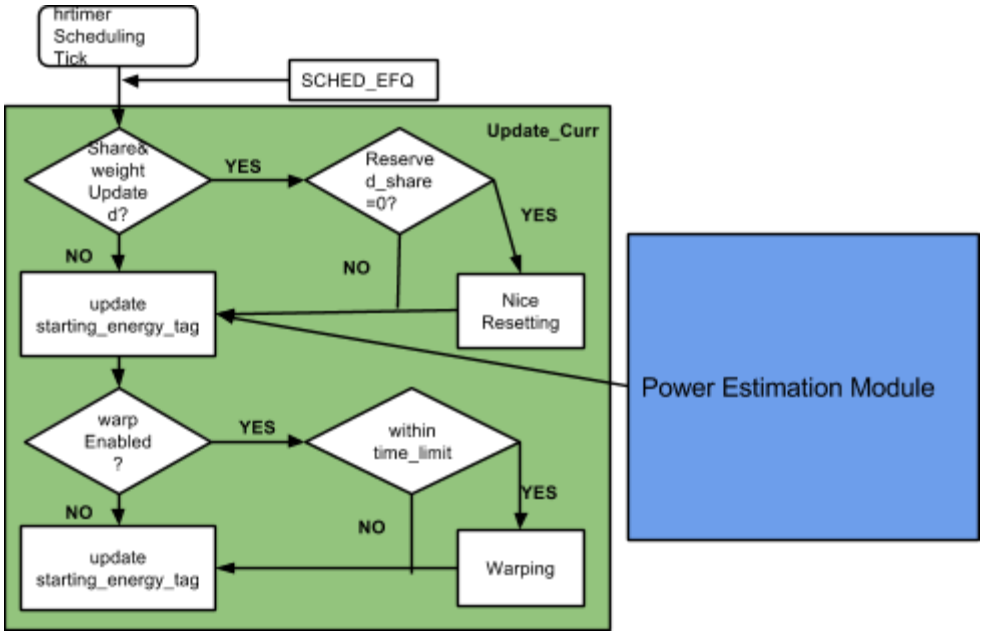


Figure 2. Flowchart of energy tag with power estimation module

Computation of energy tag is mainly taken place in the file *fair.c* under the Linux kernel directory */kernel/sched*. Figure 2 shows the flowchart under periodic scheduling tick.

hrtimer: Linux High Resolution timer.

Update_Curr: a module in *fair.c*

6. Data Analysis and Discussion

Output Generation and Analysis

Simulator Output

The output data of our simulator shows the miss rate of all time-sensitive processes under different warp configurations. The following simulation snapshots show two example configurations:

1) *warp* = 0, meaning that no warp is used for time-sensitive processes. The resulting process overall miss rate is 8%.
2) warp = 1, meaning warp is used for time-sensitive processes. The resulting process miss rate is 7%.

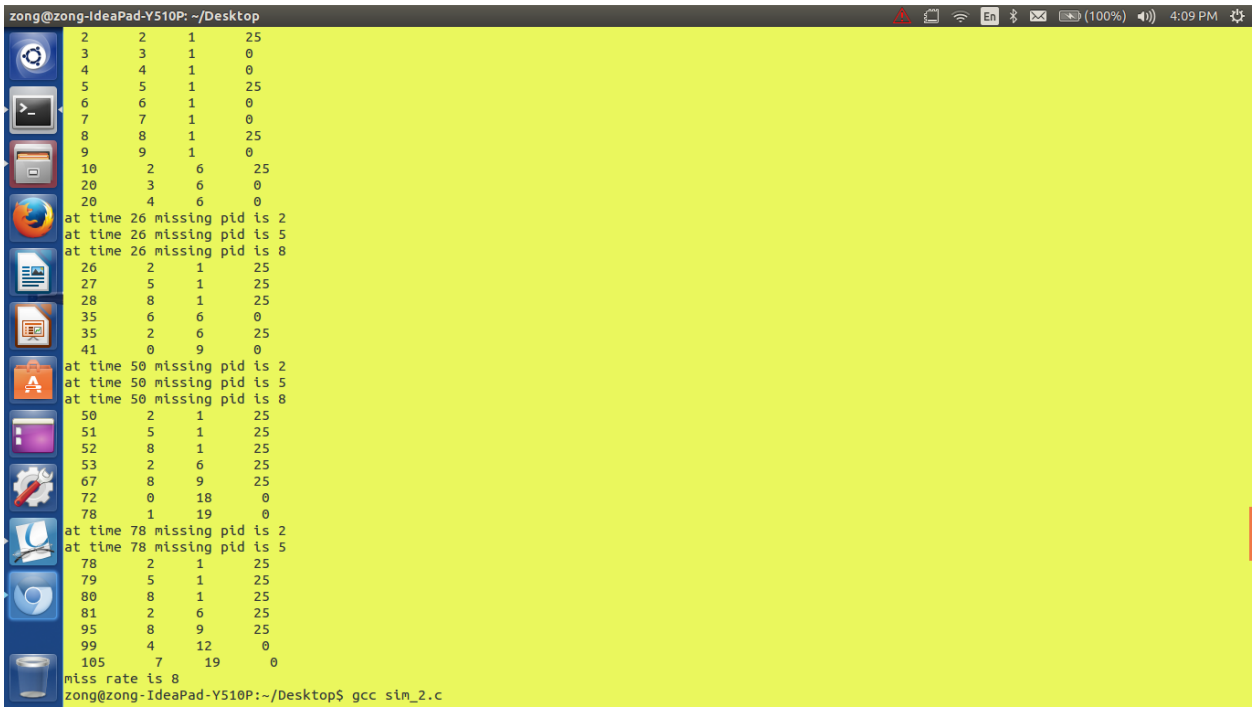


Figure 3. Simulator output snapshot: warp = 0

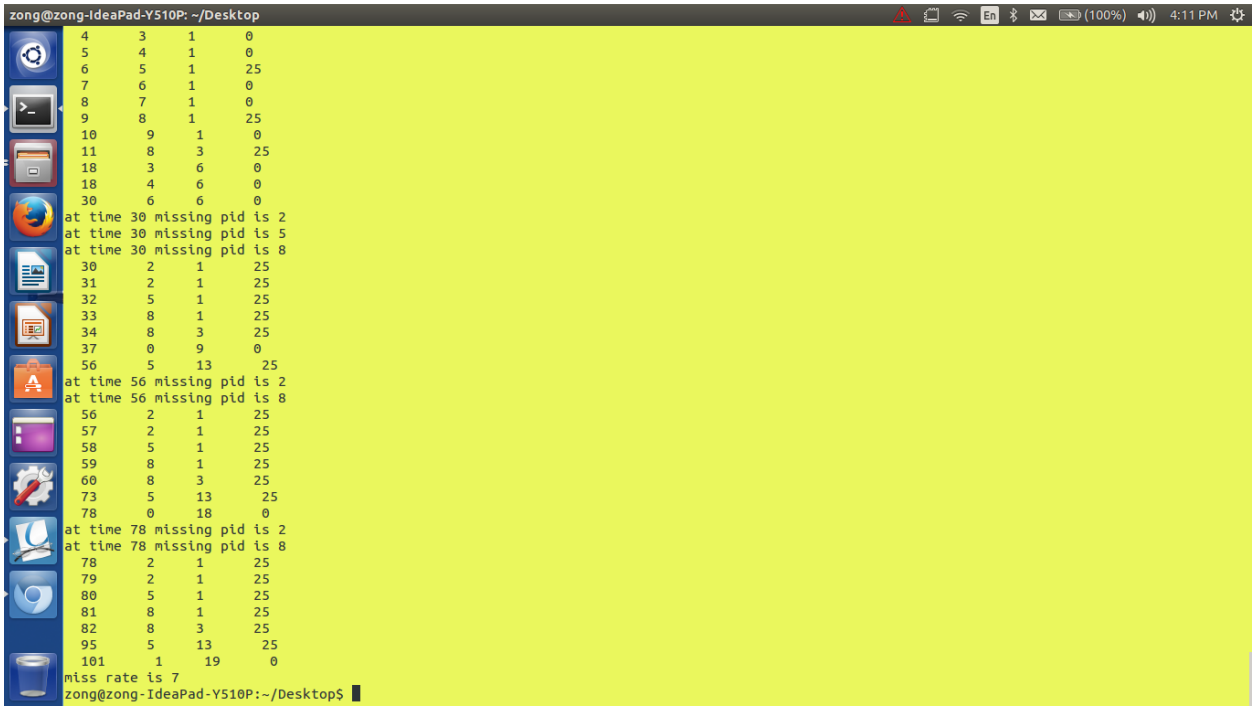


Figure 4. Simulator output snapshot: warp = 2

Expected Output

Due to lack of resources such as time and power metering device in this project, we can only generate very few data that serves to show the basic idea. However, ideally for each test case, we should expect to have the following result data charts to have a thorough evaluation of our hypothesis.

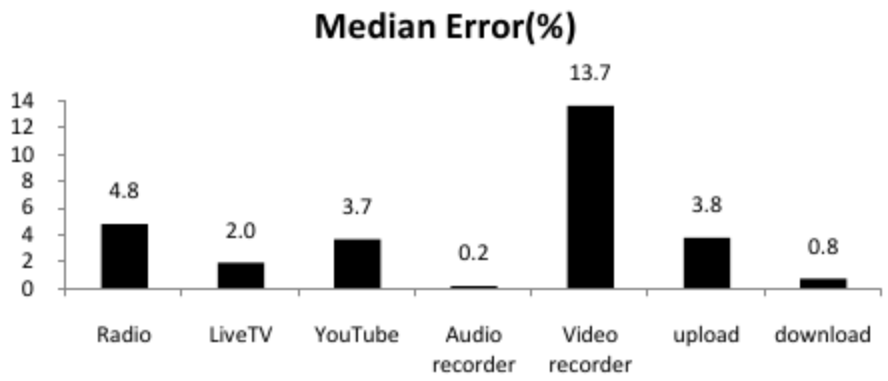
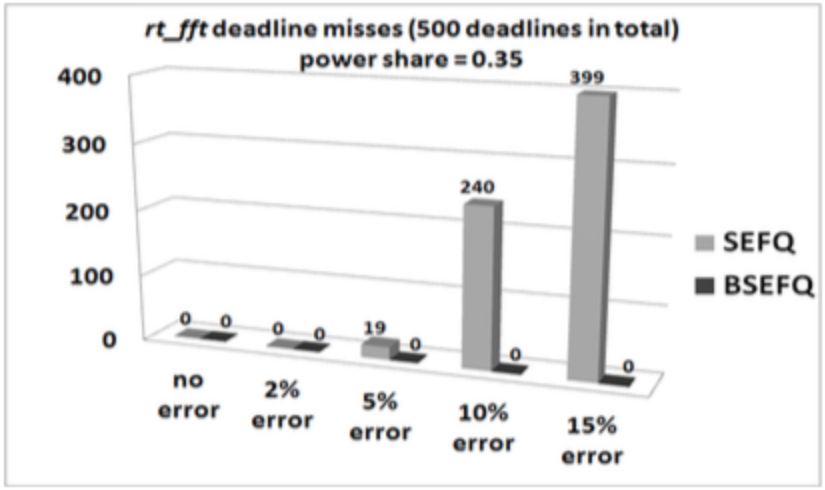


Figure 5. Expected Median percentage error in the power estimates of different test cases



a) *rt_fft* deadline misses

Figure 6. Expected Real-time performance upon different levels of energy estimation error

7. Conclusions and Recommendations

Summary and Conclusions

In this project, we are trying to find an energy saving scheduling algorithm. From the linux implementation paper[4] we found the EFQ is one of the best algorithm for cpu scheduling efficiently. However, the EFQ is highly relying on the energy estimation for each process. So we tried to build an accurate energy estimation model.

Time-constraint compliance under SEFQ is of weak robustness upon the increase energy estimation error. This problem can be solved in BSEFQ without overly reserving power shares for time-sensitive tasks. The linear regression energy estimation model is implemented for feeding the energy information in real-time to the EFQ scheduling and experiment can show the proposed combination is effective.

Recommendation for Future Studies

A more reliable and convenient process-level energy estimation method should be built and implemented with the BSEF scheduling.

In the future, in order to have a convincing results, we should still continuing to find a power metering system, which is better, more practical and easier to handle. So that we can measure the output conveniently.

8. Bibliography

[1] Duda, Kenneth J., and David R. Cheriton. "*Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler*." ACM SIGOPS Operating Systems Review. Vol. 33. No. 5. ACM, 1999.

[2] Xiao, Yu, Rijubrata Bhaumik, Zhirong Yang, Matti Siekkinen, Petri Savolainen, and Antti Yla-Jaaski. "*A system-level model for runtime power estimation on mobile devices.*" In Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom), pp. 27-34. IEEE, 2010.

[3] Olsen, C. Michael, and Chandra Narayanaswarni. "*PowerNap: An efficient power management scheme for mobile devices.*" Mobile Computing, IEEE Transactions on 5.7 (2006): 816-828.

[4] Wei, J., E. Juarez, M. J. Garrido, and F. Pescador. "*A Linux implementation of the energy-based fair queuing algorithm on an ARM-based embedded system.*" In Consumer Electronics (ICCE), 2014 IEEE International Conference on, pp. 546-547. IEEE, 2014.

[5] Parekh, Abhay K., and Robert G. Gallager. "*A generalized processor sharing approach to flow control in integrated services networks: the single-node case.*" IEEE/ACM Transactions on Networking (ToN) 1, no. 3 (1993): 344-357.

[6] BeagleBoard System Reference Manual Rev. C4, December 2009

[7] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. "*A system-level model for runtime power estimation on mobile devices.*" Proceedings of GREENCOM-CPSCOM'10, 2010.

9. Appendices

Sample code

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 4096
int Time=0;
int ProcessNum=0;
int RunningSeq;
int MissRate=0;
int VAR=1;/*trace the lowest finishing energy tag*/

struct process
{
 int pid;
 int weight;
 int starting_time;
 int starting_energy_tag;
 int finish_energy_tag;
 int iPacket;
 int *Power;
 int TotE;
 int TotTime;
```

```
 int TimeLeft;/*how many work left to be finished*/
 int FinIndex;
 int period;
 int pSeq;
 int deadline;
 int warp;
};

#define PROCESS struct process

PROCESS sched[MAX];

int max(int a1,int a2)
{
     if(a1>=a2) return a1;
     else return a2;
}

int ProcessBuild()
{
 int i,j;

 for(i=0;i<10;i++)
     {
      sched[i].pid=i;
      sched[i].weight=rand()%3+1;
      sched[i].starting_time=rand()%10;
      sched[i].TotTime=(rand()%11)+1;
      sched[i].TotE=0;
      sched[i].period=rand()%3*15;
      if(sched[i].period>0)
          sched[i].deadline=sched[i].starting_time+sched[i].period;
      else sched[i].deadline=MAX;
      sched[i].pSeq=0;
      sched[i].TimeLeft=sched[i].TotTime;
      sched[i].FinIndex=-1;/*not finished yet*/
      int Energy[sched[i].TotTime];
      for(j=0;j<sched[i].TotTime;j++)
          Energy[j]=rand()%23;
      for(j=0;j<sched[i].TotTime;j++)
          sched[i].TotE+=Energy[j];
      sched[i].Power=Energy;
      sched[i].iPacket=0;
```

```
      sched[i].finish_energy_tag=0;

sched[i].starting_energy_tag=max(VAR,sched[i].finish_energy_tag);
      sched[i].warp=0;

      ProcessNum++;
      }
 for(i=0;i<10;i++)
     {
      if(sched[i].period>0)
       {for(j=1;j<10;j++)
          {
           sched[ProcessNum]=sched[i];

sched[ProcessNum].starting_time=sched[i].starting_time+sched[i].perio
d*j;
           sched[ProcessNum].pSeq=j;

sched[ProcessNum].deadline=sched[ProcessNum].starting_time+sched[Proc
essNum].period;
           ProcessNum++;
           }
       }
      else sched[i].deadline=MAX;
     }
 return 1;
}

void top(int i)
{
 for(i;i<ProcessNum;i++)
     {
      printf("pid %d weight %d starting time %d totalenergy %d
timeleft %d period %d ",
sched[i].pid,sched[i].weight,sched[i].starting_time,sched[i].TotE,sch
ed[i].TimeLeft,sched[i].period);
      if(sched[i].FinIndex<0) printf("not finished\n");
      else if(sched[i].FinIndex==0) printf("missed \n");
          else printf("finished   \n");
     }
}

int ProcessCall()
```

```
{
 int l,i,j,k=0;
 int lowest_set=MAX;
 int lowest_seq=MAX;
 for(i=0;i<ProcessNum;i++)
      {

      if(sched[i].FinIndex>=0) continue;
      if(sched[i].starting_time>Time) continue;
      if(sched[i].TimeLeft==0) continue;
      if(Time>=sched[i].deadline)
      {
            MissRate++;
            sched[i].FinIndex=0;
            printf("at time %d missing pid is
%d\n",Time,sched[i].pid);
            continue;
      }

      if(sched[i].iPacket<=lowest_seq)
            {
            lowest_seq=sched[i].iPacket;
            j=sched[i].starting_energy_tag;
            if(j<lowest_set) k=i;
            lowest_set=j;
            }
      else continue;
      }

 if(sched[k].starting_time>Time)
 {
       Time++;
       return 0;
 }
 if(sched[k].FinIndex>=0)
 {
       Time++;
       return 0;
 }
 if(sched[k].TimeLeft==0)
 {
       Time++;
       return 0;
```

```
 }

 if(sched[k].starting_energy_tag>=sched[k].TimeLeft)
      {
      sched[k].FinIndex=1;
      Time=Time+sched[k].TimeLeft;
      sched[k].TimeLeft=0;
      printf("  %d      %d     %d      %d
\n",Time-sched[k].TimeLeft,sched[k].pid,sched[k].starting_energy_tag,
sched[k].period);


sched[k].finish_energy_tag=sched[k].finish_energy_tag+sched[k].Power[
sched[k].iPacket]/sched[k].weight;

sched[k].starting_energy_tag=max(VAR,sched[k].finish_energy_tag);
      }
 else
      {
      if((Time+sched[k].starting_energy_tag)>sched[k].deadline)
       {sched[k].FinIndex=0;
        MissRate++;
        printf("missing pid is %d\n",sched[k].pid);
        return 0;
       }

sched[k].TimeLeft=sched[k].TimeLeft-sched[k].starting_energy_tag;
      sched[k].iPacket++;
      printf("  %d      %d     %d      %d
2\n",Time,sched[k].pid,sched[k].starting_energy_tag,sched[k].period);
      Time=Time+sched[k].starting_energy_tag;

sched[k].finish_energy_tag=sched[k].finish_energy_tag+sched[k].Power[
sched[k].iPacket]/sched[k].weight;

sched[k].starting_energy_tag=max(VAR,sched[k].finish_energy_tag);
      }

}
void main(int argc, char* argv[])
{
 ProcessBuild();
 top(0);
```

```
 printf("time    pid   SET   period  \n");
 while(Time<1000) ProcessCall();
 top(0);
 printf("miss rate is %d \n",MissRate);
}
```