Implementation of Feedback Control Scheduling with EDF and Exploration on the Relationship between Miss Ratio and CPU Utilization

Han Liao and Yiming Bian
Department of Electrical and Computer Engineering for Cpr E 558 Final Project. Feedback EDF Simulation and Implementation.
lhan@iastate.edu, ybian@iastate.edu.
Cpr E 558
Simulation and Implementation

Executive Summary

In the big data era, the scale of information processed has risen to a very high level. For any processor, the ability to deal with multiple tasks in the same time is becoming necessary and critical. However, it is still a normal situation where overload happens. Comparing to a static system design that has a fixed schedule table but suffers from vulnerability when facing massive tasks, a dynamic system design enjoys more advantages and brighter future. There are several models to handle overload, such as imprecision computation model and (m,k) firm deadline model, but both of them are restricted by their property of static. The better idea to handle overload is the iteration of the input and output with the concepts of (m,k) firm scheduling and EDF scheduling, which makes the system dynamic. So, we are focusing on the implementation of a feedback EDF scheduling and intend to close the gap between real performance and expected performance of real time system. Based on this idea, we will look into the statistics and try to explore the relationship among a few parameters such as utilization, miss ratio etc.

1. Introduction

A feedback system aims at handling dynamic workloads where the job execution times vary significantly. Compared to traditional systems, it requires dynamic voltage scaling so that it would not consume energy at a same level all the time, which is considered to be a promising design to control the energy consumption issue. Feedback system uses the measured output as an input of next round to generate proper signals for actuator to operate, with the new output in the next round, it can be transmitted to the controller to generate the corresponding signals for the round after the next, which creates a loop and finally approach the expected performance. When operating a feedback system, a set point needs to be provided, usually it is known as ideal miss ratio. And the signal is passed to sensor to detect the current quality of service and calculate a measured variable, usually it is known as the current miss ratio. After that the current miss ratio and the ideal miss ratio are compared to make the decision for actuators in the next round, which would yield a new current miss ratio and would be the input of the round after the next one. With this iteration process, the current miss ratio should get close to the ideal one and later fluctuate around the set point, which is what we expect for this implementation.

2. Problem Formulation

The topic of our project is implementation of feedback EDF control system and exploration of the internal relationship between miss ratio and utilization. The thought behind this type of system is to handle dynamic workload and creating a energy-saving possibility due to dynamic

voltage scaling. It is a very good opportunity for us to understand the process of a feedback loop and some internal relationships that may be helpful for future work.

3. Methodology

3.1. Algorithm / Protocol

Our project is divided into two parts. The first part is mainly to implement the traditional EDF algorithm; the second part is to add the m, k police based on EDF. In this way, each periodic task will not have the characteristics of hard deadline. And the set point provided by the user will make the system present the dynamic performance. Therefore, the foundation of project is still the EDF algorithm building. Before we get into the algorithm and coding style in details, let’s start by outlining the code structure behind the system.

Our system compilation language is Java. The reason why we choose Java is that it has a rich resource base and clear structural characteristics. Java Swing library was heavily used in our writing of user interfaces. The system is mainly divided into three parts.

The first and obvious part is the interface presented to the user, which is mainly programmed by java.swing.* and java.awt.graphics. And since our experimental research is more about the realization of functions and research, there is no further art design and optimization of the user interface.

Secondly, it is the back-end calculation directly related to the user interface display data and the core part. Back end calculation includes data processing, analysis and feedback three common functions. The user's foreground and raw data will be processed into tasks that can be

read and calculated by the system. After receiving the confirmation information, the back end will perform various analysis and calculation on these tasks. For example, utilization test, least common multiply time calculation, priority stack build, EDF scheduler, admission controller, mandatory and optional test. Data will be accessed and transmitted between GUI class and back-end for many times. More details on this part will be explained later.

The third part is task constructor. Since our system has two completely independent processors, (EDF and Feedback EDF) constructor distinguishes two algorithms. Constructor class contains corresponding information of each task, such as Computation Time, Deadline, Period, value for m and k, Color, mandatory or optional check in current unit time. It is because of systematic and perfect data input of each user that the latter programing compilation and operation is smoother.

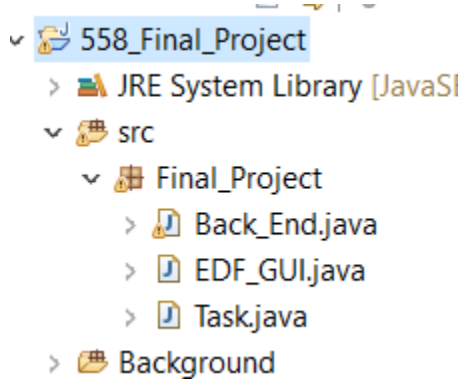


Figure 1

Previously, we have talked about GUI interface compilation mainly considering the practicality of the system and programmer experience. Due to time constraints, the GUI code will not be highlighted. In the library of javax, we mainly call and use JFrame(user interface framework), JLabel(user guide and data display), JTextArea(input reading and display of user data), JButton(back-end function implementation), JPanel(display of task contracture).Although

this is a large part of the code, it is not experimental or functional. I will not elaborate on the introduction of GUI here.

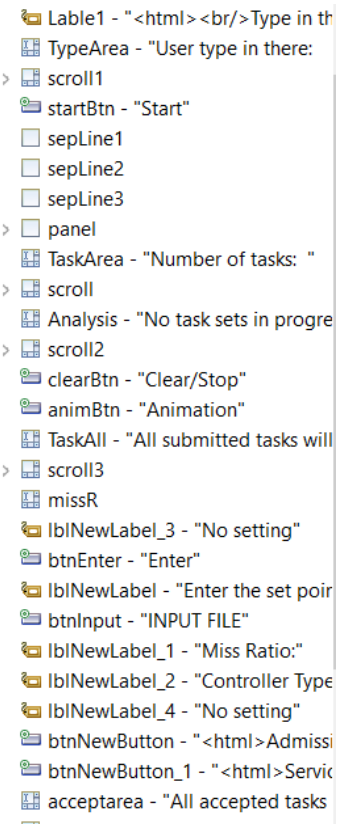


Figure 2

```
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Scanner;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.SwingConstants;
import javax.swing.Timer;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import java.awt.Font;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.JFileChooser;
```

Figure 3

The most important part of the system is the back-end class. I have summarized several major functions here:

- Traditional EDF Scheduler
- Admission Controller for Feedback System
- Priority stack build
- Mandatory and Optional check

Priority and mandatory check are the easiest parts to implement, which only need simple mathematical calculation and stack superposition. The main difficulty lies in the EDF scheduler algorithm and admission controller.

EDF scheduler algorithm Pseudo code:

```
Initialize and clear previous task list;
for (each unit time u < the lcm time)
        f: priority_build list(p) rebuilds;
        for (each task (ti) in the list)
                if u = its Period then
                        reset exe, and rest rest for this task
        for (ti in the list)
                if earliest time (early) > rest then
                        early = rest
        for each task in the list
                if rest = early and exe != 0 then
                        list usable add(ti)

        while priority stack is not empty
                pi = pop() the top
                if pi = the usable.get(ti) then
                        schedule list (schedule) add ti

        for each task in the list
                rest--;
                exe(ti)--;
```

```
        return schedule

function priority_build(task list)
        for each task in the list
                Integer array (ary) add task.getPeriod()
        Sort(ary)
        For task ti in ary
                Stack priority (p) push ti
Return p
```

We use the following notation:

- *ti*: one task in the task sets
- *u*: current unit time
- *exe*: the remaining computation time for each task
- *rest*: the remaining time to the deadline for each task
- *pi*: priority stack
- *early*: current early deadline in current unit time
- *usable*: all task satisfies the earliest deadline in current unit time
- *schedule*: the final scheduled task list

Feedback EDF System Pseudo code:

```
While Timer tm is running
If current missed < set points then
        Push the next task ti in to accepted list list
        Miss task count missed = 0
        Initialize and clear previous task list;
        Initialize scheduled scheduled
        f: priority_build list(p) rebuilds
        computation time(excuteable exe)and remaining time(rest) HashMap initialize;
        for (each unit time u < the lcm time)
                f: priority_build list(p) rebuilds;
                for (each task (ti) in the list)
                        if u = its Period then
                                reset exe, and rest rest for this task

        for (ti in the list)
                if earliest time (early) > rest then
                        early = rest
                else count++
        for (ti in the list)
```

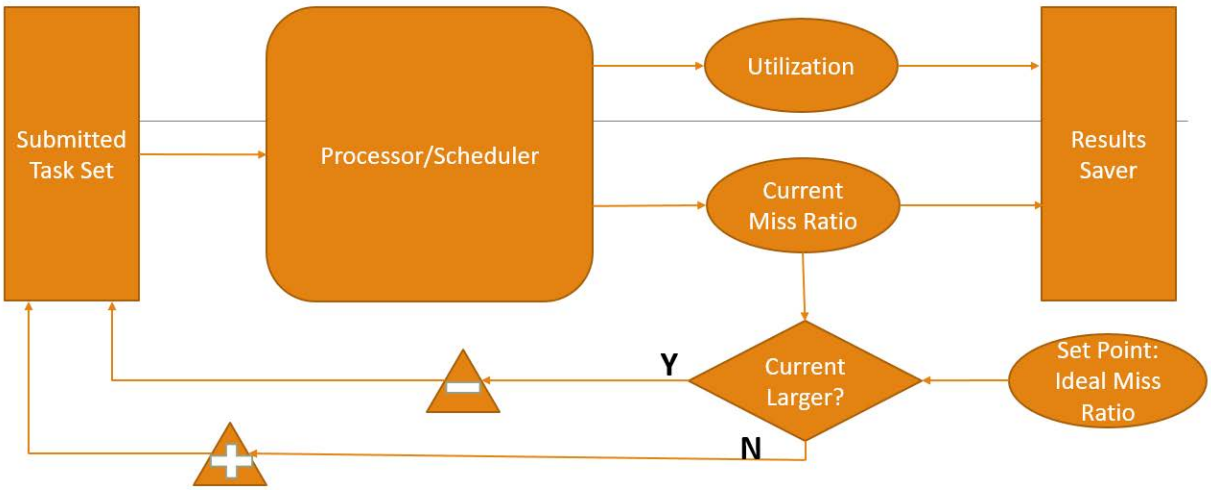
```
            if f: mandatory_check(ti) then
                    mandatory list (mi) add ti
        if mi is empty then
            if size of li = count then
                    usable list use add null
            else
                    for each ti in li
                            if exe(ti)!=0 && rest(ti)= early then
                                    use add ti
        else for each ti in mand
            if exe(mand(ti)) !=0 then
                    remain r=rest(mand(ti))

            while priority stack is not empty
            pi = pop() the top
            if pi = the usable.get(ti) then
                    schedule list (schedule) add ti

        for each task in the list
            rest--;
            exe(ti)--;
        return schedule
else
        pop the last ti in the accepted list
```

EDF's algorithm mainly has two HashMaps that continuously update the data in unit time to find the task closest to the deadline. These two maps respectively record the time taken by task computed and the remaining time needed before the deadline. If multiple tasks have the same remaining time, the priority in the stack will be used to filter.

In the feedback system, (m,k) police was added into the original feedback system, which greatly increased the compatibility of tasks and enabled many tasks to be selectively missed. In javax, we set up a timer just like animation in our system. As long as the system doesn't break down, a constant stream of submitted tasks drag more tasks into the accepted task list, and each set of tasks submits back into the back-end for computing. Calculate the current miss ratio and

then compare it with the ideal miss ratio. So this is a dynamic program, and all the immobility is based on set points entered by the user.

3.2. Illustrative Example



4. Implementation/Simulation Architecture

Java Javax Java swing Eclipse

Windows 10 Ubuntu 18.04

5. Evaluation

We have four experiment task sets: first one is barely not an overload task set, in order to test the scheduler's behavior when the system is relatively idle; the rest three task sets are all overloaded, in order to test the scheduler's behavior when the system is handling massive workload and wipe out the possibility of being occasional. We plan to illustrate two charts of each trial: the tendency of miss ratio as the time goes and the relationship between miss ratio and utilization.

Here is the experiment result of first experiment:

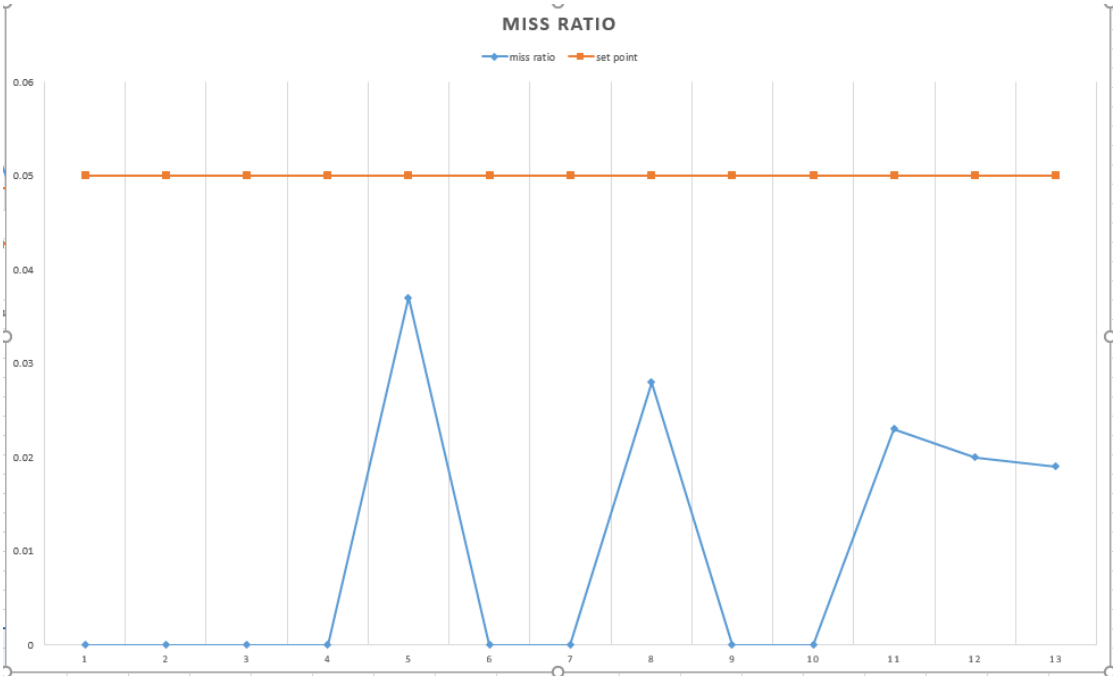


Figure 4

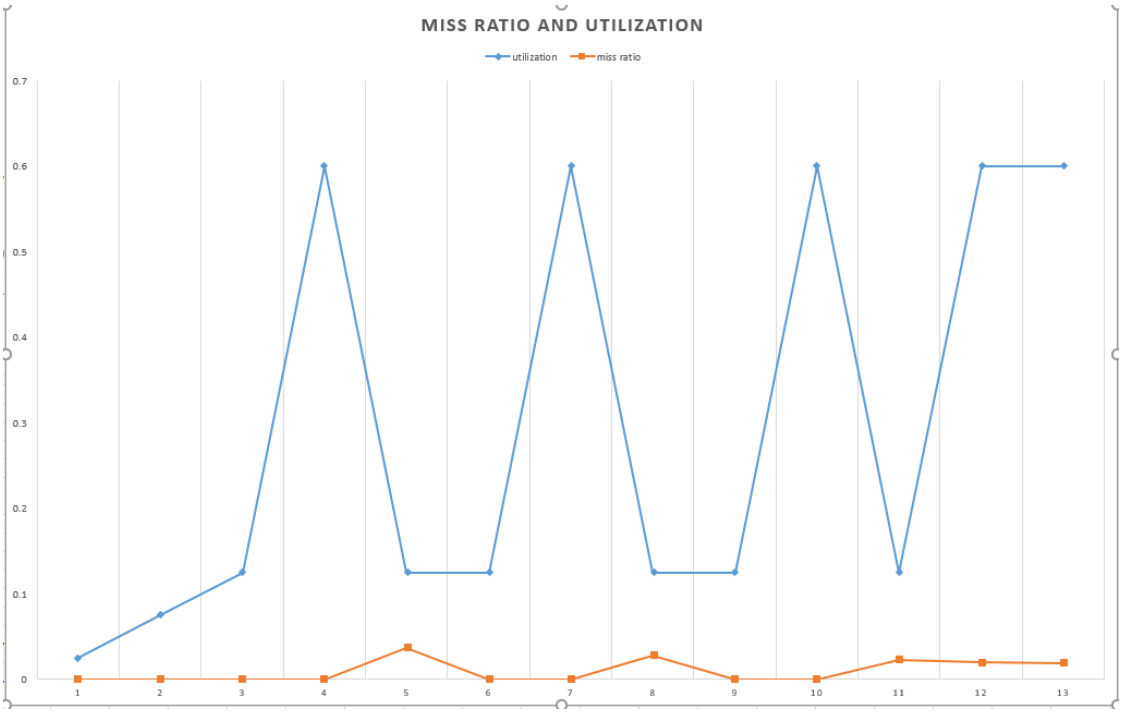


Figure 5

Here is the experiment result of second experiment:

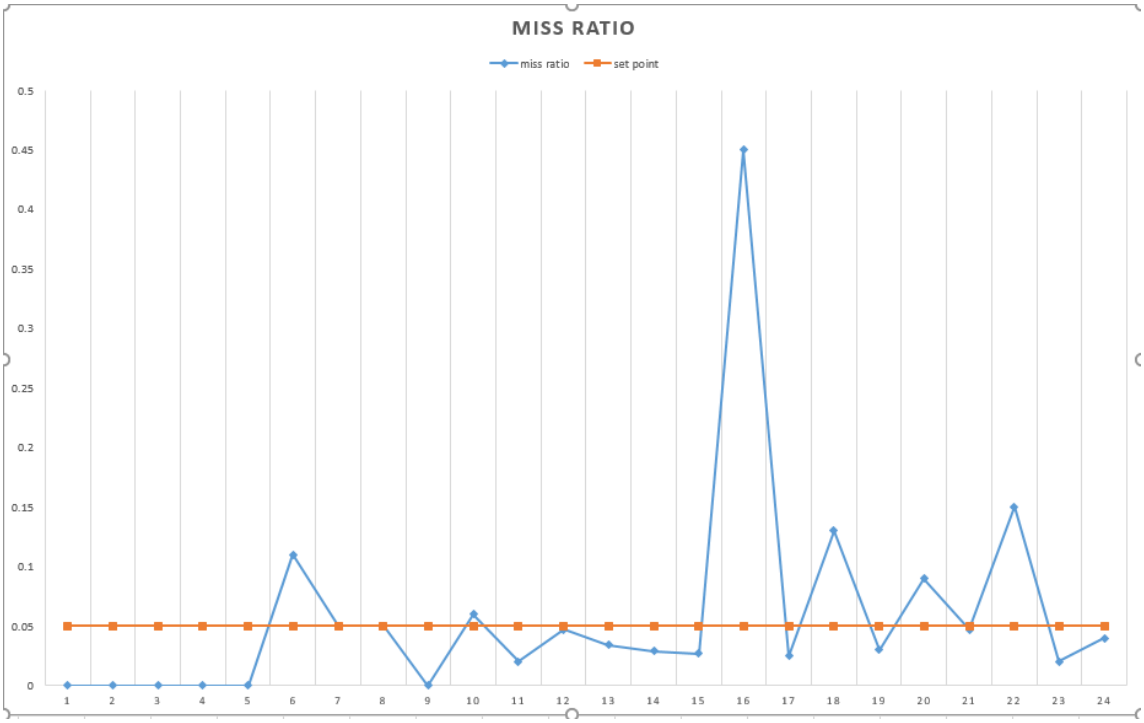


Figure 6

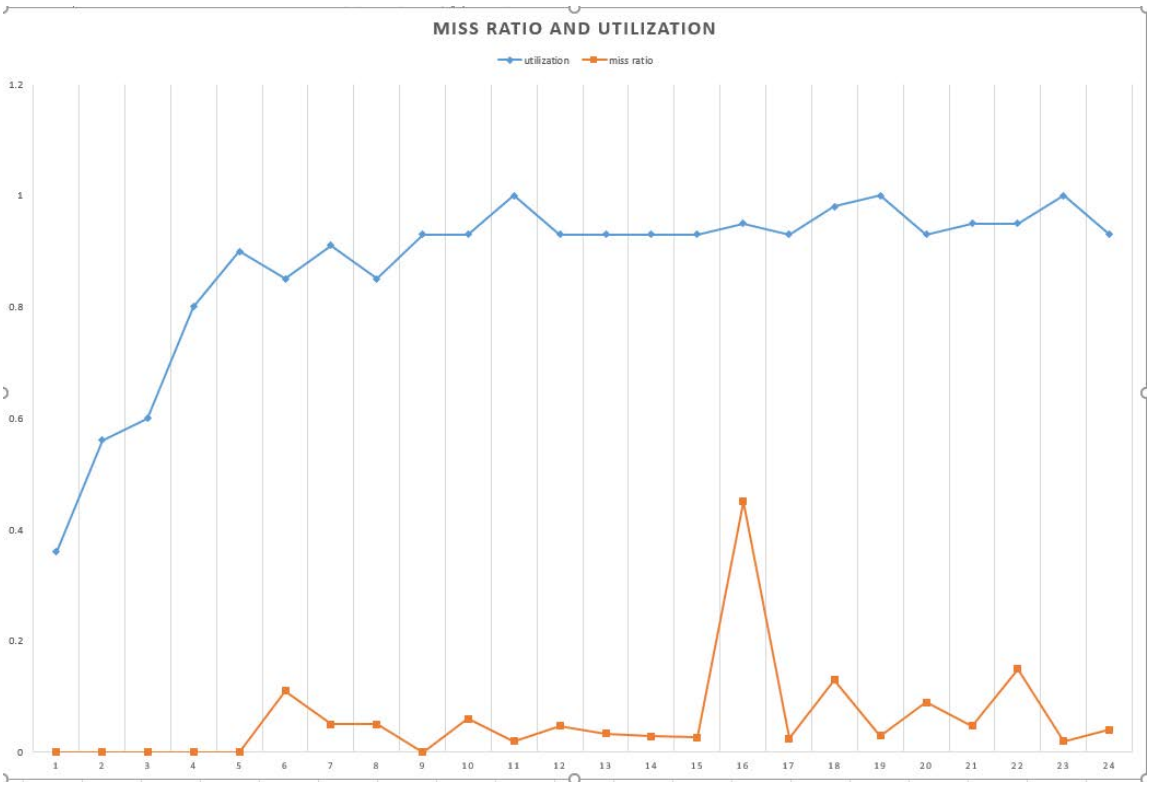


Figure 7

Here is the experiment result of third experiment:

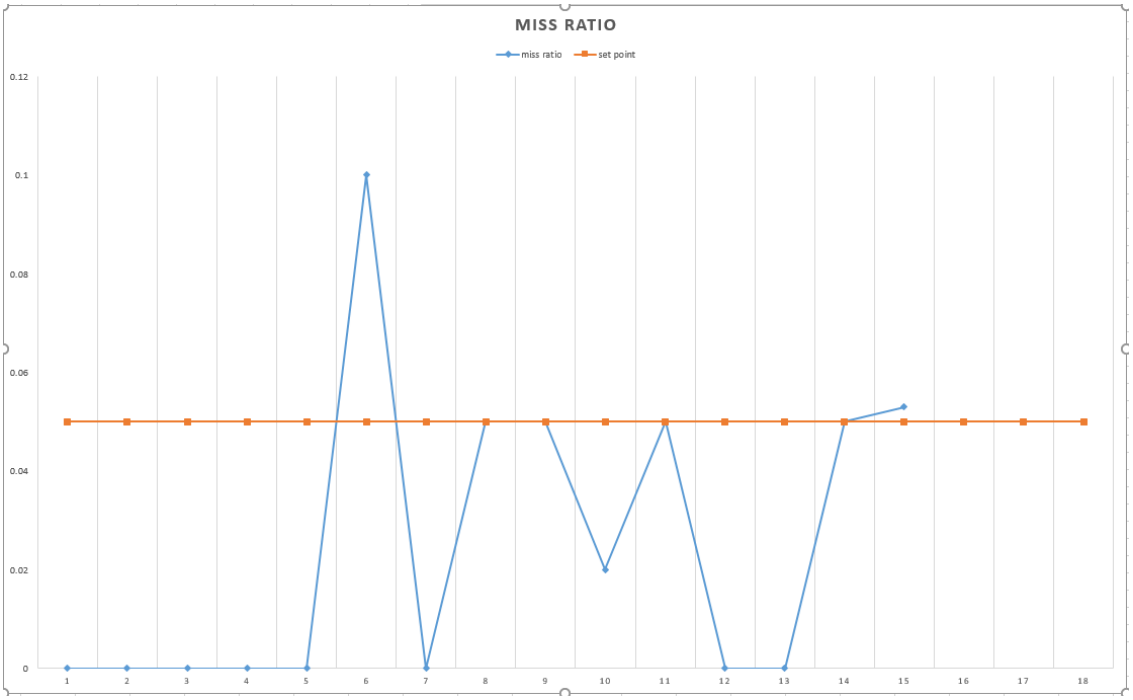


Figure 4

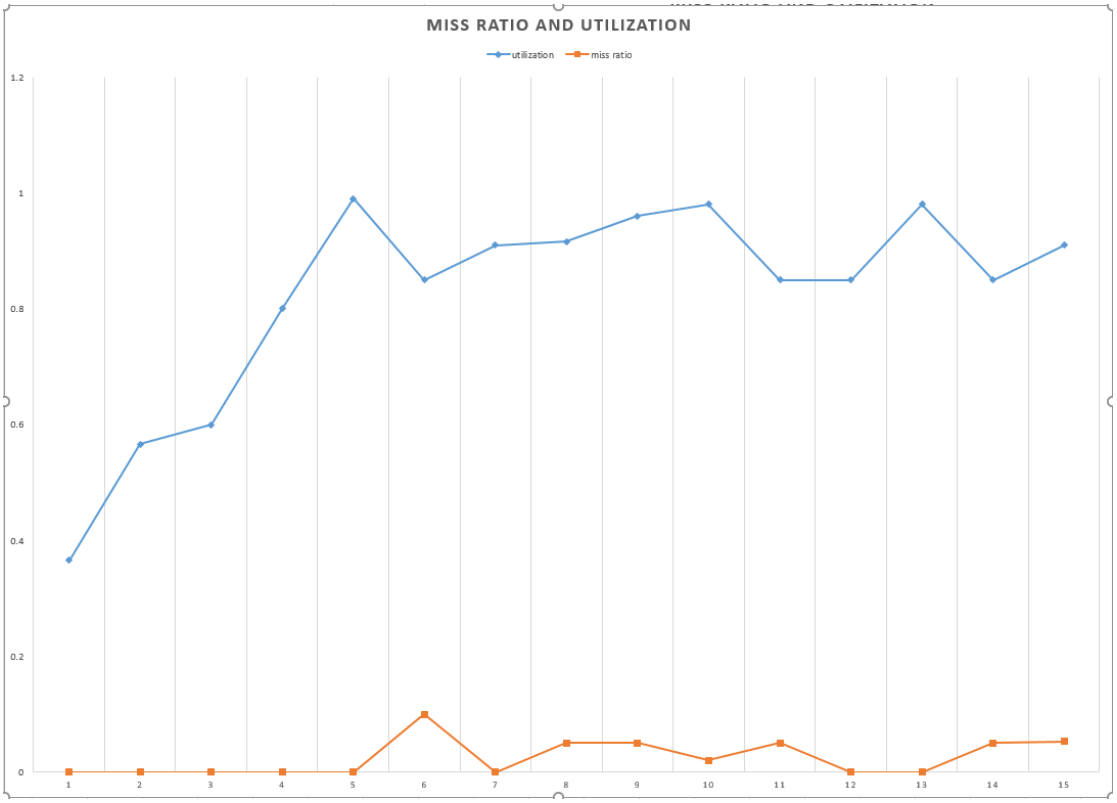


Figure 9

Here is the experiment result of last experiment:

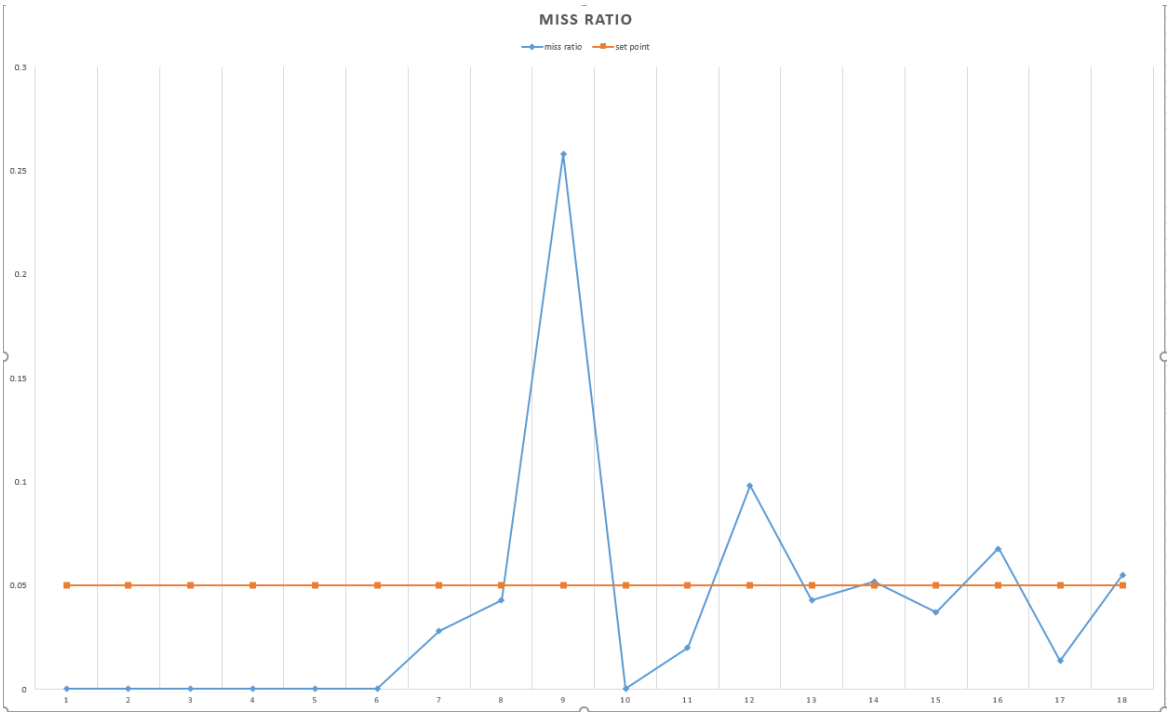


Figure 50

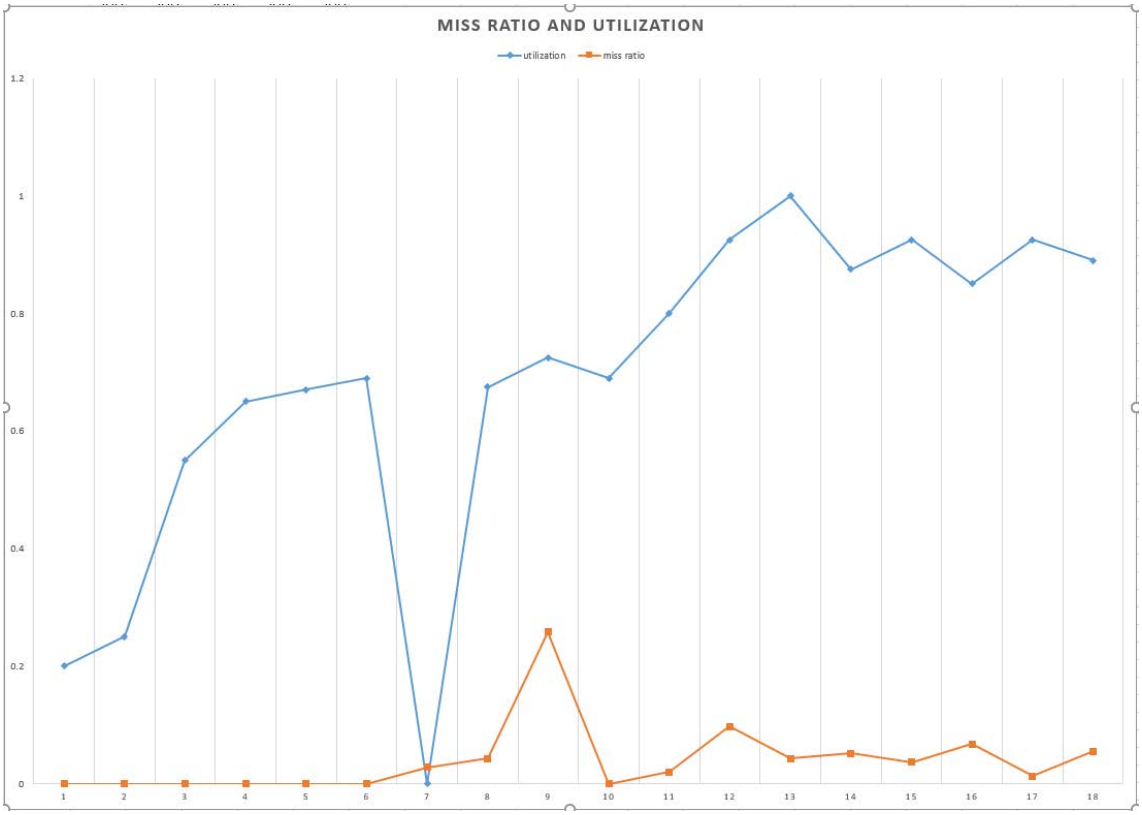


Figure 11

It is very clear that when the workload is light, there is hardly any miss so the miss ratio may never reach the set point and the feedback system behave like a normal system. As to the utilization, it is actually the sum of several ratios of computation time and period of corresponding task, so the chart looks like zig-zag.

However, when dealing with a great amount and complexity of workload, the situation turns totally different. The miss rate starts from zero because there is only a few tasks and far away from overload, then as the growth of workload, the miss ratio increases. But due to the existence of set point, the miss ratio would fluctuate around the set point and it turns out to decrease the degree of fluctuation. As to the utilization, there seems not to be a very clear relationship between it and miss ratio, but we can still tell the change pattern of utilization. It grows sharp as the workload increase and going steady around 80% and 90% under our test with our task sets.

6. Conclusions

The whole experience was fun, especially when we were running our test together. There were some bugs and misses of consideration on our implementation, but with the discussion and cooperation together, we made our work much better and well-functional. There are still much things remain to be done, for example, there should be an optimization of energy consuming because it would be wise to design the product with the characteristic of environment friendly.

7. References

[1]J.W.S. Liu, K.J. Lin, W.K. Shih, A.C. Yu, J.Y.Chung, and W. Zhao, “Algorithms for scheduling imprecise computations,” IEEE Computer, vol.24, no.5, pp.58-68, May 1991

[2] Y.F.Zhu and Frank Mueller, “Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling”, IEEE Computer Society, pp.132-141, 1997

[3]C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm," In Proc. Real-Time Systems Symposium

[4]P. Ramanathan, “Graceful degradation in real-time control applications using (m,k)-firm guarantee,” In Proc. of Fault-Tolerant Computing Symposium

[5] G. Manimaran, Overload Handling in Real-Time Systems – Part 2 Feedback Control based EDF Scheduling, Available online at: https://canvas.iastate.edu/courses/51606/files/5866358?module_item_id=1716851

Jane,W. S. & Liu. (2000). Real-Time System. Integre Technical Publishing Co., Inc

Carpenter, John. (2004). A Categorization of Real-time Multiprocessor. Scheduling Problems and Algorithms.

M.Jagadeeshraja.& B.Sasikumar. (2013). Implementation and Analysis of RMS scheduling in Real Time Operating System. Journal of Engineering, Computers & Applied Sciences (JEC&AS) Volume 2, No.11.

Yifan Zhu & F. Mueller. (2004). Feedback EDF scheduling exploiting dynamic voltage scaling. IEEE Xplore. 26 July 2004

Ajay Dudani, Frank Mueller & Yifan Zhu. (2002). Energy-conserving feedback EDF scheduling for embedded systems with real-time constraints. ACM SIGPLAN Notices. Volume 37 Issue 7, July 2002. Pages 213 - 222

Yifan Zhu & Frank Mueller. (2005). Feedback EDF scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling. ACM SIGPLAN Notices. Volume 40 Issue 7, July 2005. Pages 203-212