Interim Report 2

Neha Gholkar Xiaoqing Luo

Milestones
Week 1 :
Complete the literature survey and study about existing work (Neha and Xiaoqing) -**Done**

Week 2 :

Build vanilla scheduler simulator (Neha) - **Done**
Come up with task sets that would be fed into the simulator and perform static timing analysis considering all the possible dvfs frequencies (Xiaoqing) - **Done**

Week 3 :

1. QoS tracking (Xiaoqing)

We have implemented a QoS metric based on deadline misses. We set a window size (eg. 1000 ticks) and we define maximum number of misses that we allow in a window (eg. 10 misses). This window is a moving window and can be implemented as a cyclic array. As the window moves forward we exclude the misses that we recorded in the past.

For example, if the window size is 8, when initialized, it looks like: InitQos()

Each job has a flag to record whether it has been considered for QoS deadline misses calculation. When we detect that a job has missed its deadline we increment window's corresponding index. We check the running job and jobs in ready queue for deadline misses.

If at tick 5, a job misses its deadline, the window will be updated by UpdateQos():

CalculateQos, gives us the QoS of the system at the instant at which it is called. For now it gives us the total number of misses in the window. (from the above window Qos = 1)

If at tick 9, 2 jobs miss their deadlines, we replace (time%window_size) index with the number of recorded misses (2), the window will be look like:

At 9 CalculateQos will return 2+1=3;

If at tick 13, no new jobs are missing their deadlines we set the corresponding index to 0
(it will replace the 1 in the 5th unit of the window)

Future work on QoS is to include jobs dropped and the corresponding penalty in addition to deadline misses and its penalty.

2. Design and implement the reaction mechanism that makes decisions about the defensive frequency recovery and drop/execute a job (Neha)

Basic framework of the simulator

while(1)
{

1. For all tasks check if a job has been released at this time instance. Put the job in the ready queue
2. If a task is running
   i. if execution is not complete
      a) sys_time++;
      b) **UpdateQos**
      c) Continue;

3. If you are here : Job has just finished its execution.

Call **DVFS Algorithm to determine the frequency**

4. Schedule next job from the ready queue
5. sys_time++
6. **UpdateQoS**

}

DVFS() sets the cFrequency to the desired frequency value for the next job execution.

We implemented two algorithms:

1. Neha's Algorithm :
I preferred to take an experimental approach. My algorithm assumes that I don't have any knowledge about a job's actual execution time at a particular frequency. This algorithm doesn't look ahead while making any dvfs decisions instead it bases its decisions

1. current Qos of the system
2. whether a job has finished early / on time (at the deadline) / past the deadline (deadline missed)

Algorithm:

```
if(runtask has missed its deadline)
{
        if(num_deadlines_missed in a window is close to the max allowable misses)
                increase frequency heavily ( by 3 steps )
        else
                do nothing

}
else
{
        if(num_deadlines_missed in a window is close to the max allowable misses)
                increase frequency lightly ( by 1 step )
        else
                decrease the frequency heavily ( by 3 steps )

}
```

This is a very basic algorithm and it needs to be tuned.

The basic idea is to divide the QoS by the number of different frequency decisions that we intend to take. if max deadline misses = 10 you consider 2 distinct cases 0->5 and 5->10
In addition to this also consider whether a job of finished early contributing to the system slack or late after having missed its deadline.

2. Xiaoqing's Algorithm:

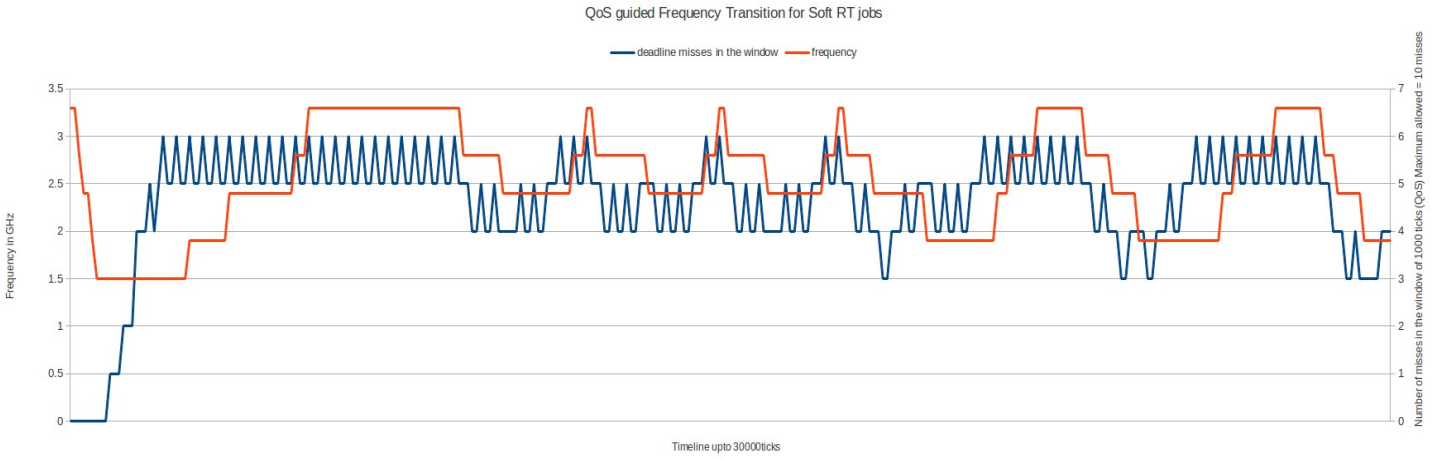
Our algorithm is based on dynamically observing the current state of Qos, and the state of current running job. We make our decision to change frequency

based on whether the job finished before deadline, and whether the CalculateQos() is larger than some threshold. We also consider about the next job in the ready queue and make the next job finish on time by selecting the appropriate frequency.

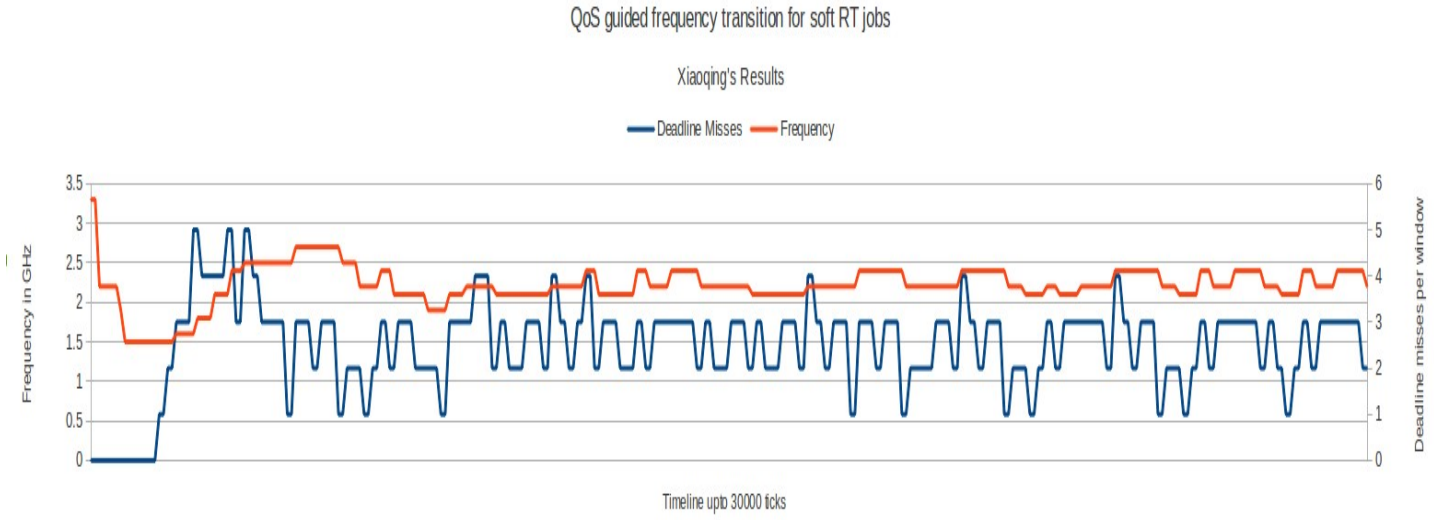
The Pseudo code of this DVFS algorithm is :

```
int DVFS( )
        int qos = CalculateQos();
        int num_miss = WINDOW*max_missing_ratio;
        if(qos > num_miss)        //if the number of misses is larger than maximum allowable misses, terminate the simulation and report failure
                exit(0)

        if no task in the ready queue  //if its idle time
                do nothing and return

        if runtask finished later than its deadline
                if qos > num_miss*P_of_QOS
                        int need_exec = next_job_execution_time – (runtask_finish_time – runtask->deadline)
                        search in the frequency<->execution_time array to set appropriate frequency   //increase the frequency
                        return the new frequency index
                else
                    do nothing  //if the Qos is very good, don’t need to increase the frequency

        if runtask finished earlier than its deadline
                if (qos > num_miss*P_of_QOS)  //means the Qos of current system is already bad, so don’t decrease frequency
                        do nothing and return
                else
                        int need_exec = next_job_execution_time – (runtask_finish_time – runtask->deadline)
                        search in the frequency<->execution_time array to set appropriate frequency   //decrease the frequency
                        return the new frequency index
```

Week 4: Run the tasks sets developed by Xiaoqing on the simulator and interpret the results (Xiaoqing, Neha)
Neha:



You can see in this result that the frequency transition follows the QoS. Its obvious because this is a history based approach. While running the simulation we see that depending on th task set we get varying average frequencies.



Xiaoqing:

In this result, you see that the frequency stabilized at 2-2.5GHz (lower than what we see in the previous approach). However here we have prior knowledge about the future job's execution times that aids in making more accurate frequency decisions.

Week 5 : Debugging and further experimentation (Xiaoqing, Neha)
Week 6 : Presentation and final report submission (Xiaoqing, Neha)
References:

- Dynamic Voltage and Frequency Scaling in Multimedia Servers, Alaa Brihi, Waltenegus Dargie, Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference
- Power-Aware CPU Management in QoS-Guaranteed Systems, Saowanee Saewong, PhD. Thesis, Carnegie Institute of Technology
- http://moss.csc.ncsu.edu/~mueller/rt/rt09/readings/projects/g5/