Real-Time Scheduling Simulations with

Periods harmonically controlled based on LCM

Shokhina Badrieva, Anastasia Golev, Alex Liew, Yumei Hou

College of Staten Island of City University of New York

2800 Victory Blvd, Staten Island, NY 10314

**Abstract**

This article presents a real time simulation of the three most popular scheduling algorithms, Rate Monotonic, Earliest Deadline and Deadline monotonic based on controlled simulated inputs. We utilized random number generators to simulate our input parameters such as arrival times, execution times, deadlines and periods for each respective process. We proposed an algorithm for modifying the generated period durations of each process so the modified period durations are close to each other and will result in the smallest possible least common multiples. This solves the problem of systems running out of memory while looking for the closest LCM especially those periods with odd or prime numbers. In extensive testing, our proposed algorithm shows Earliest Deadline scheduling have the highest success rate in feasibility.

**Introduction**

A reoccurring issue encountered by real-time systems and simulations is the generated durations of periods in scheduling algorithms, Rate Monotonic, Earliest Deadline and Deadline monotonic, are either odd numbers or prime numbers. If multiple prime numbers or odd numbers are selected as period’s duration in a simulation, the runtime length to find the LCM will become extremely long and result in stressing the system. This stress will indeed effects the system’s memory and will cause higher system overhead to execute the high number of instances of new processes. There is a need for addressing this problem to avoid system overload and memory failure.

**Related Work**

There are existing works which address this problem. [8] proposed a technique for adjusting the period durations of each process to ensure the LCM of the period durations do not explode. By doing this the algorithm guarantees the processor utilization level required by the application, and increases the chances of approaching a feasible schedule. Since [8] lowers the process’ periods the modified period would fall below the deadline values and it does nothing to address this issue. [4] proposed a technique for lowering process periods’ LCM by modifying the process’s period pi to be equal to the biggest multiple of the lowest process period that is less than or equal to it respectively pi. The issue with [10] is the modified periods could result in a significant difference between with their respective original periods. [5] proposes a method utilizing a semi-random algorithm to generate *n* process with *n* periods in a way that their least common multiple is within a certain range. [5] allows users to adjust the process periods in order to bound the least common multiple of the periods. However, in real simulation we should allow maintain some level of randomness and avoid having the user influence the simulations which may skew the simulation outcomes.

This article has the following two contributions:

an algorithm for adapting the period durations of each process so the period durations are bounded to each other, which will result in the smallest possible least common multiples and an incorporation the proposed algorithm in the simulation of three of the most popular real time scheduling algorithm, Rate Monotonic, Earliest Deadline and Deadline monotonic, which address deadline and period constraints, *pi* ≥ *di*; and

a comprehensive testing of the three algorithms using our algorithm showing results in feasibility and CPU utilizations, our new framework was able to achieve 100% feasibility in one of our simulated trials.

This paper has the following structure: Method / Approach detailing our algorithms to address exploding LCMs by bounding our period within a certain range and detailed description of our simulations; Computer Simulated Results will show our comprehensive results of our three scheduling algorithms with bounded LCM.

**Method / Approach**

To constraint the periods of each process within a certain range, we utilized the following from [8]:

,(1)

where *i*, *j*, *k*, *l*, … are integers and is bounded by the following ranges, *0* ≤ *i* ≤ *e2*, *0* ≤ *j* ≤ *e3*, *0* ≤ *k* ≤ *e5*, *0* ≤ *l* ≤ *e7*, … where *e2*, *e3*, *e5* and *e7* can be zero or positive integers. With any selection of *i*, *j*, *k*, *l*, … the array ***y*** will always have any two elements which has the same common multiples within ***y***. Figure 1, illustrate a simple example of *e2* = 1 and *e3* = 1. We substitute the values of *i*’s and *j*’s into ***y***(*i*,*j*)= *2i* *3j* which generates the following vector with the values of [1, 3, 2, 6] and then we sort the vector of *y* to get [1, 2, 3, 6]. If we look at any two elements of sorted vector ***y***, we see that any two elements are common multiples to any of the third element within the ***y***, 2 and 3 have LCM of 6. We can expand (1) by setting higher values to *e2*, *e3*, *e5* and *e7* and generating an array of ***y*** with higher values but with bounded LCM. It is also ironic to point out, 2, 3, 5, 7, … are numbers that are needed to generate vector ***y***, but are also prime numbers, unwanted periods, which are detrimental to our simulations.

We can find the lowest possible LCM in any set of period values whether they are prime numbers or odd numbers. We can utilize (1) and adjust periods by assigning periods closest to generated periods. Let’s say we have another vector *y1*, shown in figure 1 (right), having the following values [1, 2, 5, 6], where the LCM of the all the elements in vector is 30. We can use vector, ***y***, as a reference to readjust the value of 5, by comparing 5 to each of the elements in ***y***, in ascending order, until it finds the first value that is greater than 5. 6 is found to be the value to be greater than 5, then 5 is reassigned to the index of *ind*-1 in vector ***y***, which is 3. We only need to change the value 5 because the values 1, 2 and 6 already have common multiples of 6.

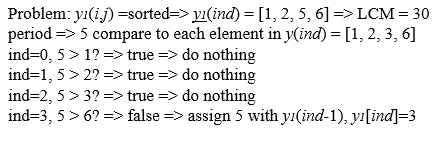
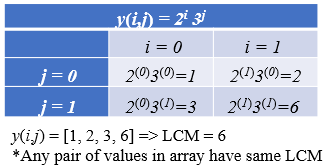


Figure 1: (left) Illustration of (1) (right) Steps of change vector ***y1*** to lower LCM

In order to properly simulate the real time scheduling algorithms, we need to ensure the deadline is always less than or equal to the period, *pi* ≥ *di*, for each process since we are either lowering the period(s) or not changing period(s). Figure 2 illustrate the complete algorithm for generating (1) array and checking each process’ period whether an adjustment is needed. If period needs to be modified, we adjust period according to steps in figure 1 (right) and check whether the adjusted period is greater or equal to the deadline. We check by using a while loop with a condition of adjusted\_period[*i*] < deadline[*i*], if true the current period is reassigned to the first period greater than the first adjusted period, illustrate in figure 2.

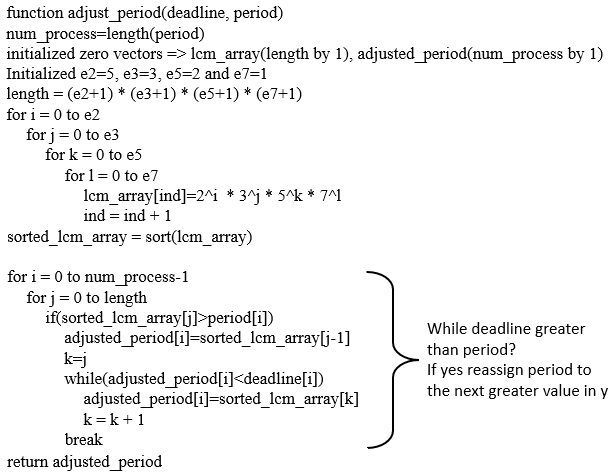


Figure 2: Pseudo Code for Proposed Algorithm

**Computer Simulation and Results**

To simulate our system, we utilized random number generators to obtain our arrival times, execution times, deadlines and periods for each process. The random numbers are constraint with the following rules:

* **Constraint 1**: at least 5 processes
* **Constraint 2:** assume all the processes arrive at different time and assume the arrival interval of the processes is exponential distribution with the mean of 20-time units.
* **Constraint 3:** the deadline of each process should be greater than its execution times, you can get the deadline by adding the process's execution times and a number **a**, and **a** is uniformly distributed on 0 to 500;
* **Constraint 4:** the period of each process should be greater than its relative deadline, you can get the period by adding the process's relative deadline and a number ***b***, and ***b*** is uniformly distributed on 0 to 1000.
* **Constraint 5:** Periods are readjusted based on (1) so that the lengths are as closely match harmonically and guarantee to be less or equal to deadline (algorithm in Figure 2).

Using (1) with the following values, *e2*=5, *e3*=3, *e5*=2 and *e7*=1 we are able to accommodate the periods generated from the specified random number generators. Our results are shown in *Table I* which shows feasibilities, CPU utilization and the run time, which is given by the calculated LCM. For each algorithm, we ran ten trials for each context switch, 0-5. Each sub-table show each of the real time scheduling algorithm, RM scheduling, EDF Scheduling and DM scheduling. Generally, while using these random numbers constraints, we see EDF has the highest feasibility success rate. RM and DM have the highest fail rate maybe due its priority of each process while EDF depends on the next earliest deadline.

The main taken aways are shown in Figure 3. The numbers in each cell are organized by the following: first row number of processes and content switch; every row after the first: process ID, arrival time, execution time, deadline and period. The first row of figure 3 shows numbers generated resulted in all three algorithms are all feasible but have low CPU utilization. One reason maybe is because the differences between some deadline values and period values are small, less than 200. The second row shows all three algorithms are infeasible from the generated numbers. From the observing the numbers that are generated on the second row, the difference between the deadlines and periods are significant, more than 500 on average.

|  |  |  |
| --- | --- | --- |
| CS=0, Trial 3 | Trial 5, CS=0 | Trial 6, CS=0 |
| Trial 10, CS=0 | Trial 5, CS=2, | Trial 1, CS=4 |

Figure 3: Some taken aways from number generated

**Conclusion**

We were able to obtain a bounded LCM using (1) and would prevent stress which would affect the system’s memory failure and overload. However, having a bounded LCM does not guarantee feasibility as proven by our experiments shown in *Table I*. To study how random numbers generators affect each real time scheduling algorithm, we can artificially generate a large number of data point of random numbers, where each observation includes number of processes, context switch, arrival time, execution times, deadlines and periods, to find which set of data points leads to feasibility or infeasibility, 2 classes. We can create a classification problem based on the numbers generated.

*Table I*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Rate Monotonic Scheduling | | | | | | | | | | | | | | | | | | | | |
|  | | O: Feasible | | | | X: Infeasible | | | |  | |  | |  | |  | | |  |  |  | |
| Context Switch  CPU Utilization | | Trial  1 | | Trial  2 | | Trial  3 | | Trial  4 | | Trial  5 | | Trial  6 | | Trial  7 | | Trial  8 | | | Trial  9 | Trial  10 | Success Rate | |
| 0  LCM | | X  .4254  151200 | | X  1.00  151200 | | O  .3695  50400 | | X  .4492  25200 | | O  .4440  75600 | | O  .3202  30240 | | X  1.00  25200 | | X  1.00  50400 | | | X  1.00  30240 | X  1.00  75600 | 3/10 | |
| 1 | | X  1.00  ﻿151200 | | X .3853  25200 | | O  .4750  75600 | | X  .49252  25200 | | X  0.5499  37800 | | O  0.5694  151200 | | O  0.5796  151200 | | O  .4709  151200 | | | X  1.00  50400 | O  .3334  151200 | 5/10 | |
| 2 | | O  0.5109  151200 | | X  1.00  75600 | | X  1.00  151200 | | X  1.00  37800 | | X  1.00  ﻿151200 | | X  1.00  ﻿151200 | | O  .2437  75600 | | X  1.00  ﻿75600 | | | O .3762  16800 | X  1.00  ﻿151200 | 3/10 | |
| 3 | | O  0.6247  75600 | | X  1.00  151200 | | X  1.00  50400 | | O  .4032  151200 | | X  1.00  151200 | | O  .3719  50400 | | X  1.00  30240 | | X  1.00  75600 | | | X  1.00  151200 | X  1.00  151200 | 3/10 | |
| 4 | | X  1.00  18900 | | X  1.00  30240 | | X  1.00  25200 | | O  .4894  30240 | | X  1.00  25200 | | X  1.00  75600 | | X  1.00  30240 | | X  1.00  30240 | | | O  .3663  75600 | X  .6605  75600 | 2/10 | |
| 5 | | X  1.00  151200 | | X  1.00  21600 | | X  1.00  151200 | | X  1.00  25200 | | X  1.00  50400 | | X  1.00  50400 | | X  1.00  151200 | | O  1.00  75600 | | | X  .4252  151200 | O  .4083  50400 | 2/10 | |
|  | Earliest Monotonic Scheduling | | | | | | | | | | | | | | | | | | | | | | |
|  | Trial  1 | | Trial  2 | | Trial  3 | | Trial  4 | | Trial  5 | | Trial  6 | | Trial  7 | | Trial  8 | | | Trial  9 | | Trial  10 | | Success Rate | |
| 0 | O  .3684  151200 | | O  .5785  151200 | | O  .3695  50400 | | O  .4423  25200 | | O  .4440  75600 | | O  .3202  30240 | | O  .5440  25200 | | O  .7111  50400 | | | O  0.4173  30240 | | X  1.00  75600 | | 9/10 | |
| 1 | O  .8853  151200 | | O  .2945  25200 | | O  0.4750  75600 | | O  .4280  25200 | | O  .5049  37800 | | O  0.5694  151200 | | O  .5796  151200 | | O  .4709  151202 | | | O  .4879  50400 | | O  .3333  151202 | | **10/10** | |
| 2 | O  0.5109  151200 | | O  .4778  75600 | | O  0.4269  151200 | | O  .4120  37800 | | X  1.00  ﻿151200 | | O  .4512  ﻿151200 | | O  .2437  75600 | | O  0.5571  75600 | | | O .3762  16800 | | O  .6491  151200 | | 9/10 | |
| 3 | O  0.6248  75600 | | O  .6456  151200 | | O  .3319  50400 | | O  .4032  151200 | | X  0.5666  151200 | | O  .3719  50400 | | X  0.9413  30240 | | O  .4427  75600 | | | O  .4859  151200 | | O  .4633  151200 | | 8/10 | |
| 4 | X  0.4552  18900 | | X  1.00  30240 | | O  .7603  25200 | | O  .4894  30240 | | O  .5266  25200 | | X  1.00  75600 | | O  .4515  30240 | | O  .6673  30240 | | | O  .3663  75600 | | O  .6341  75600 | | 7/10 | |
| 5 | O  .6558  151200 | | O  .3678  21600 | | O  .5886  151200 | | X  1.00  25200 | | X  1.00  50400 | | X  .8866  151200 | | X  1.00  151200 | | O  .5486  75600 | | | O  .4083  151200 | | O  .5761  50400 | | 6/10 | |
|  | Deadline Monotonic Scheduling | | | | | | | | | | | | | | | | | | | | | | |
|  | Trial  1 | | Trial  2 | | Trial  3 | | Trial  4 | | Trial  5 | | Trial  6 | | Trial  7 | | Trial  8 | | Trial  9 | | | Trial  10 | | Success Rate | |
| 0 | O  .3684  151200 | | O  .5785  151200 | | O  .3695  50400 | | X  .4492  25200 | | O  .4440  75600 | | O  .3202  30240 | | X  .8029  25200 | | X  1.00  50400 | | O  0.4173  30240 | | | X  1.00  75600 | | 6/10 | |
| 1 | X  1.00  ﻿151200 | | O  .2945  25200 | | O  .4749  75600 | | O  .4280  25200 | | O  .5049  37800 | | O  0.5694  151200 | | O  .5795  151200 | | O  .4709  151202 | | X  1.00  50400 | | | O  .3334  151202 | | 8/10 | |
| 2 | O  .5109  151202 | | O  .4778  75600 | | O  0.4269  151200 | | O  .4120  37800 | | X  1.00  ﻿151200 | | X  1.00  ﻿151200 | | O  .2437  75600 | | X  1.00  ﻿75600 | | O  .3762  16800 | | | X  .6533  151200 | | 6/10 | |
| 3 | O  0.6249  75600 | | X  1.00  151200 | | O  .3319  50400 | | O  .4032  151200 | | X  .6635  151200 | | O  .3719  50400 | | X  1.00  30240 | | O  .4427  75600 | | X  .4867  151200 | | | O  .46328  151200 | | 6/10 | |
| 4 | X  .5150  18900 | | X  1.00  30240 | | X  1.00  25200 | | O  .4894  30240 | | O  .5266  25200 | | X  1.00  75600 | | O  .4515  30240 | | O  .6672  30240 | | O  .3663  75600 | | | X  .8846  75600 | | 5/10 | |
| 5 | X  1.00  151200 | | O  .3678  21600 | | X  .5886  151200 | | X  1.00  25200 | | X  1.00  50400 | | X  1.00  151200 | | X  1.00  151200 | | X  .5486  75600 | | O  .4083  151200 | | | O  .5761  50400 | | 3/10 | |

**References**

[1] B. Zhao, H. Aydin, and D. Zhu, On Maximizing Reliability of Real-Time Embedded Applications Under Hard Energy Constraint, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL.6, NO. 3, AUGUST2010

[2] T. Y. Yen, W. Wolf, Performance Estimation for Real-Time Distributed Embedded Systems, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 9, NO. 11, NOVEMBER 1998

[3] Y. Shin, K. Choi, Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems, IEEE

[4] S. Knoll et al., "M&A for dynamically generating and maintaining frame-based polling schedules for polling isochronous and asynchronous functions that guaranty latencies and bandwidths to the isochronous functions," Us. Patent No. 5,742,847, Apr. 1998.

[5] J. Goossens, and C. Macq, "Limitation of the hyper-period in real-time periodic task set generation," Proc. RTS Embedded System (RTSOI), pp. 133-147, 2001.

[6] G. Liu, J. Gao, L. Zeng, Y. Hou, C. Cao, L. Tonga, Y. Wang, On-demand ventilation and energy conservation of industrial exhaust systems based on stochastic modeling, Energy and Building, Elsevier

[7] Y. Hou, J. Weng, Q. Gao, Y. Wang, M0 Khokhar, J. Liu, Considering the Patient Satisfaction and Staffing skill the Optimization of Surgical Scheduling by Particle Swarm and Genetic Algorithm

[8] J. Xu, A Method for Adjusting the Periods of Periodic Processes to Reduce the Least Common Multiple of the Period Lengths in Real-Time Embedded Systems

[9] T. F. Abdelzaher, K. S. Shin, Period-based load partitioning and assignment for large real-time applications, IEEE Transactions on Computers, 2000

[10] S Gong, JJ Han, Global emergency-based job-level scheduling for weakly-hard real-time systems, Journal of Systems Architecture, 2021 – Elsevier

[11] X. Jin; A. Saifullah; C. Lu; P. Zeng, Real-Time Scheduling for Event-Triggered and Time-Triggered Flows in Industrial Wireless Sensor-Actuator Networks, IEEE INFOCOM 2019 - IEEE Conference on Computer Communications

[12] S. Niknam, P. Wang, T. Stefanov, Hard Real-Time Scheduling of Streaming Applications Modeled as Cyclic CSDF Graphs, 2019 Design, Automation & Test in Europe Conference & Exhibition

[13] X. JIN, C. XIA, N. GUAN, C. XU, D. LI, Y. YIN AND P. ZENG, Real-Time Scheduling of Massive Data in Time Sensitive Networks with a Limited Number of Schedule Entries, IEEE Access