2011 IFIP Ninth International Conference on Embedded and Ubiquitous Computing

Generating Bounded Task Periods for Experimental Schedulability Analysis

Chaitanya Belwal and Albert M.K. Cheng
Department of Computer Science,
University of Houston, TX, USA
{cbelwal, cheng}@cs.uh.edu

Abstract - **Schedulability analysis models in embedded and real-time systems are experimentally validated using synthetic task sets, which are generated using random or pseudo-random selection algorithms. Validation of these schedulability models generally requires analyzing release of all jobs of tasks within a defined interval called the feasibility interval of the task set. The length of this interval is dependent on the hyper-period, which is the least common multiple of task periods. Hence, the time taken in experimental validations is directly proportional to the value of hyper-period, apart from the number and size of task sets. Currently, if tasks period values with low hyper-period are required, the only way to generate them is using manual or ad-hoc methods. In this paper, we present a structured method of selecting task period values from within a user-specified bounded range such that the hyper-period values of these task sets is minimized. Formula to compute maximum number of task sets of different sizes that can be generated is also derived. Finally, comparisons of hyper-period values generated from a bounded range using random selection and our method are presented.**

Keywords – Schedulability analysis, task generation, bounded task periods, Hyper-period, prime factorization

I. INTRODUCTION

In embedded and real-time systems, several theoretical schedulability algorithms and associated tests (called *schedulability models* in the rest of this paper) are validated using experimental evaluations of synthetically generated task sets. These evaluations are required to confirm the correctness and determinism in temporal properties of these models. A real-time or embedded system can continue execution for an infinite amount of time, and ascertaining temporal properties throughout this infinite period is not possible. For this purpose, real-time researchers use the notion of *feasibility interval*, which is a finite time interval within which if the task set is schedulable, it is guaranteed to be schedulable as long as the real-time system is running. For systems in which job releases are periodic, the length of feasibility interval is dependent on the *hyper-period* of the task set. Hyper-period is the finite time after which the job release pattern of tasks repeats itself. Due to this cyclic repetition of job releases, schedulability of a task set within one hyper-period is sufficient for the analysis of a task set.

The minimum value of hyper-period is the *least common multiple* (*LCM*) of the periods of all tasks present in a task set. For validation of scheduling models, several task sets are randomly or pseudo-randomly generated and analyzed through schedulability tests or simulations. Schedulability tests are classified as *necessary*, *sufficient* or *exact*. If a task set does not satisfy the necessary test it is guaranteed to be unschedulable, and if it satisfies the sufficient test it is guaranteed to be schedulable. However, if a task set satisfies the necessary test it does not guarantee schedulability, or if it fails the sufficient schedulability tests it does not guarantee unschedulability. Sufficient schedulability tests based on utilization bounds for the preemptive execution model are given in [[5],[9]]. Closed form sufficient test for fixed priority rate-monotonic (RM)-scheduling based on task periods is given in [[11]]. For RM-scheduling, polynomial time exact schedulability tests are also available [[12]].

Development of tests for scheduling in the preemptive model has been possible because the worst-case release scenario for this scheduling model can be easily determined. Liu and Layland [[9]] have shown that a task will complete execution in its worst-case response time (WCRT) if the task and all higher priority tasks are released at the same time (*synchronous* release). This is also termed as the *critical instant* of release. However, there are several execution models like transactional memory [[7]], lock-free execution [[1]], P-FRP [[3]] or atomic critical sections of Java [[10]], where there is no critical instant of release. Hence, ascertaining the schedulability of task sets by computing the response time only under critical instant is not possible for such execution models. Determining the WCRT for these models requires analyzing the response time of all the release scenarios of higher priority tasks till the hyper-period. Since the number of release scenarios is dependent on the hyper-period, the time taken to determine the WCRT is dependent on the value of the hyper-period.

Also, for these execution models no polynomial time exact schedulability tests exist. The only way to verify schedulability for a given release scenario is to simulate the execution of tasks in the feasibility interval of the task set. The simulation is performed by iterating for every discrete time unit up till the hyper-period of the task set. This makes the number of iterations to be performed and hence, the time taken to determine schedulability, directly proportional to the value of the hyper-period.

Real-time research on these execution models requires the execution of several task sets to study or verify their temporal properties. A large hyper-period value can cause modeling and analysis of such models an extremely time-consuming exercise. This problem was encountered by us during our research on P-FRP [[3]]. We had to analyze an exponential number of release scenarios that could lead to WCRT, or study several task sets to derive the best static

[1] This work is supported in part by U.S. National Science Foundation under Award no. 0720856

978-0-7695-4552-3/11 $26.00 © 2011 IEEE
DOI 10.1109/EUC.2011.39

multiprocessor partitioning scheme for P-FRP [[4]]. Even with small size task sets (3 to 6 tasks) and randomly generated task periods, some of our analysis took days and even weeks to complete using a machine with Intel dual core processors running Microsoft Windows. When some problem was detected in a method we had to perform the iteration again, and it took several weeks for us to fine tune our results and correct our algorithms. A similar problem was encountered by other real-time researchers. Anderson et al mention ([[1]], Section 4) that to reduce the time during analysis of their lock-free execution model, the task periods were selected from a fixed set of 36 values which were manually generated and were known to have low hyper-period values. For these execution models if results can be obtained quickly, more cases can be evaluated improving the accuracy of the results.

Currently, to the best of our knowledge no efficient method to generate task periods with low hyper-period values is available. Researchers have to generate such periods either manually, use ad-hoc approaches or rely on random selection which is a common method for task set generation in real-time analysis. However, this method is not good when task periods with lower hyper-period values are required. To illustrate the range of hyper-periods that can be generated through a random selection of task periods, we generated 100 task sets, with 3 tasks in each set. These task sets are unique in the sense, that no two task periods in a set are same. Consider a bounded range of [50,80] which has been selected with no particular bias, and is also used as an example for illustrating important concepts in the rest of this paper. *Fig. 1(a)* shows the sorted hyper-periods of the 100 unique task periods selected from this range through a standard random function. The average value of the 100 hyper-periods is ≈134359. *Fig. 1(b)* shows the upper and lower bounds of hyper-periods of 100 tasks that can be possible in a random selection. The average upper bound of 100 random task periods is ≈408996, while the average for the lower bound is ≈1924. It is possible to get all the task periods such that the hyper-periods are in the lower bound. However, the theoretical probability of these task periods being selected is very small as shown below:

From the bounded range [50,80], the total number of unique task sets that can be generated is $\binom{31}{3} = 4495$. The probability of selecting from the 100 task sets with the lowest hyper-periods is: $\frac{100}{4495} \cdot \frac{99}{4494} \cdot \frac{98}{4493} \approx 0.000011$.

Clearly, due to such a small probability, task periods with low hyper-periods cannot be deterministically selected from random selection. One way of generating low hyper-period task sets is to generate all the possible combinations of task periods and sort them based on hyper-period values. However, for large ranges the number of cases could be very high, and including the costs of sorting and mathematical calculations for the hyper-period, this method is computationally expensive.

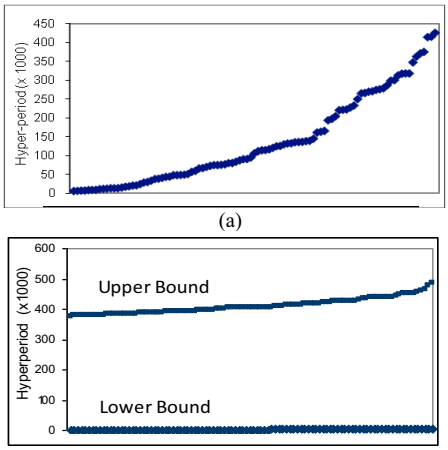


Figure 1. (a) Sorted Hyper-periods of 100 randomly generated 3-task sets (b) Lower and upper bounds of possible Hyper-periods of 100 randomly generated task sets with 3 tasks. The periods in both cases are selected from our example range [50,80].

Another solution is to choose task periods from a numerically low range. However, task sets generated in such a way will be restricted in some way. For example, consider the restriction on execution times when 5-task sets have to be generated from a period range [5,10]. The best scenario is when all tasks have the highest possible period, i.e. 10. Even in this scenario, the execution time of a task cannot be more than 2, as it will exceed the maximum allowable utilization factor of 1 for the task set [[9]]. As the number of task sets increase, the ratio between execution times and arrival periods should be low to keep the total utilization factor within allowed limits. Thus, to avoid these limitations there is a strong requirement for task sets with higher arrival periods.

In this paper, we present a deterministic technique to select task periods with low hyper-period values within a user specified numerical range. These task periods with lower hyper-periods can considerably reduce the time taken in the schedulability analysis of systems which are dependent on the feasibility interval of the task set. Our method places no restriction on the numerical value of task periods and does not require sorting of values.

A. PREVIOUS WORK AND CONTRIBUTIONS

Optimizing the hyper-period value is the focus of Goossens and Macq's paper [[14]]. As mentioned previously, in the past, real-time researchers have also used manually generated task periods with low hyper-periods. However, this technique is ad-hoc and is time-consuming especially when several task sets have to be generated.

Goossens and Macq have elegantly presented the arithmetic properties of prime numbers and the limitations

of random selection of task periods for experimental schedulability analysis. They propose a technique to deterministically generate task periods where the user specifies an upper bound of the hyper-period. This is achieved by selecting an adequate number of prime numbers from which task periods are created by multiplying these prime numbers. The prime numbers and their multiples which are used to generate task periods are randomly selected from a manually generated matrix containing only prime numbers. Goossens and Macq also present an algorithm which can generate complete task sets by random selection of task execution times.

Our work is an extension of their paper, in the sense that we do not bound the Hyper-period but allow the user to bound the task periods. Unlike [[14]], we do not use a manually generated fixed set of primes, but allow any number of primes to be algorithmically generated using the specified lower and upper bounds. Also, we do not use random functions to select task periods, but use combinatorial arithmetic, which makes our task period generation more deterministic.

B. NOTATIONS

The following are the notations used in this paper:

- Task set $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a set of n periodic tasks. Γ_n is also referred to as an n-task set
- Z^+ is the set of all positive integers
- $T_j \in Z^+$ is the **arrival period** between two successive jobs of τ_j
- $\{T_1, T_2 \ldots, T_n\}$ is a period set, where $T_i; i=1,n$ are the arrival periods of tasks $\tau_i; i=1,n$, $\tau_i \in \Gamma_n$
- The term ***n*-period set** denotes a period set with n unique elements
- C_k is the **worst-case execution time** of τ_k
- The **total utilization factor** *(U)* of a task set is the sum of ratios of processing time to arrival periods of every task. Hence, $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$
- $LCM(\Gamma_n)$ represents the **least common multiple** of task arrival periods T_1 through T_n
- $LCM(\{a_1, a_2, \ldots, a_m\})$ represents the least common multiple of the integers $a_1, a_2, \ldots, a_m$
- *PRIMES* represent the set of prime numbers
- $\binom{n}{k}$ represents the number of combination sets of size k that can be generated from a set of size n, where $k \le n$

II. GENERATING TASK PERIODS

Given a range $[t_{lower}, t_{upper}]$: $t_{upper} > t_{lower}$, from which task periods can be selected and the size of task set k we need to generate task periods which minimize the $LCM(\Gamma_n)$ of the task set. We first state some important properties of prime factors and least common multiples.

Property 1. *From the fundamental theory of arithmetic, any positive integer n can be expressed as a multiple of prime factors. Or,*

$$n = \prod_{i=1}^{q} p_i^{n_i} : p_i \in PRIMES;\ n_i,\ q \in Z^+.$$

Property 2. If $m = \prod_{i=1}^{q} p_i^{m_i}$, and $n = \prod_{i=1}^{q} p_i^{n_i} : p_i \in PRIMES;$ $n_i, m_i,\ q \in Z^+$, *the LCM of m and n, LCM({m,n}) can be expressed as:*

$$LCM(\{p,q\}) = \prod_{i=1}^{q} p_i^{\max(m_i, n_i)} .$$

Property 3. *If integers $\{a_1, a_2, a_3, \ldots, a_m\}$ have common divisors $\{d_1, d_2, \ldots, d_p\} \in PRIMES$, then the least common multiple, $LCM(\{a_1, a_2, a_3, \ldots, a_m\})$ cannot exceed*:

$$\frac{\prod_{i=1}^{m} a_i}{\prod_{j=1}^{p} (d_j)^{m-1}} .$$

Proof. Since, $a_1, a_2, a_3, \ldots, a_m$ have common divisors $\{d_1, d_2, \ldots, d_p\}$ we can write a_1 through a_m as:

$$a_1 = \prod_{j=1}^{p} d_j \cdot b_1;\ a_2 = \prod_{j=1}^{p} d_j \cdot b_2;\ \ldots; a_m = \prod_{j=1}^{p} d_j \cdot b_m$$

where, $\{b_1, b_2, \ldots, b_m\} \in$ PRIMES.

Therefore, $\prod_{i=1}^{m} a_i = \prod_{j=1}^{p} (d_j)^m \cdot \prod_{i=1}^{m} b_i$

$$\Rightarrow \frac{\prod_{i=1}^{m} a_i}{\prod_{j=1}^{p} (d_j)^{m-1}} = \prod_{j=1}^{p} d_j \cdot \prod_{i=1}^{m} b_i .$$

Clearly $\prod_{j=1}^{p} d_j \cdot \prod_{i=1}^{m} b_i$ is divisible by every integer in the set $\{\prod_{j=1}^{p} d_j \cdot b_1,\ \prod_{j=1}^{p} d_j \cdot b_2, \ldots,\ \prod_{j=1}^{p} d_j \cdot b_m\}$. Hence, $\prod_{j=1}^{p} d_j \cdot \prod_{i=1}^{m} b_i$ is a multiple of $a_1, a_2, a_3, \ldots, a_m$. Since, both d_j $(j=1,p)$ and $b_i (i=1,m)$ are prime numbers $\prod_{j=1}^{p} d_j \cdot \prod_{i=1}^{m} b_i$ is a common multiple of $\{a_1, a_2, a_3, \ldots, a_m\}$.

$\prod_{j=1}^{p} d_j \cdot \prod_{i=1}^{m} b_i$ is not guaranteed to be the LCM of the set $\{a_1, a_2, a_3, \ldots, a_m\}$ since, LCM could be factorized by the same prime divisor multiple times as per property 2. However, since $\prod_{j=1}^{p} d_j \cdot \prod_{i=1}^{m} b_i$ is the result after dividing by at least 1 common prime divisor, the actual LCM will always be lower than or equal to this number. □

Example: Consider {50,60,75}. The common prime divisor of this set is 5. Hence, as per property 3 the LCM cannot exceed 1800. The computed LCM of {50,60,75} is 300.

From property 1 we can also deduce that when two integers a_1 and a_2 are co-prime, their LCM will be their multiple $a_1 \cdot a_2$. To lower the *LCM,* task periods within a bounded range should be selected in such a way that there is at least one common divisor between each set of task periods. Hence, the basic premise of our method is to select only those task periods which are not co-prime with each other.

The pseudo-code of the method to generate task sets which are not co-prime in given in algorithm 1 (*fig. 2*). This algorithm requires a method for dynamic prime number generationand another for combination generation, both of which have been elaborated in subsequent sections. Algorithm 1 takes as input the range $[t_{lower}, t_{upper}]$ from which task periods have to be selected, the size of the period set k, as well as *count* - the number of period sets that have to be generated. It returns a hash table which contains unique period sets where no two periods are co-prime.

After all the prime numbers are generated, we compute the set of multiples of each prime number in the range $[t_{lower}, t_{upper}]$ (line 5). Since, a numerically higher divisor has greater probability of leading to a lower hyper-period (from property 1) we start from the highest possible prime number (loop in lines 4-12). We use combinatorial arithmetic to select unique subsets of size k (line 6). It could be possible for the same subset to be generated with two different prime divisors, hence we use a hash table to guarantee uniqueness of each period set. This hash table allows only a single entry for each hash code in the table where the hash code could be a simple string representation of task periods. Once the hash table has the required number of task sets, the period generation loop exits. The variable M_x refers to the set of all multiples of d_x which are present in the range $[t_{lower}, t_{upper}]$.

A. TIME COMPLEXITY

The loop in lines 4-14, will run for $|D|$ times. Let us define the variable Δ, which for any given range of task periods $[t_{lower}, t_{upper}]$: $t_{upper} > t_{lower}$, is:

$\Delta = t_{upper} - t_{lower} + 1$.

```
1.  input : [t_lower, t_upper]- Bounded
            range; k -task set size;
            count - number of period
            sets required
2.  output: H - hash table containing
            all period sets
3.  Compute D = set of all prime
                numbers in the range
                [1, t_upper]
4.  loop for each prime divisor d_x in D with
    highest prime divisor first
5.     compute set M_x = all multiples of
                d_x in the range [t_lower, t_upper]
6.      Generate (|M_x| choose k) combinations
7.      loop for each unique combination
         C in (|M_x| choose k)
8.           hash = GenerateHashCode(C)
9.           if(hash not present in  H)
10.              store (C, hash) in H
11.      end loop
12.       if (number of elements in H >
13.             count) exit loop
14. end loop
```

Figure 2. Algorithm 1 to generate low hyper-period period sets

Since, the smallest prime number is 2, number of elements of M_x in each iteration will be bounded by $\frac{\Delta}{2}$. For a task set of size k, the maximum number of combinations that will be generated for any divisor is bounded by:

$$\frac{\frac{\Delta}{2}!}{k!\cdot(\frac{\Delta}{2}-k)!}.$$

Since, this number of combinations will be generated $|D|$ times, the time complexity of algorithm 1 is bounded by:

$$O\left(|D| \cdot \frac{\frac{\Delta}{2}!}{k!\cdot(\frac{\Delta}{2}-k)!}\right).$$

B. GENERATING PRIME DIVISORS

Generation of prime numbers is one of the classical problems of number theory, for which no fixed formula exists. However, there are several standard algorithms available for prime number generation. One of the simplest is the 'Sieve of Eratosthenes' algorithm, which has a known time complexity of $O(n(\log n)(\log\log n))$ [[15]], where n is the maximum number till which primes have to be generated starting from 1.

We have made a slight modification to this algorithm such that only prime numbers greater than or equal to 2 are

generated. For a given range $[t_{lower}, t_{upper}]$, the maximum number till which prime numbers are generated is set to Δ. Prime numbers which are more than this value are guaranteed to be divisors of at the most a single value in the range $[t_{lower}, t_{upper}]$, and hence not useful for our purpose.

C. GENERATING COMBINATIONS

Combinatorial arithmetic is required to generate each of the $\binom{|M_x|}{k}$ unique combinations of size k from multiples of each prime number. The number of unique combinations of size k that can be generated from a set of size n are given by the formula: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Several combinatorial algorithms have been proposed by Knuth [[8]]. While most of these use iterative methods, Knuth also proposes a binomial tree for generating lexographic combinations. We use a modified form of this tree to generate combinations for our use and implement it using a recursive function. For generating all $\binom{n}{k}$ combinations, the root node has n children. Each child node having a value i will have children having values from i+1 to n. A node with value n has no children. Using this rule, the tree is built recursively up till level k.

D. OPTIMAL TASK SET SIZE

In this section, we find the value of task set size k for which maximum number of task sets can be generated. This is called the **optimal** task set size of a bounded range. For a task set of size k, the total number of combinations that can be generated from the prime number multiple set M_x, is expressed by $\binom{|M_x|}{k}$. Keeping $|M_x|$ constant, the value of this expression increases with k up till $\left\lceil \frac{M_x}{2} \right\rceil$, and then it starts decreasing. If we plot the values of $\binom{|M_x|}{k}$ a bell-curve is formed. *Fig. 3* shows all the points when $|M_x|$ is 30 and k varies from 0 to 30. It is clear that the value of $\binom{|M_x|}{k}$ is maximized when $k = \frac{|M_x|}{2}$.

However the size of the set M_x is not constant, and will vary for every prime divisor d_x. When $d_x = 2$, $M_x \le \left\lceil \frac{\Delta}{2} \right\rceil$, for

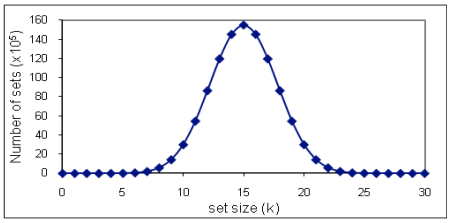


Figure 3. Number of sub-sets of size k that are possible from a set of size 30

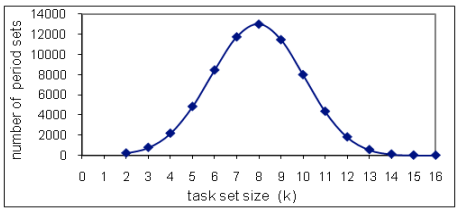


Figure 4. Number of period sets as a function of task set size (k) selected from our example range [50,80] (Δ=31)

$d_x = 3$, $M_x \le \left\lceil \frac{\Delta}{3} \right\rceil$ and so on. For any d_x, $M_x \le \left\lceil \frac{\Delta}{d_x} \right\rceil$.

The value of d_x is also bounded. Assuming a minimum task set size of 2, the prime divisor cannot be more than $\left\lceil \frac{\Delta}{2} \right\rceil$. Or, $d_x \le \left\lceil \frac{\Delta}{2} \right\rceil$. Let d_{max} be this largest prime divisor, or $d_{max} \le \left\lceil \frac{\Delta}{2} \right\rceil$. Hence, if $N_{max}(\Delta, k)$ represents the maximum number of period sets of size k that can be generated for any Δ then:

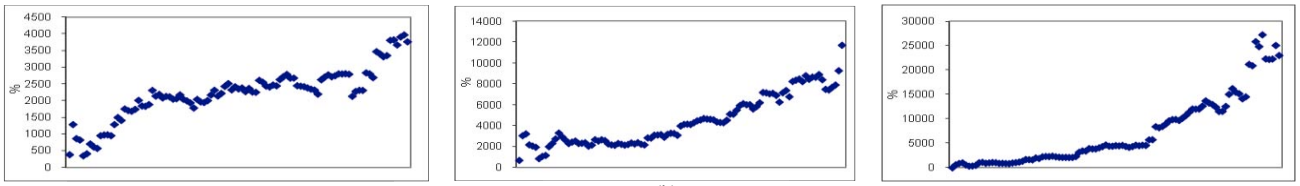
$$N_{max}(\Delta, k) \le \sum_{x=1}^{|D|} \binom{\left\lceil \frac{\Delta}{d_x} \right\rceil}{k}, \text{ where } D = \{2,3,\ldots, d_{max}\}.$$

Note that since $k \le \left\lceil \frac{\Delta}{d_x} \right\rceil$ and $d_x \ge 2$, $\Rightarrow k \le \left\lceil \frac{\Delta}{2} \right\rceil$.

Since $\left\lceil \frac{\Delta}{d_x} \right\rceil$ is variable, we cannot find the value of k which maximizes $N_{max}(\Delta, k)$, for any given Δ directly. Hence, we have plotted the change in $N_{max}(\Delta, k)$ as a function of k with Δ = 31. The results are shown in *fig. 4*. As in *fig. 3*, the curve displays a bell shape.

III. RESULTS

We have implemented algorithm 1 and the supporting algorithms for prime number generation and combinatorial B-tree, in an open release GUI-based C# program running as managed code over Microsoft's .NET framework. Named *PGTool* [[16]], researchers can use this program to generate periods with lower hyper-periods within a given range for task sets of any size. The program outputs the period set, along with its hyper-period. It also has the option of generating period sets using random selection.



(a) (b) (c)

Figure 5. Results showing the % ratio of the difference between sorted hyper-period values generated through a random algorithm vs. those by algorithm 1 for different task set sizes. The period were generated from our example range [50,80]. The % difference is in relation to the hyper-period generated by algorithm 1. (a) 4 - period sets (b) 5 - period sets (c) 6 - period sets

Using this tool we have generated 100 period sets of varying sizes using algorithm 1 and random selection method. We computed the percentage of the difference in hyper-periods relative to the hyper-period generated by algorithm 1. This value called the % ratio is computed for every period set and is stated more explicitly by the following formula:

$$\% \text{ ratio} = \frac{HP_{random} - HP}{HP} * 100, \text{ where}$$

HP : Hyper-period of periods generated through algorithm 1

HP_{random} : Hyper-period of periods generated through random selection

The hyper-periods generated through algorithm 1 and through random selection were first sorted before the difference was calculated. Hence, in computing the difference, the smallest hyper-period generated through algorithm 1 corresponds to the smallest hyper-period generated randomly, for each period set. All task periods are selected from our example range [50,80], and the % ratio's for period sets of various sizes are plotted in graphs shown in *figs. 5.*

It is interesting that with increasing period set size, the % ratio also increases. While with 4-period sets, the ratio reaches a maximum of 4500%, it reaches 25000% for 6-period sets and 120000% for 8-period sets. This shows that when schedulability analysis using simulations in the feasibility interval have to be done, using periods generated through our method will save a significant amount of time. These time savings will be even higher in larger task sets.

IV. CONCLUSION AND FUTURE WORK

We have presented a method that deterministically generates unique task periods with lower hyper-periods. These task periods are selected from a user-specified bounded range. While achieving such lower values of hyper-period through random selection is theoretically possible, it has a very low probability. Generating all possible combinations of periods and sorting them is another possible, though inefficient alternative. Our method uses a structured approach of using prime numbers along with combinatorial arithmetic to use only those values within the range which have common divisors.

Experimental analysis shows that our method generates period sets with considerably lower hyper-period values when compared with random selection. Our method does have some limitations however, with the computation time directly proportional to Δ and the period set size k. Also our method generates a lower count of period sets as compared to random selection. However, using task periods with low hyper-period saves a significant amount of time during the simulation or validation process of scheduling algorithms. Hence, the restrictions of our approach are a trade-off that has to be made for faster schedulability analysis.

REFERENCES

[1] J. H. Anderson, S. Ramamurthy. "A Framework for Implementing Objects and Scheduling Tasks in Lock-Free Real-Time Systems". *RTSS*, 1996

[2] J. Axelsson. "A method for evaluating uncertainties in the early development phases of embedded real-time systems", *RTCSA'05*, 2005

[3] C. Belwal, A.M.K. Cheng. "Determining Actual Response Time in P-FRP", *Practical Aspects of Declarative Languages (PADL)'11*, 2011

[4] C. Belwal, A.M.K. Cheng. "Partitioned Scheduling of P-FRP in Symmetric Homogenous Multiprocessors", *Technical Report number UH-CS-11-01, Dept. of Computer Science, University of Houston*, 2011

[5] E. Bini, G. Buttazzo, G.Buttazzo. "A Hyperbolic Bound for the Rate Monotonic Algorithm". *ECRTS'01*, 2001

[6] E. Bini, G. Buttazzo. "Biasing Effects in Schedulability Measures". *Proceedings of ECRTS'04, pp. 196-203*, 2004

[7] S.F. Fahmy, B. Ravindran, E.D. Jensen. "Response time analysis of software transactional memory-based distributed real-time systems", *ACM SAC Operating Systems,* 2009

[8] Donald E. Knuth, "The Art of Computer Programming", pre-fascicle 4A (a draft of Section 7.2.1.3: Generating all Combinations), Addison-Wesley,61 pages (available online at *http://www-cs-faculty.stanford.edu/~knuth/fasc3a.ps.gz*), 2004

[9] C. L. Liu, L. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *Journal of the ACM (Volume 20 Issue 1), pp. 46 - 61* , 1973

[10] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, J. Vitek. "Preemptible Atomic Regions for Real-Time Java". *RTSS'05*, *pp.62-71*, 2005

[11] W.C. Lu, H.W. Wei, K.J. Lin, "Rate Monotonic SchedulabilityConditions Using Relative Period Ratios,"*RTCSA'06 ,pp. 3-9*, 2006

[12] W.C. Lu, K.J. Lin, H. W. Wei, W.K. Shih. "Efficient Exact Test for Rate-Monotonic Schedulability Using Large Period-Dependent Initial Values". *IEEE Trans. Comput.* 57, 5, pp.648-659, 2008

[13] I.Ripoll,A.Crespo,A.K.Mok. "Improvement in feasibility testing for real-time tasks". *Real-Time Syst. 11, 1, pp. 19-39*, 1996

[14] J. Goossens, C. Macq."Limitation of Hyper-Period in Real-Time Periodic Task Set Generation". *RTS Embedded System (RTS'01), pp. 133-147*, 2001

[15] P. Pritchard. "Linear prime-number sieves: a family tree," *Sci. Comput. Programming 9:1, pp.17–35*,1987

[16] PGTool: *http://www.cs.uh.edu/~cbelwal/PGTool.exe*