Techniques For The Synthesis Of Multiprocessor Tasksets

Paul Emberson
Rapita Systems Ltd
IT Centre, York Science Park, York, YO10 5DG
paule@rapitasystems.com

Roger Stafford
CA, USA

Robert I. Davis
Department of Computer Science
University of York, York, YO10 5DD
robdavis@cs.york.ac.uk

***Abstract*—The selection of task attributes for empirical evaluations of multiprocessor scheduling algorithms and associated schedulability analyses can greatly affect the results of experiments. Taskset generation algorithms should meet three requirements: efficiency, parameter independence, and lack of bias. Satisfying these requirements enables tasksets to be generated in a moderate amount of time, allows effects of specific parameters to be explored without the problem of confounding variables, and ensures fairness in comparisons between different schedulability analysis techniques. For the uniprocessor case, they are met by the UUniFast algorithm but for multiprocessor systems, where the total desired utilisation is greater than one, UUniFast can produce invalid tasksets. This paper outlines an algorithm, Randfixedsum, for the underlying mathematical problem of efficiently generating uniformly distributed random points whose components have constant sum. This algorithm has been available via a MatLab forum for a number of years; however, this is the first time it has been formally published. This algorithm has direct application to multiprocessor taskset generation. The importance of period generation to experimental evaluation of schedulability tests is also covered.**

I. Introduction

To address demands for increasing processor performance, silicon vendors no longer concentrate on increasing processor clock speeds, as this approach is leading to problems with high power consumption and excessive heat dissipation. Instead, there is now an increasing trend towards using multiprocessor platforms for high-end real-time applications. As a result, multiprocessor task allocation and scheduling has become an important and popular area of research.

While optimal algorithms and exact schedulability tests are known for uniprocessor scheduling, multiprocessor scheduling is intrinsically a much more difficult problem due to the simple fact that a task can only use one processor at a time, even when several are free. As a result, no efficient algorithms are known that can optimally schedule general sporadic tasksets (without restrictions on deadlines). Much of the research into multiprocessor scheduling has therefore involved the analysis of heuristic scheduling policies, and the development of sufficient schedulability tests.

A number of different performance metrics can be used to assess the effectiveness of multiprocessor scheduling algorithms and their analyses. These include: optimality, comparability (or dominance) [1], utilisation bounds [2], resource augmentation or speedup factors [3], and empirical measures such as the number of tasksets that are deemed schedulable.

The research in this paper is motivated by empirical approaches to evaluating scheduling algorithm and schedulability test performance. A systematic and scientific study of the effectiveness of different scheduling algorithms and analyses requires a method of synthesising tasksets to which the scheduling algorithms and tests can be applied. We can identify three key requirements of this *taskset generation problem*: efficiency, independence, and bias.

1) *Efficient* — in order to achieve statistically significant sample sizes, large numbers of tasksets need to be generated for each taskset parameter setting (or data point) examined in experiments.
2) *Independent* — it should be possible to vary each property of the taskset independently. For example, experiments might examine the dependency of schedulability test effectiveness on the number of tasks, on taskset utilisation or on the range of task periods. The parameter of interest must be varied independent of other parameters which are held constant.
3) *Unbiased* — the distribution of tasksets generated should be equivalent to selecting tasksets at random from the set of all possible tasksets, and then discarding those that do not match the desired parameter setting.

We assume the sporadic task model commonly used in real-time systems research. A sporadic taskset comprises n tasks with the following attributes: period or minimum inter-arrival time T_i, worst-case execution time C_i and deadline D_i. The utilisation of a task is defined as $U_i = C_i/T_i$. Two important taskset parameters used for understanding the behaviour of scheduling algorithms and their analyses are the taskset cardinality n and the total taskset utilisation u. Hence we are interested in taskset generation algorithms that select utilisation values U_i so that:

$$\sum_{i=1}^{n} U_i = u \tag{1}$$

for n tasks where the target total utilisation is u. Once periods have also been generated, worst-case execution times can then

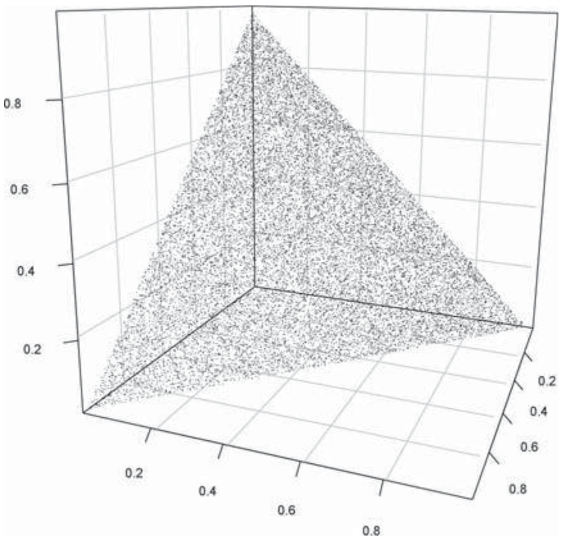

Fig. 1. $2 \cdot 10^4$ tasksets generated by UUniFast with total utilisation 0.98

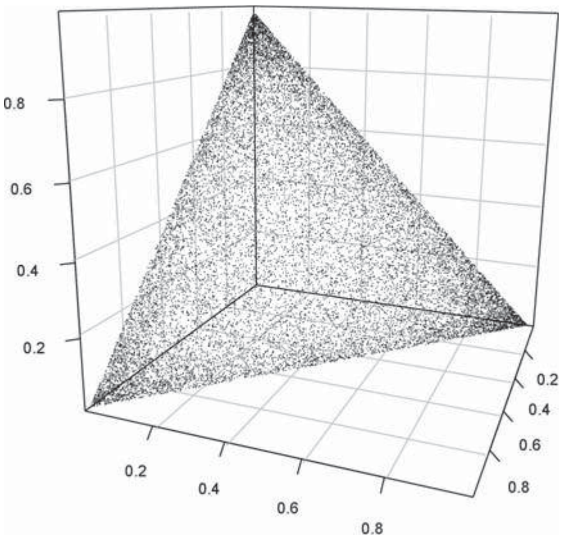

Fig. 2. Subset of 10^5 tasksets which can be scheduled by rate monotonic fixed priority scheduling (approx $1.8 \cdot 10^4$ tasksets)

be set with the formula:

$$C_i = U_i T_i \tag{2}$$

Task deadlines must also be selected. These can be set equal to T_i or randomly generated based on a proportion of the task's period or execution time. This paper focuses mainly on selecting utilisation values though a method of task period generation is given in section III.

II. Related Work

A. Uniprocessor Taskset Generation

In 2005, Bini and Buttazzo [4] created an algorithm called UUniFast that efficiently generates task utilisation values for tasksets with a chosen number of tasks and total utilisation. The distribution of utilisation values in tasksets generated by UUniFast are equivalent to uniformly sampling each task utilisation value and then only keeping those tasksets with the correct total utilisation. A taskset containing n tasks can be plotted in an n dimensional space where the utilisation of each task gives the distance from the origin in each dimension. If this is done for a set of tasksets all having the same total utilisation, then the tasksets will lie in an $n-1$ dimensional plane. Tasksets generated by the UUniFast algorithm will be evenly separated in this plane. Figure 1 shows 20000 tasksets containing 3 tasks generated by UUniFast all having a total utilisation of 0.98.

Bini and Buttazzo considered experiments which evaluated how many tasksets could be scheduled using rate monotonic fixed priority scheduling versus other scheduling policies. They noted that, if periods are uniformly sampled, tasksets are more often schedulable when the difference between the greatest and least task utilisation is large. This phenomenon is shown by figure 2. The plot shows the subset of 10^5 tasksets, again generated by UUniFast with a utilisation of 0.98, deemed schedulable using rate monotonic fixed priority scheduling with periods uniformly sampled between 10 and 10^4. 17953 tasksets are contained within this subset, i.e. a similar number of tasksets as shown in figure 1. The points in figure 2 appear more densely packed towards the edges of the plane whereas those in figure 1 are evenly distributed.

The motivation for Bini and Buttazzo's work was that previous evaluations of scheduling policies, such as by Lehoczky et al. [5], had biased results by concentrating on the area in the centre of the plane shown in figure 2 where fewer tasksets can be scheduled by rate monotonic fixed priority scheduling.

The UUniFast algorithm is efficient, allows variable independence, and generates unbiased utilisation values. UUniFast has been widely used by researchers interested in investigating the performance of scheduling algorithms and schedulability tests for single processors [6], [7].

B. Multiprocessor Taskset Generation

In the multiprocessor domain, the UUniFast algorithm has not been widely used. Researchers recognised that the algorithm cannot generate tasksets with total utilisation $u > 1$ without the possibility that some tasks will have individual utilisations that are invalid (i.e. > 1). Instead, many researchers [8], [9], [10], [11] have used an approach to taskset generation based on randomly generating an initial taskset of cardinality $|\mathcal{P}| + 1$ for the set of processors $\mathcal{P}$ and then repeatedly adding tasks to it until the total utilisation exceeds the available processing resource. This approach has the disadvantage that it confounds two variables, utilisation and taskset cardinality, and does not necessarily result in an unbiased distribution of utilisation values.

Recently, Davis and Burns [12] observed that UUniFast can be used in the multiprocessor domain, at least for some values of n and u, provided that tasksets containing invalid tasks are simply discarded. We give more details of this modified UUniFast algorithm, referred to as UUniFast-Discard in section IV-B. While UUniFast-Discard addresses a proportion of the parameter space of n and u, there are values of n and u where this approach becomes infeasible, due to the very high ratio of invalid to valid tasksets produced.

This paper addresses the problem of generating tasksets for multiprocessor systems. Stafford's *Randfixedsum* algorithm [13] is used to generate unbiased sets of utilisation values for any values of n and u.

The remainder of the paper is broken into two main sections. Section III discusses the associated issue of task period selection. Section IV explains why existing algorithms for generating tasksets with total utilisation greater than 1 are inadequate and suggests the use of the Randfixedsum algorithm. Section V concludes with a summary of the main contributions of the paper.

III. Task Period Selection

In this section, we discuss task period selection. In commercial real-time systems, it is common for systems to have tasks operating in different time bands [14] (e.g. 1ms – 10ms, 10ms – 100ms, 100ms – 1s). For example, a temperature sensor will likely sample at a lower rate than a rotation speed sensor [15].

Davis et al. [6] showed that schedulability test efficiency can be heavily dependent on the number of order of magnitude ranges of task periods (effectively the ratio between the smallest and largest task period), and that bias can result if studies do not fully explore appropriate distributions of task periods. For example, choosing task periods at random according to a uniform distribution in the range $[1, 10^6]$ results in 99% of tasks having periods greater than 10^4, thus the effective ratio of maximum to minimum task period is far less than might be expected (closer to 10^2 than 10^6 for small tasksets).

To avoid these problems, a log-uniform distribution of task periods can be used, with tasksets generated for different ratios of the minimum (T_{min}) to the maximum (T_{max}) task period. The parameter T_g defines the granularity of the periods chosen (which are all multiples of T_g).

$$r_i \sim U(\log T_{min}, \log(T_{max} + T_g)) \tag{3}$$

$$T_i = \left\lfloor \frac{\exp(r_i)}{T_g} \right\rfloor T_g \tag{4}$$

The uniform random values r_i produced are assumed to lie in the range $[\log T_{min}, \log(T_{max} + T_g))$. T_{min} and T_{max} should be chosen as multiples of T_g.

Note that when applying equation (2) the worst-case execution time is usually rounded to the nearest integer which will affect the distribution of actual utilisations in the generated taskset. Changing the unit of time to use larger numeric values will decrease this loss of accuracy.

The effects of different period sampling algorithms were examined with some simple experiments. In each case 1000 tasksets were generated and tested using exact schedulability analysis for fixed priority pre-emptive scheduling on a uniprocessor [16]. Periods were sampled from either a uniform or log-uniform distribution within a certain range. The correct total utilisation for a chosen number of tasks was achieved

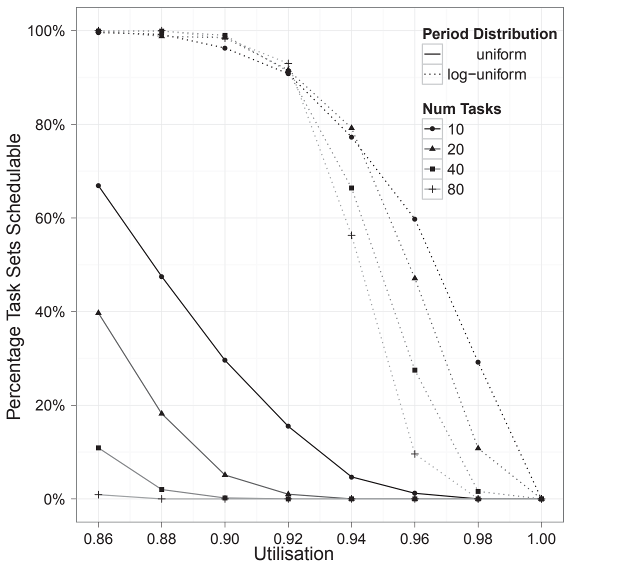

Fig. 3. Comparison of taskset schedulability for different size uniprocessor tasksets generated with uniform and log-uniform period distributions in the range $[10, 10000]$.

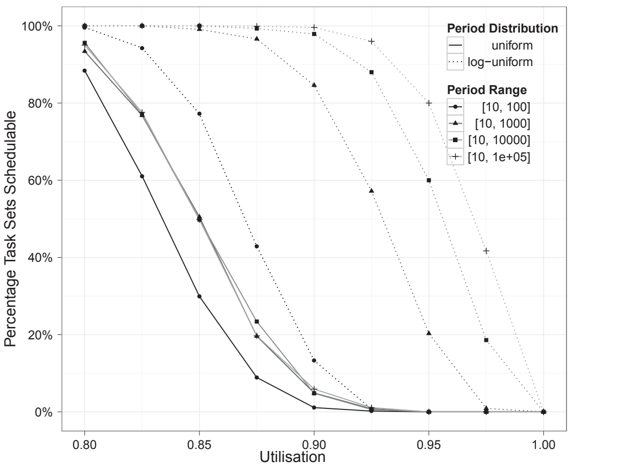

Fig. 4. The effect of changing the range of periods on taskset schedulability for tasksets of size 20 generated with uniform and log-uniform period distributions.

with the UUniFast algorithm. Taskset deadlines were set equal to their periods and priorities assigned according to rate monotonic priority ordering.

Figure 3 shows the proportion of schedulable tasksets for varying taskset cardinality and total utilisation. The period range was set to $[10, 1000]$ for all experiments. The plot shows that many more tasksets are schedulable when taskset periods are sampled from a log-uniform distribution for all utilisation levels up to 0.98. The difference in the number of schedulable tasksets is also much smaller at lower utilisation values over the different taskset sizes when using log-uniform sampling.

Lehoczky [5] calculated that tasksets with a greater range of periods would be easier to schedule using exact rate monotonic analysis. Lehoczky assumed periods were sampled from a uniform distribution. This is supported by the results shown in figure 4. The graph shows the proportion of schedulable tasksets for different period ranges and total utilisations. All tasksets were of size 20. These results show the phenomenon described by Davis [6] that, when a uniform distribution of periods is used, the number of schedulable tasksets does not continue to increase for large period ranges because nearly all period values will be of the same magnitude. The period range has a much larger effect on schedulability of periods sampled from a log-uniform distribution. In fact, for uniform period sampling, there is no significant increase in schedulability as the range widens to more than a factor of 100 and many results overlap on the graph. Even a range whose maximum is only 10 times greater than its minimum produces more schedulable tasksets with a log-uniform period distribution than ranges over 4 orders of magnitude when uniform sampling is used.

IV. Task Worst Case Execution Time Generation

Rather than generate worst case execution time (WCET) values directly, it is more common to generate task utilisation values then calculate WCET values from equation (2). This is done since the total taskset utilisation is an often used covariate in experiments with schedulability tests. This total taskset utilisation value is written as u in this paper. The other common covariate is taskset cardinality and this should be possible to control independently from taskset utilisation.

A. UUniFast

Motivated by the need for an unbiased distribution of tasksets, Bini and Buttazzo [4] decided that task utilisation values should be sampled from a uniform distribution but with the constraint that they summed to a constant desired total taskset utilisation. An algorithm for doing this is to randomly select utilisation values $x_1, \ldots, x_{n-1} \sim U(0,1)$ and then set $x_n = 1 - \sum_{i=1}^{n-1} x_i$. However, if the sum term is greater than 1, the set must be discarded and the operation repeated. If successful, utilisation values are set according to $U_i = ux_i$. Bini and Buttazzo call this algorithm UUniform and explain that it is infeasible in practice since the probability that the sum of the first $n-1$ values is less than u is $1/(n-1)!$ [4].

The UUniFast algorithm [4] is an efficient equivalent of the above algorithm. The principle of the algorithm is to first sample a value which represents the sum of $n-1$ task utilisation values and then set a task utilisation value to the difference between the required total and this sampled value. This is then repeated for each task with the sampled value in the previous iteration acting as the required total.

The probability density function for the sum of m independent random variables uniformly selected from $[0,1]$ is

$$UniSumPdf(x;m) = \frac{1}{(m-1)!} \sum_{k=0}^{\lfloor x \rfloor} (-1)^k \binom{m}{k} (x-k)^{m-1} \tag{5}$$

We refer to this distribution as the UniSum distribution. It is adapted from Hall's derivation for the density of the mean of m independent uniform random variables [17]. It is a piecewise function where each region $[a, a+1]$ for $a = 0, \ldots, m-1$ is defined by a different polynomial of degree $m-1$. Therefore, if we wish to sample a value which represents the sum of $n-1$ utilisation values as required for UUniFast, m is set to $n-1$. The graph of this probability density function is shown in figure 5. The domain which must be sampled from for UUniFast is $[\max(u-1,0), u]$. For uniprocessor tasksets, $u \leq 1$. The relevant area of the graph is highlighted in figure 5. The cumulative distribution function in this region is proportional to $x^m = x^{n-1}$ and is easily invertible. UUniFast makes use of this fact to perform inverse transform sampling in order to obtain values for the sum of $n-1$ values. The UUniFast algorithm is given below.

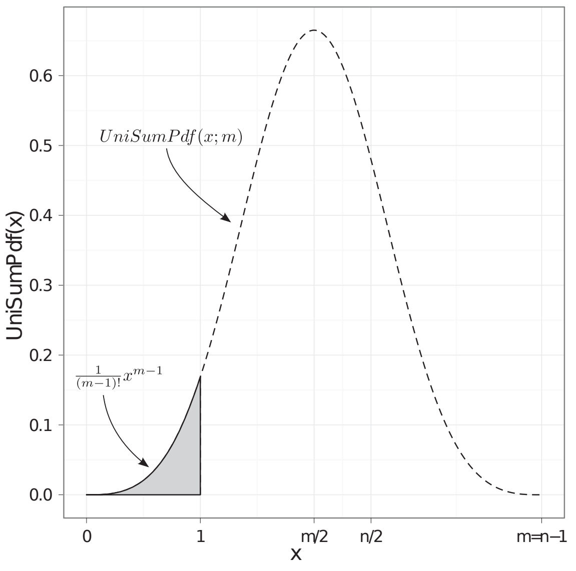

Fig. 5. UniSum probability density function which could be used to sample the sum of $n-1$ values.

```
Let $r_1, \ldots, r_{n-1} \sim U(0,1)$
$s_n = u$
$s_{i-1} = s_i * {r_{i-1}}^{1/(i-1)}$ for $i = n, \ldots, 2$ and $s_0 = 0$.
$u_i = s_i - s_{i-1}$
```

There are a few points of note regarding extending UUniFast for total taskset utilisation values $u > 1$. The distribution given by equation (5) is symmetrical about $(n-1)/2$. If an algorithm can sample values for $0 \leq u \leq n/2$ then sampling values for a total utilisation $u' > n/2$ can be obtained by sampling values with $u = n - u'$ and then using $u'_i = 1 - u_i$ for each task utilisation value.

The complex piecewise nature of the UniSum distribution makes it difficult to sample from in the general case. The sum of n independent random variables will approach a normal distribution but the accuracy of the approximation is heavily dependent on the number of tasks and region of the distribution being sampled from. Saddlepoint approximations [18], [19] are more accurate. However, in either case, sampling from a

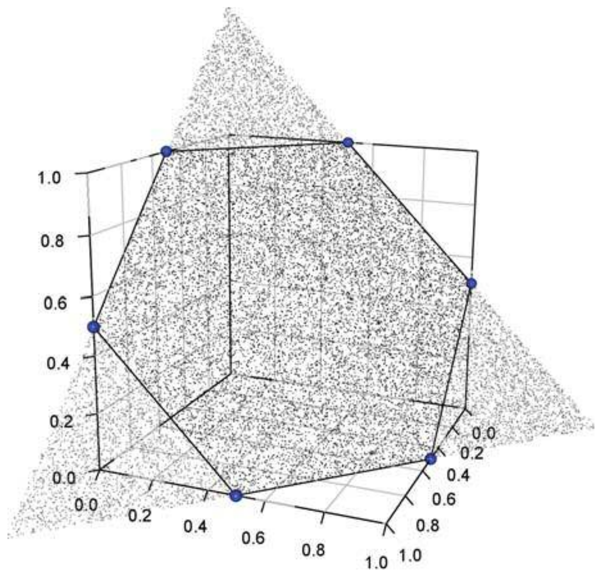

Fig. 6. Tasksets generated with UUniFast-Discard for $n = 3, u = 1.5$

truncated section of the distribution is difficult to do efficiently since it usually requires rejecting a number of samples as well as calculating the distribution itself.

B. UUniFast-Discard

UUniFast-Discard is a simple extension to UUniFast suggested by Davis and Burns [12]. This algorithm applies UUniFast unchanged for values of $u > 1$ and then discards any tasksets which contain an individual task utilisation greater than 1. The issue with this algorithm is that it becomes increasingly inefficient as the value of u approaches $n/2$. Figure 6 shows this effect for $n = 3$ and $u = 1.5$. The valid tasksets lie inside the marked hexagon but the area of the plane within which tasksets are generated is 50% larger than this meaning $1/3$ of tasksets will be discarded in this case. The algorithm becomes extremely inefficient for large values of n with values of u close to $n/2$.

Davis and Burns [12] used a pragmatic discard limit of 1000 to avoid UUnifast-Discard making intractable attempts to find valid tasksets. This limit restricts the maximum number of attempts at taskset generation to 1000 times the number of tasksets required.

C. Randfixedsum Algorithm

Stafford's Randfixedsum [13] was designed to efficiently generate a set of vectors which are evenly distributed in $n - 1$ dimensional space and whose components sum to a constant value. The key to its efficiency is that it does not require any random samples to be rejected. It can be applied directly to the problem of task utilisation generation with a chosen constant total taskset utilisation. This algorithm was made public with an open source Matlab implementation accompanied by a document explaining the theory behind the algorithm. However, it has not been formally published before.

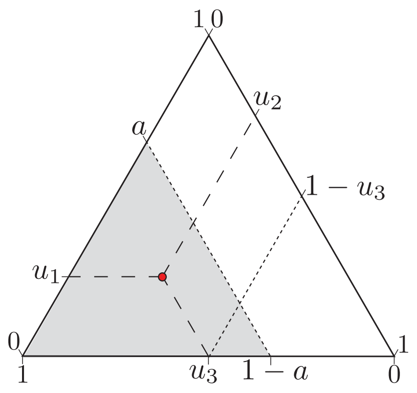

Fig. 7. Triangular axes for plotting triplets with sum 1

To explain how the Randfixedsum algorithm works, we will first turn to the case of $n = 3$ and $u = 1$. All valid tasksets with cardinality 3 and total utilisation 1 can be plotted on triangular axes, each of length 1. Such a set of axes is shown in figure 7. As noted by Stafford [13], if the points in the triangle are evenly distributed then the number of points inside any area within the triangle will be proportional to that area. A smaller triangle can be created as shown in figure 7 by drawing a line between $(a, 0, 1 - a)$ and $(0, a, 1 - a)$. The area of the large triangle is $\frac{\sqrt{3}}{4}$ and the area of the smaller shaded triangle is $\frac{\sqrt{3}}{4}a^2$. For the correct proportion of points to lie inside the shaded triangle compared to the whole, the probability of a point being inside the shaded triangle should be a^2. This is equivalent to requiring $P(u_3 > (1 - a)) = P((1 - u_3) < a) = a^2$. This can be done by selecting a uniform random value r_2 and then setting $u_3 = 1 - r_2^{1/2}$ as is done in UUniFast. Following this, a value u_2 is selected between 0 and $1 - u_3$ along a line. Any segment of this line should contain a number of points proportional to its length. This is done in UUniFast by setting $u_2 = (1 - u_3) - r_1$ where r_1 is another uniform random value.

Extending the concept above to several dimensions, it can be seen that UUniFast will evenly distribute points inside an $n - 1$ dimensional simplex. Stafford's algorithm divides up the valid region of points into multiple $n - 1$ dimensional simplexes and then applies an algorithm similar to UUniFast to select points within a randomly chosen simplex. By making the probability of selecting each simplex proportional to its volume, points are evenly distributed throughout the entire valid region. The remainder of this section describes how the simplexes are generated.

To divide the valid region into simplexes, the centre point at $(u/n, u/n, \ldots, u/n)$ is chosen. From here, we select a point by moving to 1 or 0 in one of the dimensions and then move to the centre of the boundary that was hit. For example the point $(0, u/(n - 1), \ldots, u/(n - 1))$ or the point $(1, (u - 1)/(n - 1), \ldots, (u-1)/(n-1))$. This is done repeatedly until we reach a point where the sum of 0s and 1s is exactly $k = \lfloor u \rfloor$. At this stage, if another 0 or 1 is selected, then the only way to maintain the constant sum is to pick a point outside the valid

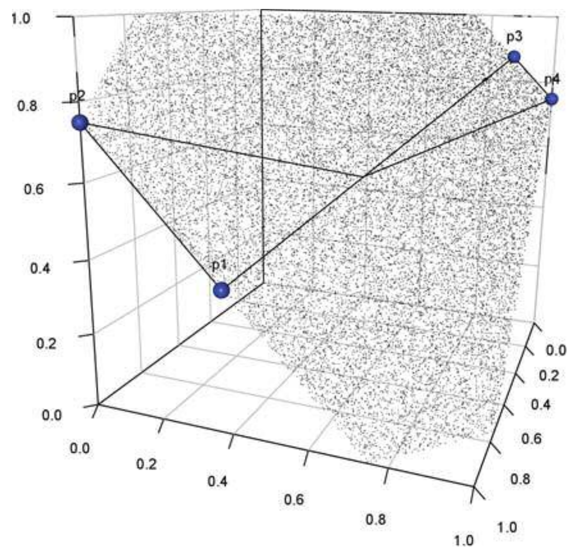

Fig. 8. Two types of simplex generated by Stafford's algorithm for $n = 3$, $u = 1.75$

region. The 1s can be selected for any k of $n-1$ dimensions and the sequence of points (including the initial centre point) used to construct the simplex can be ordered in $n!$ ways. This creates $\binom{n-1}{k}n!$ different simplexes.

Figure 8 shows the $\binom{2}{1} = 2$ types of simplex for $n = 3$ and $u = 1.75$. $3! = 6$ of each type of simplex are needed to cover the entire valid region. Stafford's algorithm calculates the hypervolume of each type of simplex and uses this for its probability of selection. Points are then evenly distributed inside each simplex. The final stage of Stafford's algorithm is to randomly permute the order of dimensions within each point to get coverage of the whole valid region. Stafford's algorithm is available online [13] written in the Matlab language. We aim to implement the algorithm within a taskset generation tool which will be made publicly available.

V. Conclusion

The research described in this paper was motivated by the need for taskset generation algorithms to support the study of scheduling algorithm and schedulability test effectiveness for multiprocessor real-time systems.

The main contributions of the paper are as follows:

- Investigation of how sampling periods from uniform and log-uniform distributions affects the schedulability of tasksets running on a single processor using fixed priority scheduling.
- The application of Stafford's Randfixedsum algorithm to the selection of task utilisation values for tasksets with a total utilisation greater than 1. This algorithm generates an unbiased distribution of task utilisation values, and is capable of doing so for any valid values of taskset utilisation and taskset cardinality.

If the experimental region of interest is where the total taskset utilisation is either very small or large compared to the taskset cardinality then UUniFast-Discard is efficient is simple to implement. As the taskset utilisation approaches $n/2$ from either above or below, the algorithm is much less efficient and impractical for larger tasksets.

The existing Matlab implementation of Randfixedsum is highly efficient in all regions of the parameter space. We therefore recommend its use in multiprocessor taskset generation. We aim to make implementations of this algorithm in other languages available shortly.

Acknowledgments

The authors would like to thank Enrico Bini for his contribution to discussions about the applicability of the UUniFast algorithm to the multiprocessor case. The research was funded in part by the EPSRC Tempo project.

References

[1] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*. Chapman & Hall/CRC, 2004, ch. 30.

[2] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001, pp. 193–202.

[3] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995, pp. 214–221.

[4] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, May 2005.

[5] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of the 10th Real Time Systems Symposium (RTSS 1989)*, 1989, pp. 166–171.

[6] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1261–1276, 2008.

[7] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.

[8] M. Bertogna, "Real-time scheduling for multiprocessor platforms," Ph.D. dissertation, Scuola Superiore SantAnna, Pisa, 2007.

[9] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*. IEEE Computer Society, 2007, pp. 149–160.

[10] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, 2009.

[11] T. Baker, M. Cirinei, and M. Bertogna, "Edzl scheduling analysis," *Real-Time Systems*, vol. 40, no. 3, pp. 264–289, December 2008.

[12] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, December 2009, pp. 398–409.

[13] R. Stafford. (2006) Random vectors with fixed sum. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/9700

[14] A. Burns and G. Baxter, "Time bands in systems structure," in *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Springer London, 2006, pp. 74–88.

[15] J. W. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[16] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

[17] P. Hall, "The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable," *Biometrika*, vol. 19, no. 3/4, pp. 240–245, 1927. [Online]. Available: http://dx.doi.org/10.2307/2331961

[18] H. E. Daniels, "Saddlepoint approximations in statistics," *The Annals of Mathematical Statistics*, vol. 25, no. 4, pp. 631–650, December 1954.

[19] J. L. Jensen, *Saddlepoint Approximations*. Oxford University Press, May 1995.