On the Evaluation of Schedulability Tests for Real-Time Scheduling Algorithms

Robert I. Davis[1,2]
[1]*Real-Time Systems Research Group, Department of Computer Science, University of York, UK.*
[2]*INRIA, Paris, France.*

Abstract

This short paper discusses the criteria and methods that can be used to evaluate the performance of schedulability tests for real-time scheduling algorithms. We summarize the different theoretical and empirical methods that can be used and outline their advantages and disadvantages. The main focus of the paper is on empirical techniques. Here we set out some of the potential pitfalls, and describe a de-facto standard approach based on visualizing results using success ratio and weighted schedulability plots. We discuss how these can be augmented using other graphs such as difference plots and frequency distributions for breakdown utilization. For more complex task models, we note that a consistent set of parameters can be obtained from benchmarks, and we show how a moderate number of benchmarks can be used to produce a large number of related task sets with a variety of utilization levels suitable for use in empirical evaluation. Finally, we remark on the dearth of real-time benchmarks, and call for more benchmarks or benchmark generators to be developed in conjunction with industry.

This paper accompanies an invited talk given at the WATERS workshop in 2016.

1. Introduction

The performance of schedulability tests for real-time scheduling algorithms can be compared in a number of different ways. These can be broadly classified into two categories:

- *Theoretical methods* such as deriving dominance relationships, utilisation bounds [21], or resource augmentation and speedup factors results [18]. These approaches typically give a worst-case comparison against a specific competitor, i.e. an alternative schedulability test for the same or a different scheduling algorithm.
- *Empirical methods* involve evaluating schedulability tests on a large number of task sets of different utilisation levels. These approaches typically facilitate an average-case comparison against a number of different competitors.

The main focus of this paper is on empirical methods for comparing the performance of different schedulability tests. In the following, we sometimes discuss comparisons between scheduling algorithms, by this we mean between exact schedulability tests for those algorithms. Before covering the various empirical methods in detail, we first summarise the main theoretical and empirical methods and discuss their advantages and disadvantages.

1.1. Theoretical methods

Dominance Relationships: show that one schedulability test always outperforms another. For example, schedulability test A is said to *dominate* schedulability test B if every task set that is schedulable according to test B is also schedulable according to test A, and there are some task sets that are schedulable according to test A but not according to B. If two schedulability tests deem precisely the same sets of tasks as schedulable, then the tests are said to be *equivalent*. If there are some task sets that are schedulable according to test A and not according to test B, and vice versa, then the tests are said to be *incomparable*.

Proving dominance relationships has the following obvious advantage: the dominant method is shown to always be better, examples include exact versus sufficient schedulability tests, and EDF v. fixed priority scheduling on a single processor [21]. Disadvantages are that the dominance relationship typically only holds for a simplified model, for example EDF dominates fixed priority scheduling only for simple models where we neglect to include scheduling overheads or cache related pre-emption delays [22]. Further, dominance relationships give no indication how good the schedulability tests (or algorithms) actually are; a dominant test may still have poor performance, just not quite as poor as that of the one it dominates!

Utilisation Bounds [21] provide a simple way of comparing different scheduling algorithms. The bound for a given scheduling algorithm is the largest utilisation value such that all task sets with utilisation no greater than that value are guaranteed to be schedulable (according to an exact test). Examples include the famous Liu and Layland bounds for EDF (1.0) and fixed priority ($\ln(2) \approx 0.69$) scheduling on a single processor. The main advantage of a utilisation bound is that it illustrates the worst-case behaviour for any implicit-deadline[1] task set, and so can be used as a simple, linear-time schedulability test. The disadvantages are that the bound only applies to a simple system model (i.e. implicit-deadlines, no overheads etc.), and the worst-case behaviour may only exist for specific corner cases that are of little interest in practice. (The average case *breakdown utilisation* for implicit deadline task sets is approx. 0.88 [20] or higher when biases in task set generation are avoided [6]. This is substantially above

[1] In an implicit-deadline task set all tasks have deadlines equal to their periods.

the worst-case value of 0.69. Thus most task sets seen in practice are schedulable with utilisation much higher than the bound).

Speedup Factors [18]: indicate the factor by which the speed of a system would need to increase so that any task set that was schedulable under algorithm A is guaranteed to become schedulable under algorithm B. The advantage of deriving speedup factors is that they illustrate the worst-case performance that one scheduling algorithm can have relative to another. For example, the speedup factor for fixed priority pre-emptive scheduling versus EDF is $1/\Omega \approx 1.76$ for constrained-deadline task sets on a single processor [11]. Speedup factors can also be used to explore sub-optimality with respect to an optimal algorithm, for example comparing non-pre-emptive algorithms against pre-emptive EDF [13]. The disadvantages of using speedup factors as a metric are that the worst-case behaviour may exist only for corner cases that are of little interest in practice. For example the corner cases that result in a speedup factor of 2 for fixed priority pre-emptive v. EDF scheduling of arbitrary-deadline task sets require that some tasks have an infinitesimally small period, while others have an infinite period [12]. For most task sets, the speedup factor is less than 1.1. Finally, care is needed in interpreting speedup factor results used to discriminate between different schedulability tests. For example, surprisingly the speedup factors comparing fixed priority versus EDF scheduling remain unchanged when moving from an exact test for fixed priority scheduling to a simple linear time test [25]. This does not, however, imply that the speedup factor comparing a linear-time test for fixed priority scheduling to an exact test for the same is 1. Trivially we know this is not the case, since there are task sets that are schedulable according to an exact test that are not schedulable according to the linear test.

1.2. Empirical methods

Simulations or *scenario based assessments* simulate the execution of a task set over a long time period, and are typically repeated for multiple task sets. Such simulations are useful as a way of exploring average-case behaviour. They also form a *necessary* schedulability test, in the sense that if a task misses a deadline during the simulation, then the task set can be declared unschedulable. The absence of any deadline misses does not in general prove schedulability; however, in some circumstances it can, for example with periodic task sets if the simulated interval is sufficient to ensure that the schedule repeats [15] and the scheduling algorithm is sustainable with respect to task execution times [3]. The disadvantages of simulation are that there is typically no guarantee that worst-case behaviours will be observed unless the worst-case scenario (pattern of arrivals) is known. Further, the worst-case scenario may be very different for different scheduling algorithms. Thus comparisons based on one type of scenario e.g. synchronous release may bias the results in favour of one algorithm over another [24].

Real experiments: involve running real-code, or in some cases synthetically generated task code, on *real* hardware. Such experiments have similar advantages to simulation (exploring average-case behaviour, and acting as a necessary schedulability test). However, they also have the advantage that they include all of the actual overheads incurred, and can also be used to collect overhead measurements that can later be used in simulation or schedulability tests that have been extended to account for such overheads [7]. The disadvantages of such an experimental approach are that there are no guarantees that the worst-case behaviour has been observed, unless the worst-case scenario is known. (Determining the worst-case scenario may be complicated by the presence of overheads). Also, setting up an experiment on real hardware is typically more time consuming than using simulation, and may be difficult to reproduce precisely if the initial hardware state cannot be completely controlled.

Case studies: one or more example task sets are taken from industrial applications. Typically, the case study provides specific parameter values (e.g. periods, execution times, for tasks), and in some cases may provide code from which other parameter values can be derived. For example, code from the Mälardalen benchmark suite [16] can be used to obtain not only Worst-Case Execution Times (WCETs), but also traces of address accesses and hence a characterisation of memory demand and cache usage [1]. These parameters can then be used in schedulability analysis which accounts for memory bus load and cache related pre-emption delays, as well as processor usage. The advantages of using information from case studies include, certainty that the parameter values used are realistic (at least for one application area), and the ability to obtain consistent parameter values for each task. Disadvantages include potentially very limited coverage of the parameter space e.g. using just one example may hide issues elsewhere. More generally, there can be questions as to whether the case study is really representative; i.e. is it similar to applications from other industries? We note that limited coverage of the parameter space can in some cases be mitigated by distilling information from representative case studies and using it to creating multiple similar systems or configurations. This is done by NETCARBENCH to create sets of messages for research into Controller Area Network (CAN) [8], and has been suggested in the context of the AMALTHEA project, as a way of providing automotive benchmarks for free [19].

Empirical evaluation: involves using large numbers of synthetically generated task sets to evaluate the performance of schedulability tests. The advantages of this approach are that, if properly designed, it can give good coverage of the parameter space, and thus provide a fair and unbiased comparison. Care is however needed to achieve this. The disadvantages include uncertainty as to whether the values covered are representative of real systems, and the consideration of overheads, which is often neglected.

The remainder of this paper focuses on systematic methods for the *empirical evaluation* of schedulability tests. In the next section, we discuss key aspects of empirical evaluation. In Section 3, we briefly recall the sporadic task model, before describing a framework (Section 4) for empirical evaluation. In Section 5 we discuss the different types of experiments that can be used to evaluate performance and the corresponding graphs that can be used to visualise the results.

2. Key Aspects of Empirical Evaluation

The empirical evaluation of schedulability tests relies on generating a large number of task sets with parameters chosen from some appropriate distributions. The performance of different schedulability tests is then compared by determining task set schedulability according to each test and providing graphs that enable these results to be interpreted. This may be done in a simple way by plotting a graph of the *success ratio* (i.e. the proportion of task sets that are deemed schedulable by each test) at different utilisation levels, or by using more sophisticated approaches such as *weighted schedulability* metrics [4].

In the empirical evaluation of schedulability tests, the following aspects are important:

1. S*ystematic approach*: It is important to ensure adequate coverage of the full range of realistic parameter settings, with appropriate default values used, typically in the middle of the realistic range. The opposite of this is so called *cherry picking* where specific parameter values are chosen to highlight the benefit of a particular method, for obvious reasons this should be avoided.
2. *Avoiding bias* and *confounding variables*: Other aspects that can degrade evaluation quality are unintentional bias in the distributions used for certain parameters such as task execution times or periods, and the use of methods which confound two or more variables; for example, task set utilisation and cardinality (number of tasks). Examples of both are given in Section 4.
3. *Statistical confidence*: How many times have we as reviewers looked at graphs of success ratios for different algorithms and seen two lines very close together and wondered if the results really are significant, or how much they might change by simply using a different random seed in the task set parameter generation? By giving information about variation, such questions can be answered.
4. *Standardisation*: If everyone used the same framework and parameter settings for the evaluation of schedulability tests, then the real-time research community would benefit from having many papers containing results that were directly comparable to each other (transitivity). This would be a great advantage in terms of making comparisons and seeing which methods were the most effective. Standardisation would also greatly aid *reproducibility*.

3. System model, terminology and notation

In this section, we briefly recall the sporadic task model which introduces key task parameters that need to be considered in the evaluation of schedulability tests.

Sporadic task model: We assume the system comprises a static set of n tasks that are scheduled to execute on m processors (m = 1 for a single processor system). We assume that each task gives rise to a potentially infinite sequence of jobs. Each job may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task. Each task τ_i is characterised by: its relative *deadline* D_i, *worst-case execution time* C_i, and minimum inter-arrival time or *period* T_i. A task's *worst-case response time* R_i is defined as the longest time from a job of the task arriving to it completing execution. The processor utilisation U_i of task τ_i is given by C_i / T_i. The utilisation of the task set is the sum of the utilisations of all of its tasks.

4. A Framework for Empirical Evaluation

In this section, we propose a framework for empirical evaluation which could potentially be used to provide a de facto standard set of evaluation experiments used in the majority of research into schedulability tests. This baseline could then be extended to consider additional parameters appropriate to the specific problem.

The empirical evaluation of schedulability tests is underpinned by methods for task set generation. We now consider this topic in more detail.

To thoroughly examine the effectiveness of a schedulability test, it is necessary to generate a large number of task sets with different parameter settings that cover in an unbiased way, the range of possible task sets that could occur in practice. Further, this needs to be done in a way that does not confound variables. For example, generating task sets by a process of repeatedly adding tasks to get higher utilisation values confounds task set cardinality and utilisation. It results in a strong correlation between the two, making it impossible to determine if some aspect of the performance of a schedulability test is affected by the number of tasks or by the task set utilisation.

The two primary inputs typically used for task set generation are the task set cardinality n and the required utilisation U. (These parameters are controlled for in a systematic way, since they have such an impact on schedulability test performance).

4.1. Generating task utilisation values

Given requested values for n and U, we need to first generate a set of *n* task utilisation values that add up to *U*. Further this need to be done in a way that results in an unbiased and uniform distribution. In simple terms this is equivalent to repeatedly choosing *n* task utilisation values at random from a uniform distribution in the range 0 to 1 and then only keeping those task sets where the total utilisation adds up to *U*.

There are a number of published methods which are capable of achieving this. For uniprocessor systems, the Uunifast method [6] is the most effective and can typically be implemented in less than 10 lines of code. For multiprocessor systems ($m > 1$) then the Uunifast-discard technique [10] provides a simple to implement extension that builds on Uunifast (just a few lines of code more). Uunifast-discard is effective down to about 2 tasks per processor. Below this level, for example 9 tasks on an 8 processor system, it will not be effective in generating task sets (too many trials without finding task sets that match the criteria). Further, the more complex RandFixedSum method [14] can be used for any valid combination of task set cardinality and utilisation (an open source Matlab implementation of RandFixedSum is available, see [14]).

All three methods take as inputs *n* and *U* and output a set of *n* utilisation values U_i that sum to *U*.

4.2. Distribution of task periods

The next stage in task set parameter generation is to select a set of task periods or minimum inter-arrival times. The execution time values can then be derived as $C_i = U_i T_i$.

Task periods can be selected from a distribution; however, which distribution should be used? Many scheduling papers use a uniform distribution between two values (min and max periods). These values can then be modified to give different ranges of task periods. However, we contend that this is not a good method to use [9]. The reason being that if the range is say 10 to 1,000,000 then on average 99% of all of the periods generated will be in the range 10,000 to 1,000,000. In effect the range of task periods is limited to just two orders of magnitude rather than the intended 5. Put another way, there is virtually no appreciable difference between the average case behaviour for experiments conducted with a range of periods of 10 to 1,000,000 and those conducted with a range of task periods of 10,000 to 1,000,000.

To avoid this problem, we recommend the use of a log-uniform distribution of task periods, again between some min and max values that can be varied. Implementation of random selection from a log-uniform distribution is simple, since it equates to making a random pick from a uniform distribution between the log of the min and max periods and then raising the base of the logarithm to the power of the value obtained to give the period. Again this is typically less than 10 lines of code. We note that fixed priority pre-emptive scheduling is more effective when there is a large spread of task periods; hence it appears to be more effective when a log-uniform rather than a uniform distribution is used (assuming the same range of periods).

While randomly chosen task periods are needed to explore the full range of schedulability test performance, many real systems effectively constrain task periods to a set (or sets) of harmonic values. For evaluation purposes, such harmonic or semi-harmonic sets can be produced via the *bag of primes* method [23] where a set of prime numbers (with duplicates) are placed in the bag and then some number of them are selected at random (without replacement). The selected values are then multiplied together to obtain the task period. This method has the advantage that it constrains the Least Common Multiple (or hyperperiod) of every task set generated to be no larger than the product of the values in the bag of primes.

An alternative approach is simply to specify a set of permitted task periods with harmonic relationships and then pick from that set at random. For example, the task periods used in automotive systems are typically from the set of values (1, 2, 5, 10, 20, 50, 100, 200, and 1000ms) [19].

We note that neither of the methods that generate sets of harmonic periods can provide the same coverage as random generation; however, it can be argued that the results from studies of such task sets may be more representative of task sets found in real systems. Best practice would therefore be to generate both types of task periods and conduct evaluations using both to explore whether the different distributions have a significant effect on schedulability test performance. We note that it is highly likely that they will do, particularly for fixed priority scheduling algorithms; (the utilisation bound for implicit deadline task sets with harmonic periods is 1.0, compared to approx. 0.69 for arbitrary non-harmonic values).

4.3. Distribution of task deadlines

The simplest approach here is to set task deadlines equal to their periods (implicit deadlines); however, in some cases schedulability tests are sensitive to the gap between a job's deadline and its next release. (This can be the case for multiprocessor schedulability tests with carry-in interference [10]).

Two alternative methods of setting task deadlines are prevalent in the literature. The first is to choose the deadline at random (uniform distribution) between the task's worst-case execution time and its period. The second is to vary deadlines in lock-step with periods i.e. $D_i = xT_i$ where *x* is a variable that is used to control the deadlines generated. This enables graphs to be drawn showing how the weighted schedulability metric (see section 5.2) varies with the deadline to period ratio *x*.

To generate arbitrary deadlines, one might choose deadlines in some range, such as $[C_i, kT_i]$ where *k* takes a value of 2, or 4, or 10. Note in this case it may be more appropriate to use a log-uniform distribution of values. We note; however, that in practice, most arbitrary deadlines are multiples of the task period, since they originate from requirements on the size of the buffers needed for inputs and outputs to the tasks.

5. Experiments and graphs

A number of different types of graphs can be used to illustrate the performance of schedulability tests. These include success ratio, weighted schedulability metrics, frequency distributions of breakdown utilisation, and box and whisker plots of metrics such as response times.

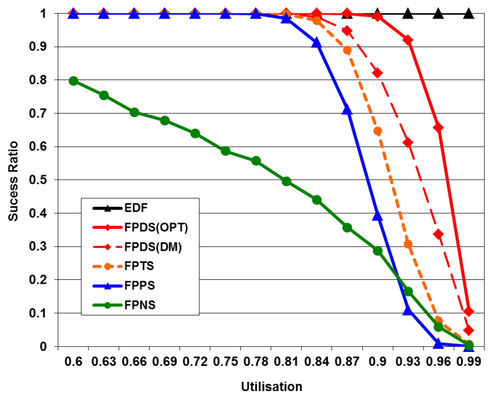

Figure 1: Success ratios v. utilisation

5.1. Success ratio

The simplest type of experiment is to plot the success ratio (i.e. the proportion of task sets that are deemed schedulable by each test against utilisation). An example of such a graph is given in Figure 1.

Since utilisation has such a strong impact on schedulability, then fixing utilisation at a single value and plotting how the proportion of schedulable task sets changes with some other parameter can potentially produce misleading results, or at least results which may change radically if a different value of utilisation were chosen. Instead, we recommend using the weighted schedulability measure [4] discussed below.

5.2. Weighted schedulability metrics

Success ratio graphs have the disadvantage that if we want to vary another parameter as well as utilisation, then we need a whole sequence of graphs for each value of the other parameter or a 3-D plot, which is typically hard to interpret and soon becomes cluttered if there is more than one surface displayed. It is important to vary parameter values to adequately cover the parameter space, since some schedulability tests / scheduling algorithms may be sensitive to a particular parameter, for example the range of task periods and deadlines (as is the case with non-pre-emptive algorithms), or the number of tasks.

Typically, it is not possible to cover the entire parameter space via simple success ratio plots as this would result in too many combinations (1000s of graphs). One useful approach is to vary one parameter at a time while holding the others constant at some appropriate default values. The *weighted schedulability* measure [4] can then be used to illustrate how schedulability varies with each parameter.

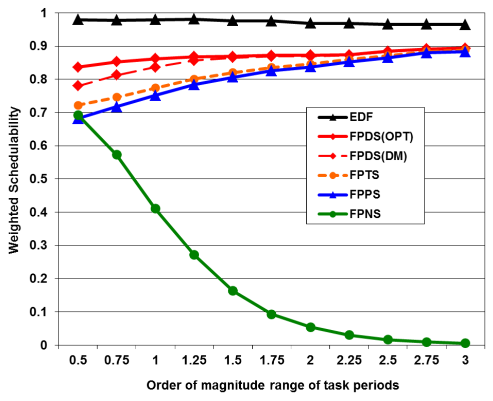

Figure 2: Weighted schedulability versus period range

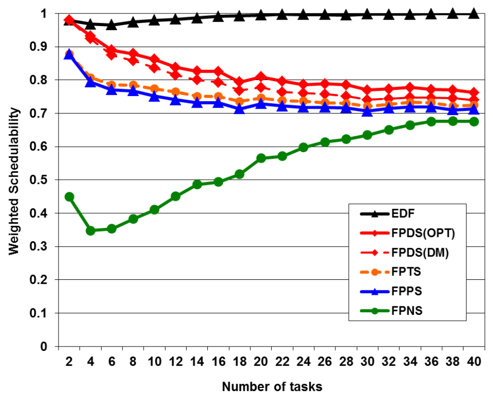

Figure 3: Weighted schedulability versus task set size.

The weighted schedulability measure $Z_y(p)$ for schedulability test y is determined as a function of parameter p. For each value of parameter p, this measure combines results for all of the task sets generated for all of a set of equally spaced utilisation levels. Let $S_y(\tau, p)$ be the binary result (1 or 0) of schedulability test y for a task set τ with parameter value p.

$$Z_y(p) = \sum_{\forall \tau} \frac{S_y(\tau).U(\tau)}{U(\tau)}$$

where $U(\tau)$ is the utilisation of task set τ. The weighted schedulability measure thus reduces what would otherwise be a 3-dimensional plot to 2 dimensions [4]. Weighting the individual schedulability results by task set utilisation reflects the higher value placed on being able to schedule higher utilisation task sets. Examples of weighted schedulability graphs are shown in Figure 2 and Figure 3.

5.3. Breakdown utilisation frequency distribution

When comparing different scheduling algorithms or different priority assignment policies it is sometimes interesting to show the frequency distribution of the *breakdown utilisation* [20]. The breakdown utilisation is the maximum utilisation which can be achieved by scaling the execution times of all of the tasks in the task set by the same factor, without the task set becoming unschedulable. The frequency distribution shows the variability across different task sets and can highlight clear differences between scheduling algorithms or priority assignment policies – see Figure 4.

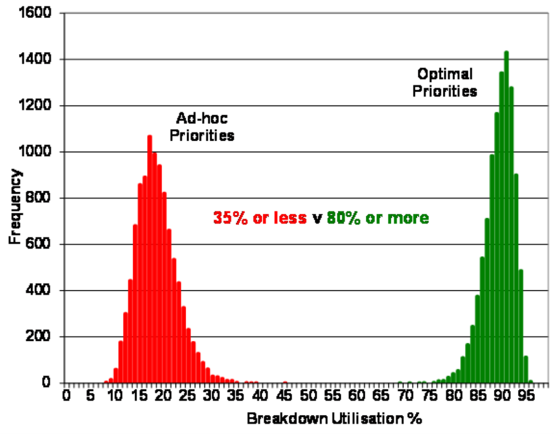

Figure 4: Breakdown Utilisation

5.4. Variation and confidence in the results

When examining schedulability via success ratio or weighted schedulability plots, then the values plotted relate to the number of task sets that are deemed schedulable by the test at each utilisation level; however, no information is given about the variation which might occur in these results if the experiment were repeated multiple times using different random seeds, or indeed the confidence that we can have that the results show a significant difference between two algorithms.

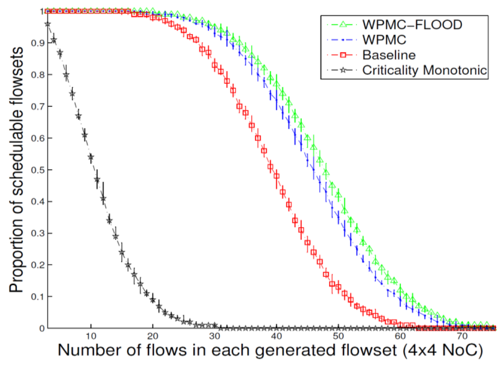

Figure 5: Variation shown on success ratio graph

In order to determine this variation, it is necessary to repeat the experiment multiple times (e.g. 100 times) and examine the distribution of the values returned. This variation can be plotted see Figure 5 as an example, showing a vertical bar between the 25 and 75 percentiles.

5.5. Difference measures

As a final note on success ratio and weighted schedulability plots, one might naively assume that if the line for algorithm A is completely above that for algorithm B, this implies some form of dominance. However it may not even be indicative of a (very) weak form of dominance with respect to the task sets studied. The line for algorithm A may be above that for B, simply because there are many task sets that are schedulable under algorithm A, but not under B. This does not, however, rule out there also being quite a few task sets that are schedulable under algorithm B, but not under algorithm A. Such differences can be illustrated by plotting the number of task sets schedulable with A and NOT with B and vice-versa. Non-zero values for both lines implies *incomparability* of the two algorithms – see Figure 6.

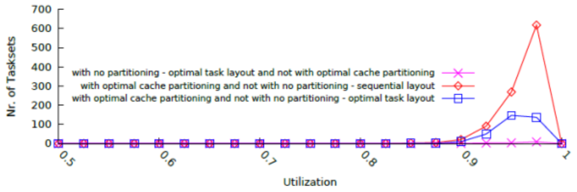

Figure 6: Difference graph showing incomparability

5.6. Box and whisker plots

While schedulability tests give yes/no answers to whether or not a particular task set is schedulable, it is sometimes useful to look at other results such as response times or the number of times a job misses its deadline in some long simulation run. Here substantial variation in values can be expected between different task sets or groups of task sets. Thus it is important that the results are presented along with measures indicating their variability. Box and whisker plots are useful in this respect; they show not only the median (50 percentile) values but also the 5, 25, 75, and 95 percentiles as well as outliers – see Figure 7.

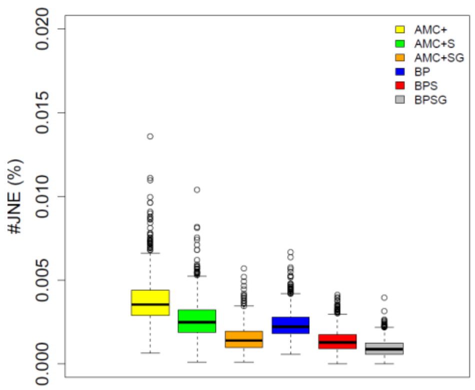

Figure 7: Box and whisker plot showing variance via percentiles and outliers

5.7. Computational complexity and run-time

When evaluating the performance of schedulability tests, it is also important to consider the computational complexity of the test, be it linear, quadratic, polynomial, pseudo-polynomial, or exponential, and also the run-time of the test on task sets of a practical size. Detailed investigation is often warranted into how the run-time of a test changes as different task set parameters are varied. As an example, Figure 8 shows the average run-time of two exact tests for fixed priority pre-emptive scheduling on a single processor. Response Time Analysis (RTA) [17] [2] is pseudo-polynomial in complexity, whereas the Hyperplanes Exact Test (HET) [5] is exponential in the number of tasks [9]. When the range of task periods is limited, the HET test typically requires fewer operations; whereas a large range of task periods can result in very long run-times for that test. More detailed information can be obtained from frequency distributions, see Figure 9.

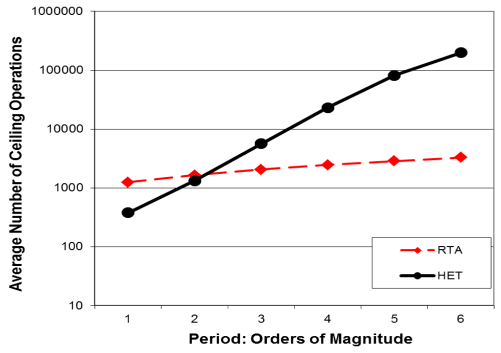

Figure 8: Run-time for exact tests in terms of the number of ceiling operations

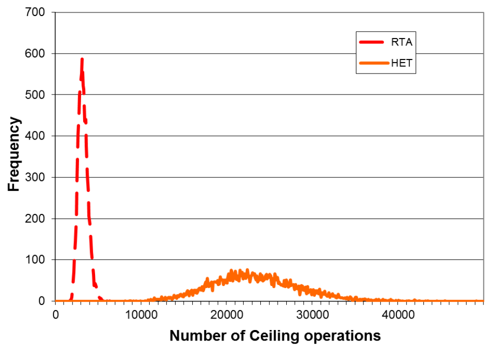

Figure 9: Frequency distribution of the run-time for exact tests in terms of the number of ceiling operations

6. Making task sets from benchmarks

The main disadvantage of entirely synthetic task set generation, as described in Section 4, is that it is difficult to generate appropriate additional parameter values, for example in addition to WCETs, other information may be needed such as the number of memory accesses, as well as Evicting Cache Blocks (ECBs), Useful Cache Blocks (UCBs) etc. In contrast, the main disadvantage of benchmark or case study tasks or task sets is that there are so few of them that it is very difficult to get a systematic view of algorithm or schedulability test performance. Ideally 1000s of task sets are needed.

One solution to these problems is to combine benchmark information with synthetic generation of some parameters. We now describe a simple method for doing just that. We assume that task WCETs, memory accesses, cache usage etc. for each benchmark program are fixed according to the system configuration studied (they may still depend on processor speed, cache size etc.) and have been derived from the benchmark code.

The method of task set generation proceeds as follows, assuming as inputs the task set cardinality and desired task set utilisation. For each task required in a task set, we pick a benchmark program at random[2] from the available set (ideally the number of benchmarks should be substantially larger than the task set cardinality). Next, we generate the task utilisation values using Uunifast, Uunifast-discard, or RandFixedSum as appropriate. The period of each task is then a free variable which is synthesized from the utilisation value chosen for the task and the task's WCET. All of the other task parameters remain as inherited from the benchmark. This process enables a large number of task sets of different utilisation values to be generated from a limited number of benchmarks. These task sets can then be used in the same way as synthetic task sets in empirical evaluations, i.e. as the basis for success ratio – see Figure 10, and weighted schedulability experiments.

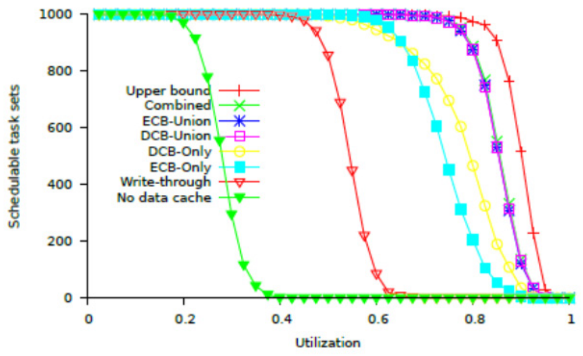

Figure 10: Success ratio plot for task sets generated from benchmarks

The advantages of this approach are that it allows for more detailed and realistic information to be input into task set generation. The set of parameters used are completely consistent with the benchmark code (e.g. WCET, UCBs, ECBs, memory accesses etc. are all consistent). The disadvantages of this approach are that the task sets generated are all grounded in, and hence share similarities with the benchmarks used. They are representative of those benchmarks, but may not be representative of other programs. Further, the distribution of task periods becomes correlated with the benchmark WCETs. This means that if there is a very wide range of benchmark WCETs, the resulting tasks will have a wide

[2] Typically, this selection is made with replacement.

range of periods. This may be problematic when investigating non-pre-emptive scheduling algorithms. An example of this method is given in [1].

7. Summary, Recommendations and Discussion

In this short paper, we presented some of the pros and cons associated with using theoretical and empirical methods to assess the performance of scheduling algorithms and schedulability tests. We discussed a framework for empirical evaluation. This consists of providing baseline results using success ratio plots with task parameters set to realistic default values, and then using weighted schedulability plots to examine how the results change as each relevant parameter is varied over a broad range while keeping the other parameters constant at the default values. We suggested ways of showing statistical confidence in the results obtained, and illustrated a number of different metrics that can be used to examine performance, such as frequency distributions for the breakdown utilisation, and difference plots counting how many task sets are schedulable according to one algorithm, but not with another and vice-versa. We also showed that it is important to examine both the theoretical complexity of schedulability tests and their actual run-times on realistic task sets, since there can be substantial differences in run-time as task set parameters are varied.

7.1. A de-facto standard

Having a de-facto standard set of experiments that we all use to examine the performance of schedulability tests would:

- Make it easier to review and assess work.
- Make reproducing results easier.
- Facilitate direct comparison between results in different papers.
- Provide a set of experiments that we all expect to see.

It would need some agreement on the set of experiments expected, and some standardisation of the details i.e. agreement on some reasonable, and representative default values. Perhaps this is something the WATERS community can progress.

7.2. Call for more Benchmarks

This paper and the presentation it is based on deliberately focused on a simple sporadic task model for single and multiprocessor systems. Much of today's research needs more complex models, for example that provide parameter values for the number (and potentially the pattern) of memory accesses, and the dependencies / interaction / communication between tasks. As the task models get more complex, it becomes harder to be sure that synthetically generated task sets are really representative of those in today's or tomorrow's real systems.

One way of bridging this gap is to obtain a larger number of more comprehensive benchmarks. These benchmarks can then be mined to produce meaningful information covering a wide range of parameters (WCETs, traces of memory accesses, UCBs, ECBs, communication with other tasks etc.). They can be used to set up appropriate ranges and default values for these parameters, which can be fed into more comprehensive task set generation methods. Further, following the simple approach suggested in Section 6, a moderate number of representative benchmarks is all that is needed to directly produce a large number of task sets of different utilisation levels.

We encourage researchers to try and obtain and publish benchmarks that can be freely used by all in our community.

Acknowledgements

This work was partially funded by the UK EPSRC MCC project (EP/K011626/1), and the Inria International Chair program. EPSRC research data management: No new primary data was created in this study. The graphs in this paper have been reproduced with permission from various papers co-authored by researchers in the Real-Time Systems Research Group at the University of York. These papers are not cited where the graphs are simply used as examples and the research in those papers is not referred to as such.

References

[1] S. Altmeyer, R.I. Davis, L. Indrusiak, C. Maiza, V. Nelis, J. Reineke, "A Generic and Compositional Framework for Multicore Response Time Analysis". In proceedings Real-Time Networks and Systems (RTNS), pages 129-138, 4-6th Nov 2015.

[2] N.C. Audsley, A. Burns, M. Richardson , A.J. Wellings., "Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling". Software Engineering Journal, 8(5), pp 284-292, 1993.

[3] S. Baruah, A. Burns "Sustainable scheduling analysis". Proceedings Real-Time Systems Symposium (RTSS), pp 159–168, 2006.

[4] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability," In proceedings of OSPERT, pp. 33-44, Brussels, Belgum, 2010.

[5] E. Bini and G.C. Buttazzo. "Schedulability Analysis of Periodic Fixed Priority Systems". *IEEE Transactions on Computers*, 53(11):1462–1473, November 2004.

[6] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. Real-Time Systems, 30(1-2):129–154, 2005.

[7] B. B. Brandenburg. 2011. "Scheduling and Locking in Multiprocessor Real-Time Operating Systems", PhD Thesis, The University of North Carolina at Chapel Hill.

[8] C. Braun, L. Havet, and N. Navet, "NETCARBENCH: a benchmark for techniques and tools used in the design of automotive communication systems," in 7th IFAC International Conference on Fieldbuses and Networks in Industrial and Embedded Systems, pp. 321–328, 2007. Available at http://www.netcarbench.org.

[9] R.I. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems". IEEE Transactions on Computers, September 2008 (Vol. 57, No. 9) pp. 1261-1276.

[10] R.I. Davis, A. Burns, "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". Real-Time Systems 47 (1) pp1-40, 2011.

[11] R.I. Davis, T. Rothvoß, S.K. Baruah, A. Burns "Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling". Real-Time Systems, Vol. 43, No. 3, pp. 211-258, November 2009.

[12] R.I. Davis, A. Burns, S. Baruah, T. Rothvoss, L. George, O. Gettings, "Exact comparison of fixed priority and EDF scheduling based on speedup factors for both pre-emptive and non-pre-emptive paradigms". Real-Time Systems, Vol. 51, No. 5, pp. 566-601, Sept 2015.

[13] R.I. Davis, A. Thekkilakattil, O. Gettings, R. Dobrin, S.Punnekkat, "Quantifying the Exact Sub-Optimality of Non-Preemptive Scheduling". In proceedings Real-Time Systems Symposium (RTSS), 1-4th Dec 2015.

[14] P. Emberson, R. Stafford, R.I. Davis "Techniques For The Synthesis Of Multiprocessor Tasksets". In proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) , pp. 6-11, July 6th, 2010.

[15] J. Goossens, E. Grolleau, L. Cucu-Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms" Real-Time Systems, pp. 1-25, 2016.

[16] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. "The Mälardalen WCET benchmarks – past, present and future". In WCET, pages 137–147, July 2010.

[17] M. Joseph and P.K. Pandya. "Finding Response Times in a Real-time System". *The Computer Journal*, 29(5):390–395, October 1986.

[18] B. Kalyanasundaram, K. Pruhs, "Speed is as powerful as clairvoyance". *In Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 214-221, 1995.

[19] S. Kramer, D. Ziegenbein and A. Haman, "Real world automotive benchmark for free" WATERS workshop, 2015.

[20] J.P. Lehoczky, L. Sha, Y. Ding. 1989. "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour". In Proceedings Real-Time Systems Symposium (RTSS), pp. 166–171.

[21] C.L. Liu, J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of the ACM*, 20(1) pages 46-61, 1973.

[22] W. Lunniss, S. Altmeyer, R.I. Davis, "A Comparison between Fixed Priority and EDF Scheduling accounting for Cache Related Pre-emption Delays". Leibniz Transactions on Embedded Systems (LITES), Vol. 1, No. 1, April 2014.

[23] C. Macq and J. Goossens, "Limitation of the hyper-period in real-time periodic task set generation," in Proceedings of the 9th international conference on real-time systems, pp. 133–148, 2001.

[24] R.S. De Oliveir, A. Carminati, R.A. Starke, "On using adversary simulators to evaluate global fixed-priority and FPZL scheduling of multiprocessors". Journal of Systems and Software. 2013 Feb

[25] G. von der Bruggen, J-J. Chen, R. I. Davis, and W-H. K. Huang, "Exact Speedup Factors for Linear-Time Schedulability Tests for Fixed-Priority Preemptive and Non-preemptive Scheduling". Information Processing Letters.