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

Robert I. Davis$^{1,2}$,

$^1$Real-Time Systems Research Group, University of York, UK
$^2$INRIA, Paris, France
Outline

- Introduction
  - Different ways of comparing schedulability tests
- Advantages and disadvantages of different approaches
- Key aspects in Empirical Evaluation
- Task set generation
  - Methods and pitfalls
  - Taking a systematic approach
- Some suggestions
- Task set generation from case studies
- Questions and Open Discussion
Comparison of schedulability tests for real-time scheduling algorithms

- **Exact tests**
  - All task sets are correctly classified by the test as either schedulable or unschedulable
  - Comparison of exact tests is in effect a comparison of the algorithms

- **Sufficient tests**
  - May classify some task sets that are in fact schedulable as unschedulable, but not vice-versa
  - Often trade effectiveness for efficiency

- **Evaluation**
  - Interested in guaranteed real-time performance – i.e. from whatever tests are available
Comparison of schedulability tests for real-time scheduling algorithms

- **Theoretical methods**
  - Dominance relationships
  - Utilisation bounds
  - Resource augmentation bounds or speedup factors
  
  Typically give a worst-case comparison

- **Empirical methods**
  - Comparisons using (many) task sets

  Typically give an average-case comparison
Theoretical methods

- **Dominance relationships**
  - Show that one test / algorithm always outperforms another

**Advantages**
- Dominant method always better
- Examples: Exact v. sufficient tests, EDF v. FP

**Disadvantages**
- Typically only applies to a simplified model e.g. no scheduling overheads, no CRPD etc.
- Gives no indication how good the methods actually are (dominant method may still have poor performance)
Theoretical methods

- **Utilisation Bounds**
  - All task sets with utilisation no greater than the bound are guaranteed to be schedulable

**Advantages**
- Illustrates worst-case behaviour for any implicit deadline task set ($D = T$)
- Examples: EDF v. FP ($U = 1$ versus $U = 0.69$)

**Disadvantages**
- Worst-case behaviour may exist only for corner-cases that are of little interest in practice
- Only applies to simple model, implicit deadlines, no overheads etc.
Theoretical methods

- **Speedup Factors**
  - Factor by which the speed of the system needs to be increased, so that any task set that was schedulable under algorithm B is guaranteed to become schedulable under algorithm A

**Advantages**
- Illustrates worst-case performance relative to a different algorithm (or test)
- Used to explore sub-optimality w.r.t an optimal algorithm
- Examples: FP v. EDF, constrained deadlines $S = \frac{1}{\Omega}$
Theoretical methods

- Speedup Factors
  - Disadvantages
    - Worst-case behaviour may exist only for corner-cases that are of little interest in practice
    - May not discriminate well between tests
    - Recent (as yet unpublished) work shows that speedup factors for FP-P v EDF-P and FP-NP v. EDF-NP appear to be the same when simple linear tests are used for FP as they are when exact tests are used
Empirical methods

- **Empirical evaluations**
  - Using synthetically generated task sets to evaluate schedulability tests

- **Simulations**
  - Using synthetically generated task sets to evaluate scheduling algorithms via simulated execution

- **Experiments**
  - Running real or synthetic task sets on real hardware

- **Case studies**
  - Empirical evaluations or simulations, using tasks / task parameters derived from real applications

**Main Focus of this talk is Empirical evaluations**
Empirical methods: pros and cons

- Simulations
  - Simulate the execution of a task set over a long time period, repeat for multiple task sets

Advantages

- Useful to explore average case behaviour
- Useful as a form of necessary schedulability test: deadline misses prove that the task set is not schedulable (but no misses don’t prove schedulability)

Disadvantages

- Typically no guarantee that worst-case behaviours are seen unless the worst-case scenario is known
- Worst-case scenario may be very different for different algorithms e.g. FP and EDF
Empirical methods: pros and cons

- **Experiments**
  - Running real or synthetically generated tasks on real hardware

**Advantages**
- As per simulation (useful to explore average case behaviour, and acts as a necessary test)
- Includes all **overheads** on the actual hardware
- Can be used to collect overhead measurements to include in a model

**Disadvantages**
- Typically no guarantee that worst-case behaviours are seen unless the worst-case scenario is known
Empirical methods: pros and cons

- **Case Studies**
  - One or more example task sets taken from industry
  - Typically the case study provides specific parameter values, or they may be obtained from the code

**Advantages**
- The parameter values used are realistic
- Detailed information available via analysis of code

**Disadvantages**
- Is the case study representative?
- Limited coverage of the parameter space (e.g. one task set) may hide issues elsewhere
Empirical methods: pros and cons

Empirical evaluation
- Generate large numbers of task sets with parameters chosen in an appropriate way
- Evaluate schedulability test performance on these task sets

Advantages
- Can provide good coverage of the parameter space
- Can provide a fair (unbiased) comparison, but care is needed to achieve this

Disadvantages
- Are the parameter values covered representative of real systems?
- What about overheads?
Sporadic task model: as an example

- **Sporadic task model**
  - Static set of \( n \) tasks \( \tau_i \) with priorities 1..\( n \)
  - Bounded worst-case execution time \( C_i \)
  - Sporadic/periodic arrivals: minimum inter-arrival time \( T_i \)
  - Relative deadline \( D_i \)
  - Utilisation \( U_i = C_i / T_i \)
  - Independent execution (no resource sharing)
  - Independent arrivals (unknown a priori)

- **Processors**
  - \( m \) processors (multiprocessor)
  - \( m = 1 \) (uniprocessor)
Empirical evaluation

- **Basic approach**
  - Generate large numbers of task sets with parameters chosen in an appropriate way
  - Determine the performance of different schedulability tests on these task sets
  - Plot graphs e.g. success ratio, weighted schedulability, frequency distributions etc. to illustrating performance

There are a number of key aspects to this
Empirical evaluation: key aspects

- **Systematic approach**
  - Ensure adequate coverage of full range of realistic parameter setting (i.e. avoid *cherry-picking*).

- **Avoid bias and confounding variables**
  - Examples: unintended bias in distributions of execution times, periods etc.
  - Some methods can confound variables, correlating them.

- **Statistical confidence**
  - How might the results have changed with a different random seed.

- **Standardisation of methods**
  - Enables direct comparison between results in different research papers (transitivity), aids reproducibility etc.
Empirical evaluation

- **Aim**
  - 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

- **Basic framework**
  - Baseline approach to task set generation
  - Extensible as further parameters are needed
Task set generation: a systematic approach

- **Primary inputs**
  - Task set cardinality $n$, and Utilisation $U$

- **Utilisation**
  - Given $n$ and $U$ for the task set generate a set of $n$ unbiased utilisation values for the tasks that add up to $U$
    - **Uunifast** – for single processor systems
    - **Uunifast-discard** – for multiprocessor ($n > 2m$)
    - **RandFixedSum** – for multiprocessor

- **Avoids bias and confounding variables**
  Iteratively creating task sets by adding a task to a previous task set confounds (correlates) utilisation and the number of tasks, making it difficult to see the influence of these individual factors on schedulability
Task set generation: Uunifast

- **What does it do**
  - Utilisation values produced have the same distribution as obtained by choosing sets of $n$ values at random from a uniform distribution $[0, U]$ and then only taking those sets that sum to $U$

- **Code**

```c
UUnifast(n, Ut)
{
    SumU = Ut;
    for (i = 1 to n-1)
    {
        nextSumU = SumU * pow(rand(), 1/(n-i));
        U[i] = SumU - nextSumU;
        sumU = nextSumU;
    }
    U[n] = SumU;
}
```
Task set generation: Uunifast-discard

- **Problem with Uunifast**
  - For $U > 1$ Uunifast can generate utilisation values $>1$ which are invalid for individual tasks

- **What does Uunifast-discard do**
  - Simply throws away task sets with invalid tasks, proven to produce an unbiased uniform distribution of utilisation values
  - Works well for $n > 2m$, but too many discards (invalid tasks) for smaller $n$
  - For $n$ closer to $m$ need to use a more general method provided by **Randfixedsum**
Task set generation: Randfixedsum

- **What does Randfixedsum do**
  - General algorithm derived by Roger Stafford for creating vectors uniformly distributed in an \( n-1 \) dimensional space whose components sum to a constant value
  - Can be used to generate utilisation values for multiprocessor task sets
  - Efficient since no random values need to be excluded
  - Open source MatLab implementation available
Task set generation: Task Periods

- **Periods can be selected from some distribution**
  - Which distribution(s) should we use?
  - Limit periods to a range between a min and max value

- **Uniform?**
  - Using a uniform distribution has some issues
  - Want to be able to vary range of task periods, since this is an important parameter w.r.t. non-preemptive scheduling and complexity of some schedulability tests
  - With a period range of \([10, 1,000,000]\) then roughly 99% of periods are in \([10,000, 1,000,000]\) i.e. 2 orders of magnitude when we expected 5
  - Uniform distribution not effective in showing differences due to range of periods
Task set generation: Task Periods

- **Log-Uniform?**
  - Random selection from a log-uniform distribution: random pick from a uniform distribution between the logs of the min and max periods and then raise the base of the log to the power of the value chosen to obtain the period
  - Expected number of tasks in any order of magnitude range is the same e.g. [10,100], [100,1000] etc.
  - Avoids previous issues with uniform distribution

- Note Fixed Priority scheduling is more effective when there is a larger spread of periods, hence FP is more effective with Log-Uniform than with Uniform distributions with the same period range
Task set generation: Task Periods

- **Harmonics**
  - Task periods in real systems tend to be chosen from a set (or sets) of harmonic values
  - This can be simulated using the **bag of primes** method

- **Bag of primes method**
  - A set of small prime numbers (with some repeats) are chosen as a basis (e.g. 2,2,2,2,3,3,3,5,5...) and placed in the bag
  - A number of values are then selected at random from the bag without replacement
  - The product of the values chosen gives the task period
  - The LCM of task periods is limited to the LCM of all values in the bag
Task set generation: Task Periods

- **Harmonics – alternative method**
  - Simply specify a set of possible values, for example as may be used in automotive systems (5, 10, 20, 50, 100, 250, 1000ms)
  - Chose values at random from the list
  - Again the hyperperiod is limited to the LCM of the values specified

- **Notes**
  - Since harmonic and non-harmonic periods can differently impact schedulability (e.g. FP has a utilisation bound of 1 for harmonic task sets, and 0.69 for non-harmonic) best practice would be to repeat expts with both distributions
Task set generation: Task Deadlines

- **Deadlines**
  - Implicit deadlines equal to period
  - Constrained deadlines
    - Chosen at random between $C$ and $T$
    - Varied in lock step as a proportion of period
Evaluation Framework: Baseline

- **Baseline settings**
  - Determine realistic settings as defaults for parameter values and vary utilisation

- **Success ratio plots**

- Typically need about 1000 task sets per utilisation level
Evaluation Framework: Weighted schedulability

- **Varying parameters**
  - Need to vary parameters to cover a wide range of possible parameter values
  - Important to do this as some schedulability tests / algorithms may be sensitive to a particular parameter e.g. range of task periods, number of tasks, etc.
  - Typically not possible to cover the whole parameter space via simple success ratio plots – too many combinations (1000s of plots)
  - Can vary one parameter while holding others constant at default values
  - Use weighted schedulability plots to illustrate variation w.r.t. each parameter
Evaluation Framework: Weighted schedulability

- **Weighted schedulability**
  - Combines results for all of the task sets generated for all of a set of equally spaced utilisation levels (i.e. from a line on a success ratio plot)
  
  \[
  Z_y(p) = \sum_{\forall \tau} \frac{S_y(\tau).U(\tau)}{U(\tau)}
  \]
  
  - Effectively the area under the success ratio curve but weighted by utilisation – gives more emphasis to scheduling high utilisation task sets
  - Reduces multiple success ratio plots to a single weighted schedulability graph
Evaluation Framework: Weighted schedulability

- Examples of weighted schedulability graphs

- Typically need about 100 task sets per utilisation level, since there are usually at least 10 utilisation levels that make up each data point
Evaluation Framework: Frequency distributions

- Frequency distribution of breakdown utilisation
Evaluation Framework: Confidence intervals

- How confident are we the picture wouldn’t change if we run the experiment again with a different random seed?
- Multiple runs to show percentiles for each data point
Evaluation Framework: Difference measures

- One line being above another does not imply dominance
  - Can plot number of task sets schedulable with test A but not with test B and vice-versa to show *incomparability*
Evaluation Framework: Variability: box and whisker plots

- Schedulability is a binary result (yes/no)
  - Interesting to look at other metrics and consider their variability
Empirical evaluation: Task sets from case studies / benchmarks

- **Case studies / benchmarks:**
  - Typically provide a small number of tasks / task sets
  - Can provide other detailed information e.g. WCETs, memory accesses, UCBs, ECBs used in CRPD analysis etc.
  - However, large numbers of task sets are needed for evaluation purposes

- **Making task sets from benchmarks**
  - Random selection of tasks from (larger) benchmark set
  - Chose utilisation values using Uunifast etc.
  - Compute period = C/U (can therefore use real WCETs)
Empirical evaluation: Task sets from case studies / benchmarks

**Advantages:**
- More detailed and realistic information input into task set generation
- Task parameters take on real values e.g. WCETs of actual code

**Disadvantages**
- All task sets generated share similarities since they are generated from the same limited set of benchmarks, so only representative of the input benchmarks
- Period distribution correlates with WCET distribution
- May need to exclude some benchmarks to control range of task periods (e.g. when investigating non-preemptive algorithms)
Empirical evaluation: Task sets from case studies / benchmarks

- Example with task set generation using data from Malardalen benchmarks
Empirical evaluation: Recap

- **Empirical evaluation**
  - Investigates schedulability test / scheduling algorithm performance w.r.t. large number of synthetically generated task sets

- **Evaluation framework:**
  - Baseline results using success ratio plots (from realistic default values)
  - Weighted schedulability results varying each relevant parameter over a broad range, keeping other parameters constant at default values
  - Consider statistical confidence in results
  - Use other metrics to illustrate specific properties
Empirical evaluation: A suggestion

- **A de-facto standard: If we all used the same framework for evaluation this would:**
  - Make it easier to review / assess different work
  - Make reproducing results easier
  - Facilitate direct comparison between results in different papers
  - Provide a set of expts we expect to see in papers

- Would need to agree on the set of experiments expected, and some de-facto standard details such as defaults, parameter ranges etc.
Open discussion

- More complex task models needed
  - Presentation deliberately restricted to a simple task model
  - Many other attributes need to be modelled
  - Interaction / communication between tasks
  - Multiprocessor – cross core contention – memory demand and processor demand
Open discussion

- Few real benchmarks available to build upon
  - Use of synthetic task sets v. case studies, both have their pros and cons
  - Useful to build task sets from benchmarks - some caveats in doing so
Open discussion

- Is some form of standard framework useful?
  - Use the same task set generators?
Open discussion

- Can we improve how we evaluate schedulability tests for real-time scheduling algorithms?
Questions?