Schedulability and Response-time Analyser

Michael Brockway

October 15, 2012

1 Response-time Equation Solver – Principles

We are given a set of periodically, pre-emptively scheduled tasks, t, with *fixed* priorities, computation times C_t, periods T_t, and deadlines D_t; and *without* interference between tasks. The present software assumes jitter and context switch times are negligible, and that there is no blocking of tasks due to competition for system resources. Other versions of this software include jitter in the analysis and indicate how context-switch times and blocking times may be included.

The *response times* r_t of the tasks are (fixed point) solutions of the equations (one for each task)

$$r_t = C_t + \sum_{u \in hp(t)} \ulcorner (r_t)/T_u \urcorner . C_u \tag{1}$$

$hp(t)$ denotes the set of tasks of priority higher than t.

A task t is deemed to *meet its deadline* if $r_t \leq D_t$.

The equations (an instance of (1) for each task t in the set) are solved iteratively for the r_t by setting, for each t, $r_t = \sum_{u \in hp(t)} C_u$ initially and then repeatedly using equations (1) to compute a new set of r's from the previous set, until no r values change.

If at any stage, for some task t, $r_t \geq T_t$, the system is unschedulable.

The software also computes the CPU *utilisation* of the task set,

$$U = \sum_t C_t/T_t \tag{2}$$

When there are n tasks in the set, prioritised *rate-monotonically* (in order of period, shorter period $\rightarrow$ higher priority), then a *sufficient* condition for schedulability is

$$U \leq n(2^{1/n} - 1)$$

Also, if an *earliest-deadline-first* scheduling discipline is used rather than fixed priorities, the task set is schedulable provided $U \leq 1$; although the response-time analysis is irrelevant in this case, of course.

In the present implementation, the task set is stored as an array of records of task attributes including the tasks name, its (maximum) computation time, period and deadline. Priority order (hi $\rightarrow$ lo) is the order of the tasks in the array. (So $\sum_{u \in hp(t)}$ is just the sum over previous tasks in the array.)

You have three options.

1. You can run the analysis "raw" – with priorities in the order you entered the task data;
2. You can ask the software to pre-process the array by sorting it into ascending order by task periods, reflecting a *rate-monotonic* schedule;
3. You can to pre-process the array by sorting it into ascending order by task deadlines, reflecting a *deadline-monotonic* schedule. Do not confuse this with EDF scheduling!

There are implemetations in C and in Java.

2 Response-time Equation Solver – Using it

Tasks are input to an array in priority order (hi $\rightarrow$ lo), unless you propose to use one of the preprocess sorting options. So for task t in the array, the sum in (1) above is over all tasks u earlier than t in the array.

The program reads the task information from a text file containing one line for each task, consisting of a task name (with no spaces), a C value, a T value and a C value. Lines beginning `//` and empty lines are ignored. For example (`data1.txt` in this folder) -

```
//name  C   T     D
task_1  20  100   80
task_2  30  150   60
task_3  90  1000  1000
task_4  60  1000  600
```

C and Java versions of the solver are provided, with source files `RespTime.c`, `RespTime.java` for your interest. Run them at a command prompt: one of

```
...$ RespTime <data-file> [r|d] [> <log-file>]
...$ java -jar RespTime.jar <data-file> [r|d] [> <log-file>]
```

Run with the `r` option if you want the task list sorted into rate-monotonic order before the response-times calculation. Run with the `d` option if you want it sorted into deadline-monotonic order. Run with neither of these options if you want the task list left in the order you put in the data file.

Use a redirection `> logfile` to send the output to a text file; otherwise it is displayed on the console. You can also *append* output to a file with `>> logfile` and, in Unix, pipe the output to the `tee` command to send the output to both the screen and a log file (also has an append option).

3 Extensions

The simple response-time analysis implemented in these software tools can be "improved" in various ways, for instance,

3.1 Jitter

If each task in the set has, additionally, a jitter (random delay in task release) attribute, J, this may be added as a data-attribute of a task, the function to initialise task data structures upgraded appropriately, and the solve() method upgraded so as to solve iteratively a set of equations

$$r_t = C_t + \sum_{u \in hp(t)} \lceil (r_t + J_u)/T_u \rceil . C_u \tag{3}$$

(one instance for each task t in the set).

As an exercise you might implement this in C or Java.

3.2 Blocking Time

If each task in the set has, additionally, a blocking-time attribute, B, this may be added as a data-attribute of a task, the function to initialise task data structures upgraded appropriately, and the solve() method upgraded so as to solve iteratively a set of equations

$$r_t = C_t + B_t + \sum_{u \in hp(t)} \lceil (r_t + J_u)/T_u \rceil . C_u \tag{4}$$

(one instance for each task t in the set).

EXERCISE: Implement this upgrade (leave jitter out if you wish).

3.3 Context Switch Time

Scheduling overheads can be modelled by allowing a finite amount of time s for context-switching between tasks. In this case the relevant set of equations to solve is, for each t in the task set,

$$r_t = C_t + B_t + 2s + \sum_{u \in hp(t)} \lceil (r_t + u.J)/u.T \rceil (u.C + 2s) \tag{5}$$

EXERCISE: Implement this.

3.4 Pessimistic

This response-time analysis is "worst-case" as it assumes each task released simultaneously with all higher-priority tasks (at the "critical instant"). There

is work in the literature which also tries to obtain "best-case" estimates; for instance,

- Ola Redell and Martin Sanfridson (2002), *Exact Best-Case Response Time Analysis of Fixed Priority Scheduled Tasks*, in proceedings of Euromicro Conference on Real-Time Systesm (ECRTS2002), pp 165-172, downloadable from `http://www.artes.uu.se`
- William Henderson *et al* (2001), *Improving the Accuracy of Scheduling Analysis Applied to Distributed Systems*, in Intl. Journal of Real-Time Systems, vol 20, pp 5-25; downloadable from `http://computing.unn.ac.uk`