Choosing task periods to minimise system utilisation in time triggered systems

Alan Burns *, Robert Davis

Real-Time Systems Research Group, Department of Computer Science, University of York,
Heslington, York YO1 5DD, United Kingdom

Received 12 April 1995; revised 15 April 1996
Communicated by R.S. Bird

Abstract

The analysis associated with priority based scheduling usually assumes that task periods are fixed by operational requirements. In this paper we consider systems in which the overriding requirement is to recognise and respond to events originating in the system's environment. An algorithm is presented which can deterministically choose task periods so that event deadlines are satisfied, whilst attempting to maximise the number of such events which can be dealt with. Although it is shown that a simple optimal priority ordering scheme is not possible, an effective heuristic for choosing priorities is presented.

Keywords: Real-time systems; Scheduling

1. Introduction

The fundamental requirement for hard real-time systems is predictability. Systems must be guaranteed to respond to the occurrence of events in their external environment within a specified time (the event deadline). The software implementation of such systems typically comprises a set of processes or tasks. These tasks may be executed periodically, or sporadically in response to the external events. In safety critical real-time systems, it is often desirable to poll for all external events rather than use interrupts and sporadic tasks. To each such event is assigned a periodic task and a deadline by which the event must be noted and acted upon. In this paper we consider the use of fixed priority scheduling to manage these periodic tasks, and determine (off-line) that all event deadlines are satisfied.

Fixed priority scheduling has been the focus of considerable research activity over the last decade. Initial work centred upon Rate Monotonic Scheduling [7] where each task is required to complete before its next release. In effect this implies that each task \( \tau_i \) has a deadline, \( D_i \), equal to its period, \( T_i \). More recent work has considered systems with \( D_i < T_i \) [6] or \( D_i > T_i \) [5,8]. In the former case a simple algorithm provides an optimal priority assignment: task priorities are ordered according to deadline, the shorter the deadline the higher the priority. When tasks have \( D_i > T_i \) a more complex priority assignment scheme is needed [3].
To determine if a task set is schedulable it is necessary to know the period and the worst case computation time for each task. From this information the worst case completion time (or response time), $R_i$, for each task can be calculated (see Section 2 for a description of this analysis).

In order for a periodic task to handle an asynchronous event its period must be sufficiently short so that it is guaranteed to process the event by the deadline of that event. For each task the worst possible time for an event to occur is just after the task checked for it. Time $T_i$ will pass before the task is invoked again (in the worst case), a worst case response time of $R_i$ thus generates the following necessary relation for all tasks (i.e. $\forall i$),

$$T_i + R_i \leq E_i, \quad (1)$$

where $E_i$ is the deadline of the event polled for by task $\tau_i$.

With Rate Monotonic Scheduling (i.e. $D = T$), relation (1) can be satisfied by fixing each task’s period:

$$T_i = \frac{E_i}{2}. \quad (2)$$

Note it is assumed that the event polled for by task $\tau_i$ cannot recur within $E_i$.

This simple formulation is however inefficient. Table 1 contains a simple task set catering for six events (the computation time for each task is represented by $C$). The periods of each task are obtained from Eq. (2). Note the apparent total utilisation of the task set is 122% and hence it would appear that the system cannot be guaranteed. In Section 3 we illustrate how the tasks can be scheduled to meet the event deadlines – and present the analysis that allows this behaviour to be guaranteed.

### Table 1

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$E$</th>
<th>$T$</th>
<th>$C$</th>
<th>Utilisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>16</td>
<td>8</td>
<td>3</td>
<td>37.5%</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>10</td>
<td>3</td>
<td>30%</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>30</td>
<td>15</td>
<td>3</td>
<td>20%</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>40</td>
<td>20</td>
<td>3</td>
<td>15%</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>48</td>
<td>25</td>
<td>3</td>
<td>12%</td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>80</td>
<td>40</td>
<td>3</td>
<td>7.5%</td>
</tr>
</tbody>
</table>

### 2. Response time calculations

To introduce the analysis developed in this paper, a simple computational model is used. Extensions to this model are possible without significant extra complexity. The task set to be analysed is assumed to consist of a fixed number of independent periodic tasks. All system overheads, e.g. context switching, clock interrupt handling, determining next task to run etc., are ignored (assumed to have zero cost).

The work of Joseph and Pandya [4] and later Audsley et al. [1] gives the following equation for the worst case response time of a given task $\tau_i$ (for $R_i \leq T_i$),

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lfloor \frac{R_j}{T_j} \right\rfloor C_j, \quad (3)$$

where

- $R_i$ is the worst case response time of task $\tau_i$. This is defined as the longest time between the task being invoked and the time it completes its execution.
- $C_i$ is the worst case execution time of task $\tau_i$.
- $T_i$ is the period of task $\tau_i$.
- $hp(i)$ is the set of all tasks of higher priority than task $\tau_i$.

Time is modeled using the nonnegative integers; hence all task attributes ($C_i$, $T_i$, etc.) have positive integer values.

The most severe restriction of this simple model is task independence. Any realistic system will require task interaction (for example, to exchange data). In terms of response time calculation the only difference task interactions make is the inclusion of a blocking term ($B_i$) into Eq. (3). The value of $B_i$ is the maximum time $\tau_i$ has to wait for a lower priority task to complete a dependent operation (for example, to exit a data area requiring mutual exclusion). Different models of interaction give rise to different values of these blocking factors. This issue is, however, not a topic for this paper. All the response time equations given below could have a $B_i$ term added. For simplicity of presentation, we assume that the blocking times are zero, and hence we do not explicitly include $B_i$.

The analysis makes no assumptions about the priority ordering used, except that priorities are unique (i.e., no tasks can share the same priority).

No simple solution to Eq. (3) exists since the term $R_i$ appears on both sides. However, a recurrence rela-
Table 2

<table>
<thead>
<tr>
<th>Task</th>
<th>$T$</th>
<th>$C$</th>
<th>$R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$T_2$</td>
<td>10</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>$T_3$</td>
<td>15</td>
<td>3</td>
<td>15</td>
</tr>
</tbody>
</table>

It can be formed:

$$w_{t}^{n+1} = C_t + \sum_{j \in hp(i)} \left\lfloor \frac{w_{t}^{n}}{T_j} \right\rfloor C_j.$$  \hspace{1cm} (4)

At this point, the $(q+1)$th invocation completes before the next release and hence the worst case response time of any invocation of $T_i$ is given by:

$$R_i = \max_{q=0,1,2,...} (R_i(q) - qT_i).$$  \hspace{1cm} (7)

Note $q$ is bounded when the overall utilisation is less than 100% [8].

3. Choosing task periods

The analysis in the previous section, in common with most studies on fixed priority scheduling, assumes that task periods are derived from requirements and are thus fixed. In this section we choose task periods so that relation (1) is satisfied. Three models (of increasing complexity and realism) are presented. The models have different assumptions about the computation times of tasks:

1. All releases of tasks have the same computation time.
2. Task computation times differ (but are the same for different releases of the same task).
3. Each task has two computation times corresponding to whether the event is observed or not.

In the first model a single parameter, $C_t$, is applicable to all tasks. The second model has a distinct $C_t$ per task. The final, most realistic model distinguishes between an execution of the task that handles an event, and one in which the event is missing. It can be assumed that the handling of an event will take considerable more computation time. Hence this last model has two parameters per task, $C_t^T$ (which is the time required to test for the event) and $C_t^H$ (the extra time needed to handle the event if it has occurred).

The motivation for presenting the first very simple model is that an optimal priority ordering scheme exists for it. This priority ordering scheme is not optimal for the other models, but proves to be a good heuristic for finding an effective priority ordering.

3.1. First model – Single $C$ for all tasks

If a task’s response time $R_t$ is less than its period $T_t$, then relation (1) and Eq. (3) can be easily combined (as the actual value of $R_t$ is independent of $T_t$). For other values of $R_t$, Eq. (5) must be solved for
the bounded set of $q$ values. Eq. (7) then provides a value of $R_i$, and relation (1) a value of $T_i$. Each time $T_i$ changes, the response time calculations must be repeated (as $R_i$ is dependent on $T_i$). Fortunately this relation is monotonic (i.e. $T$ is never increased and $R$ is never reduced); eventually either $R_i$ and $T_i$ become stable or $T_i$ takes on a value less than zero. In the latter case we conclude that the system is not schedulable.

These observations lead to the following algorithm for choosing the maximum task periods such that relation (1) is always (just) satisfied:

\[
\begin{align*}
\text{order all tasks by priority} \\
\text{for all tasks from high priority to low loop} \\
\quad \text{calculate response time } R \text{ using eqn(4)} \\
\quad \text{if } R \geq E \text{ then} \\
\quad\quad \text{exit (event deadline cannot be guaranteed)} \\
\quad \text{else} \\
\quad\quad T := E - R \\
\quad\quad \text{if } R \leq T \text{ then} \\
\quad\quad\quad \text{exit (event deadline guaranteed)} \\
\quad\quad \text{else} \\
\quad\quad\quad \text{loop} \\
\quad\quad\quad \quad \text{recalculate } R \text{ using eqn(5)} \\
\quad\quad\quad \quad \text{if } R \text{ has not changed then} \\
\quad\quad\quad \quad\quad \text{exit (event deadline guaranteed)} \\
\quad\quad\quad \quad \text{else} \\
\quad\quad\quad \quad\quad T := E - R \\
\quad\quad\quad \quad\quad \text{if } T \leq 0 \text{ then} \\
\quad\quad\quad \quad\quad\quad \text{exit (event deadline cannot be guaranteed)} \\
\quad\quad\quad \quad\quad \quad \text{end} \\
\quad\quad\quad \text{end loop} \\
\quad\quad \text{end} \\
\quad \text{end} \\
\end{align*}
\]

When this algorithm is applied to the task set given in Table 1 the following results are obtained (see Table 3). Note that the apparent utilisation has been reduced from 122% (when $T = 0.5E$) to under 88%, and that all event deadlines are satisfied. It should also be noted that all the task’s response times are less than their periods, and hence only the first part of the algorithm was needed for this example.

### 3.1.1. Priority assignment

This analysis assumes that task priorities have already been given. The algorithm used, mapped event deadlines on to task priorities - the shorter the deadline, the higher the priority. For the simple model with all task's having the same computation time it is possible to prove that this algorithm is optimal.

**Definition 1.** A priority ordering scheme is optimal if it can schedule any task set that can be scheduled by any other scheme.

**Definition 2.** Event Deadline Monotonic (EDM) ordering of priorities assigns the higher priorities to the tasks with the shorter event deadlines:

$$\forall i, j: E_i < E_j \Rightarrow P_i > P_j.$$  

**Theorem 3.** If all computation times are equal and response times are no greater than task periods then Event Deadline Monotonic (EDM) ordering of priorities is optimal.

**Proof.** Following the proof that deadline monotonic ordering [6] is optimal for fixed period tasks (when $R_i < T_i$), assume that there exists a task set $\Gamma$ that can be scheduled by priority ordering scheme $X$. Let $\tau_i$ and $\tau_j$ be two tasks within $\Gamma$ that are given consecutive priorities but are not ordered according to the EDM scheme, i.e. $P_i > P_j$, but $E_i > E_j$. Note that if no two such tasks exist then $X$ is EDM.

Consider the consequences of swapping the priorities of $\tau_i$ and $\tau_j$.

Note first, that all tasks with priorities greater than that of $\tau_j$ are unaffected.

Next consider the two tasks themselves. Task $\tau_j$ was schedulable at the old priority level and so is certain to be schedulable at the higher level (as $E_i > E_j$). Moreover, as it requires the same computation time as $\tau_i$, then the response time $R_j^0$ with the new priority is equal to the response time $R_i^0$ with the old priority.

Consider $\tau_i$: its new response time, $R_i^0$, cannot be shorter than $R_i^0$ as it requires the same computation time. As the interference due to higher priority tasks
is the same regardless of the ordering of $\tau_i$ and $\tau_j$, the only way that $R_j^i$ could be greater than $R_i^j$ is if $\tau_j$, which now has a higher priority, executes a second time before $\tau_i$ can complete, i.e. $T_j^i < R_i^j$. But $T_j^i$ must be greater than $T_j^i$ (as its response time has reduced) and $T_j^i \geq R_j^i$, by definition. Hence $T_j^i \geq R_j^i$ and $\tau_i$ is schedulable (as $E_i > E_j$) with its new priority position, and $R_i^j = R_i^j$.

Finally, it is necessary to show that all lower priority tasks remain schedulable. To do this we shall prove that at all times the interference on these lower priority tasks is less with the priority ordering $\{\tau_j, \tau_i\}$ than it was with the ordering $\{\tau_i, \tau_j\}$.

The interference for priority ordering $\{\tau_i, \tau_j\}$ is given by (for time $t$, $t > 0$):

$$\left[\frac{t}{T_i^0}\right] C + \left[\frac{t}{T_j^0}\right] C,$$

where $C$ is the computation time of the two tasks.

This is equivalent to

$$\left[\frac{t}{E_i - R_i^0}\right] C + \left[\frac{t}{E_j - R_j^0}\right] C.$$

The new load is

$$\left[\frac{t}{E_i - R_i^0}\right] C + \left[\frac{t}{E_j - R_j^0}\right] C.$$

When the relation between the old and new response times is substituted, the difference in load becomes:

$$\left[\frac{t}{E_i - R_i^0}\right] C + \left[\frac{t}{E_j - R_j^0}\right] C$$

$$- \left[\frac{t}{E_i - R_i^0}\right] C - \left[\frac{t}{E_j - R_j^0}\right] C.$$

For this difference to be positive requires the following inequality to be true:

$$\left(\frac{t}{E_i - R_i^0}\right) + \left(\frac{t}{E_j - R_j^0}\right)$$

$$- \left(\frac{t}{E_i - R_i^0}\right) - \left(\frac{t}{E_j - R_j^0}\right) \geq 2.$$

The "2" comes from the removal of the ceiling functions. This reduces to

$$(R_j^0 - R_i^0)(E_i - E_j)$$

$$\times [(E_j - R_j^0) + (E_i - R_i^0)] \geq 2/t.$$

3.2. Second model – One $C_i$ per task

As the general analysis outlined in Section 2 dealt with tasks having non-identical computation times, the algorithm for choosing task periods derived above is equally applicable to this more general case. For example, Table 4 illustrates the application of the algorithm to a new task set (task are listed in priority order with $\tau_1$ having the highest priority).

Unfortunately, for this model, there is no simple algorithm for assigning priorities to the tasks. To illustrate this consider the tasks given in Table 5.

If the task set consists of tasks $\tau_a$, $\tau_b$ and $\tau_x$ then the only priority ordering that will meet the event deadlines is $\{\tau_a, \tau_b, \tau_x\}$. However, if the task set is $\tau_d$, $\tau_b$ and $\tau_y$ then the only feasible ordering is $\{\tau_b, \tau_d, \tau_y\}$. Hence a subset of the tasks cannot be considered in isolation, all tasks must be included before an effective order can be determined. To search for a feasible priority ordering could take up to $N!$ cases (where $N$ is the number of tasks). This is clearly not feasible

<table>
<thead>
<tr>
<th>Task attributes</th>
<th>$E$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_a$</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_b$</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_x$</td>
<td>43</td>
<td>13</td>
</tr>
<tr>
<td>$\tau_y$</td>
<td>36</td>
<td>11</td>
</tr>
</tbody>
</table>
for any nontrivial size of $N$, and hence a heuristic algorithm would be beneficial.

In general computation times will be similar to each other, and for most tasks, in any task set, $R$ will be less than $T$. Where some values of $C$ are very large (by comparison) it may be that utilisation is reduced by ordering the tasks in a non-EDM way, i.e.

$$\left(\frac{C_i}{E_i - R_i^p}\right) + \left(\frac{C_j}{E_j - R_j^p}\right)$$

$$< \left(\frac{C_i}{E_i - R_i^a}\right) + \left(\frac{C_j}{E_j - R_j^a}\right)$$

even though $E_i > E_j$. This check is easily carried out (it only involves switching two tasks and recalculating response times), and leads to the following algorithm:

```
order all tasks using EDM
test_for_schedulability
if not schedulable then
do for I in 1..N-1 do calculate response times for tasks I and I+1 reorder I and I+1 if utilisation is lowered od
test_for_schedulability (exit if true)
if order unchanged then exit (failure)
od
end
```

Where this algorithm fails is when a task set has the (unfortunate) property that, although the utilisation has been decreased by the reordering, the interference at some specific time $t$ has increased. And this time coincides with the response time of some lower priority task. With the task set in Table 5, the “correct” ordering is $r_0, 76$, but for $7_Y$ this ordering is inappropriate and the reverse is best.

Notwithstanding the existence of counter-examples to the proposed priority assignment algorithm, the scheme has worked well in simulation studies (which considered over 33 million task sets). With small task sets (where the $N!$ different orderings could all be evaluated), the scheme found 99.90% of the feasible task sets. Specifically EDM itself found 99.82% and the reordering gave the extra 0.08%.

### 3.3. Third model – Two computation times per task

With this more realistic model the execution time of the task when it is handling an event ($C_i^H + C_i^T$)

| $\tau_1$ | 25 | 1 | 2 | 3 | 22 | 3 | 22 |
| $\tau_2$ | 30 | 1 | 4 | 8 | 22 | 8 | 22 |
| $\tau_3$ | 100 | 1 | 10 | 19 | 81 | 19 | 81 |
| $\tau_4$ | 150 | 1 | 15 | 43 | 107 | 43 | 107 |
| $\tau_5$ | 200 | 1 | 15 | 75 | 125 | 71 | 129 |
| $\tau_6$ | 400 | 1 | 15 | 185 | 215 | 106 | 294 |
| $\tau_7$ | 800 | 1 | 20 | 214 | 586 | 196 | 604 |
| $\tau_8$ | 1000 | 1 | 20 | 428 | 572 | 273 | 727 |
| $\tau_9$ | 1800 | 1 | 20 | 834 | 966 | 323 | 1477 |

is assumed to be significantly greater than when the event is missing ($C_i^T$). In a given window of time $W$, the maximum number of releases of task $\tau_i$ is given by

$$\left\lceil \frac{W}{T_i} \right\rceil.$$  

Further the window of time over which events may have occurred which are then dealt with in the interval $W$ is given by

$$\left\lceil \frac{W}{T_i} \right\rceil T_i.$$  

Hence the maximum number of events which may be handled in time $W$ is

$$\left\lceil \frac{W}{T_i} \right\rceil T_i.$$  

The response time equation for $R \leq T$ is now:

$$R_i = C_i^T + C_i^H + \sum_{\forall j \in \text{hp}(i)} \left[ \left\lceil \frac{R_i}{T_j} \right\rceil C_j^T + \left\lceil \frac{R_i}{E_j} \right\rceil C_j^H \right]. \tag{8}$$

Table 6 provides a more extensive task set and illustrates the difference between the task response times and periods that are calculated by the third model via Eq. (8) (i.e. $R^3$ and $T^3$) and those that are generated by the second model given in Section 3.2 (i.e. $R^2$ and $T^2$). Task priorities are assigned according to EDM.

Note that there is little difference for the higher priority tasks but the lower priority ones do exhibit significant differences. Overall the utilisation is reduced from 81.59% for the second model to 77.78% for the third.

The response time equation for $R > T$ can be derived using the same method given for standard tasks in Section 2.
Simulation studies again showed that the EDM priority ordering scheme was an effective heuristic for this third model.

4. Sensitivity analysis

The above algorithms will, in the worst case, lead to tasks that will only just meet the event deadline. In safety critical systems some tolerances may be required. This is easily accommodated by setting the deadlines to be tighter than actually necessary.

An alternative approach is to undertake a sensitivity analysis and choose task periods based upon the results of that analysis. Consider a task set that is schedulable (i.e. all event deadlines are met). By what percentage can each event deadline be reduced and the system remain schedulable? This value gives a measure of how close the system is to being unable to guarantee the event deadlines. By using the task period values obtained by this process a system is defined that is balanced in its tolerance of any event deadline failure.

5. Conclusions

With a purely periodic system, task periods must be chosen so that the resulting implementation is sufficiently reactive. In this paper we have shown how event deadlines can be used to systematically fix task periods so that resource utilisation is minimised. The Event Deadline Monotonic priority ordering scheme was shown to be an effective (though not optimal) means of maximising the chance of finding a feasible implementation.

References