An Integrated Approach to Scheduling in Safety-Critical Embedded Control Systems

I. Bate (iain.bate@cs.york.ac.uk) and A. Burns (alan.burns@cs.york.ac.uk)
Real-Time Systems Group, Department of Computer Science, University of York, York, United Kingdom

Abstract. This paper describes an approach that has been developed over a number of years for the job of scheduling systems and providing evidence that timing requirements are met. The approach has been targeted at the safety-critical systems domain, and more specifically the development of control systems for jet engines. The work provides a usable computational model that supports the reuse of legacy systems. In addition, timing analysis has been developed that features low pessimism, low computational complexity and that is robust to change. The contributions of this paper are to show how standard timing analysis is often insufficient for real systems, presenting extensions to the standard analysis to give an integrated approach to verification, and providing a case study that demonstrates the appropriateness and benefits of the overall technique.

1. Introduction

Safety-critical systems are different to other systems because a failure to meet a requirement may lead to a catastrophic effect, e.g. an accident leading to loss of life. For this reason, there is a greater emphasis on verification and validation of software in safety-critical systems than with other types of system. In terms of scheduling, this leads to systems having hard deadlines which are those that have to be met under all circumstances. An evaluation of industrial practice for avionic systems confirms that the majority of safety-critical systems still employ a static scheduler. The cyclic scheduler being the most often used form of static scheduler.

Locke (Locke, 1992), amongst others, describe a number of problems with the static scheduler. These include having a restricted computational model (i.e. supporting only periodic tasks with a limited range of iteration rates) and poor maintainability (i.e. small changes within the system may have wide-spread effect). Development programmes such as Integrated Modular Avionics (IMA) call for much greater flexibility within systems. In some cases it is felt that it would be virtually impossible to achieve the ultimate goals of IMA using static scheduling (Grigg and Audsley, 1997). For instance, it is felt that the IMA standard ARINC 653 (ARINC, 1996) that uses static scheduling for partition

and communications scheduling may affect the ability to synthesise and maintain schedules (Audsley and Wellings, 1996). By moving to fixed priority scheduling the problems associated with static scheduling can be alleviated, resulting in significant technical and commercial benefits. This paper will address how this transition can be performed providing the maximum technical benefit in a cost effective manner without a significant risk to projects.

The contribution of this paper is to extend existing scheduling approaches to better address the specific scheduling needs of safety-critical embedded systems and in particular engine control systems. These systems have a number of characteristics and requirements that make them different to other real-time systems and to typical academic case studies. Examples of distinguishing features of a Safety Critical System are: a failure, e.g. a timing overrun, can have catastrophic consequences, system timing requirements (such as latency or event sequence) are not given in a form directly relating to scheduling (e.g. deadlines and offsets), and the temporal requirements include the co-ordination of multiple components. Also the system may contain legacy components which should be supported by the scheduling approach.

Section 2 provides evidence of why the standard assumptions are inappropriate and what the assumptions for scheduling should include. Section 3 extends the standard theory to reduce any limitations and provide an approach capable of developing uniprocessor systems. Section 4 shows how the approaches developed for uniprocessor systems can be migrated to distributed systems. Section 5 discusses some of the problems that were faced when moving the techniques into industrial practice. Finally, section 6 provides a summary of the contributions of this paper.

2. Limitations of Standard Approaches

There are many papers and books that describe the standard approach to scheduling and timing analysis, including (Joseph and Pandya, 1986; Katcher et al., 1993; Audsley et al., 1993; Burns and Wellings, 2001). The following subsections provide an argument of why the typical scheduling assumptions are invalid in practice and how the standard theory needs to be extended for use in real systems.

2.1. Preemptive Scheduling

The majority of papers on fixed priority scheduling make the assumption that the computational model is preemptive, i.e. tasks can in-

terrupt one another so that the highest priority runnable task is always the one being executed. However there are two other alternative approaches, non-preemptive scheduling where tasks are run to completion after they have begun executing and cooperative scheduling where tasks are executed until at pre-defined points during their execution they voluntarily suspend themselves. Non-preemptive scheduling is the one used in static scheduling. The following are advantages of the non-preemptive computational model over the other forms of scheduling.

1. Legacy software produced for a static scheduler can be reused without the need for significant re-design and re-verification. To safely use application software in a preemptive or cooperative environment, it is necessary to ensure that the effect of suspending the task's execution on the data and control flow cannot lead to ambiguities in the system performance. For example, interrupting software in the middle of writing back a set of new data could leave the system in an unsafe state. The problem of moving legacy software, designed based on an assumption of a non-preemptive environment, to a preemptive environment is worse than moving to a cooperative environment.

2. A preemptive or cooperative flow of control introduces a great deal more paths into the software which makes verification, in particular structural testing, more difficult.

The key disadvantage of the non-preemptive scheduler is that once the lower priority tasks have begun executing, higher priority tasks that become runnable are prevented from executing - this is referred to as blocking. Blocking makes it more difficult to meet the deadlines of tasks which often leads to tasks being broken up into smaller parts to meet temporal constraints. Hence the software engineering process is fundamentally affected by what is, in practice, a small aspect of the system.

For the first-time use of the technology in the safety-critical systems domain where approaches tend to be less ambitious, the advantages of non-preemptive scheduling are considered greater than the disadvantages and therefore the non-preemptive approach is to be adopted. However, the intention is to slowly migrate to a preemptive computational model and hence any techniques developed should ideally be usable with little modification in a preemptive scheduling framework. For similar reasons, the initial work is to be performed with entirely periodic tasks since these are currently used in static scheduling and the problem of verification is eased. However the approach should support a future transition to task sets with both periodic and aperiodic

tasks. Based on this restricted computational model, the standard timing analysis will have new sources of pessimism. For example with a non-preemptive scheduler, tasks cannot be interfered with after they have commenced execution. Whereas with a preemptive scheduler, it is assumed tasks can be interfered with until they complete their execution. Therefore, sources of pessimism in the timing analysis need to be identified and eliminated, where possible.

2.2. Precedence Relations

An important requirement for embedded control systems is the need to support transactions. A transaction is considered to be a number of tasks executing in a pre-defined order within an end-to-end deadline - often referred to as latency. An example of a transaction is where a task reads information from a number of sensors, another task processes the information and the final task outputs the information to an actuator. It is important to maintain the freshness of data, that is if the sensor data becomes too old before it contributes to an output of the system then its usefulness is diminished; indeed it may even become hazardous. This is particularly important in replicated systems where outputs have to be compared to determine the validity of data. If the outputs are not based on input data from a small time window, then the comparison is more difficult to do with confidence. Since the standard approaches for fixed priority scheduling do not accommodate the use of transactions, the approaches will need to be extended.

Some work has been performed to develop approaches to solve similar problems to ours (Gerber et al., 1994; Yerraballi, 1996; Gutierrez et al., 1995). These approaches suffer from similar common problems, which are:

1. an assumption is made that the task attributes are completely flexible. However in many cases a task may have multiple requirements placed upon it, e.g. it could be part of more than one transaction. Therefore changing the task's attributes to meet one requirement could cause another requirement previously met to be no longer satisfied;

2. the approaches are not intuitive, leading to difficulty in explaining the techniques and performing the techniques manually. The ability to explain the techniques to non-specialists is essential for certification and technology transfer. The ability to perform the techniques manually is essential as part of the review process which is an integral part of the development of any safety-critical system.

2.3. Poor Jitter Control

A key disadvantage of the standard approaches for fixed priority scheduling when compared to static scheduling is that jitter can be worse. In many cases a bound on output jitter is not specified in systems scheduled statically because it is assumed that jitter is negligible. However this assumption no longer valid with fixed priority scheduling because a task can be executed any time between its release and its deadline. Within control systems, jitter causes a reduction in the signal-to-noise ratio of the system which could lead to instability. Hence, a mechanism for controlling the jitter suffered by particular tasks is needed.

2.4. Support for Offsets

The vast majority of papers make the assumption that all tasks have a zero offset from a fixed time reference and have deadlines not greater than their period. Liu and Layland (Liu and Layland, 1973) show that under these condition the system has a single critical instance for all tasks. Offsets may be used in systems for a number of reasons, including guaranteeing the separation between two tasks, spreading the resource load of tasks with tight deadlines so that the system's tasks are schedulable and for guaranteeing tasks maintain a particular precedence order. However for a system with offsets there may no longer be a critical instance. When tasks have offsets, Leung and Merril (Leung and Merril, 1980) provides exact analysis that involves showing every instance of every task on a particular processor is schedulable over the period [Maximum Offset of Any Task, 2×Least Common Multiple of the Tasks' Periods + Maximum Offset of Any Task). A problem with this approach is that the exact analysis can be intractable; dependent on the periods of the tasks being analysed. Hence, a practical mechanism for using and verifying the use of offsets is needed that has lower computational complexity than the exact test without introducing unmanageable pessimism.

2.5. Completing the Specification of Scheduling Requirements

The specification of timing requirements are incomplete in a number of respects related to the means by which requirements can be met. For instance, the systems engineers may produce a requirement that a control system is stable within certain bounds. This system requirement leads to timing requirements for jitter and period which can be met in a number of ways. For example, a stability requirement can be met by tasks having a small period and a relaxed jitter constraint, or a larger

period but a tighter jitter constraint. Related to this, there are other requirements that are flexible. If an offset is used to ensure a task only executes after another has completed, then the offset can be increased from the minimum that ensures the execution order to provide benefits to the system. The advantages of doing this could include increasing the likelihood of the system meeting its timing requirements and increasing the robustness to change.

2.6. Kernel Overheads

Kernel overheads in a system can consume significant amounts of processing time. If the overheads becomes too large or if they occur at the wrong time, then it can cause the system to be unschedulable. Ignoring the overheads means that the analyses cannot guarantee to correctly predict whether deadlines are met. There are papers by Strosnider (Katcher et al., 1993) and Burns (Burns et al., 1995b), amongst others, that modify the standard timing analysis for certain forms of overhead. However, to our knowledge no published work investigates how to trade-off the different design decisions that need to be made when moving from a static scheduler. An objective for this work is to control the amount of overheads and when they occur, and account for this overhead in the timing analysis using the existing theory as a basis.

2.7. Fault Tolerance

One of the key differences between non-safety-critical systems and safety-critical systems is the need for high reliability and availability which is provided through fault tolerance. In terms of scheduling there are two main aspects to fault tolerance, which are: detecting when a task executes for longer than expected, i.e. there is a timing overrun; and detecting when a sporadic task is released at a higher rate than expected. Detecting faults related to sporadic tasks is often handled by a simple test within the task release part of the kernel software.

The mechanism often used in static scheduling to detect timing overruns is to enforce a rule that no task is executing when there is a clock tick, i.e. a periodic interrupt used to trigger various kernel services. Then, the timing watchdog simply has to check whether there is a task executing when the clock tick occurs. When moving from static scheduling, the mechanism for detecting timing overruns may have to be altered and the timing analysis extended to account for the new overheads. With fixed priority scheduling, there are many mechanisms for detecting these forms of timing faults that have different detection times, granularity of identification and overheads. However,

to our knowledge no published work investigates how to trade-off the different design decisions that need to be made. Therefore an objective of this work is to investigate how to perform the necessary trade-off within the constraints of our problem domain.

2.8. Distribution

Few safety-critical systems are implemented as distributed systems. However the scheduling and timing analysis of these distributed systems is often made considerably simpler by an assumption that there are no temporal relationships between the processors. For future systems, much greater levels of integration are going to be introduced in order to reduce the number of processors needed and to increase the system's effectiveness both functionally and non-functionally (Edwards, 1994). An example of this is if engine and flight control systems are integrated, then the control algorithms used for both can be optimised based on greater levels of shared information to improve performance and efficiency. Having a single processor execute the software for both these functions is difficult due to the processing needs, the physical separation issues and the impact on fault tolerance.

The key challenge when meeting temporal requirements in a distributed system is handling distributed transactions in a maintainable way without too much pessimism. The techniques developed for a uniprocessor system need to be migrated to a distributed system in a manner that is acceptable to industry and their certification authorities.

3. Extending the Standard Approaches

This section presents our work to extend the standard analysis to resolve the limitations discussed in section 2. During this work, four constraints were used to guide the work in order to reduce opposition to the use of the overall technique.

1. *Certification* - whether sufficient evidence of the approach's integrity can be generated to satisfy the certification authorities.

2. *Understanding* - a key criterion is whether the technique can be understood in enough detail by non-specialists so that it is adopted. It is accepted that this criterion is difficult to measure.

3. *Reuse* - wherever possible the changes to the scheduling software should not affect the rest of the system allowing the majority of the components of existing systems to be reused.

4. *Sufficiency* - whether the technique is flexible and efficient enough to allow the engineer to design, implement and maintain the system with acceptable levels of effort. To meet the previous constraints, it is necessary to develop techniques that are not optimal in the general sense. This effectively reduces the likelihood that the approach may find a solution when one actually exists. Therefore the developed techniques should be tailored to be as practical and effective as possible for the intended domain.

3.1. Moving to Non-Preemptive Scheduling

Harter (Harter, 1984; Harter, 1987) presents the first work that attempts to solve the computational complexity problems of the exact analysis. Harter developed timing analysis that could be performed in pseudo-polynominal time to show whether the task set is schedulable, i.e. all tasks meet their deadline. The standard timing analysis for each individual processor is solved using equation (1) taken from (Harter, 1987). The analysis is valid for task sets with a critical instant. Harter's analysis assumes there are a fixed number of tasks, all of which have a fixed unique priority, zero offset, and the deadlines are not greater than the period.

$$R_i = C_i + B_i + I_i \tag{1}$$

where i is a task in the set of tasks for a given node
R_i is the Worst-Case Response Time (WCRT) of task i
C_i is the worst-case execution time of task i
B_i is the worst-case blocking time suffered by task i
I_i is the worst-case interference suffered by task i

The blocking time, B_i, is the longest time that a lower priority task can prevent task i when it is runnable. The blocking time is dependent on the computational model that is being used. In an idealised preemptive model, the blocking time should be zero. However cases exist, particularly with shared resources, where some blocking may need to be accounted for.

The interference a task suffers is the maximum utilisation from the critical instant for its higher priority tasks before it executes for the first time. The interference is calculated using equation (2) which represents the sum of the utilisations over the duration of interest for all the higher priority tasks than task i. The utilisation is the product of the number of times the task can execute and its worst-case execution time. The number of times a higher priority task can execute is found by rounding up the result of the time during which interference may occur (i.e. the

response time of the task being analysed) divided by the period of the higher priority task.

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2}$$

where $hp(i)$ is the set of higher priority tasks than task i

Equation (1) is solved by forming a recurrence equation as shown in equation (3).

$$R_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{3}$$

with $R_i^0 = C_i$
which terminates when $R_i^{n+1} = R_i^n$, or $R_i^{n+1} > D_i$.
where D_i is the deadline of task i,
T_i is the period of task i

The final value for the worst case response time is taken when the analysis converges, i.e. $R_i^{n+1} = R_i^n$, or when the worst-case response time exceeds the task's deadline, i.e. $R_i > D_i$.

The analysis in this section was also derived and published independently by Joseph and Pandya as the Time Dilation Algorithm (Joseph and Pandya, 1986; Joseph, 1985), and by Audsley et al (Audsley et al., 1991).

3.1.1. *Improved Blocking Model*

A drawback of using a non-preemptive scheduling model, instead of a preemptive model, is that tasks tend to suffer greater blocking. The blocking time caused by lower priority tasks for the non-preemptive model is expressed in equation (4). In many system, the blocking time can be prohibitive, especially when it is considered that the lower priority tasks are often the most computationally intensive.

$$B_i = max_{k \in lp(i)}(C_k) \tag{4}$$

where $lp(i)$ is the set of lower priority tasks than task i

Section 2.1 has already stated that the computational model may consist entirely of periodic tasks. Therefore, timing analysis is presented that reduces pessimism based on a computational model that features mainly periodic tasks. Under these circumstances, blocking is reduced by noting that low priority periodic tasks that are always released at the same time as task i cannot cause blocking. This is captured by equation (5).

$$\begin{aligned} \forall n, m &\in \mathbb{N} \\ k \in lp(i) &: (T_k \neq nT_i \wedge T_i \neq mT_k) \vee R_k > T_i \\ B_i &= max_{\forall k}(R_k - \Delta) \end{aligned} \tag{5}$$

where Δ is one clock cycle

If aperiodic tasks feature in the set of tasks with a lower priority than the task being analysed, then equation (4) must be used.

Theorem 1
A lower priority task, k, cannot block a task if the lower priority task is always released at the same time as an instance of task i and it always completes before the next instance of task i.

Proof
If task k is always released at the same time as task i (i.e. the condition in equation (6) holds), then task i always completes before task k commences executing by virtue of its higher priority. If this condition holds, then task k can't block task i if it always completes before task i is re-released (i.e. the condition in equation (7) holds).

$$T_i = nT_k \vee mT_k = T_i, \text{ where } n, m \in \mathbb{N} \tag{6}$$

$$R_k \leq T_i \tag{7}$$

□

Instead of the usual expression, $B_i = max_{\forall k}(C_k)$, equation (5) uses the term $B_i = max_{\forall k}(C_k - \Delta)$.

Theorem 2
The blocking time is represented by one clock cycle less than the maximum worst-case execution time of any task that can cause blocking.

Proof
The blocking time is reduced since at least one clock cycle of task k must already have occurred otherwise the higher priority task would execute. Hence, the maximum of $(C_k - \Delta)$ for all lower priority tasks that can cause blocking (k) represents a safe estimate. □

3.1.2. *Improved Interference Model*
Another area where pessimism could be introduced is in the interference model. The purpose of this section is to investigate where the pessimism arises and how it may be reduced. The interference term, given in equation (2), is taken from the standard timing analysis and represents

the maximum amount of time higher priority tasks can execute before the task being analysed.

Equation (2) is pessimistic, because the numerator of the interference term assumes a preemptive model. With non-preemption, a higher priority task cannot commence executing if a lower priority task has begun executing. Instead, the higher priority task must wait for the lower priority task to complete. Therefore, the interference can be improved as shown in equation (8). In this equation, the numerator is reduced by the worst-case execution time of task i, task i being the task whose interference is being calculated. However, to prevent anomalies due to edge effects, caused by a task being released at the same time as another is dispatched, then *one* clock cycle (i.e. Δ) is added. In other words, the extra clock cycle ensures task i has actually commenced its execution.

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i - C_i + \Delta}{T_j} \right\rceil C_j \tag{8}$$

3.1.3. *Optimality of DMPO with the Restricted Computational Model*
Leung and Whitehead (Leung and Whitehead, 1982) show that the DMPO (Deadline Monotonic Priority Ordering is where the highest priority tasks have the shortest deadline) approach is optimal for the preemptive computational model based on the assumptions that all tasks have zero offsets and deadlines less than or equal to their corresponding period.

However, the change of computational model and associated analysis means that DMPO is no longer optimal as shown by the following example. Table I shows that the blocking caused by task C with DMPO leads to an unschedulable task set. Table II shows that if the priorities of tasks B and C are swapped, then the task set is schedulable. Hence, DMPO is sub-optimal for a non-preemptive scheduling model. Audsley (Audsley, 1993) provides an optimal algorithm for assigning priorities, however this is computationally complex. For this reason and because the cases are rare in which DMPO does not find the schedulable solution, DMPO has been used during the course of this work.

3.2. Support for Precedence Relations

This section presents our extensions to the standard scheduling approach so that evidence is provided that the requirements of precedence

Table I. Schedulability Results with DMPO

Id	T	D	C	B	P	R	Met?
A	4	4	2	3	1	5	N
B	16	15	1	0	2	3	Y
C	16	16	4	0	3	7	Y

Table II. Schedulability Results without DMPO

Id	T	D	C	B	P	R	Met?
A	4	4	2	2	1	4	Y
B	16	15	1	0	3	11	Y
C	16	16	4	0	2	6	Y

and end-to-end deadlines for a uniprocessor system are met. Our approach consists of two independent parts, which are deriving the task attributes (without knowledge of the tasks' worst-case execution times) so that the timing requirements may be met, and then using the tasks' worst-case execution times to verify that the timing requirements are met. This section provides the first part of the approach (i.e. the task attribute assignment); the second part (i.e. the timing analysis) has already been covered in section 3.1.

The proposal is that the necessary timing requirements can be handled by setting deadlines to appropriate values. The system's timing requirements are then verified by simply proving that the tasks' offsets are enforced in the scheduler and using timing analysis to show the tasks' deadlines are met. An alternative approach would be to meet the transaction's requirements by just setting priorities. The disadvantages of the alternative approach are that analysis other than the standard timing analysis is needed to verify the system's timing properties and it is harder to manually validate that the requirements are met. It should be noted that the ability to manually validate results is important during the review phases of the overall certification process.

The approach to meeting the transaction requirements is based on reducing the tasks' deadlines in a systematic manner until the requirements are met and that the tasks' deadlines are reduced by the minimum amount. Characteristics of the transaction's requirement include a task's period could be longer than the period of the task

that follows it in the transaction and the transaction's deadline could be greater than its period. Having unequal task periods means it may not be necessary or possible to reduce all the deadlines so that perfect precedence between the tasks of the transaction is maintained.

The task attributes are generated using Algorithm 1. The motivation behind the algorithm is that it does not simply rely on the tasks' deadlines being reduced so that perfect precedence is maintained. Instead, while the transaction requirement is not met, the longest deadline is reduced by Δ, and if this causes any of the preceding sequence of tasks in the transaction to have the same deadline, then their deadlines are also reduced by Δ. Since the algorithm does not simply enforcing precedence, the resultant deadlines should be larger and hence the likelihood of meeting the timing requirements is increased. The benefit of the approach in Algorithm 1 is that if the deadlines are larger, then there is a greater chance of a schedulable solution. The algorithm also contains a manipulation of the *TDAC* (Task_Deadlines_Are_Changing) flag. The use of this flag becomes apparent later.

Algorithm 1 - *for Dealing with Transaction Deadlines*

for each task i in the transaction
  if the following task j has an equivalent deadline then
    reduce the task i's deadline by Δ
    assign the value of false to the TDAC flag
  while (transaction deadline not met AND (all tasks' deadlines > 0))
    assign the value of true to the TDAC flag
    reduce the longest deadline of any task in the transaction by Δ
    for each task in the transaction
      if the task j has an equivalent deadline then
        reduce the task's deadline by Δ
        assign the value of false to the TDAC flag

3.2.1. *Justifying Correctness*

The key to the integrity of the approach is ensuring the correctness of the checks that the tasks' deadlines and the transaction's deadline are met. The task's deadlines are checked using the timing analysis already discussed in section 3.1. There are two phases to the calculation of the transaction's response time, which are: establishing the particular release of each task in the transaction, and the completion time for the releases of interest. This is carried out by starting with the first task and working through the tasks in the defined precedence order. Equation

(9) can be used for calculating the task instance that is relevant to the transaction.

$$n_t = \left\lceil \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t + \Delta}{T_t} \right\rceil \tag{9}$$

where t is the t^{th} task (assuming tasks are ordered according to precedence) in the transaction,
n_t is the instance of the t^{th} task, where $n_t \in \mathbb{N}$, and
n_1 is 1

The worst-case response time of the ${n_t}^{th}$ instance of task t, $R_t^{n_t}$, from the transaction's critical instance (i.e. time zero) is given in equation (10).

$$R_t^{n_t} = (n_t - 1)T_t + R_t \tag{10}$$

Using an initial value of $n_1 = 1$, the response time of each task in the transaction can be calculated by starting with the first task and taking each task in turn. However for the verification to be useful, the correctness of equation (9) needs to be justified.

Theorem 3
The value, $R_t^{n_t}$, represents the worst-case response time of the t^{th} task of the transaction, where the tasks are ordered as defined by the precedence constraint.

Proof
Consider task *t-1*, which is the task in the transaction that precedes the task whose instance is required. The worst-case response time of the n^{th} instance of the task *t-1*, $R_{t-1}^{n_{t-1}}$, can be represented by equation (11).

$$R_{t-1}^{n_{t-1}} = (n_{t-1} - 1)T_{t-1} + R_{t-1} \tag{11}$$

The next instance of task t must satisfy the condition in equation (12), i.e. the response time of task t must be after task $t-1$, but the previous instance of task t must be before task $t-1$.

$$(n_t - 2)T_t + R_t < R_{t-1}^{n_{t-1}} < (n_t - 1)T_t + R_t \tag{12}$$

Therefore using equation (10),

$$(n_t - 2)T_t + R_t < (n_{t-1} - 1)T_{t-1} + R_{t-1} \tag{13}$$

$$n_t < \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + 2T_t}{T_t} \tag{14}$$

and the instance of task t that maintains precedence, n_t, must complete after the ${n_{t-1}}^{th}$ instance of task *t-1*,

$$(n_t - 1)T_t + R_t > (n_{t-1} - 1)T_{t-1} + R_{t-1} \tag{15}$$

$$n_t > \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t}{T_t} \tag{16}$$

We can also state that

$$\begin{aligned} i &\in \{\text{Tasks in the Transaction}\} \\ \forall i &: n_i \geq 1 \end{aligned} \tag{17}$$

The conditions in equations (14), (16) and (17) are satisfied by the value of n_t in equation (18). The reason this value is used is because it is the smallest value that satisfies the necessary conditions which means that the results value of n_t and $R_t^{n_t}$ are kept to a minimum.

$$n_t = \left\lceil \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t + \Delta}{T_t} \right\rceil \tag{18}$$

□

3.2.2. *Example of the Approach*
For example, consider the transaction requirements of:

1. task precedence constraint is task A followed by task B followed by task C followed by task D,
2. an iteration rate of 100 and
3. an end-to-end deadline of 145.

The periods of tasks A, B, C and D are 50, 100, 100 and 50 respectively. In this case, task B cannot always follow task A because task B has a slower update rate.

Without altering tasks' deadlines, the requirement is not met because the transaction's response time is 250. By an approach based on enforcing precedence by setting deadlines (i.e. $D_A = 50 - 3\Delta$, $D_B = 50 - 2\Delta$, $D_C = 50 - \Delta$ and $D_D = 50$), the transaction requirements could be met with a worst-case transaction response time of 50. However using Algorithm 1 larger deadlines are assigned, the task attributes are (i.e. $D_A = 50$, $D_B = 100 - 2\Delta$, $D_C = 100 - \Delta$ and $D_D = 50$), as shown in Figure 1. From this Figure, it can be seen that the task attributes result in the transaction deadline being met. The benefit of the larger deadlines is that the tasks have a lower priority and hence the likelihood of finding a schedulable task set is increased.

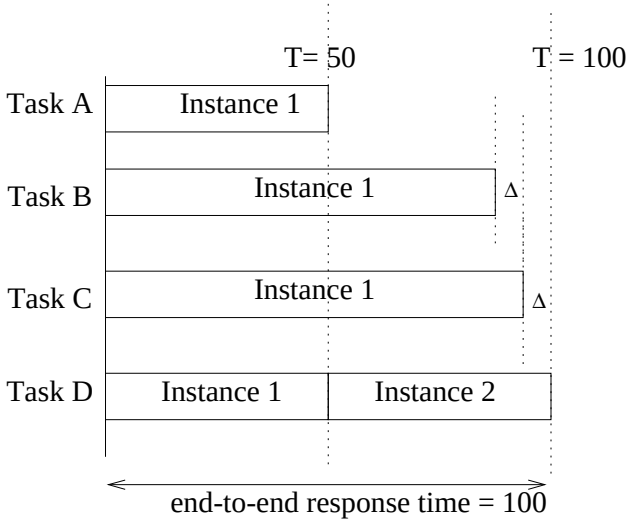

Figure 1. Attributes produced with Algorithm 1

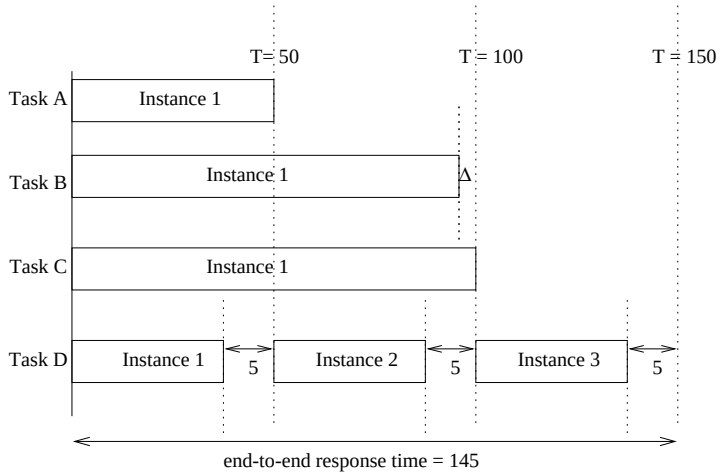

Figure 2. Alternative attributes

3.2.3. *Optimality of the Approach*

With any approach it is important to ascertain whether it is optimal. If it is not optimal, then any deficiencies should be highlighted so that appropriate action can be taken where necessary. First impressions suggest that the approach may be optimal because the deadlines of the tasks are the maximum possible whilst still meeting the timing requirements. However, trying to prove optimality is difficult because of the effect of complex interactions with other tasks and transactions.

Figure 2 presents an alternative solution to the one derived in the previous section using Algorithm 1. It is impossible to judge which of these is the better solution without knowledge of all the tasks' execution times, priorities and deadlines. On the one hand, the task attributes in Figure 1 cause less interference through task D. While the task attributes in Figure 2, cause less interference through tasks B and C. Hence, it can not be stated that the approach in Algorithm 1 is optimal.

In our experience using the task attribute assignment approach, no specific examples have been found where the issue raised in this section has caused problems. A case study on a real system is presented in (Bate, 1998) along with a discussion of potential strategies that can be followed if problems are encountered.

3.3. Controlling Jitter

For similar reasons to those given in section 3.2, the jitter constraint should be handled using offsets and deadlines rather than priorities. If a task has a deadline equal to the jitter constraint, then the constraint is met because the window of allowed execution is constrained between the critical instance and the time allowed by the jitter constraint. However, using this approach would be pessimistic since there is a minimum processing time for each task that adds no variability, i.e. the task's execution time can only vary between its best-case response time and its worst-case response time. Therefore, the deadline can be calculated using equation (19).

$$D_i = J_i + BCRT_i \tag{19}$$

where $BCRT_i$ is the best case response time of task i.

The best case response times of tasks can be generated in two stages. Initially, the value can be set to zero, but later it can be calculated using either exact analysis techniques or the method proposed by Harbour et al (Gutierrez et al., 1998).

The analysis given in equation (19) could cause problems when the transaction deadline is less than or equal to its period, and there is a jitter constraint placed on the last task in the transaction. This would lead to all the preceding tasks having tight deadlines, which could affect schedulability. The solution to the problem is the use of offsets to constrain the variation in the task's execution at the end of the allowed execution time of the transaction.

The approach for choosing tasks' deadlines and offsets can be represented by Algorithm 2. Deadlines are used to control the jitter. However, the actual algorithm is motivated by the desire to spread resource

usage through the transaction's available execution window and hence improve the likelihood that the timing requirements are met. The deadline relative to the task's release is constrained using Algorithm 2. The algorithm only alters the task's release time in the case of the last task in the transaction where the transaction deadline is less than or equal to the transaction's period. In this case, all preceding tasks in the transaction have their deadline capped at the last task's release time.

The approach defined in Algorithm 2 cannot be described as optimal. If a large number of tasks (specifically ones that are not part of the particular transaction) are already phased to execute during the time interval $[O_i, D_i]$, then the approach may lead to an unschedulable solution. In this circumstance an approach based entirely on equation (19) and Algorithm 1, instead of Algorithm 2, may increase the likelihood of a schedulable solution. However, in practice this is unlikely to be the case and our use on real systems has not caused any problems. To our knowledge, there is no optimal approach published for dealing with jitter.

Algorithm 2 - *Algorithm for Dealing With Jitter*

for each task (denoted i) in the system
  if task i has a jitter constraint then
    if task i is not the last task in the particular transaction then
      $D_i = J_i + BCRT_i$
      assign the value of true to the TDAC flag
    else
      if the transaction deadline > transaction period then
        $D_i = J_i + BCRT_i$
        assign the value of true to the TDAC flag
      else
        if $D_i >$ transaction deadline then
          $D_i =$ transaction deadline
          assign the value of true to the TDAC flag
        $O_i = D_i - (J_i + BCRT_i)$
        for all preceding tasks (denoted j) in the transaction
          if transaction deadline is not met then
            if $D_j >$ transaction deadline - $(J_i + BCRT_i)$ then
              $D_j =$ transaction deadline - $(J_i + BCRT_i)$
              assign the value of true to the TDAC flag

An integrated task attribute assignment approach is given in Algorithm 3. The motivation behind this algorithm is to combine the approaches derived so far in a manner that accounts for tasks that

have to satisfy multiple timing requirements. Firstly, the tasks' offsets and deadlines are constrained to allow for jitter. Then, tasks' deadlines are altered so the transactions' deadlines are met in an iterative fashion until the task's deadlines no longer change. When the deadlines are no longer changing (indicated by the TDAC flag being false at the end of the while loop), then we can consider the interactions of multiple timing requirements to have been accounted for.

Algorithm 3 - *Overall Algorithm for Priority Assignment*

```
initialise TDAC flag to true
apply algorithm 2
while the TDAC is true
  assign the value of false to the TDAC flag
  for each transaction in the set of transactions
    apply algorithm 1
if (transaction deadline not met AND (all tasks' deadlines > 0)) then
  write error message to standard output
else
  apply priorities to the tasks by the DMPO approach
  perform timing analysis of the task set
```

This section has shown how the standard scheduling approach has been extended to control jitter and how the task attribute assignment approaches in this section and section 3.2 may be combined.

3.4. Support for Offsets

Experience in performing timing analysis of real industrial systems leads to the observation that there is likely to be a pattern in the types of offset used. The use of offsets is likely to be analogous to that of the cyclic scheduler, i.e. the processing frame is split into manageable chunks with tasks allocated to the different partitions using offsets. Table III can be used to illustrate the type of requirements that result. In Table III, tasks A, B, C and D have offsets such that their computation is relatively evenly spaced over the period of 25000 units.

Based on this observation, this section presents a tractable but non-exact 'composite' approach to the problem of analysing task sets featuring offsets. The motivation behind this approach is that if a composite task with zero offset can represent the tasks with non-zero offsets and the same period, then the offsets are eliminated and the analysis

presented in section 3.1 may be employed. Hence computational complexity is reduced and existing (and accepted) analyses can be used. The approach takes advantage of the fact that the offsets are relatively evenly spread through time, partially due to the static origin of the requirements, i.e. from statically scheduled systems. The benefit of the composite task approach is that the computational complexity is kept sufficiently low, while allowing resource usage to be spread through time.

Table III. Example task set

Id	T	O	D	C
A	25000	0	6000	2000
B	25000	6250	12000	1500
C	25000	13000	18000	1500
D	25000	18000	25000	1500
E	50000	0	50000	2000
F	100000	0	100000	1000
G	200000	0	200000	1000
H	1000000	0	1000000	2500

The following are the steps followed to derive each composite task.

1. Define a set, ST, that consists of the tasks with the same period and non-zero offsets, and a maximum of one arbitrarily chosen task (if one exists) with the same period and zero offset.

   For the task set in Table III, the set ST consists of four tasks B, C and D (by virtue of having the same period and non-zero offset), and task A (by virtue of having the same period and zero offset).

2. Define a set, ST1, that is equivalent to the set ST except for any task with zero offset in set ST has its offset changed to equal its period.

   For the task set in Table III, the set ST1 consists of four tasks B, C and D as well as one other denoted as X. Task X has a period equal to the period of tasks B, C and D (i.e. 25000) and an offset equal to its period.

3. A sequence, ST2, is defined which is an ordered (by increasing value of offset) version of set ST1.

For the task set in Table III, the sequence ST2 is defined as:

a) *task B (offset is 6250) followed by*
b) *task C (offset is 13000) followed by*
c) *task D (offset is 18000) followed by*
d) *task X (offset is 25000).*

i.e. $ST2 = \{B,\ C,\ D,\ X\}$

4. A sequence, ST3, is defined with the same order as sequence ST2, but with value equal to the offset of the member in sequence ST2 divided by the index (range 1..N) into the set.

   For the task set in Table III, the sequence ST3 is defined as:

$$\begin{aligned} ST3 &= \left\{ \frac{O_B}{\text{index of member B in sequence ST2}}, \right. \\ &\quad \frac{O_C}{\text{index of member C in sequence ST2}}, \\ &\quad \frac{O_D}{\text{index of member D in sequence ST2}}, \\ &\quad \left. \frac{O_X}{\text{index of member X in sequence ST2}} \right\} \\ &= \left\{ \frac{6250}{1}, \frac{13000}{2}, \frac{18000}{3}, \frac{25000}{4} \right\} \\ ST3 &= \{6250, 6500, 6000, 6250\} \end{aligned} \tag{20}$$

5. A sequence, ST4, is defined which is an ordered version of the worst-case execution times of the tasks in set ST. The ordering is in accordance with decreasing value of worst-case execution time.

   For the task set in Table III, the sequence ST4 is defined as $\{C_A, C_B, C_C, C_D\} = \{2000, 1500, 1500, 1500\}$.

6. The composite task is given a period equal to the minimum value of any member in the sequence ST3.

   For the task set in Table III, the period of the composite task is the minimum of 6250, 6500, 6000 and 6250. Therefore, the period of the composite task is 6000.

7(a). In a simplified version of the composite analysis, the composite task is given a worst-case execution time equal to the first member in the sequence ST4, or in other words the member with the maximum worst-case execution time.

For the task set in Table III, the composite task's worst case execution time is 2000.

7(b). Alternatively, the worst-case of the composite task for a particular response time R_i can be calculated using equation (21).

$$C_{COMP} = \frac{\sum_{\forall q:q \in ST} \left\lceil \frac{R_i - O_q}{T_q} \right\rceil ST4(\#q)}{\left\lceil \frac{R_i}{T_{COMP}} \right\rceil} \tag{21}$$

where ST4 (#q) represents the q^{th} member of ST4, and
#q is the index into the set ST in the range
[1,Number of Tasks in Set ST].

For the task set in Table III with a value of R_i of 27000, the worst-case execution time is calculated as shown below. The value is effectively reduced from 2000 to 1700, which is an improvement of 15%.

$$\begin{aligned}
C_{COMP} &= \frac{\sum_{\forall q:q \in ST} \left\lceil \frac{R_i - O_q}{T_q} \right\rceil ST4(\#q)}{\left\lceil \frac{R_i}{T_{COMP}} \right\rceil} \\
&= \frac{\sum_{\forall q:q \in \{tasks\ A,\ B,\ C,\ D\}} \left\lceil \frac{R_i - O_q}{T_q} \right\rceil ST4(\#q)}{\left\lceil \frac{27000}{6000} \right\rceil} \\
&= \frac{\left\lceil \frac{R_i - O_A}{T_A} \right\rceil ST4(1) + \left\lceil \frac{R_i - O_B}{T_B} \right\rceil ST4(2)}{5} \\
&+ \frac{\left\lceil \frac{R_i - O_C}{T_C} \right\rceil ST4(3) + \left\lceil \frac{R_i - O_D}{T_D} \right\rceil ST4(4)}{5} \\
&= \frac{\left\lceil \frac{27000-0}{25000} \right\rceil 2000 + \left\lceil \frac{27000-6250}{25000} \right\rceil 1500}{5} \\
&+ \frac{\left\lceil \frac{27000-13000}{25000} \right\rceil 1500 + \left\lceil \frac{27000-18000}{25000} \right\rceil 1500}{5} \\
\therefore C_{COMP} &= 1700
\end{aligned} \tag{22}$$

8. The composite task is given a deadline equal to the minimum relative deadline for any task in the set ST. The relative deadline is classed as the task's deadline minus the task's offset.

For the task set in Table III, the composite task's deadline is equal to the minimum of:

a) $D_A - O_A = 6000$

b) $D_B - O_B = 6250$

c) $D_C - O_C = 5000$

d) $D_D - O_D = 7000$

$\therefore D_{COMP} = 5000$

9. Replace the tasks that form the composite task in the task set to be analysed (i.e. the members of set ST) with the composite task.

   For the task set in Table III, tasks A, B, C and D are replaced with the task COMP. The resulting task set consists of the tasks COMP, E, F, G and H as shown in Table IV. There are two columns of worst-case execution times in Table IV. C_{SIMP} is the worst-case execution time calculated using the simplified approach in step 7(a). Whereas, C_{IMPR} is the improved value of worst-case execution time calculated using the approach in step 7(b).

Table IV. Example task set after the composite task replaces the tasks with offsets

Id	T	O	D	C_{SIMP}	C_{IMPR}
COMP	6000	0	5000	2000	1700
E	50000	0	50000	2000	2000
F	100000	0	100000	1000	1000
G	200000	0	200000	1000	1000
H	1000000	0	1000000	2500	2500

The above steps ensure that task COMP will always induce equal (or more) interference than the tasks it replaces (in the analysis) (Bate, 1998). To understand whether the composite form of analysis is effective requires comparison with the exact approach. Analysis was performed with task set characteristics generated pseudo-randomly within defined ranges. The task set characteristics were:

1. The iteration rates in the range [25, 1000].
2. The worst-case execution time of the tasks are in the range (0, 2.5].
3. The offsets were assigned either randomly within the range (0, task iteration rate).

4. The deadlines were maximised so that:

   a) Tasks without offsets have a deadline equal to their period.

   b) Tasks with offsets have a deadline equal to the offset of the next task in the sequence *ST2*. In the case of the task with the largest offset at a particular iteration rate (i.e. the last member of sequence *ST2*), the deadline is equal to the period of the task.

5. A value for effectiveness is produced over a 1000 samples. Effectiveness is the percentage of task sets calculated as schedulable, compared to exact analysis.

Figure 3 shows the effectiveness of the composite approach (labeled COMP) with C_{COMP} simply taken to be the maximum WCET of any task in sequence ST4 compared to the actual composite approach (labeled IMPR) where C_{COMP} is calculated using equation (21). For comparison purposes, results are provided for the analysis where offsets are ignored (labeled SIMP). The two numbers (X,Y) at the end of each label (e.g. COMP_X_Y) indicate the resource range (i.e. the resource utilisation of the task sets is in the range [X%,Y%)) for the task sets. Figure 3 shows that the improved form of composite approach performs more effectively (up to 50% more effective) than either of the other two approaches. The Figure shows the significant benefit of the improved approach for calculating C_{COMP}. The benefit is particularly apparent for higher utilisation levels when the removal of any pessimism is particularly important. The lines labeled FREE will be explained in the next section.

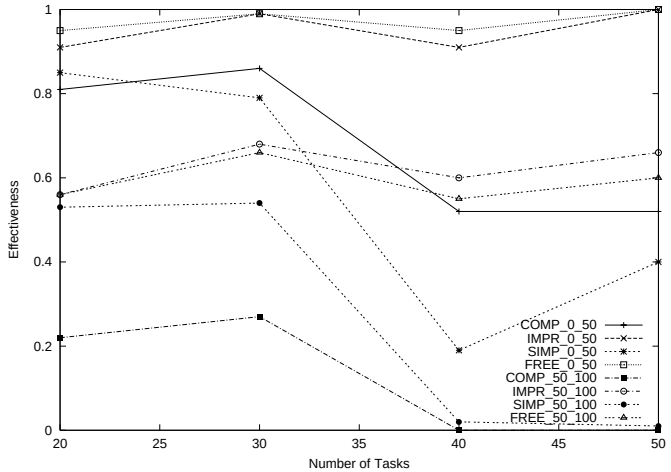

Figure 3. Comparison of the Approaches for Offset Analysis

This section has shown how the standard timing analysis can be extended to cater for offsets in a tractable manner with improved pessimism over an approach that ignores offsets. One of the principal benefits of the technique is the fact the existing analysis can be used with a limited amount of pre- and post-processing. There are a number of benefits of being able to reuse the analysis including, existing tools and training can still be used and only a limited amount of extra certification evidence is needed.

3.5. Completing the Specification of Scheduling Requirements

The use of offsets is an example of how the flexibility in the task attributes can be taken advantage of. The rigorous treatment of offsets is a new area that allows scope to optimise the chosen timing constraints. Consider a transaction requirement specified for two tasks which has a jitter constraint placed on the second task. As long as the necessary precedence is maintained for the two tasks (i.e. the offset of the second task is not reduced) and the second task meets its required deadline, then the size of the second task's offset may be increased.

Algorithm 4 - *Algorithm for determining a task's offsets*

for each task in the set ST (loop_index =0,1, ..)
  calc. offset $= \frac{loop\_index}{\text{no. of tasks in set ST}}$ **Period of tasks in set ST**
  if (the current offset of the task < calculated offset)
    offset = calculated offset

Algorithm 4 provides an approach for choosing the offset of tasks in a flexible manner. The motivation behind this approach is that increasing the tasks' offsets so that their release and execution is more evenly spread can help improve the schedulability of the task set. Figure 3 shows how this approach (labeled FREE) helps improve schedulability compared to the composite analysis without it (labeled COMP). The results show that in general using Algorithm 4 improves the chance of a schedulable solution being found.

3.6. Modifying the Infrastructure to Release Tasks and Accounting for the Overheads

In general, there are two principal forms of task release, with 'tick driven' the system has a periodic interrupt that triggers the task release mechanism and other kernel services, and 'time driven' where the task

release mechanism is executed when an event occurs (e.g. rather than after the periodic interrupt, when a task completes execution tasks are released based on the value of a real-time clock). In (Audsley et al., 1996) it was shown that there were significant problems with both of these approaches.

The tick driven approach can result in high amounts of release jitter or kernel overheads dependent on the clock tick rate chosen. Audsley (Audsley, 1993) shows that the release jitter for a given task can be calculated using equation (23). The overhead can be calculated using equation (24) that effectively models the overhead as a task with period T_{clk} and worst-case execution time C_{clk} (Katcher et al., 1993; Burns et al., 1995b). It should be noted that the clock tick preempts the execution of other tasks. If the clock tick rate is too low, then the tasks may suffer too much release jitter to be schedulable. If the clock tick rate is too high, then the kernel overheads are increased because too many checks are performed.

$$J_i = T_{clk} - gcd(T_{clk}, T_i) \tag{23}$$

where gcd is the greatest common divisor

$$\begin{aligned} M &= \left\lceil \frac{T_{min}}{T_{clk}} \right\rceil \\ C_{clk} &= MC_{first} + (\#j - M)C_{sub} \\ U_{overhead}(t) &= \frac{t}{T_{clk}} C_{clk} \end{aligned} \tag{24}$$

where $U_{overhead}(t)$ is the utilisation due to the overhead of the clock tick at time t

In contrast, the time driven approach eliminates the release jitter problem but the method of releasing tasks is a fundamental change which affects reuse (particularly of the safety evidence) and the kernel overheads may be higher due to the checks being performed frequently.

In (Audsley et al., 1996) a hybrid approach to task release is proposed that is a compromise of the two techniques. The hybrid approach releases the majority of tasks based on a clock tick, with a few tasks (those whose rate is not a harmonic of the clock tick rate) released in a time driven manner. The tasks released in a time driven manner incur an overhead of C_{time} for each time they are released. The clock tick rate, T_{clk}, should be chosen using equation (25) so that the minimum value for the overhead is obtained.

$$U_{overhead} = \frac{lcm(T_j)}{T_{clk}}(MC_{first} + (\#j - M)C_{sub}) \tag{25}$$

where C_{first} is the cost of releasing the first task,
C_{sub} is the cost of releasing the subsequent tasks,
j is a task in the set of tasks to be executed,
T_{clk} is the clock tick rate that is an integer multiple of the clock tick rate
$\#j$ is the number of tasks in the task set, and
$lcm(T_j)$ is the least common multiple of the periods of the set of tasks

It is shown in (Bate, 1998) that the hybrid release mechanism provides better worst-case response times than the time driven approach when the period of the task in question is greater than the period of the clock tick. Otherwise, the time driven approach provides the better response times. The reason is the hybrid scheduling approach causes less overheads than the time driven approach, however the hybrid approach phases most of its overheads (due to the task that models the clock tick) at the critical instant which causes more impact on the tasks with a shorter period than the clock tick. Since the majority of tasks should tend to have a period greater than or equal to the clock tick period, then this should not cause too great a problem. In addition, the system's responsiveness is improved because tasks not released at a harmonic of the clock tick period do not suffer any release jitter.

3.7. Fault Tolerance

An additional responsibility for the infrastructure is to detect timing overruns. With a static scheduler, timing overruns are detected in a tick driven manner with a simple test performed. A similar approach can be used with fixed priority scheduling but with a different test. The test needs to determine whether sufficient execution has occurred between clock ticks. The timing analysis is modified as shown in equations (26) and (27). It should be noted that the timing watchdog can preempt other tasks so the numerator in the ceiling function is not reduced by $C_i - \Delta$.

$$R_i = C_i + B_i + I_i + I_{TW} \tag{26}$$

$$I_{TW} = \left\lceil \frac{R_i}{T_{TW}} \right\rceil C_{TW} \tag{27}$$

where I_{TW} is the interference due to the timing watchdog software,
C_{TW} is the worst-case execution time of the timing

watchdog software, and
T_{TW} is the period of the timing watchdog software.

An alternative approach is the countdown timer watchdog where each time a task commences execution, a countdown internal timer is started. The duration of the countdown timer is greater than the worst-case execution time of any task in the task set. When the task execution is complete, the countdown timer is restarted. If the countdown timer reaches zero, then fault recovery is performed. It is assumed that if a task executes for longer than its worst case execution time, then a failure has occurred. Therefore, the response time to a fault is equal to the duration of the countdown timer. This approach has clear advantages that the response time is faster and the task causing the fault is identified. However, the approach was not advocated in an industrial field study because it would require a change to the hardware for what is one of the most important components as far as safety is concerned, and hence the cost and risk would be too great.

4. Distributed Scheduling

The real challenge with distributed scheduling is the efficient implementation and verification of transactions involving more than one processor. This is because the scheduling of a particular task is no longer influenced just by the other tasks executing on the processor it is resident upon. Instead, the influence of other processors becomes relevant, which makes the synthesis, change control, and maintenance problems more complex (Burns et al., 1995a).

The aim of this section is to demonstrate some of the issues involved in scheduling for distributed systems and how a smooth transition can be made from an uniprocessor to a distributed architecture. A key component of this work is trying to maximise the reuse of the techniques developed for the uniprocessor systems. The reasons for this are:

1. There may be a great deal of time, money and effort invested in the existing theory and tools. It would be beneficial to allow the tools to be reused.

2. Treating each processor individually eases the synthesis and maintenance effort by facilitating a modular approach.

The early work on distributed scheduling using fixed priority scheduling was an event-driven approach by Tindell and Clark (Clark and Tindell, 1994). Their approach uses either periodic or aperiodic tasks

for the first task in the transaction. Subsequent tasks/messages are triggered as sporadic tasks when their preceding task/message has completed. These subsequent tasks/messages are modelled as periodic tasks (period = minimum inter-arrival time of the sporadic) with release jitter equal to the worst-case response time of the preceding task/message. The advantage of the release jitter approach is that an extended form of uniprocessor analysis may be used. However, there are four principal disadvantages of the release jitter approach, which are:

1. Sporadic tasks are frowned upon in safety critical systems due to the difficulty in analysing their behaviour - refer to section 2.1 for discussion.

2. Small changes on one processor can easily lead to the whole system needing to be re-verified, i.e. it is not robust to change.

3. The pessimism in the timing analysis can make the scheduling of the system difficult (Gutierrez et al., 1998; Burns et al., 1995a; Sun and Liu, 1996; Gutierrez et al., 1997).

4. However, the main issue is that the release jitter accumulates through the course of the transaction which can become prohibitive due to the impact on the control system's stability.

Our work shows how time driven scheduling can be used in an effective manner, keeping pessimism to a minimum and easing the problems of maintenance. The original work on this approach was performed by Liu and Sun (Sun and Liu, 1996), and was referred to as the Phase Modification Approach. By giving a task an offset such that its dispatch (or release) is always greater than the worst case response time of the event trigger (i.e. the worst case arrival time of the message) then precedence is maintained, even across a distributed system (as long as a global time-base is supported). The difference between our approach and the original is that their approach uses exact analysis which may lead to an intractable solution (refer to section 2.4 for details) whereas our approach uses the composite analysis presented in section 3.4 and it takes advantage of the flexibility in task attribute assignment to increase robustness to change.

Our approach requires the uniprocessor task attribute assignment to be integrated with the distributed scheduling approach. Figure 4 shows that at a uniprocessor level, the task attributes are calculated using Algorithm 3 with allowance made for some tasks already having attributes (offsets and deadlines) brought about from distributed transactions. In these cases, Algorithm 5 (in Algorithm 5 element refers to the tasks

and messages in a particular transaction) is applied which enforces the rules: an offset of a task can never be reduced and a deadline of a task can never be increased. The motivation behind Algorithm 5 is that increasing the tasks' offsets so that their release and execution is more evenly spread can help improve the schedulability of the task set. Figure 4 also shows the four main features related to distributed task attribute assignment, which are:

1. the analysis of messages on the databus is performed using the timing analysis for tasks by assuming a message on a databus is analogous to a task on a processor,
2. expansion of the approach to allow for multiple components, i.e. more than one processor and a databus,
3. the feedback of timing analysis results back into the uniprocessor attribute assignment, and
4. that the distributed timing analysis is carried out until convergence is achieved.

Algorithm 5 - *Algorithm for determining a task's offsets*

for each element in precedence order (loop_index =0,1, ..)
  slot time = $\frac{loop\_index}{\text{no. of elements}}$**Transaction Deadline**
  if (WCRT of the preceding element in the transaction< slot time)
    offset = slot time
  else
    offset = WCRT of the preceding element in the transaction
  deadline = offset of the next element in the transaction

One of the benefits of altering the offsets is that maintainability is enhanced by reducing the amount of re-verification when the system changes. The reason is that within defined bounds a change on one processor does not necessitate a system wide re-verification, i.e. it is not holistic. Figure 5 illustrates how the timing characteristics of processor 1 may be modified until $R_A \geq O_B$, without affecting the scheduling of processor 2, since $D_A = O_B$. Therefore, if the software on one processor is modified, then only that processor needs to be re-analysed as long as the overall system is schedulable. When the timing requirements are no longer met, the timing analysis and task attribute assignment for the whole system is repeated.

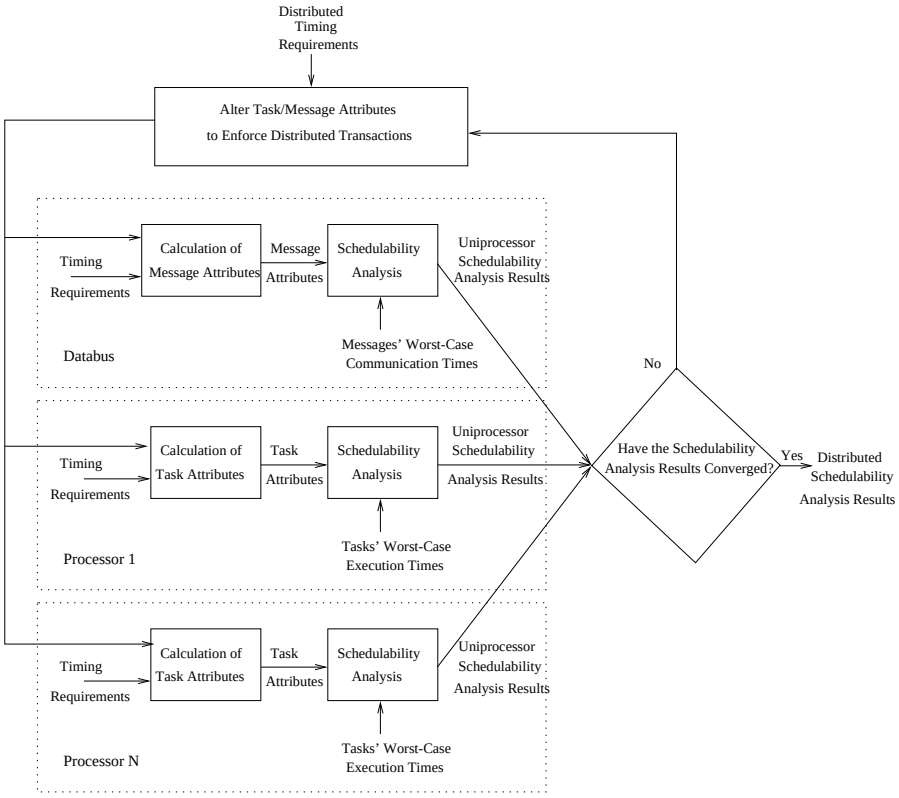

Figure 4. Approach to Meeting Timing Requirements in Distributed Systems

Again to show the approaches effectiveness, simulations have been performed with purely pseudo-random task set characteristics. The characteristics are:

1. The iteration rates were in the range [25,1000].

2. The worst-case execution time of tasks are in the range (0,2.5].

3. The worst-case communication time of messages are in the range (0,0.25].

4. The number of nodes is in the range of [2,6].

5. The number of tasks (N) is in the range [10,30].

6. A number of transactions in the range [1, N]. A transactions has a number of tasks (= the number of nodes) each task executing on a different processor. The transaction deadline is equal to its period.

7. A value for effectiveness is produced over a 1000 samples. Effectiveness is the percentage of task sets calculated as schedulable, compared to exact analysis.

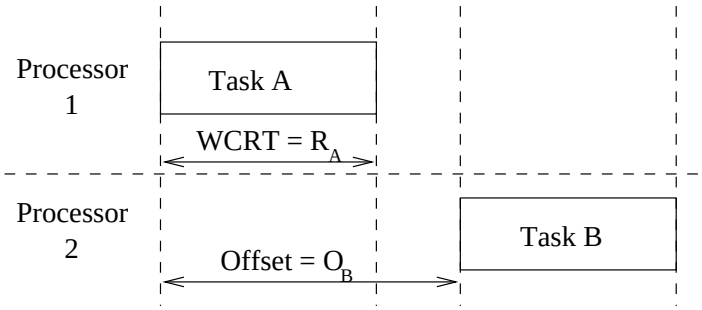

Figure 5. Phasing of Tasks on Different Processors

The simulation results for 10 and 30 tasks per processor are presented in the graphs contained in Figures 6 and 7 respectively. For each analysis technique there are two lines, one of the lines is the results for resources in the range [0%,50%) and the other the results for resources in the range [50%,100%].

The composite analysis (labeled COMP) generally has less pessimism than the release jitter approach (labeled RJ). In particular, when the resource level is higher (i.e. in the range [50%,100%]) the composite analysis performs up to 10% better than the release jitter approach. The improvement resulting from the composite approach is significant on top of the other benefits. Therefore, the approach is viewed as a viable solution for distributed scheduling and timing analysis. The release jitter approach only performs better when the resource level is small which is a consequence of the tasks' response times being smaller, which leads to the offset (or release jitter - the relevant one is dependent on the technique being used to enforce precedence) being small. When the offset or release jitter is small, the release jitter analysis has less pessimism than the composite offset approach.

This section has shown how the uniprocessor scheduling and timing analysis approaches in sections 3 can be extended for distributed systems in a manner that reduces pessimism and enhances robustness to change.

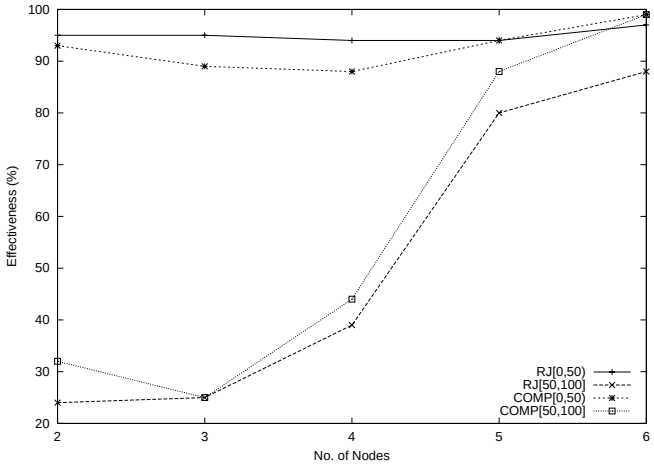

Figure 6. Comparison of the Simulation Results For 10 Tasks

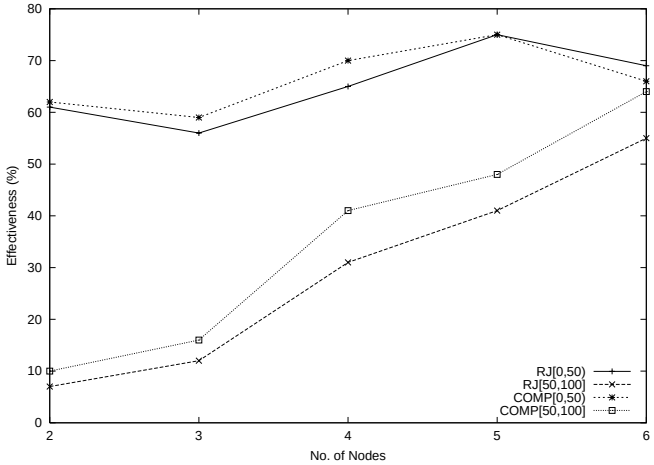

Figure 7. Comparison of the Simulation Results For 30 Tasks

5. Industrial Take-Up

There were a number of challenges when moving the techniques discussed in this paper into industrial practice, including:

1. *Cost* - Justifying there was a cost benefit in adopting the technology since this is often more important than technical benefits.

2. *Education* - It is important that most of a company's engineers and managers understand the technology well enough to be convinced and judge any potential impact on the way they do their jobs. This challenge required suitable training to be developed.

3. *Process* - Safety-critical systems are developed to a stringent process. The impact of the new technology had to be understood. The key difference was that the verification strategy relied more heavily on analysis rather than test (Bate et al., 1996).

4. *Requirements* - One of the key benefits of fixed priority scheduling over cyclic scheduling is the greater level of flexibility in the computational model. However to take advantage of this flexibility requires that the current timing requirements are changed. In a highly complex system, such as an aircraft engine controller, this is a significant undertaking. In addition, jitter constraints that had previously not been specified had to be derived.

However the key challenge was that the work needed to satisfy the certification authorities that the fixed priority approach provides a scheduler with at least the same integrity as the cyclic executive. Table V summarises a safety analysis that has been performed using a technique presented by Burns and McDermid (Burns and McDermid, 1994). The principle behind the technique is that the four main properties (functionality, resource, timing, safety) are examined and recorded for whether the property is met, the nature of the evidence and the assumptions that are made.

Table V shows that both static analysis and dynamic testing has been performed to show the correct operation of the system, and to bound memory and processing resource usage. This assumes fault-free conditions. Failure analysis has been performed that shows detectable timing faults are recovered assuming the timing watchdogs do not also fail. For the systems in question, hazards caused by dual random failures are normally accepted as being sufficiently remote, assuming there is no possible common cause and individual failures have a sufficiently low probability. The challenges presented by technology transfer were handled sufficiently well that the work in this paper has been adopted by Rolls-Royce for use on a project (Hutchesson and Hayes, 1998).

Table V. Summary of the Kernel Safety Analysis.

Property	Nature of Property	Nature of Evidence	Assumptions
Functionality	Dispatcher Correctness (tasks are scheduled at the correct rate and the correct order)	Tests using the actual hardware and software showed that the dispatcher met the requirements.	The kernel infrastructure operates in fault-free conditions.
	Operational Correctness (system invariants are not affected)	The scheduler is produced using statically stored variables – invariants should not be affected.	The scheduler is produced to an appropriate standard.
Timing	In all cases the performance of the system is predictable and schedulable.	Timing analysis has been performed that shows the system is schedulable.	The kernel infrastructure operates in fault-free conditions.
Resource	Memory usage is predictable and within the allowable limits.	The scheduler is produced using statically stored variables – resource usage proportional to the (static) number of tasks.	The kernel infrastructure operates in fault-free conditions.
Failure Behaviour	In the event of a timing overrun within the system, the fault will be identified within the appropriate time.	The tick driven timing watchdog is the same as that used for the cyclic executive scheduler which has previously been tested. Appropriate hazard analysis of the timing watchdog also exists.	The timing watchdog does not also fail.

6. Conclusions

The contribution of this paper has been to extend existing scheduling approaches to better address the specific scheduling needs of safety-critical embedded systems and in particular engine control systems. During the course of this paper, techniques have been presented that

are aimed at supporting smooth transitions from statically scheduled systems to fixed priority scheduled systems and from uniprocessor to distributed architectures. Further details of the work and the rationale behind the work are contained in (Bate, 1998). The work has been tailored in order to fulfill a number of objectives set out in section 3:

1. *Reuse* - periodic task release and non-preemptive scheduling has been used so that the actual software and the process of producing it changes by the minimum possible,

2. *Certification and Understanding* - the approach comes with analysis that is acceptable to industry and the certification authorities as demonstrated by Rolls-Royce's use of the approach on a project (Hutchesson and Hayes, 1998), and

3. *Sufficiency* - a method for task attribute assignment has been developed that is tailored for the needs of the application. The resulting analysis has acceptable levels of pessimism, computational complexity and release jitter. For large scale systems development, a greater benefit may be the fact the scheduling analysis is resilient to change within defined bounds.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments on this paper. This paper is an extended version of a paper that appeared at the 6th International Conference on Real-Time Computing Systems and Applications in 1999.

References

ARINC: June 17th, 1996, *ARINC 653: Avionics Application Software Standard Interface (Draft 15)*. Airlines Electronic Engineering Committee (AEEC).

Audsley, N. C.: 1993, 'Flexible Scheduling of Hard Real-Time Systems'. Ph.D. thesis, Department of Computer Science, University of York.

Audsley, N. C., I. J. Bate, and A. Burns: 1996, 'Putting Fixed Priority Scheduling into Engineering Practice for Safety Critical Applications'. In: *Proceedings of the Real-Time Technology and Applications Symposium*. pp. 2–10.

Audsley, N. C., A. Burns, M. Richardson, K. Tindell, and A. J. Wellings: 1993, 'Applying new scheduling theory to static priority pre-emptive scheduling'. *Software Engineering Journal* **8**(5), 284–92. Softw. Eng. J. (UK).

Audsley, N. C., A. Burns, M. Richardson, and A. J. Wellings: 1991, 'Hard Real-Time Scheduling: The Deadline Monotonic Approach'. In: *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*. pp. 127–132.

Audsley, N. C. and A. J. Wellings: 1996, 'Analysing APEX Applications'. In: *Proceedings Real Time Systems Symposium*. pp. 39–44.

Bate, I.: 1998, 'Scheduling and Timing Analysis for Safety-Critical Systems'. Ph.D. thesis, Department of Computer Science, University of York.

Bate, I. J., A. Burns, J. A. McDermid, and A. J. Vickers: 1996, 'Towards a Fixed Priority Scheduler for an Aircraft Application'. In: *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*. L'Aqulia, Italy, pp. 34–39.

Burns, A., N. C. Audsley, and A. J. Wellings: 1995a, 'Real-Time Distributed Computing'. *In Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems* pp. 34–40.

Burns, A. and J. A. McDermid: 1994, 'Real-time safety-critical systems: analysis and synthesis'. *Software Engineering Journal* **9**(6), 267–81. Softw. Eng. J. (UK).

Burns, A., K. Tindell, and A. J. Wellings: 1995b, 'Effective Analysis for Engineering Real-Time Fixed Priority Schedulers'. *IEEE Transactions on Software Engineering* **21**(5), 475–480.

Burns, A. and A. J. Wellings: 2001, *Real-Time Systems and Programming Languages (3rd edition)*. Addison Wesley.

Clark, J. A. and K. Tindell: 1994, 'Holistic Schedulability Analysis for Distributed Hard Real Time Systems'. *Microprocessing & Microprogramming* **50**(2-3), 117–134.

Edwards, R. A.: 1994, 'ASAAC Phase I Harmonized Concept Summary'. In: *Proceedings ERA Avionics Conference and Exhibition*. London, UK.

Gerber, R., S. Hong, and M. Sabsena: 1994, 'Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes'. *In Proceedings of the 15th IEEE Real-Time Systems Symposium* pp. 192–203.

Grigg, A. and N. C. Audsley: 1997, 'Towards the Timing Analysis of Integrated Modular Avionics Systems'. *In Proceedings ERA Avionics Conference and Exhibition* pp. 3.1.1–3.1.12.

Gutierrez, J., J. Garcia, and M. Harbour: 1995, 'Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems'. *IEEE Parallel and Distributed Systems* pp. 124–131.

Gutierrez, J., J. Garcia, and M. Harbour: 1997, 'On the Schedulability Analysis for Distributed Real-Time Systems'. In: *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*. pp. 136–143.

Gutierrez, J., J. Garcia, and M. Harbour: 1998, 'Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems'. In: *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*. pp. 35–44.

Harter, J.: 1984, 'Response times in level-structured systems'. Technical Report CU-CS-269-94, Department of Computer Science, University of Colorado, USA.

Harter, J.: 1987, 'Response times in level-structured systems'. **5**(3), 232–248. ACM Transactions on Computer Systems.

Hutchesson, S. and N. Hayes: 1998, 'Technology Transfer and Certification Issues in Safety Critical Real-Time Systems'. In: *Digest of the IEE Colloquium on Real-Time Systems*.

Joseph, M.: 1985, 'On a problem in real-time computing'. *Information Processing Letters* **20**(4), 173–177.

Joseph, M. and P. Pandya: 1986, 'Finding response times in a real-time system'. *Computer Journal* **29 A02**(5), 390–5.

Katcher, D., H. Arakawa, and J. Strosnider: 1993, 'Engineering and analysis of fixed priority schedulers'. **19**(9), 920–34. IEEE Transactions on Software Engineering.

Leung, J. Y. T. and M. L. Merril: 1980, 'A Note on Preemptive Scheduling of Periodic Real-Time Tasks'. *Information processing Letters* **11**(3), 115–118.

Leung, J. Y. T. and J. Whitehead: 1982, 'On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks'. *Performance Evaluation (Netherland)* **2**(4), 237–250.

Liu, C. L. and J. W. Layland: 1973, 'Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment'. **20**(1), 40–61.

Locke, C.: 1992, 'Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives'. *Real-Time Systems Journal* **4**(1), 37–53.

Sun, J. and J. Liu: 1996, 'Synchronization protocols in distributed real-time systems'. In: *Proceedings of The 16th International Conference on Distributed Computing Systems*. pp. 38–45.

Yerraballi, R.: 1996, 'Scalability in Real-Time Systems'. Ph.D. thesis, Computer Science Department, Old Dominion University.