# A Probabilistic Framework for Schedulability Analysis

A. Burns, G. Bernat, and I. Broster

Real-Time Systems Research Group
Department of Computer Science
University of York, UK

**Abstract.** The limitations of the deterministic formulation of scheduling are outlined and a probabilistic approach is motivated. A number of models are reviewed with one being chosen as a basic framework. Response-time analysis is extended to incorporate a probabilistic characterisation of task arrivals and execution times. Copulas are used to represent dependencies [1].

## 1 Introduction

Scheduling work in real-time systems is traditionally dominated by the notion of absolute guarantee. The load on a system is assumed to be bounded and known, worst-case conditions are presumed to be encountered, and static analysis is used to determine that all timing constraints (deadlines) are met in all circumstances.

This deterministic framework has been very successful in providing a solid engineering foundation to the development of real-time systems in a wide range of applications from avionics to consumer electronics. The limitations of this approach are, however, now beginning to pose serious research challenges for those working in scheduling analysis. A move from a deterministic to a probabilistic framework is advocated in this paper where we review a number of approaches that have been proposed. The sources of the limitations are threefold:

1. Fault tolerant systems are inherently stochastic and cannot be subject to absolute guarantee.
2. Application needs are becoming more flexible and/or adaptive – work-flow does not follow pre-determined patterns, and algorithms with a wide variance in computation times are becoming more commonplace.
3. Modern super-scalar processor architectures with features such as cache, pipelines, branch-prediction, out-of-order execution etc. result in computation times for even straight-line code that exhibits significant variability. Also, execution time analysis techniques are pessimistic and can only provide upper bounds on the execution time of programs.

Note, these characteristics are not isolated to so called ‘soft real-time systems’ but are equally relevant to the most stringent hard real-time application. Nevertheless, the early work on probabilistic scheduling analysis has been driven by a wish to devise effective QoS control for soft real-time systems [1, 17, 18, 38].

In this paper we consider four interlinked themes:

[1] This paper is approximately the same as the one at EmSoft2003

1. Probabilistic guarantees for fault-tolerant systems
2. Representing non-periodic arrival patterns
3. Representing execution-time
4. Estimating extreme values for execution times.

In the third and fourth themes it will become clear that one of the axioms of the deterministic framework – a well founded notion of worst-case execution time – is not sustainable. The parameterisation of work-flow needs a much richer description than has been needed hitherto.

The above themes are discussed in Sections 3 to 5 of this paper. Before that we give a short review of standard schedulability analysis using a fixed priority scheme as the underlying dispatching policy (see Burns and Wellings [11] for a detailed discussion of this analysis). We restrict our consideration to the scheduling of single resources – processors or networks. In Section 6 we bring the discussion together and draw some conclusions.

## 2 Standard Scheduling Analysis

For the traditional fixed priority approach, it is assumed that there is a finite number ($N$) of tasks ($\tau_1$ .. $\tau_N$). Each task has the attributes of minimum inter arrival time, $T$, worst-case execution time, $C$, deadline, $D$ and priority $P$. Each task undertakes a potentially unbounded number of invocations; each of which must be finished by the deadline (which is measured relative to the task's invocation/release time). All tasks are deemed to share a *critical instance* in which they are all released together; this is often taken to occur at time $0$. It is important to emphasise that the standard analysis assumes that the two limits on load (minimum $T$ and maximum $C$) are actually observed at run time. No compensation for average or observed $T$ or $C$ is accommodated.

We assume a single processor platform and restrict the model to tasks with $D \leq T$. For this restriction, an optimal set of priorities can be derived such that $D_i < D_j \Rightarrow P_i > P_j$ for all tasks $\tau_i, \tau_j$ [26]. Tasks may be periodic or sporadic (as long as two consecutive releases are separated by at least $T$). Once released, a task is not suspended other than by the possible action of a concurrency control protocol surrounding the use of shared data. A task, however, may be preempted at any time by a higher priority task. System overheads such as context switches and kernel manipulations of delay queues etc. can easily be incorporated into the model [21][10] but are ignored here.

The worst-case response time (completion time) $R_i$ for each task ($\tau_i$) is obtained from the following [20][2]:

$$R_i \;=\; C_i \;+\; B_i \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

where $\mathbf{hp}(i)$ is the set of higher priority tasks (than $\tau_i$), and $B_i$ is the maximum blocking time caused by a concurrency control protocol protecting shared data.

To solve equation (1) a recurrence relation is produced:

$$r_i^{n+1} \;=\; C_i \;+\; B_i \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \qquad (2)$$

where $r_i^0$ is given an initial value of 0. The value $r^n$ can be considered to be a computational window into which an amount of computation $C_i$ is attempting to be placed. It is a monotonically non-decreasing function of $n$. When $r_i^{n+1}$ becomes equal to $r_i^n$ then this value is the worst-case response time, $R_i$ [10]. However if $r_i^n$ becomes greater than $D_i$ then the task cannot be guaranteed to meet its deadline, and the full task set is thus unschedulable. It is important to note that a fixed set of time points are considered in the analysis: 0, $r_i^1$, $r_i^2$, ..., $r_i^n$.

Table 1 describes a simple 4 task system, together with the worst-case response times that are calculated by equation (2). Priorities are ordered from 1, with 4 being the lowest value, and blocking times have been set to zero for simplicity. Scheduling analysis is independent of time units and hence simple integer values are used (they can be interpreted as milliseconds).

| Task | $P$ | $T$ | $C$ | $D$ | $B$ | $R$ | Schedulable |
|---|---|---|---|---|---|---|---|
| $\tau_1$ | 1 | 100 | 30 | 100 | 0 | 30 | TRUE |
| $\tau_2$ | 2 | 175 | 35 | 175 | 0 | 65 | TRUE |
| $\tau_3$ | 3 | 200 | 25 | 200 | 0 | 90 | TRUE |
| $\tau_4$ | 4 | 300 | 30 | 300 | 0 | 150 | TRUE |

**Table 1.** Example Task Set

All tasks are released at time $0$. For the purpose of schedulability analysis, we can assume that their behaviour is repeated every LCM, where LCM is the least common multiple of the task periods. When faults are introduced it will be necessary to know for how long the system will be executing. Let $L$ be the lifetime of the system. For convenience we assume $L$ is an integer multiple of the LCM. This value may however be very large (for example LCM could be 200ms, and $L$ fifteen years!).

## 3 Probabilistic Guarantees for Fault-tolerant Systems

In this review we restrict our consideration to transient faults. Castillo *at al* [13] in their study of several systems indicate that the occurrences of transient faults are 10 to 50 times more frequent than permanent faults. In some applications this frequency can be quite large; one experiment on a satellite system observed 35 transient faults in a 15 minute interval due to cosmic ray ions [12].

Hou and Shin [19] have studied the probability of meeting deadlines when tasks are replicated in a hardware-redundant system. However, they only consider permanent faults without repair or recovery. A similar problem was studied by Shin et al [36]. Kim et al [22] consider another related problem: the probability of a real-time controller meeting a deadline when subject to permanent faults with repair.

To tolerate transient faults at the task level will require extra computation. This could be the result of restoration and re-execution of some routine, the execution of an exception handler or a recovery block. Various algorithms have been published which attempt to maximise the available resources for this extra computation $[37, 35, 3]$. Here we consider the nature of the guarantee that these algorithms provide. Most approaches make the common *homogeneous Poisson process* (HPP) assumptions that the fault arrival rate is constant and that the distribution of the fault-count for any fixed time interval can be approximated using a Poisson probability distribution. This is an appropriate model for a random process where the probability of an event does not change with time and the occurrence of one fault event does not affect the probability of another such event. A HPP process depends only on one parameter, viz., the expected number of events, $\lambda$, in unit time; here events are transient faults with $\lambda = 1/MTBF$, where $MTBF$ is the Mean Time Between transient Faults[2]. Per the definition of a Poisson Distribution,

$$\mathbf{Pr}_n(t) \;=\; \frac{e^{-\lambda t}(\lambda t)^n}{n!} \tag{3}$$

gives the probability of $n$ events during an interval of duration $t$. If we take an event to be an occurrence of a transient fault and $Y$ to be the random variable representing the number of faults in the lifetime of the system ($L$), then the probability of zero faults is given by

$$\mathbf{Pr}(Y = 0) = e^{-\lambda L}$$

and the probability of at least one fault

$$\mathbf{Pr}(Y > 0) = 1 - e^{-\lambda L}$$

Modelling faults as stochastic events means that an absolute guarantee cannot be given. There is a finite probability of any number of faults occurring within the deadline of a task. It follows that the guarantee must have a confidence level assigned to it and this is most naturally expressed as a probability. One way of doing this is to calculate the worst case fault behaviour that can (just) be tolerated by the system, and then use the system fault model to assign a probability to that behaviour. Two ways of doing this have been studied in detail.

– Calculate the maximum fault arrival rate that can be tolerated [9] – represented by $T_F$, the minimum fault arrival interval.
– Calculate the maximum number of faults each task can tolerate before its deadline [31].

The first approach is more straightforward (there is only a single parameter) and is reviewed in the following section. The basic form of the analysis is to obtain $T_F$ from the task set, and then to derive a probabilistic guarantee from $T_F$. An alternative formulation is to start with a required guarantee (for example, probability of fault per task release of $10^{-6}$) and to then test for schedulability. This is the approach of Broster et al [6] and is outlined in Section 3.2.

[2] MTBF usually stands for mean time between failures, but as the systems of interest are fault tolerant many faults will not cause system failure. Hence we use the term MTBF to model the arrival of transient faults.

### 3.1 Probabilistic Guarantee for $T_F$

Let $F_k$ be the extra computation time needed by $\tau_k$ if an error is detected during its execution. This could represent the re-execution of the task, the execution of an exception handler or recovery block, or the partial re-execution of a task with checkpoints. In the scheduling analysis the execution of task $\tau_i$ will be affected by a fault in $\tau_i$ or any higher priority task. We assume that any extra computation for a task will be executed at the task's (fixed) priority[3].

Hence if there is just a single fault, equation (1) will become [33][7][4]:

$$R_i \;=\; C_i \,+\, B_i \,+ \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \,+\, \max_{k \in \mathbf{hep}(i)} (F_k) \tag{4}$$

where $\mathbf{hep}(i)$ is the set of tasks with priority equal or higher than $\tau_i$, that is $\mathbf{hep}(i) = \mathbf{hp}(i) \cup \{\tau_i\}$.

This equation can again be solved for $R_i$ by forming a recurrence relation. If all $R_i$ values are still less than the corresponding $D_i$ values then a deterministic guarantee is furnished.

Given that a fault tolerant system has been built it can be assumed (although this would need to be verified) that it will be able to tolerate a single isolated fault. And hence the more realistic problem is that of multiple faults; at some point all systems will become unschedulable when faced with an arbitrary number of fault events.

To consider maximum arrival rates, first assume that $T_f$ is a known minimum arrival interval for fault events. Also assume the error latency is zero (this restriction is easily removed [9]). Equation (4) becomes [33, 7]:

$$R_i \;=\; C_i \,+\, B_i \,+ \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \,+\, \left\lceil \frac{R_i}{T_f} \right\rceil \max_{k \in \mathbf{hep}(i)} (F_k) \tag{5}$$

Thus in interval (0 $R_i$] there can be at most $\left\lceil \frac{R_i}{T_f} \right\rceil$ fault events, each of which can induce $F_k$ amount of extra computation. The validity of this equation comes from noting that fault events behave identically to sporadic tasks, and they are represented in the scheduling analysis in this way [2].

Table 2 gives an example of applying equation (5). Here full re-execution is required following a fault (ie. $F_k = C_k$). Two different fault arrival intervals are considered. For one the system remains schedulable, but for the shorter interval the final task cannot be guaranteed. In this simple example, blocking and error latency are assumed to be zero. Note that for the first three tasks, the new response times are less than the shorter $T_f$ value, and hence will remain constant for all $T_f$ values greater than 200.

The above analysis has assumed that the task deadlines remain in effect even during a fault handling situation. Some systems allow a relaxed deadline when faults occur (as long as faults are rare). This is easily accommodated into the analysis.

---

[3] Recent results had improved the following analysis by allowing the recovery actions to be executed at a higher priority [27].

[4] We assume that in the absence of faults, the task set is schedulable.

| Task | $P$ | $T$ | $C$ | $D$ | $F$ | $R$ | $R$ |
|---|---|---|---|---|---|---|---|
| | | | | | | $T_f = 300$ | $T_f = 200$ |
| $\tau_1$ | 1 | 100 | 30 | 100 | 30 | 60 | 60 |
| $\tau_2$ | 2 | 175 | 35 | 175 | 35 | 100 | 100 |
| $\tau_3$ | 3 | 200 | 25 | 200 | 25 | 155 | 155 |
| $\tau_4$ | 4 | 300 | 30 | 300 | 30 | 275 | UNSCH |

**Table 2.** Example Task Set - $T_f$ = 300 and 200

## Limits to Schedulability

Having formed the relation between schedulability and $T_f$, it is possible to apply sensitivity analysis to equation (5) to find the minimum value of $T_f$ that leads to the system being just schedulable. As indicated earlier, let this value be denoted as $T_F$ (it is the threshold fault interval).

Sensitivity analysis [39, 24, 23, 34] is used with fixed priority systems to investigate the relationship between values of key task parameters and schedulability. For an unschedulable system it can easily generate (using simple branch and bound techniques) factors such as the percentage by which all $C$s must be reduced for the system to become schedulable.

Similarly for schedulable systems, sensitivity analysis can be used to investigate the amount by which the load can be increased without jeopardising the deadline guarantees. Here we apply sensitivity analysis to $T_f$ to obtain $T_F$.

When the above task set is subject to sensitivity analysis it yields a value of $T_F$ of 275. The behaviour of the system with this threshold fault interval is shown in Table 3. A value of 274 would cause $\tau_4$ to miss its deadline.

| Task | $P$ | $T$ | $C$ | $D$ | $R$ |
|---|---|---|---|---|---|
| | | | | | $T_F = 275$ |
| $\tau_1$ | 1 | 100 | 30 | 100 | 60 |
| $\tau_2$ | 2 | 175 | 35 | 175 | 100 |
| $\tau_3$ | 3 | 200 | 25 | 200 | 155 |
| $\tau_4$ | 4 | 300 | 30 | 300 | 275 |

**Table 3.** Example Task Set - $T_F$ set at 275

In the paper cited earlier for this work, formulae are derived for the probability that during the lifetime of the system, $L$, no two faults will be closer than $T_F$. This is denoted by $Pr(W < T_F)$; where $W$ denotes the actual (unknown) minimum inter-fault gap. Of course, $Pr(W < T_F)$ is equivalent to $1 - Pr(W \geq T_F)$. The exact formulation is

$$
\begin{aligned}
Pr(W \geq T_F) &= \sum_{n=0}^{\infty} P_{n,\,(T_F/L)}\; e^{-\lambda L}\frac{(\lambda L)^n}{n!} \\
&= e^{-\lambda L}\left\{1 + \lambda L + \sum_{n=2}^{\left\lceil \frac{L}{T_F} \right\rceil} \left(1 - (n{-}1)\Big(\frac{T_F}{L}\Big)\right)^n \frac{(\lambda L)^n}{n!}\right\} \\
&= e^{-\lambda L}\left\{1 + \lambda L + \sum_{n=2}^{\left\lceil \frac{L}{T_F} \right\rceil} \frac{\lambda^n}{n!}\,(L - (n{-}1)T_F)^n\right\} \qquad (6)
\end{aligned}
$$

this leads to

$$
Pr(W \geq T_F) = e^{-\lambda L}\left\{1 + \lambda L + \sum_{n=2}^{\infty} \frac{(\lambda L - (n-1)\lambda T_F)_+^n}{n!}\right\} \qquad (7)
$$

Fortunately upper and lower bands can also be derived.

**Theorem 1.** *If $L/(2T_F)$ is a positive integer then*

$$
Pr(W{<}T_F) < 1 + \left[e^{-\lambda T_F}\,(1 + \lambda T_F)\right]^{\frac{L}{T_F}-1} - 2\left[e^{-2\lambda T_F}\,(1 + 2\lambda T_F)\right]^{\frac{L}{2T_F}}
$$

**Theorem 2.** *If $L/(2T_F)$ is a positive integer then*

$$
Pr(W{<}T_F) > 1 - \left[e^{-\lambda T_F}\,(1 + \lambda T_F)\right]^{\frac{L}{T_F}}
$$

which gives rise to the approximations

**Corollary 1.** *An approximation for the upper bound on $Pr(W{<}T_F)$ given by Theorem 1 is $\frac{3}{2}\lambda^2 L T_F$, provided that $\lambda T_F$, $\lambda^2 L T_F$ are small, and $L{\gg}T_F$.*

**Corollary 2.** *An approximation for the lower bound on $Pr(W{<}T_F)$ given by Theorem 2 is $\frac{1}{2}\lambda^2 L T_F$, provided only that $\lambda T_F$, $\lambda^2 L T_F$ are small.*

The important upper bound approximation of Corollary 1 can be written in the form $\frac{3}{2}(\lambda L)(\lambda T_F)$. It will often be the case that $\lambda T_F < 10^{-2}$; indeed this constraint allowed the approximations to deliver useful values. But $\lambda L$ can vary quite considerably from $10^{-2}$ or less in friendly environments to $10^3$ or more in long-life, hostile domains.

The example introduced in earlier had a $T_F$ value of 275ms. Table 4 gives the upper bound on the probability guarantee for various values of $\lambda$ and $L$ (in seconds).

A typical outcome of this analysis is that in a system that has a non-stop run-time ($L$) of 10 hours with a mean time between transient faults of 1000 hours and a tolerance of faults that do not appear closer than 1/100 of an hour, the probability of missing a

| | $\lambda$ | | |
|---|---|---|---|
| $L$ | $1$ | $10^{-2}$ | $10^{-4}$ |
| $1$ | $1.1{\times}10^{-4}$ | $1.1{\times}10^{-8}$ | $1.1{\times}10^{-12}$ |
| $10^1$ | $1.1{\times}10^{-3}$ | $1.1{\times}10^{-7}$ | $1.1{\times}10^{-11}$ |
| $10^2$ | $1.1{\times}10^{-2}$ | $1.1{\times}10^{-6}$ | $1.1{\times}10^{-10}$ |
| $10^4$ | $1$ | $1.1{\times}10^{-4}$ | $1.1{\times}10^{-8}$ |

**Table 4.** Upper bound on Non-Schedulability due to Faults.

deadline is upper bounded by $1.5{\times}10^{-7}$. A lower bound is also derived (Corollary 4) and this yields a value of $0.5{\times}10^{-7}$. For these parameters the exact analysis produces a value very close to $1.0{\times}10^{-7}$.

When $\lambda L{<}10^{-2}$, $\lambda L$ approximates the probability of any fault happening during the mission of duration $L$. So, $\frac{2}{3}(\lambda T_F)^{-1}$ represents the gain in reliability that is achieved by the use of fault tolerance, under the other assumptions stated. For example, in Table 4, when $\lambda = 10^{-2}$ and $L = 1$ the gain is approximately $10^6$.

## 3.2 Scheduling Analysis for Probabilistic Events

In contrast to the above approach Broster et al [6] produce a response time profile that is a direct result of the probabilistic fault model. They do not assume a single $T_F$ value. Their analysis was originally derived for CAN scheduling but is generalised here. Rather than consider the lifetime of the system, $L$, the analysis is formulated in terms of the likelihood of failure per execution of the task. There is an obvious straightforward relationship between these two formulations.

It was noted earlier that standard response analysis for fixed priority scheduling looks at a series of time points, 0, $r_i^1$, $r_i^2$, ..., $r_i^n$. (we shall ignore the task subscript in the following description). This approach assumes no faults. It is possible to generalise the scheme by replacing the single point $r^1$ by a series of points that are obtained by assuming one fault, two faults etc. That is $r(0)^1$, $r(1)^1$, $r(2)^1$, ..., $r(m)^1$ (with $r(0)^1 = r^1$). The fault model (eg. Poisson) is able to provide a value for the probability of $s$ faults in $r(s)^1$ – see equation(3).

Once all probabilities are obtained that are greater then the required guarantee (eg. $10^{-6}$) the sequence is terminated and for each value of $r(s)^1$ a set of secondary points are obtained $r(s,q)^2$ with the second parameter, $q$, being the number of faults in the interval $[r(s)^1, r(s,q)^2)$, $q = 0, 1, 2, ...$. This process is repeated and at each interaction, events (faults) with a likelihood below the threshold of interest are ignored. Although the overall search space is potentially large, the trimming that occurs due to dismissing rare events, leads to a scheme has been shown to be tractable for real sized problems. Once all sequences terminate, deadlines are checked and a probability distribution for response times is obtained.

**Scheduling Analysis for Required Probabilistic Guarantee** An alternative way of formulating the question of a probabilistic guarantee of schedulability is to calculate the worst-case response time when fault occurrences, below the threshold of interest, are ignored. Note this is not the same question as that addressed by Broster. To answer this formulation of the question is much easier as only a single iteration of the scheduling equation is needed. First equation (4) is solved assuming zero faults (let this value be represented by $R(0)$ – we again ignore the task subscript). Then the fault model, the threshold and this value $R(0)$ are used to estimate the number of faults (of interest) in the interval. Let this value be $S_1$. From equation (8) we can now calculate a new value for response time, $R(S_1)$:

$$R_i(S_1) \;=\; C_i \;+\; B_i \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(S_1)}{T_j} \right\rceil C_j \;+\; S_1 \max_{k \in \mathbf{hep}(i)} (F_k) \qquad (8)$$

The number of faults of interest in $R(S_1)$ is then calculated. If this new value, $S_2$, is equal to $S_1$ then the formulation is stable and $R(S_1)$ is the worst case response time. Alternatively equation (8) is solved for $S_2$ and the process continues until either stability is obtained or a response time greater than deadline is calculated and unschedulability is proclaimed.

To illustrate the approach consider the small example given earlier for the other approach. If we set the threshold value to $10^{-6}$ and assume $\lambda$ is obtained from a mean time between errors of 0.1 seconds then the same response time give in Table 3 are observed (eg. 60, 100, 155 and 275).

Note that the above analysis is relatively straightforward when a Poisson derived fault model is assumed. Nevertheless, the framework can still be used if other arrival distributions are more appropriate.

### 3.3 Summary

The schemes reviewed in this section all have a common theme. Scheduling approaches are used to maximise the effective resources that can be made available, when required, for fault tolerance. Then limits to schedulability are derived in conjunction with the probability of those limits being observed during execution. This furnishes the probabilistic guarantee. Alternatively, a standard yes/no guarantee is obtained while faults below a threshold of likelihood are ignored.

## 4 Probabilistic Guarantees with Non-Periodic Work

Initially scheduling analysis assumed a purely periodic work flow [28]. The sporadic jobs were incorporated by assuming a minimum arrival time, that in the worst case was exhibited by the system. In effect a sporadic job behaved exactly the same as a periodic one. Response time analysis, as outlined in Section 2 can actually deal with a much more general model of non-periodic work. Let $A_k(t)$ be defined to be a function that delivers the maximum number of arrivals of task $k$ in any interval [0,t). Then equation (1) becomes

$$R_i \;=\; C_i \;+\; B_i \;+\; \sum_{j \in \mathbf{hpp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \;+\; \sum_{k \in \mathbf{hpn}(i)} A_k(R_i) C_k \qquad (9)$$

where $\mathbf{hpp}(i)$ is now the set of higher priority periodic tasks, and $\mathbf{hpn}(i)$ the set of higher priority non-periodic tasks.

Although a useful generalisation, equation (9) is still a deterministic one. It assumes that the worst-case number of arrivals of all sporadics tasks will occur with probability one. To deal with non-periodic tasks that follow a stochastic model a different framework is needed. First, some form of probabilistic density function will be needed for all sources of sporadic work. If nothing is known about the arrival pattern of work then clearly no guarantee, not even a probabilistic one, can be given. One method of incorporating this stochastic work load is to use the same approach as for fault tolerance. After all, faults handling routines are, from a scheduling point of view, just one form of non-periodic work. The approach outlined in Section 3.2 can then be applied.

A probability threshold for the system must be defined. This is the value below which events are sufficiently rare to be ignored. Let $\rho$ be this threshold value. We re-defined the function $A$ given earlier in this section as follows: $A_k(\rho, t)$ is the number of arrival events in any interval of length $t$ with a probability of more than $\rho$. So $A_k(10^{-6}, 30)$, for example, would give the result 2 if the probability of 3 or more arrivals in 30 time units is less than $10^{-6}$ (and the probability of 2 is more than this value). Equation (9) then becomes:

$$R_i \;=\; C_i \,+\, B_i \,+\, \sum_{j \in \mathbf{hpp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \,+\, \sum_{k \in \mathbf{hpn}(i)} A_k(\rho, R_i) C_k \qquad (10)$$

This approach contains some explicit assumptions that would need to be clarified. For example, it assumes each source of arrivals is independent of each other; also that the computation time of the job is independent of the arrival behaviour. The existence of correlations would complicate the analysis – but pessimistic assumptions may be relatively straightforward to incorporate (see later discussion on the use of Copulas).

Care must of course be taken with choosing the probability threshold for the system. If an application is, by its specification, meant to deal with rare events then the threshold must be chosen so that such events (at least one in any small interval) are always incorporated into the run-time behaviour that is being analysed.

## 5 Representing Execution Time

The above discussions have generalised the notion of work flow by allowing the arrival of work to be described stochastically. However the worst-case resource requirement of each job is still represented by a single parameter $C$. This represents the maximum processor (resource) time needed by the job on each and every arrival. In developing a general framework for scheduling analysis, where application code and processor behaviour combined to produce a rich execution profile, it is not surprising that this single parameter approach is becoming limited in its application. Even a two (average and worst-case) or three (add minimum) parameter scheme is far from adequate.

In other works [8, 16, 4] we have argued that it is now inadequate to use analysis alone to obtain a single worst-case execution time (WCET) value. Rather a combination of analysis and measurement must be used to obtain a probabilistic representation of the entire execution profile of the task. Moreover, this probability density function must extend beyond observed data to predict the likelihood of experiencing, during the real execution of the system, extreme (large) values for execution time. Data obtained from measurement of relatively straightforward code illustrates two general characteristics of execution time profiles (let $O$ be the maximum observed value during measurement).

- Large observed values for computation time may be sufficiently rare that for non-hard systems it would be inappropriate for any schedulability test to assume this value for every task's execution (ie. $O$ is too large to use).
- Large observed values for computation time may not represent the worst-case that will be experienced during real execution, and extrapolations beyond observed values will be needed for some hard real-time systems (ie. $O$ is too small to use).

The alternative to simple parameterisation is to model execution time as a random variable following some probability distribution. These distributions (execution time profiles) being derived from measurement. But the granularity of measurement remains an open issue. Three levels are possible:

1. The basic block - a sequence of instructions.
2. The task - which consists of a number of basic blocks.
3. The system - which consists of a number of tasks.

If measurement is used at the task level then knowledge about the structure of the task is being ignored, however uncertainties arising from the interactions of basic blocks are being sampled. If analysis is used at the task level (with measurements only being done for basic blocks) then the rules for combining the execution time profiles need to be articulated. A similar trade-off exists at the system level.

In the work we have undertaken, we have used measurement only at the basic block level and hence we must address the issue of how to combine probability distributions. If independence could be assumed then standard statistical methods could be applied. Unfortunately there seems to be ample evidence that this assumption would be overtly optimistic. A series of basic blocks may be strongly correlated. Moreover a series of task executions within a schedule may also be dependent upon one another. Indeed the execution of the same task, one or more times, within the response time of a lower priority task may exhibit a strong correlation. These may be positive (a long execution time is more likely to be followed by another large one) or negative (long will induce a short one next time).

## 5.1 Use of Copulas

Copulas are a general mathematical tool to construct multivariate distributions and to investigate dependence structures between random variables [32]. A copula is basically a joint distribution function with uniform marginals. The main feature is that they allow one to separate the marginal distributions from the dependency between the two random variables, therefore given a joint probability distribution it is possible to characterize it uniquely with the marginal distributions and a copula. Similarly, given two marginal distributions and a copula, it is possible to derive the joint distribution and this is unique.

The importance is that the copula captures the *dependence structure* between random variables. So given two joint distributions with different marginal distributions but that capture the same dependency process, they would have the same copula.

There are two additional results of importance for this analysis, the first one is that the set of copulas is a partially ordered set and there exist two special copulas, called the lower and upper Fréchet bounds that characterize the maximum and minimum dependence between random variables.

The problem of timing analysis can be formulated as follows, if $X$ and $Y$ are two random variables that represent the execution time of two blocks of code with respective distribution functions $F_X(t)$ and $F_Y(t)$, we want to determine the distribution of $Z = X + Y$ which is the execution time of $X$ followed by $Y$, $F_Z(t)$. If $X$ and $Y$ are independent, the probability density function of $Z$ corresponds to the standard convolution of the probability density functions of $X$ and $Y$. However, if this hypothesis is not correct then we can use the theory of copulas to construct $F_Z(t)$.

If the joint distribution is known, (or its copula) then the distribution of $Z$ is a straightforward generalisation of the convolution but using the joint distribution instead. More importantly, if the dependency is not known, then it is possible to find upper and lower bounds of the distribution function for *any* possible dependency between the marginal distributions [5, 29, 14]. Some generalisations of these results allow to tighten even more these bounds if partial knowledge of the dependence is known.

### 5.2 Representing Extreme Execution Times

It was noted earlier that for some hard real-time systems execution time values beyond what have been observed during tests need to be taken into account if very low levels of failure are to be tolerated. One means of addressing this issue is to apply the branch of statistics concerned with extreme values. One of the three extreme value distributions is used to 'fit' the data and then give predictions beyond the observed data range. We have had some success [16] in fitting the Gumble distribution but it is still not clear if this is a general purpose technique. What this approach provides is a probability distribution for the worst-case value for a task's execution. One useful result of this study is that a collection of tasks has a bounded behaviour. Let $C_1..C_N$ be the worst-case times derived from the above approach with probability threshold $\rho$; then the sequential execution of each task will have a total expected execution time of $C_1 + C_2 + C_3 + .. + C_N$ with probability bound $\rho$.

## 6 Other Relevant Work on Probabilistic Analysis

There have been some other approaches using probabilistic methods in real-time systems. The work of Diaz et.al. [15] computes probability distributions of the response times of entirely periodic (fixed release times) task systems with random execution times. The work relies on the independence of the execution times of the different tasks. The work improves on an earlier work by Gardner et.al. [18]. The works of Nissanke [25] and Eles [30] also tackle this problem. However, none of these approaches address the issue of extreme distributions or dependencies between execution times or task arrivals.

## 7 Conclusion: A Probabilistic Framework

Bringing together the above approaches we are able to postulate one means of constructing a scheduling framework that can deal with stochastic parameterisation of work flow. The following are the main components of such a framework.

1. All tasks have an arrival pattern expressed as $A(\rho, t)$ - the number of instances of the task likely to occur in any interval of length $t$, where the probability of greater than $A(\rho, t)$ occurring is less than $\rho$.
2. All tasks have an execution profile derived that extends beyond the data observed during test.
3. A threshold probability is defined for the system. Events (task arrivals or execution times) with a likelihood of occurring less than this threshold are ignored. The threshold could be expressed as a likelihood of failure per execution of any task of interest.
4. A worst-case response time of each task is calculated from the above data as follows:
   - An initial estimate, $R_0$, is obtained by assuming all tasks arrive once with execution times derived from their profiles and $\rho$.
   - The number of tasks arriving in $R_0$ is derived (using $A(\rho, R_0)$ and any dependency relationships).
   - A conservative copula is used to combine the execution profiles of those jobs.
   - A new value for $R_0$ (i.e. $R_1$) is obtained by using the threshold probability value on this derived distribution.
   - Repeat until a stable value of $R$ is obtained (or $R$ expands beyond the task's deadline).

It would be at least theoretically possible to vary the probability threshold to derive a relation between response time and this threshold.

In conclusion, we have argued in support of the developed the notion of a probabilistic assessment of schedulability and shown how it can be derived from the stochastic behaviour of the work that the real-time system must accomplish. Many aspects of this framework require significant further study, and we aim to continue with this line of investigation.

## References

1. A.K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain*, pages 123–132. IEEE Computer Society Press, 1998.
2. N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.
3. G. Bernat and A. Burns. New results on fixed priority aperiodic servers. In *20th IEEE Real-Time Systems Symposium*, Phoenix. USA, December 1999.

4. G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real–time systems. In *Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, pages 279–288, Austin, Texas, USA, 2002.
5. G. Bernat and M. Newby. Probabilistic WCET analysis, an approach using copulas. Technical report, Department of Computer Science University of York, Technical Report, 2003.
6. I. Broster, A. Burns, and G. Rodríguez-Navas. Probabilistic analysis of CAN with faults. In *Proceedings of the 23rd Real-time Systems Symposium*, Dec 2002.
7. A. Burns, R. I. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. *Euromicro Real-Time Systems Workshop*, pages 29–33, June 1996.
8. A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings 12th EUROMICRO conference on Real-time Systems*, 2000.
9. A. Burns, S. Punnekkat, L. Strigini, and D.R. Wright. Probabilistic scheduling guarantees for fault-tolerant real-time systems. In *Proceedings of the 7th International Working Conference on Dependable Computing for Critical Applications. San Jose, California*, pages 339–356, 1999.
10. A. Burns and A. J. Wellings. Engineering a hard real-time system: From theory to practice. *Software-Practice and Experience*, 25(7):705–26, 1995.
11. A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley Longman, 3rd edition, 2001.
12. A. Campbell, P. McDonald, and K. Ray. Single event upset rates in space. *IEEE Transactions on Nuclear Science*, 39(6):1828–1835, December 1992.
13. X. Castillo, S.P. McConnel, and D.P. Siewiorek. Derivation and Calibration of a Transient Error Reliability Model. *IEEE Transactions on Computers*, 31(7):658–671, July 1982.
14. H. Cossette, M. Denuit, and E. Marceau. Distributional bounds for functions of dependent risks. Technical report, Bulletin suisse des actuaires., 2001.
15. J.L. Diaz, D. F. García, K. Kim, C.-G. Lee, L.L. Bello, J.M. Lopez, S.L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *22nd IEEE Real-Time Systems Symposium.*, Austin, TX. USA, 2002.
16. S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Proceedings IEEE Real-Time Systems Symposium*, 2001.
17. M. K. Gardner. *Probabilstic Analysis and Scheduling of Critical Soft Real-time Systems*. PhD thesis, University of Illinois, Computer Science, Urbana, Illinois, 1999.
18. M. K. Gardner and J.W. Lui. Analysing stochastic fixed-priority real-time systems. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 1999.
19. C.-J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Computers*, 46(12):1338–1356, 1997.
20. M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, 29(5):390–395, 1986.
21. D.I. Katcher, H. Arakawa, and J.K. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 19, 1993.
22. H. Kim, A.L.White, and K. G.Shin. Reliability modeling of hard real-time systems. In *Proceedings 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, pages 304–313. IEEE Computer Society Press, 1998.
23. M. H. Klein, T. A. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: A Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.
24. J.P. Lehoczky, L. Sha, and V. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. Tech report, Department of Statistics, Carnegie-Mellon, 1987.

25. A. Leulseged and N. Nissanke. Stochastic analysis of periodic real-time systems. In *9th Intl. Conf. on Real-Time and Embeded Computing Systems and applications (RTCSA 2003)*, Taiwan, 2003.
26. J.Y.T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, 1982.
27. G. Lima and A. Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *IEEE Transactions on Computer Systems (to appear)*, 2003.
28. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.
29. G.D. Makarov. Estimates for the distribution function of a sum of two random variales when the marginal distributions are fixed. *Theory Probab. Appli.*, 26:803–806, 1981.
30. S. Manolache, P. Eles, and Z. Peng. Memory and time efficient schedulability analysis of task sets with stochastic execution time. In *Proceedings 13th EUROMICRO conference on Real-time Systems*, 2001.
31. N. Navet, Y.-Q.Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *Journal of Systems Architecture*, 46(1):607–617, 2000.
32. R.B. Nelsen. *An introduction to Copulas*. Springer, 1998.
33. S. Punnekkat. *Schedulability Analysis for Fault Tolerant Real-time Systems*. PhD thesis, Dept. Computer Science, University of York, 1997.
34. S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Proceedings of the Conference of Advances in Computing Science - ASIAN '97*, pages 72–82. Springer, 1997.
35. S. Ramos-Thuel and J.P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems. In *Proceedings of 14th IEEE Real-Time Systems Symposium*, pages 160–171, December 1993.
36. K. G. Shin, M. Krishna, and Y. H. Lee. A unified method for evaluating real-time computer controllers its application. *IEEE Transactions on Automatic Control*, 30:357–366, 1985.
37. M. Silly, H. Chetto, and N. Elyounsi. An optimal algorithm for guaranteeing sporadic tasks in hard real-time systems. In *Proceedings 2nd IEEE Symposium on Parallel and Distributed Systems*, pages 578–585, 1990.
38. T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S. Liu. Probabilisitc performance guenrantee for real-time tasks with varying computation times. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 164–173, 1995.
39. S. Vestal. Fixed Priority Sensitivity Analysis for Linear Compute Time Models. *IEEE Transactions on Software Engineering*, 20(4):308–317, April 1994.