

Tracking Real-Time Systems Requirements

Aloysius K. Mok[†]
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
mok@cs.utexas.edu

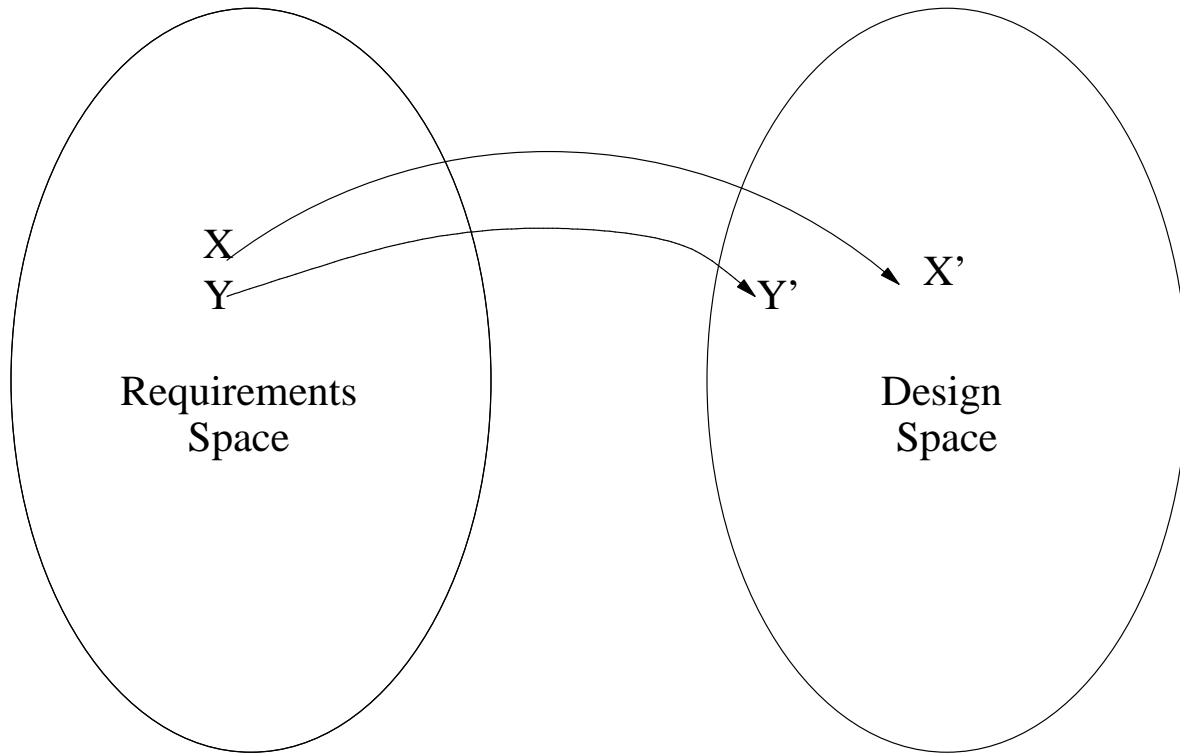
Abstract

One of the hard problems in maintaining real-time systems requirements is to keep track of the impact of resource usage on the applications. Often times, it is not sufficient to keep track of upper bounds since some requirements such as jitter are also sensitive to the lower bounds and the resource scheduling algorithm employed. In the case of non-preemptive scheduling, real-time requirements can be missed when resource utilization is decreased as in a CPU upgrade, even though there are no jitter constraints. In this paper, we define the notion of *robustness* for real-time performance requirements and discuss the tracking of sensitivity of real-time application requirements with respect to resource usage by formal method. We shall draw as an example the performance requirements of the avionics software of the Boeing 777 aircraft.

1. Introduction

A problem in engineering large complex software systems is the sensitivity of a design to changes in the requirements. If we view each step of the design process as a mapping from a requirements space to an (abstract) design space, the sensitivity problem may be viewed as a relation, let us call it the *tracking relation* between some appropriately defined *difference metric* in the requirements space and the corresponding difference metric induced in the design space, where the difference metric measures the magnitude of some aspect of a change in the requirement/design space. There are some properties of the tracking relation that are obviously desirable. For example, the tracking relation should preserve *locality*: differences confined to a locality in the requirements space should induce differences confined to a locality in the design space, and *scalability*: a small difference in the requirements space should induce a small difference in the design space. Of course, how a difference metric is defined should reflect the aspect of requirements capture under consideration. For example, the difference metric meant to capture locality in the requirements space may reflect the number of functionalities/components that are affected by a change in the requirement, and the difference metric meant to capture scalability in the requirements space may reflect the increase in system load in a requirements change. The idea of tracking relation is illustrated in the following figure.

[†] The research reported here is supported partially by a grant from the Office of Naval Research under contract number N00014-98-1-0704.



$$\text{Want } \|Y' - X'\| \sim \|Y - X\|$$

Figure 1. Tracking Relation

In the following, we shall illustrate the tracking relation concept by considering a specific aspect of real-time systems design, specifically, the relation between a change in the real-time performance requirement and the schedulability of the design solution. Intuitively, if we make the real-time performance requirement of an application less stringent, we should expect the design solution to require at most the same amount of computing resources. A mapping from requirement to design is *robust* if a less demanding requirement will not cause a performance failure in the design. We shall formalize the concept of robustness in the context of real-time scheduling theory. The schedulability problem for preemptive schedulers was first discussed in [Liu&Layland 73]. We include below our own proofs for theorems 3 and 4.

2. Some Definitions

Let us first define some terminology. We shall assume that time is discrete and all timing parameters are integers. A slight modification is needed for the discussion below to apply to continuous time.

A *sporadic task* is characterized by a pair: $T_i = (C_i, P_i)$, where each request for service of T_i requires C_i units of CPU time to satisfy and two successive requests of T_i must be

separated by at least P_i time units. Suppose M is a set of n sporadic tasks $\{(C_1, P_1), \dots, (C_n, P_n)\}$ where C_i, P_i are respectively the computation time and the minimum separation between successive requests for sporadic task T_i . A preemptive fixed-priority (PFP) scheduler is used to schedule tasks in M . A PFP scheduler always selects for execution the task that has the highest priority. Unless otherwise stated, we shall adopt the convention that task T_i is assigned a higher priority than task T_j iff $i < j$. In the following, we talk about schedules for M that are produced by a PFP scheduler. We call these schedules PFP schedules.

Suppose r is a request for a task T_i that occurs at time t in a schedule s . Then the response time of r is defined to be $t' - t$ where t' is the time at which r is satisfied by the completion of the instance T_i in s corresponding to r . Given a priority assignment, a task is scheduleable iff all of its requests have response time no bigger than its minimum separation in every PFP schedule. A feasible schedule is one in which every task in the task set is scheduleable.

We can now define the robustness property as follows. The requirements space is the set of sporadic task sets. A design is a priority assignment to the tasks in a sporadic task set. Suppose we reduce the computation time or increase the minimum separation of a task in a given task set, we should expect the same design to work. In other words, a priority assignment that results in all the tasks in a task set being scheduleable should preserve scheduleability if the computation time is reduced or minimum separation increased for some task in the task set. We say that a priority assignment is *robust* if this is indeed the case. We shall see that the RMA priority assignment (defined below) is robust for the PFP scheduler.

3. The Robustness of PFP Priority Assignment

A task T_i is said to have an outstanding computation at time t in a schedule s iff a request r for T_i occurs at time t' , $t' < t$, and r has not been satisfied at time t . Unless every request is satisfied before the arrival of the next request, it is possible for a task to have multiple outstanding computations at a time instant.

For a schedule s , a task T_i and time values t, t' , define $N(i, s, t, t')$ to be the number of requests of T_i that appear in the interval $[t, t')$ in s . Notice that $N(i, s, t, t')$ is bounded from above by $\lceil (t' - t)/P_i \rceil$, and is exactly equal to $\lceil (t' - t)/P_i \rceil$ if a request for T_i occurs at t and T_i issues a request every P_i time units thereafter.

Lemma 1

Suppose s is a schedule such that there is no outstanding computation for any task at time t_0 , for some $t_0 \geq 0$. Suppose a request r for task T_n occurs at t_0 in s , and r is satisfied at time t_1 . Then the assertions (I1), (I2) below must hold.

$$(I1) \quad \forall t_0 < t < t_1, \quad C_n + \sum_{1 \leq i < n} C_i \cdot N(i, s, t_0, t) > t - t_0$$

$$(I2) \quad C_n + \sum_{1 \leq i < n} C_i \cdot N(i, s, t_0, t_1) = t_1 - t_0$$

Proof:

Since there are no outstanding computations at time t_0 and the scheduler does not idle the processor whenever there is an outstanding computation, a simple induction on t shows that $C_n + \sum_{1 \leq i < n} C_i \cdot N(i, s, t_0, t)$ is the amount of computation time that is needed to satisfy the request of T_n at t_0 plus the requests for all the higher priority tasks in the interval $[t_0, t)$ in s . QED

Lemma 2

Suppose s is a schedule such that there is no outstanding computation for any task at time t_0 , for some $t_0 \geq 0$, and a request r for task T_n occurs at t_0 . Let s' be a schedule in which there are no outstanding computations at time t_0 , all tasks request simultaneously at time t_0 , and all tasks request at their maximum rates thereafter. If the request r in s has a response time $= u$ and the request for task T_n at time t_0 in s' has a response time $= u'$, then $u' \geq u$.

Proof:

Let the completion times of the requests for T_n that occur at time t_0 in s and s' be respectively t_1 and t'_1 . Assume the contrary: $u' < u$, i.e., $t'_1 < t_1$.

By applying assertion (I1) of lemma 1 to schedule s and assuming $t'_1 < t_1$, we have

$$C_n + \sum_{1 \leq i < n} C_i \cdot N(i, s, t_0, t'_1) > t'_1 - t_0$$

Bounding $N(i, s, t_0, t'_1)$ by $\lceil (t'_1 - t_0)/P_i \rceil$, we have

$$C_n + \sum_{1 \leq i < n} C_i \cdot \lceil (t'_1 - t_0)/P_i \rceil > t'_1 - t_0$$

Since all tasks request at their maximum rate from time t_0 in schedule s' ,

$$\forall 1 \leq i < n, N(i, s', t_0, t'_1) = \lceil (t'_1 - t_0)/P_i \rceil$$

Substituting $N(i, s', t_0, t'_1)$ for $\lceil (t'_1 - t_0)/P_i \rceil$, we have

$$C_n + \sum_{1 \leq i < n} C_i \cdot N(i, s', t_0, t'_1) > t'_1 - t_0$$

However, by applying assertion (I2) of lemma 1 to schedule s' where t'_1 is the completion time for the request at t_0 , we have

$$C_n + \sum_{1 \leq i < n} C_i \cdot N(i, s', t_0, t'_1) = t'_1 - t_0$$

Thus we have a contradiction. QED

Let M be a set of sporadic tasks. Let s be a PFP schedule of M in which all tasks request simultaneously at time 0, and all tasks request at their maximum rates thereafter. A task T in M is said to pass its *critical – instant test* if the response time of the request for T at time 0 in the schedule s is no bigger than its minimum separation parameter.

Suppose s' is a schedule in which there are no outstanding computations at time t_0 , for some $t_0 \geq 0$, all tasks request simultaneously at time t_0 , and all tasks request at their maximum rates thereafter. Since the schedule s and the suffix of s' after t_0 are identical, the response time of the request for a task T at time t_0 in s' does not exceed T 's minimum separation parameter iff T passes its critical instant test.

Theorem 3 ([Liu&Layland 73])

If a task T passes its critical-instant test, then T is scheduleable by a PFP scheduler.

Proof:

Suppose T has the n^{th} highest priority (i.e., T is T_n and has a lower priority than tasks T_1, \dots, T_{n-1}). We need to show that every request of T_n must have a response time no bigger than P_n in any PFP schedule. Let s be a PFP schedule. Without loss of generality, we shall disregard the scheduling of the tasks $\{T_i | i > n\}$, i.e., we consider only the n highest priority tasks. For any time value t_0 such that: (P1) there is no outstanding computation for T_n at time t_0 in s , and (P2) there is a request for T_n at t_0 , we shall show that the request at t_0 must have a response time $\leq P_n$.

Let t_x be the biggest time value, $0 \leq t_x \leq t_0$ such that there is no outstanding computation for any task at time t_x . Since there is no outstanding computation at time 0, t_x must exist. Notice that if $t_x \neq t_0$, then the processor cannot idle in the interval $[t_x, t_0]$ and only tasks with priority higher than n are executed in $[t_x, t_0]$. Now consider a schedule s' such that there are no requests for the tasks T_1, \dots, T_{n-1} before t_x in s' , and the requests for these tasks at or after t_x in s' occur at the same time as those in s . Also let the first request for T_n in s' occur at time t_x . By construction of s' , the response time of this request is equal to $t_0 - t_x$ plus the response time of the request for T_n which occurs at t_0 in schedule s . Since there are no outstanding computations for T_1, \dots, T_n at t_x in s' , and T_n passes its critical-instant test, the response time of the first request of T_n in s' must be at most P_n by lemma 2, and therefore the request for T_n at t_0 in s must have a response time not exceeding P_n .

The arrival time of the first request of T_n trivially satisfies (P1) and (P2), and hence the first request must have response time $\leq P_n$. Suppose the first i requests of T_n have response time $\leq P_n$. Then the $(i+1)^{\text{th}}$ request must satisfy (P1) and (P2), since the i^{th} and $(i+1)^{\text{th}}$ requests are separated by at least P_n time units. Hence the $(i+1)^{\text{th}}$ request must also have response time $\leq P_n$. QED

Given a task set M and a priority assignment, let s be the PFP schedule of M such that all tasks request simultaneously at time 0, and all tasks request at their maximum rates thereafter. We call s the critical schedule of M .

Corollary

If every task in a sporadic task set M meets its first deadline in the critical schedule of M , then M is scheduleable by a PFP scheduler.

Proof: Immediate.

The Rate Monotonic Assignment (RMA) of priorities:

Suppose $M = \{(C_1, P_1), \dots, (C_n, P_n)\}$ is a set of n sporadic tasks, and task T_i has higher priority than task T_j if $i < j$. Then the priority assignment of tasks in M is consistent with RMA if $P_i \leq P_j$, $1 \leq i, j \leq n$.

Theorem 4 ([Liu&Layland 73])

Suppose M is a set of tasks whose priority assignment is consistent with RMA. Then M is schedulable iff its critical schedule is feasible.

Unless otherwise stated, we shall refer to the critical schedule of a task set as one corresponding to a RMA-consistent assignment of priorities. For preemptive schedulers, reducing the computation time of a task in a schedulable task set will not cause the resulting task set to be unschedulable. To see this, suppose T has the n^{th} highest priority in the task set. Let the response time of the first request of T be x in s , the critical schedule, and let its response time be y in s' , the critical schedule after the computation time of a higher priority task T_k has been reduced by some $\delta > 0$. If $y > x$, then applying assertion (I1) of lemma 1 to the critical schedule s' yields

$$C_n + \left(\sum_{\substack{1 \leq i < n \\ i \neq k}} C_i \cdot \lceil x/P_i \rceil \right) + (C_k - \delta) \cdot \lceil x/P_k \rceil > x$$

Rewriting this inequality,

$$C_n + \sum_{1 \leq i < n} C_i \cdot \lceil x/P_i \rceil - \delta \cdot \lceil x/P_k \rceil > x$$

However, applying assertion (I2) to the critical schedule s yields

$$C_n + \sum_{1 \leq i < n} C_i \cdot \lceil x/P_i \rceil = x$$

which is a contradiction.

Similarly, it can be shown that increasing the period of any task in a schedulable task set will not cause the resulting task set to be unschedulable by a PFP scheduler. Thus, we have the following theorem.

Theorem 5

The RMA assignment of priorities for the PFP scheduler is robust.

In general, it can be seen that any feasible priority assignment for the PFP scheduler is robust. Unfortunately, this is not the case if the scheduler is non-preemptive.

4. Loss of Robustness in Non-preemptive Schedulers

In real-life systems, such as the Boeing 777 Integrated Airplane Information Management System (AIMS), not all tasks can be scheduled preemptively. The AIMS system, running on the ARINC 659 platform, requires high resource utilization and performance guarantees while providing strict partitioning of functions on a multiprocessor platform. The scheduling problem involves pre-scheduling of both computational and communication resources and involves both deadline and jitter requirements. A typical real-time requirement AIMS takes the form:

[Data] from <process> ([which runs at] <rate> <duration>) to <process>
of aggregate data transmission length <xfer duration> with minimum
latency <latency bound>.

Maximum latency <latency bound> means that the time from the start of execution of the sending process to the end of execution of the receiving process must not be less end than the specified bound. A +-500 usec jitter requirement (deviation from ideal period) across the board can be assumed, to both data and process start and end times.

Each of the above type of requirements involves three tasks, one application task for the sending process, one application task for the receiving process and one communication task for the transmission of data over the bus connecting the processors executing the application tasks. In total, there are 155 applications tasks and 951 communications between these tasks.

Whereas the application tasks may be preemptively scheduled on a processor, the communication tasks are inherently non-preemptive as limited by the minimum size of a message. If we view the scheduling of messages on the bus as a single resource scheduling problem, a PFP scheduler is inappropriate for the communication resource. A non-preemptive fixed priority (NPF) scheduler is one that always selects among all ready tasks the one that has the highest priority for execution until completion, i.e., once a task starts execution, no preemption is allowed, where a task is ready at time t if it has a request which arrives no later than time t and which has not been allocated execution time. We now show that the RMA priority assignment of NPF is not robust.

Consider the following task set with 3 tasks: $\{T_1 = (3, 5), T_2 = (2, 10), T_3 = (4, 20)\}$. With the RMA assignment, this task set is schedulable by a non-preemptive fixed priority scheduler as is shown by the timing diagram in the figure 2. However, the task set becomes unschedulable if we reduce the execution time of T_2 from 2 to 1. Hence, NPF scheduler is not robust with respect to reduction of execution time requirement of a task.

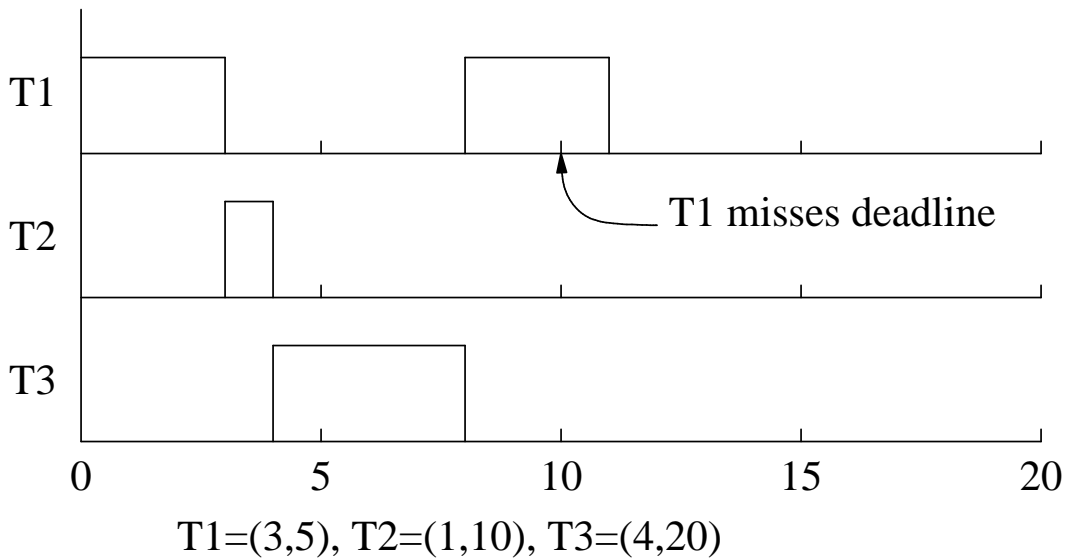
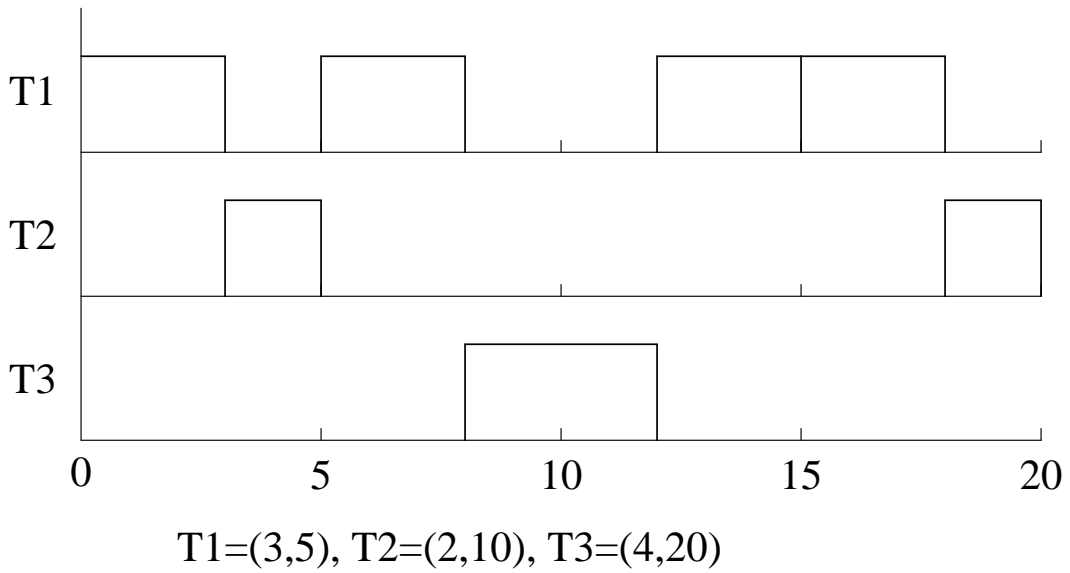
Next, consider the task set with 3 tasks: $\{T_A = (1, 4), T_B = (3, 8), T_C = (6, 16)\}$. Again, with the RMA assignment, this task set is schedulable by a non-preemptive fixed priority scheduler as is shown by the timing diagram in the figure 3. However, the task set becomes unschedulable if we increase the period of T_A from 4 to 5. Hence, NPF scheduler is not robust with respect to reduction of execution frequency of a task.

Lastly, consider the following task set with 3 tasks: $\{T_1 = (30, 50), T_2 = (20, 100), T_3 = (40, 200)\}$. With the RMA assignment, this task set is schedulable by a non-preemptive fixed priority scheduler as is shown by the timing diagram in the figure 4. However, the task set becomes unschedulable if we reduce the execution times of all three tasks by 10%. Hence, NPF scheduler is not robust with respect to improvement in CPU speed.

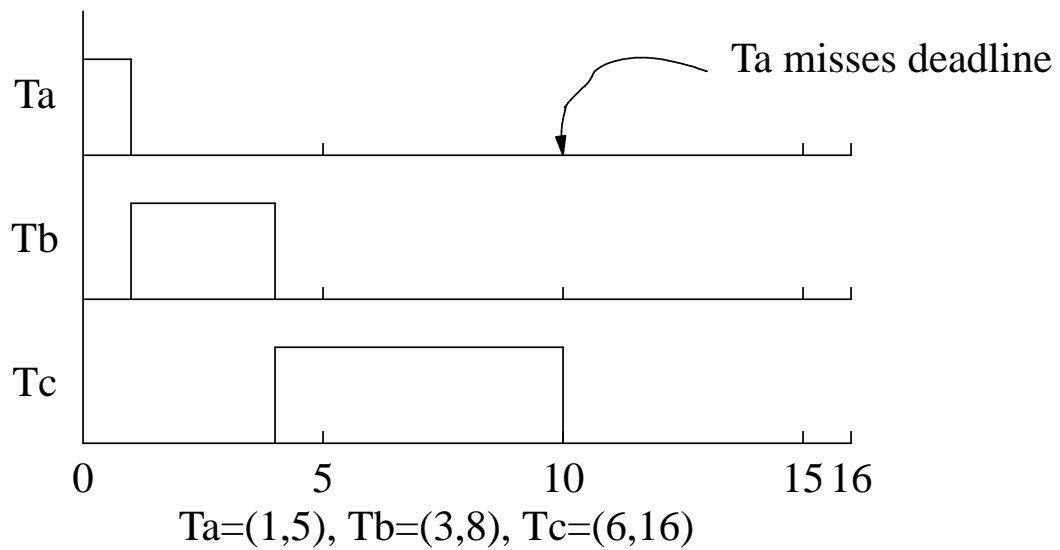
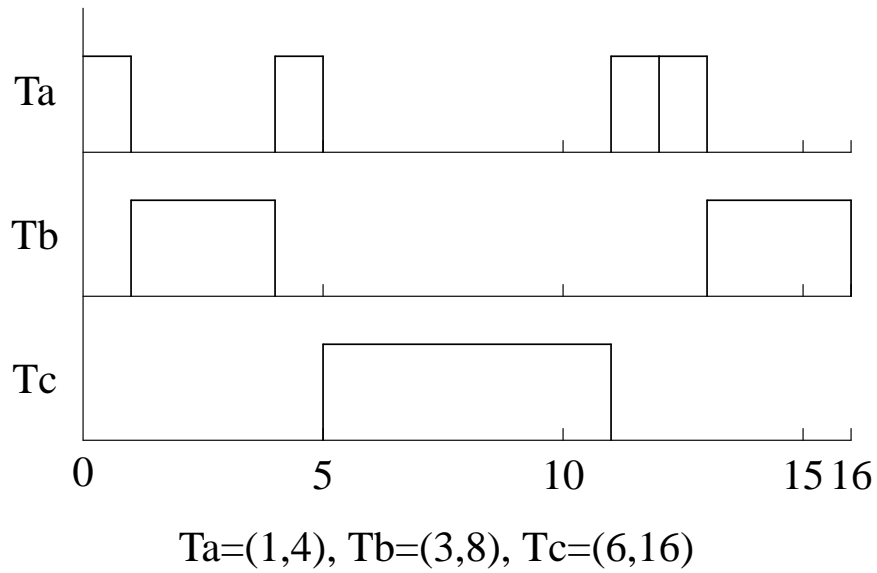
The above three counter-examples establishes

Theorem 6

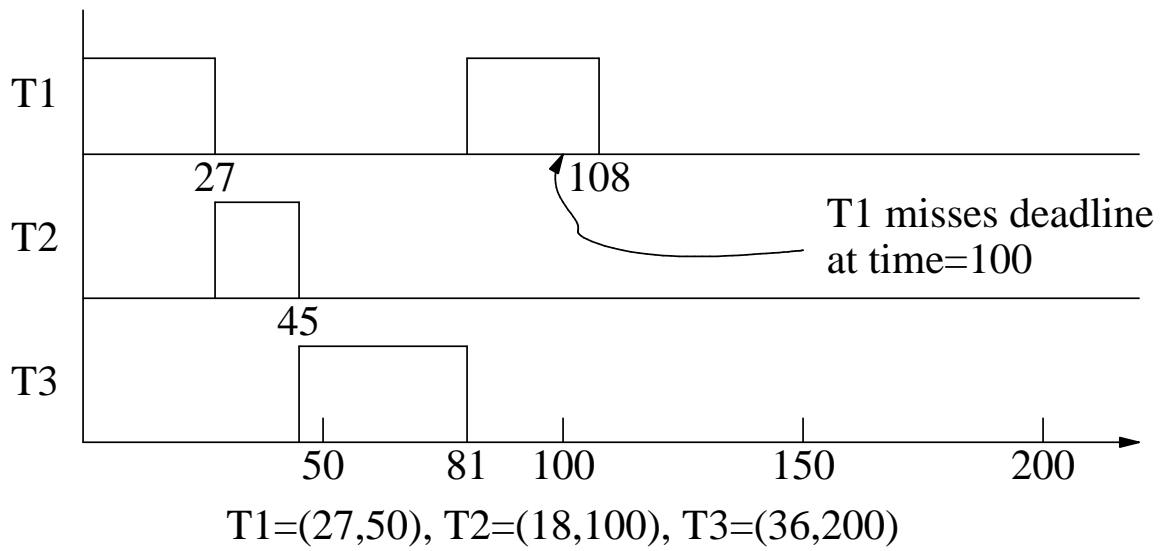
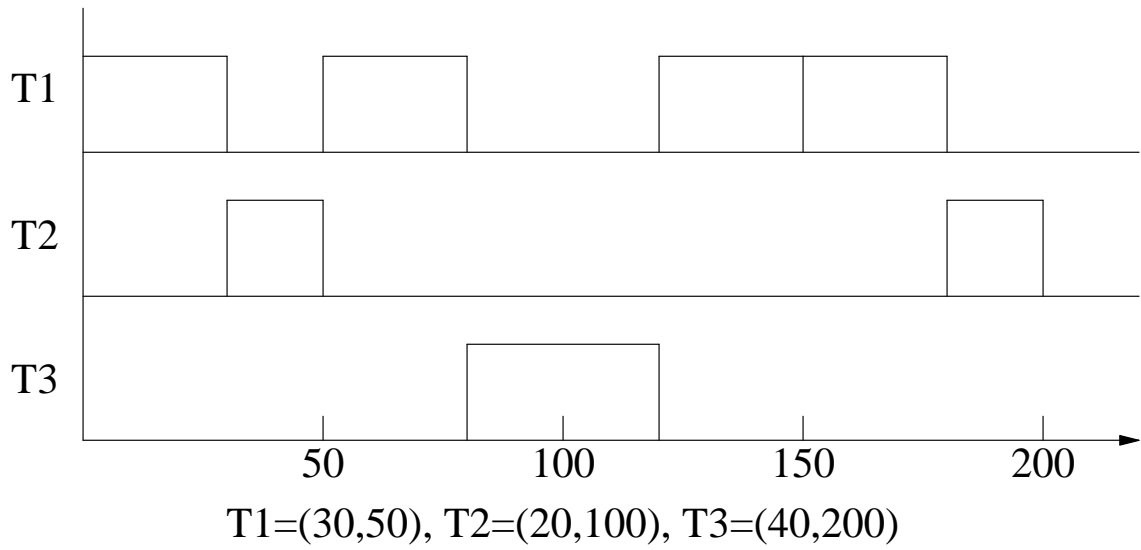
The RMA assignment of priorities for the NPF scheduler is not robust.



REDUCING C2 FROM 2 TO 1 CAUSES DEADLINE MISS
Figure 2. LOSS OF ROBUSTNESS WITH RESPECT TO DECREASED COMPUTATION TIME



INCREASING P_a FROM 4 TO 5 CAUSES DEADLINE MISS
Figure 3. LOSS OF ROBUSTNESS WITH RESPECT TO INCREASED PERIOD



**FASTER CPU CAUSES T1 TO MISS DEADLINE
 ALL EXECUTION TIMES DECREASED BY 10%**

Figure 4. LOSS OF ROBUSTNESS WITH RESPECT TO IMPROVED CPU SPEED

The real-time scheduling solution adopted by the Boeing 777 AIMS relies on the use of *cyclic executives* which require the precomputation of static schedules which are then repeated at run time. In [MTR 96], we describe in detail a tool which automates the computation of the cyclic executives for requirements such as the Boeing 777 AIMS.

5. Conclusion

The robustness property discussed above is an example of the tracking relation in mapping requirements to design. Whereas the lack of robustness with respect to real-time performance requirements can be overcome by design automation tools such as the MSPRTL tool described in [MTR 96], the more challenging problem is to achieve other properties such as locality and scalability simultaneously with performance robustness.

Bibliography

- [Liu&Layland 73] C. L. Liu and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of ACM*, vol. 20, no. 1, January 1973.
- [MTR 96] A. K. Mok, Duu-Chung Tsou and R. C. M. de Rooij, "The MSPRTL Real-Time Scheduler Synthesis Tool," *Proceedings of the 17th IEEE Real Time Systems Symposium*, December 1996, pp. 118--128.