Doctor of Philosophy Dissertation

Real-time Control Design with Resource Constraints

by

Yifan Wu

Supervisor: Giorgio Buttazzo

Scuola Superiore Sant’Anna, Pisa
2009

Abstract

Nowadays, most control systems are manipulated by control algorithms that are implemented as software components. In many practical applications, such software components execute on platforms characterized with constrained resources, such as in embedded systems. To ensure the correct timing behavior of the control software, real-time kernel is employed to schedule several such components running on the same platform and sharing the same limited resources. However, the integration of real-time systems and control systems needs careful analysis and deliberate design with new techniques.

In this dissertation, the design problem for real-time control systems in resource-constrained platforms is investigated. The background knowledge of real-time systems and control systems is introduced, and the state of the art is reviewed. Several new approaches are proposed to contribute to the existing technologies.

The limited-preemption is utilized to improve the responsiveness of control tasks and hence improve the control performance, without jeopardizing the schedulability of the whole system.

A general framework for real-time control design is presented where the delay and jitter effect are incorporated into the performance optimization. The resource constraints are characterized using the convex approximation of the EDF deadline space, and a two-step procedure for period and deadline selection is proposed.

The utilization of multiprocessor platform is investigated. A search algorithm is presented to exploit the internal parallelism of an application and partition it into several flows that are then implemented using resource reservation to guarantee the timely behavior and provide temporal isolation. The benefits of using this method is shown by a control application involved with a ball-and-plate system.

Acknowledgements

TODO:[To write]

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Problem 3
1.3 Structure 4

2 Background **5**
2.1 Real-time systems 5
2.1.1 Task model 5
2.1.2 Real-time Scheduling 8
2.1.3 Schedulability analysis 11
2.1.4 Sensitivity analysis 13
2.1.5 Resource reservation 14
2.2 Control Systems 15
2.2.1 Control systems analysis 18
2.2.2 Computer-controlled systems 23
2.3 Real-time Control Integration 26
2.3.1 Control loop timing 26
2.3.2 State of the art 30
2.3.3 Analysis Tools 33
2.4 Conclusion 36

3 Improved responsiveness using limited-preemption **37**
3.1 Introduction 37
3.2 Related Work 37
3.3 System Model 38
3.3.1 Placement of NP chunks 39
3.3.2 Timing Attributes 39
3.4 Response Time Analysis 40
3.4.1 Worst-Case Response Time of EDF 40
3.4.2 Worst-Case Response Time of LP-EDF 41
3.5 Experimental Results 44
3.5.1 Experiment Setup 44
3.5.2 Responsiveness Results 45

3.5.3 Control Performance Results 49
3.6 Conclusion 51

4 Parameter Selection in an Integrated Framework 53
4.1 Introduction 53
4.2 Related Work 55
4.3 System model 57
4.4 The General Framework 58
4.4.1 Integrated Design approach 58
4.4.2 The performance loss index 59
4.4.3 The optimization problem 60
4.5 Linking Task Parameters to Control Performance 61
4.5.1 Characterization of the delay and jitter 61
4.5.2 Performance loss index derivation 62
4.5.3 Quantitative performance degradation analysis 63
4.5.4 Example of analysis results 65
4.6 Resource Constraints Characterization 67
4.7 Experimental results 69
4.7.1 The control systems 70
4.7.2 Experimental setup 71
4.7.3 Period and deadline selection 72
4.8 Conclusion 74

5 Parallel control application on multiprocessor platform 77
5.1 Introduction 77
5.2 Related Work 78
5.3 System model and background 80
5.3.1 Terminology and notation 81
5.3.2 Demand Bound Function 83
5.3.3 The (α, Δ) server 84
5.4 Partitioning an application into flows 85
5.4.1 Assigning deadlines and activations 86
5.4.2 Bandwidth requirements for a flow 90
5.4.3 The branch and bound algorithm 91
5.5 Experimental results 94
5.5.1 Effectiveness evaluation 94
5.5.2 A Control Example 96
5.6 Conclusion 100

6 Conclusion 103
6.1 Summary 103
6.2 Future Work 104

A Proof of the quasiconvexity of optimization problem (5.15) 105

B Deadline and activation assignment for Γ_{bap} **108**

Appendices **105**

References **109**

Chapter 1

Introduction

1.1 Motivation

Computer-controlled systems are nowadays prevailingly employed in many fields. Practically speaking, almost all the control systems are now computer-controlled systems rather than analog systems, due to various advantages brought by the underlying computing platform such as enhanced functionality and extended flexibility.

The traditional design approach for computer-controlled systems often involves two steps. In the first step, the controller is obtained through discretization of the design in continuous time domain, or by directly using sampled control theory. It is often assumed a constant sampling period in this step. Once acquired in its discretized form, during the second setup, the controller is implemented as a software component and then scheduled to execute on computing platform where, in most cases, real-time systems are used to guarantee the timely behavior. The equidistant sampling period assumption, among others, helps build up a separation between the control community and the scheduling community. The control engineers are relieved from no worrying about the implementation details and how scheduling is performed, while computer engineers are allowed to focus on the platform issues and pay no attention to the potential impact on control performance. From a historical point of view, such a separation facilitates both sides to concentrate on its own area and produce considerable outcomes.

However, there are various problems induced by the ignorance of the mutual influence. The control community often assumes a too simple task model and a deterministic platform. The influence of the shared and limited resources is usually not considered during the controller design. In fact, the controller is sometimes envisioned to run as a simple loop on a dedicated computing unit. On the other hand, the scheduling community assumes the controller can always be model as a task and makes scheduling design trade-off without consideration of the control performance.

In practice, many control applications are nowadays running on systems characterized with non-determinism and timing uncertainties, which might be induced by several sources:

- *Low-cost mass-market products.* Most of the control applications are implemented using inexpensive hardware. Only in extreme applications, such as nuclear power plants, can the cost of the computing hardware be neglected in the overall development cost [Cer03]. These low-cost hardware platforms are often characterized with limited resources, such as computing capacity, memory, battery power, etc. Meanwhile, due to the same reason on price cut, the usage of mass-market hardware and operating systems reduces the cost and increases the flexibility of the system design, however leading to less efficiency and predictability than the ad-hoc solution, e.g. Application-Specific Integrated Circuits (ASIC).

- *Multitasking environment.* Due to the need of low cost solution and demand for high functionality, there is trend to incorporate more software components into the same computing platform. Such software components include user interface, logging, or even other control tasks. In this multitasking environment, multiple tasks compete for the shared resources, leading to task status like preemption and blocking. Besides, caches are used to improve the overall performance, but may give rise to cache misses resulting in inconstant and unpredictable computation time.

- *Networked control system.* When subsystems inside the control loop are connected using networks, extra delays are introduced into the system such as network interface delay, queuing delay, transmission delay, propagation delay, link layer resending delay, etc. Moreover, packet loss should also be considered.

In other words, the implementation issues have significant impact on the original control design, and hence the separation of control design and computer implementation is no longer capable of sufficing the needs of the modern control systems. In general, the negative effects of the separated design approach include:

- *Impaired schedulability.* When control engineers assign parameters such as periods to the controller tasks without considering the limited resource constraints, the set of software components may not be schedulable by the execution platform.

- *Degraded control performance.* The scheduling-induced delay and jitter bring non-determinism which violates several assumptions made in the control design, such as equidistant sampling and zero-or-constant latency, leading to degradation of control performance or even instability.

- *Repetitive design process.* The whole design procedure tends to be repetitive and tedious whenever the system is found to be unschedulable or the control goal can not be met with the provided platform.

Therefore, the integration of control and real-time scheduling is necessary to bridge the two communities, and mutual understanding is encouraged to achieve better system design. This dissertation aims to explore several possible ways to improve performance and increase flexibility for the real-time control co-design in resource-constrained systems.

1.2 Problem

The control and scheduling co-design problem is formally defined in [Cer03] as:

> Given a set of processes to be controlled and a computer with limited computational resources, design a set of controllers and schedule them as real-time tasks such that the overall control performance is optimized.

and an alternative view to this is to minimize the resource usage while still meeting the performance requirement. There are indeed several specified types of resources within real-time control systems, e.g. CPU, memory, I/O ports, battery, etc. However, without loss of generality, this work will focus on the CPU time.

The integrated design of control and scheduling needs knowledge from both area. To exploit the potential in the integration, several factors can be utilized:

- *Control Theory*. New theories and design criterion help to overcome or compensate the extra delay and jitter caused by the scheduling platform.
- *Task model*. Dedicated task model can be used to better fit for the controller implementation.
- *Scheduling policy*. Modification of existing scheduling policies may benefit the system performance.
- *Design framework*. To avoid the repetitive design process, new design framework is desirable.
- *Hardware*. With the advent of multiprocessor platform, the potential of computation capability is largely extended, which can be employed to augment the system performance.

This work will review existing researches utilizing one or several of these factors, and present some new methodologies. In particular, a scheduling policy different from the traditional one is employed to improve control performance. A design framework is proposed to relief the traditional repetitive design process. Dataflow programming model is suggested to be used for enhancing applications on multiprocessor platform.

1.3 Structure

The main consideration of this dissertation is to make real-time control co-design with respect to the intersection of control and real-time scheduling domains. The outline is described as follows:

Chapter 2 shows the background of the work. It first gives a brief introduction of the real-time systems theory and control theory. It then talks about the main issues in the real-time control integration. It reviews the state-of-the-art technologies following several major directions. Finally, it introduces several tools that help to analyze and simulate the real-time control systems.

Chapter 3 suggests to use scheduling policy where preemptions are limited to enhance the system behavior. This method allows to improve control performance without affecting the schedulability of the whole system.

Chapter 4 presents a general framework which treats the real-time control co-design as an optimization problem, so that the traditional repetitive design process is avoided. The proposed framework allows to select task periods and deadlines under schedulability constraints, taking into account the effect on control performance from the scheduling-induced delay and jitter.

Chapter 5 proposes to use dataflow programming model and multiprocessor platform to improve the system performance and flexibility. In particular, a method is presented to partition the control application into several flows, which then assigned to resource reservation onto multiprocessors.

Chapter 6 concludes the contents of the dissertation and gives suggestion on the future work.

Chapter 2

Background

2.1 Real-time systems

Real-time systems are computing systems that must react within precise time constraints to events in environment [But97]. The correctness of a real-time system depends on not only its computed values but also the time at which the results are produced [Sta88]. There are numerous applications where real-time systems play a crucial role, including flight control systems, vehicle collision avoidance, military appliance, industrial automation, etc.

The essential goal of applying real-time theory is to ensure the timing behavior of the system. Testing, to some extend, provides a partial verification of the system behavior but fails to make such guarantee. Therefore, rather than using average measures in general purpose systems, the full predictability of the real-time system's timing behavior is only achievable by elaborated analysis using pessimistic assumptions.

2.1.1 Task model

As illustrated in Figure 2.1, the computation entity that a real-time system deals with is named *task*, denoted as τ_i, which is usually characterized by the following parameters:

- **Arrival time** a_i is the time at which a task becomes ready for execution. It is also referred to as *activation time* or *release time* (denoted by r_i);
- **Start time** s_i is the time at which a task starts executing;
- **Finishing time** f_i is the time at which a task finishes its execution;
- **Absolute deadline** d_i is the time before which a task should complete its execution;
- **Worst-case execution time (WCET)** C_i is the maximum time needed for the processor to execute the task without interruption;

- **Relative deadline** D_i is the deadline with respect to the arrival time, that is $D_i = d_i - a_i$.

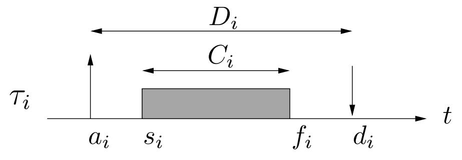


Figure 2.1: A typical real-time task.

Depending on the *task criticality*, that is the strictness of complying with the timing constraint imposed by the deadline, tasks can be distinguished into two classes:

A *Hard task* must finish its execution within the relative deadline D_i or otherwise causes catastrophic consequences to the system.

A *Soft task* only decreases performance if it misses a deadline.

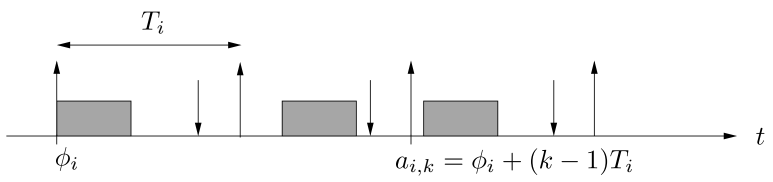
A real-time task τ_i usually generates an infinite number of identical activities, called *instances* or *jobs*, denoted by $\tau_{i,k}$, $k \in \mathbb{N}$ [1]. Depending on the regularity of the activation mode of jobs, tasks can be classified as *periodic task* and *aperiodic task*, corresponding to the *time-triggered* and *event-triggered* fashion, respectively. A periodic task regularly generates its jobs every T_i, defined as the *period* of the task. Hence, if the activation time of the first instance is denoted by ϕ_i (the *release phase*), then the arrival time of the k-th job is $a_{i,k} = \phi_i + (k-1)T_i$. On the contrary, an aperiodic task activates it jobs at an irregular rate. To perform off-line guarantee of the criticality of the aperiodic tasks, it is most of interest to analyze the peak-load situation by assuming its maximum arrival rate. That is the time between two successive activations of an aperiodic task is delimited by a minimum value. The task is then called *sporadic task*, and this minimum value is defined as *minimum inter-arrival time*, also denoted by T_i. Figure 2.2 shows an example of task instances for a periodic task and a sporadic task.

In most cases, a periodic or sporadic task can be completely characterized by the 3-tuple (C_i, D_i, T_i). The acquisition of the worst-case execution time C_i usually resorts to either static analysis involving both software code and hardware platform, or measurement-based approaches. The relative deadline D_i and the period (or the minimum inter-arrival time) T_i are typically specified by the system designer, where D_i is often set equal to T_i.

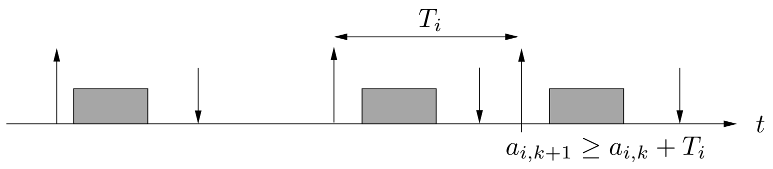
Task constraints

The task model described above concerns *timing constraints* of the real-time tasks. Besides, there are other types of constraints that can be additionally imposed, listed

[1] In this dissertation, $\mathbb{N}$ denotes the set of positive integers $\{1, 2, ...\}$, while $\mathbb{N}_0 = \mathbb{N} \cap \{0\}$ means the set of non-negative integers $\{0, 1, 2, ...\}$.



(a) Activation of a periodic task τ_i



(b) Activation of a sporadic task τ_i

Figure 2.2: Sequence of instances for a periodic task and a sporadic task.

as follows.

- *Precedence constraints.* The precedence relations between tasks can be described using a directed acyclic graph (DAG) G, where tasks and relations are represented by nodes and arrows, respectively. Notation $\tau_i \prec \tau_j$ denotes that τ_i is a *predecessor* of τ_j, meaning that τ_j cannot start executing before the completion of τ_i. Notation $\tau_i \rightarrow \tau_j$ denotes that τ_i is an *immediate predecessor* of τ_j, meaning that there is an arc directed from τ_i to τ_j.

  An example of a precedence graph G is shown in Figure 2.3. It is clear that only task τ_1 has no predecessors which means it can start executing at any time. Once τ_1 completes execution, task τ_2 and τ_4 are able to start. τ_3 has to wait for the completion of τ_2, while τ_5 can not start until both τ_2 and τ_4 finish executing. Tasks with no predecessors, e.g. τ_1, are called *beginning tasks* or *root nodes* in the DAG. Tasks with no successors, e.g. τ_3 and τ_5, are called *ending tasks* or *leaf nodes* in the DAG.

$$\begin{tikzcd}
\tau_1 \arrow[r] \arrow[dr] & \tau_2 \arrow[r] \arrow[dr] & \tau_3 \\
 & \tau_4 \arrow[r] & \tau_5
\end{tikzcd}$$

Figure 2.3: A precedence graph.

- *Shared resource constraints.* This type of constraints requires synchronization mechanism, e.g. semaphores, to achieve mutual exclusion among different tasks and thus keep data consistency. Resource access protocols, such as Priority Inheritance and Priority Ceiling [SRL90], are necessary to avoid the priority inversion phenomenon.

2.1.2 Real-time Scheduling

A real-time system considers a set of concurrent tasks and manages to assign the processor to these tasks according to some predefined criterion, called *scheduling policy*. A *schedule* is a particular assignment of tasks to the processor, so that each task is executed until its completion. It can be mathematically described as:

> Given a task set $\tau = \{\tau_1, \ldots, \tau_n\}$, a schedule is a mapping $\sigma : \mathbb{R}^+ \rightarrow \mathbb{N}$ such that $\forall t \in \mathbb{R}^+, \exists t_1, t_2 \ : \ t \in [t_1, t_2)$ and $\forall t' \in [t_1, t_2) \ : \sigma(t) = \sigma(t')$. In other words, $\sigma(t)$ is a step function as follows:
>
> $$\sigma(t) = \begin{cases} i > 0 & \text{if } \tau_i \text{ is running} \\ 0 & \text{if the processor is idle} \end{cases}$$

An example of a schedule and the corresponding $\sigma(t)$ function is shown in Figure 2.4. At time instants t_1,t_2,t_3,t_4 and t_5, the processor performs a *context switch*. At time instant t_2, the execution of τ_1 is suspended and the processor is assigned to task τ_2 according to the scheduling decision (usually due to τ_2 having a higher priority). This operation is called *preemption*.

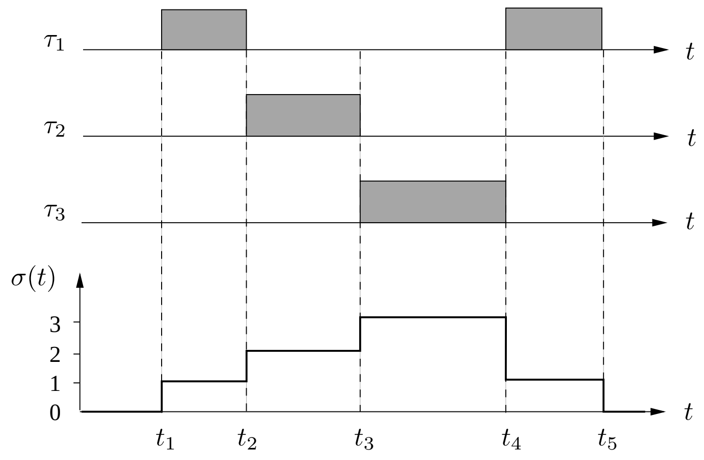


Figure 2.4: Example of schedule and $\sigma(t)$.

A schedule is said to be *feasible* if all the tasks are able to complete within a set of constraints, e.g. to all finish execution before deadlines. A task set τ is said to be *schedulable* if there exists a feasible schedule for it.

In general, the scheduling problem deals with the problem to assign m processors $P = \{P_1, P_2, \ldots, P_m\}$ and l types of resources $R = \{R_1, R_2, \ldots, R_l\}$ to n tasks $\tau = \tau_1, \tau_2, \ldots, \tau_n$ in order to complete all tasks under the imposed constraints [Bla96]. This problem has been shown to be NP-complete in its general form [GJ79], and hence several assumptions have to be made to reduce its complexity, e.g. restrict to uniprocessor platform, remove the precedence and shared resource constraints, or assume homogeneous task sets (with only periodic or only aperiodic tasks). These assumptions lead to various scheduling polices that are typically classified as follows:

- *Off-line.* The schedule of the task set is decided before the actual task activation, due to the fact that the complete information of the task set is known to the scheduling policy. In this case, the resulting schedule is stored in a table and the activation of tasks are accordingly triggered by the dispatcher during runtime. Therefore this kind of scheduling is also called *table-driven scheduling* [BS88].

- *On-line.* The scheduling decision is made during runtime, whenever a new task arrives or a running task terminates. This offers more flexibility than the off-line scheduling but also introduces larger overheads. A typical diagram of the implementation of the on-line scheduling is shown in Figure 2.5. Tasks that are activated will be first put into a *ready queue*, waiting for execution. The ready queue is sorted by the *scheduler* with respect to the ordering of priorities. The first task in the queue with the highest priority is dispatched to execute on the processor. If the task is preempted during the execution, it is put back to the ready queue. Otherwise, it will terminate after its completion. Notice that the scheduler acts like a priority assigner to tasks within the ready queue. Therefore, the algorithm used by the scheduler, i.e. the *scheduling algorithm*, is also referred to as *priority assignment scheme*.

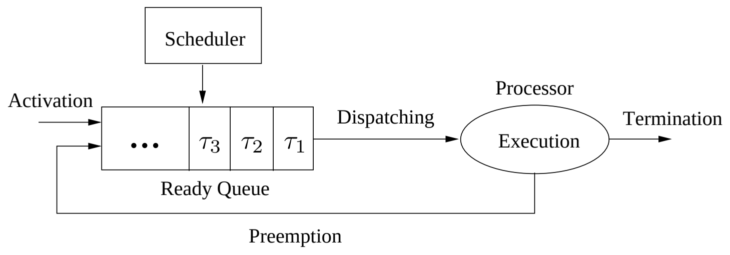


Figure 2.5: Diagram of on-line scheduling.

- *Preemptive.* Preemptive scheduling algorithms allow the running task to be interrupted at any time. The interruption causes the suspension of the currently running task and the assignment of the processor to a chosen task.

- *Non-preemptive.* With non-preemptive scheduling algorithms, once the task is started, it is executed by the processor until completion. Therefore there is no interference from other tasks.

With the advent of multiprocessor computing platform, there is increasing attention paid beyond the classic uniprocessor area. The scheduling algorithms in multiprocessor context can be further classified into:

- *Global.* A system-wide queue is used for all the ready tasks. The dispatcher then picks several of them to each execute on an available processor.

- *Partitioned.* Tasks are statically assigned to one of the processors and each processor keeps a ready queue. The scheduling algorithm plays locally on each ready queue as for a uniprocessor.

Figure 2.6 illustrates the difference between global scheduling and partitioned scheduling.

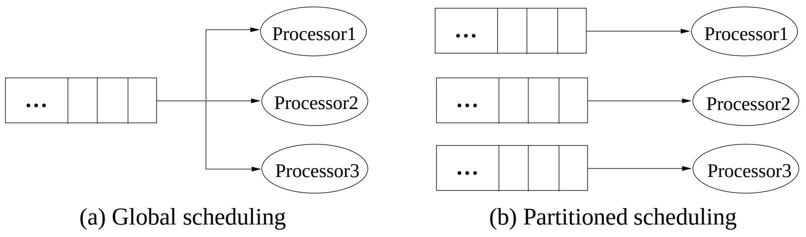


Figure 2.6: Difference between global scheduling and partitioned scheduling.

Optimal scheduling algorithms

To ensure the predictable behavior of a real-time system (especially hard real-time system), the feasibility of the task set should be guaranteed before the execution of tasks assuming worst-case scenario. The feasibility analysis for a task set means to find a feasible schedule if there exists one. However, it will be intractable if this is performed by checking the schedulability of the task set under numerous scheduling algorithms. Therefore it is important to introduce the concept of *optimality*, which refers to the fact that if the task set is not schedulable under the optimal scheduling algorithm, then it will not be schedulable under any other scheduling algorithms in the same category, that is using the same assumptions.

Concerning scheduling independent task set on uniprocessor, there exist two major scheduling algorithms, *Rate-Monotonic* (RM) and *Earliest-Deadline-First* (EDF), both considered as optimal scheduling algorithms in their respective categories.

- *Rate-Monotonic scheduling.* It assigns higher priorities to tasks with higher activation rates. Since the priorities are assigned according to static parameters and can be decided before runtime, it is considered as *fixed-priority* scheduling algorithm. It has been proven in [LL73] that RM is optimal among all the fixed-priority scheduling, if $D_i = T_i$ for all the tasks. Notice that when $D_i \leq T_i$, the *Deadline-Monotonic* (DM) [LW82] scheduling algorithm is optimal, where priorities are assigned according to the relative deadlines. Actually, RM is just a special case of DM.

- *Earliest-Deadline-First scheduling.* It assigns priorities to tasks according to the time to their absolute deadlines. The shorter time to the deadline, the higher priority is given to the task. Therefore, EDF is considered as a *dynamic-priority* assignment. Actually, as shown in [Der74], EDF is optimal among all the dynamic-priority scheduling algorithms, for either periodic or aperiodic task set.

2.1.3 Schedulability analysis

With the concept of optimality, the feasibility of the task set is verified by performing the schedulability analysis under the optimal scheduling algorithm. Some simple results on the schedulability analysis methods of Rate-Monotonic and Earliest-Deadline-First scheduling are hereby presented.

Utilization bound

Given a set of n periodic tasks, the *utilization* (also called *bandwidth*) U_i of each task is defined as the ratio between computation time and period, that is $U_i = C_i/T_i$, and the *total utilization* U of the task set is the sum of the utilization of all the tasks:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.1}$$

A sufficient condition for the schedulability of a task set under Rate-Monotonic scheduling is [LL73]:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{2.2}$$

The right part of the inequality can be interpreted as the utilization bound of a task set in terms of schedulability under RM, and it converges to $\ln 2 \approx 0.69$ as $n \to \infty$. This bound is expanded in [BBB03] named as *the hyperbolic bound*:

$$\prod_{i=1}^{n} (U_i + 1) \leq 2 \tag{2.3}$$

whose geometrical interpretation can be found later in Section 2.1.4.

Under EDF scheduling, the utilization bound approach applies as a necessary and sufficient condition when $D_i = T_i$ for all the tasks [LL73]. All the tasks meet their deadlines if and only if

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.4}$$

which implies that the processor can be always fully utilized.

Response time analysis

Given the following definition:

- **Response time** $R_{i,k}$ of a job $\tau_{i,k}$ is defined as the time between the arrival time and the finishing time, i.e.

$$R_{i,k} = f_{i,k} - a_{i,k} \tag{2.5}$$

- **Worst-case response time** R_i of task τ_i is the maximum $R_{i,k}$ of all jobs:

$$R_i = \max_k R_{i,k} \tag{2.6}$$

a necessary and sufficient schedulability condition for RM has been presented in [JP86] where the worst-case response time of each task is calculated and compared with its corresponding deadline. The worst-case response time R_i of task τ_i under RM is acquired by taking into account all the interference from other tasks with higher priorities and can be computed using the following recursive equation:

$$R_i = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.7}$$

where $hp(\tau_i)$ represents the set of tasks with higher priorities than τ_i. Therefore, the task set is schedulable if and only if $R_i \leq D_i$ for all the tasks. Notice that this condition is suitable for $D_i \leq T_i$.

The worst-case response time computation for EDF is more complicated than that for RM, and thus is not suggested for the schedulability analysis. However, it can still be used as a measure of task responsiveness, and hence will be described in Chapter 3.

Processor demand criterion

Concerning EDF scheduling, the condition in Eq. (2.4) is only necessary when $D_i \leq T_i$. To perform the exact schedulability analysis in this case, [BRH90] presented the processor demand criterion whose basic idea is that during any time interval $[t, t+L)$, the processing time required by the task set must not exceed L. The processor demand is defined as the cumulative computation time required by

Therefore, assuming a set of periodic tasks with deadlines less than periods is schedulable by EDF if and only if

$$\forall L \geq 0 \quad \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq L \tag{2.8}$$

where the set of check points of L can be restricted to a limited number (see [BRH90] for more details).

2.1.4 Sensitivity analysis

While schedulability analysis tries to answer the yes-or-no question of whether a task set is schedulable or not, the *sensitivity analysis* solves the problem of how much changes of parameters can be made keeping the task set feasible. Rather, it gives a measure of the affordable modifications on task parameters in the sense that the feasibility of the task set is not jeopardized. This is useful, for instance, when a certain degree of uncertainty is allowed for the WCET estimation in system design stage, or when deciding how to change the task periods in order to bring an unfeasible task set back to feasible.

The sensitivity analysis usually finds a *feasibility region* within which the task set is feasible. Therefore, the feasibility margin can be interpreted as the distance from the current parameter setting to the boundary of the feasibility region. An example of the feasibility region in terms of computation time under fixed priority scheduling [BDNB06] is shown in Figure 2.7. Task set τ is unfeasible because

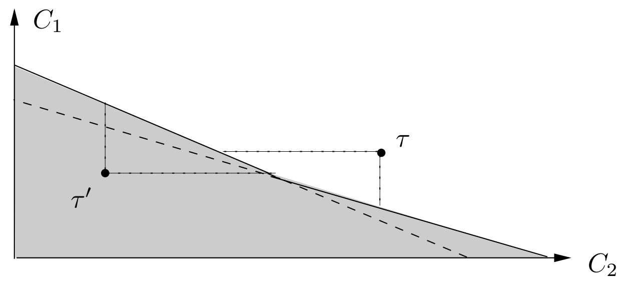


Figure 2.7: Sensitivity analysis in the C-space with 2 tasks.

it is outside the feasibility region (the C-space) denoted by the gray region. The distance from the point of the current computation times to the boundary of the feasibility region measures how much modification needed to make τ feasible. The dotted vertical and horizontal lines mean the minimum required changes on only C_1 or C_2, respectively. On the other hand, task set τ' is feasible since it resides within the C-space, and its location tells the margin of its feasibility. In the rest of this section, some results of sensitivity analysis related to this dissertation will be briefly described.

Utilization space

The utilization bound described in Section 2.1.3 for schedulability analysis can be envisioned as the *feasible utilization region* in the coordinate system of task utilization (the *utilization space* or *U-space*). An illustration involving 2 tasks is shown in Figure 2.8. Notice that the feasible region for 2 tasks under RM scheduling, given by either Eq. (2.2) or Eq. (2.3), is smaller than the feasible region under EDF scheduling restricted by the $U = 1$ bound. However, recall that the utilization bound condition for RM is only sufficient, which means the region between the hy-

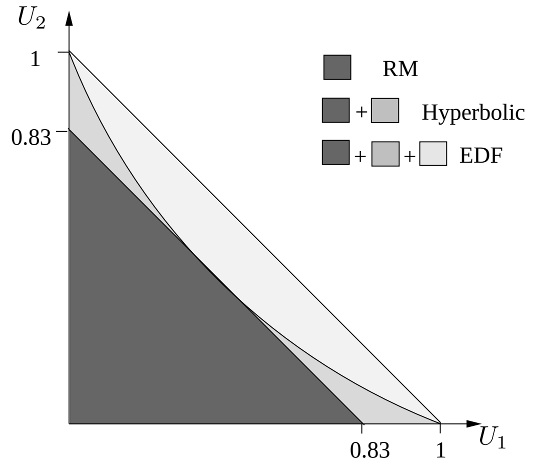


Figure 2.8: The feasible utilization region in the U-space of 2 tasks.

perbolic bound and the $U = 1$ bound remains unknown in terms of schedulability under RM.

Being $U_i = \frac{C_i}{T_i}$ and C_i fixed, the utilization space can also be directly related to the f-space, where frequency $f_i = \frac{1}{T_i}$. The exact feasible f-region under RM is described in [BDNB06].

EDF Deadline space

In [BB09b], the authors give the description of the deadline space under EDF scheduling. Given the computation times $\mathbf{C} = (C_1, \ldots, C_n)$ and the periods $\mathbf{T} = (T_1, \ldots, T_n)$ of n tasks, the exact feasible deadline region is given by the following formula:

$$\mathbb{S} = \bigcap_{\mathbf{k} \in \mathbb{N}^n} \bigcup_{i : k_i \neq 0} \{\mathbf{D} \in \mathbb{R}^n : D_i \geq \mathbf{k} \cdot \mathbf{C} - (k_i - 1)T_i\}$$

The geometrical interpretation of a simple task set consisting of 2 periodic tasks with parameters of $\mathbf{C} = (2, 3)$ and $\mathbf{T} = (4, 7)$ is plotted in Figure 2.9. The EDF deadline space and its usage in helping real-time control design will be detailed in Chapter 4.

2.1.5 Resource reservation

The *resource reservation* reflects the idea of reserving a certain amount of resource for one or a group of computing activities. In case of CPU time, the processing capacity of a cpu can be partitioned into a set of reservations, each equivalent to a virtual processor with reduced speed. Resource reservation can be implemented using server mechanism in the operating system, where each server may host one or several tasks. A desirable property of resource reservation is to provide *temporal*

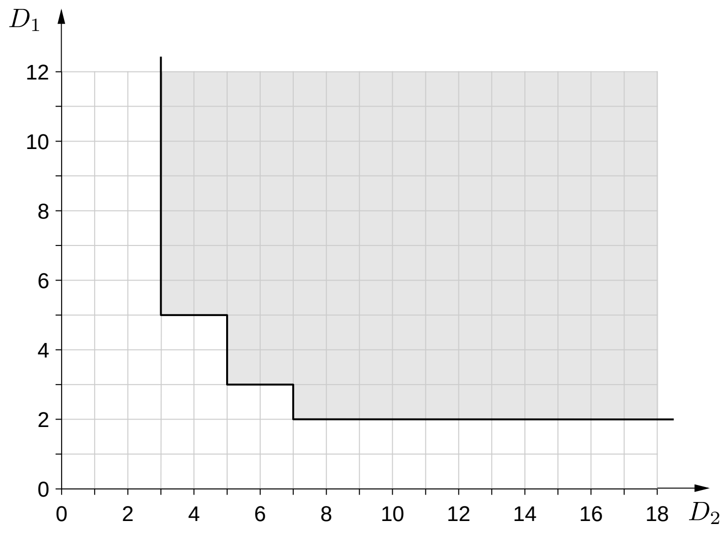


Figure 2.9: An example of the feasible region in EDF D-space of 2 tasks.

isolation between applications, in the sense that the occurrence of misbehavior such as overruns within one server will not affect the rest of the system. This can be applied to hybrid task sets comprising hard/soft real-time tasks and/or non real-time tasks, where the timing constraints of hard real-time tasks are guaranteed to be met while the average response times of soft and non real-time tasks are reduced.

An example of such a mechanism is presented in [AB98] where the Constant Bandwidth Server (CBS) is introduced to reserve a specified bandwidth to each server that is characterized by a pair (Q_s, T_s), where Q_s is the maximum budget and T_s is the server period. The ratio $U_s = Q_s/T_s$ is called server bandwidth. The bandwidth isolation property of the CBS reveals the fact that, in any time interval L, a task served by a CBS with bandwidth U_s will never demand more than U_sL.

Besides CBS, there are numerous other different server mechanisms proposed in the literature. To ease the analysis and provide common interface between different mechanisms and implementations, the (α, Δ) parameter pair has been proposed in [FM02] to characterize resource reservation mechanisms. Briefly speaking, α represents the bandwidth, and Δ means the time granularity, that is the maximum time an application may need to wait for being assigned some resource by the server. The (α, Δ) server will be explained further in Chapter 5.

2.2 Control Systems

Control system theory has continuously served the modern industry and society for almost two centuries. There are various definitions of control systems, from different points of view. The one from [CDHB04] is cited here:

A *control system* is an interconnection of components forming a system configuration that will provide a desired response.

The basic assumption for analysis of control systems is the cause-effect relationship for the components in a system. Therefore, a system component or *plant* [2] can be represented as a block, which accepts input and produces output, as shown in Figure 2.10. This cause-effect relationship representation allows to view the plant as a 'Black Box', facilitating the decomposition and analysis of the whole system.

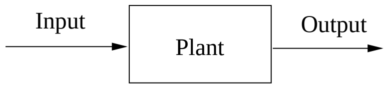


Figure 2.10: Plant as a block.

To obtain the desired output, an actuator is connected to the plant, which forms up a *open-loop control system*, shown in Figure 2.11. However, the open-loop

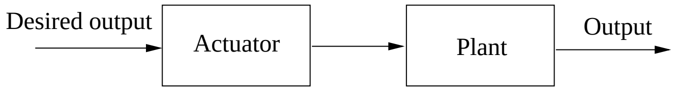


Figure 2.11: Open-loop control system.

control system usually fails to serve the purpose in the sense of producing desired response, if the precise information of the system is not available or an unexpected disturbance occurs.

To overcome the deficiency of the open-loop system, the revolutionary idea of feedback is introduced into the control system, where the actual output signal of the plant is measured and fed back to compare with the desired output value. The controller then makes decision on the compared result, that is the difference between two signals (which is also the reason why a negative operator is used on the feedback signal), and take action on the controlled plant. Such a system is called *feedback control system* or *closed-loop control system*. Figure 2.12 illustrates the diagram of the general form of a feedback control system. The signals in the closed loop are:

- r is the desired output, often called *reference*;
- y is the measurement of the actual output, called *output variable*;
- e is the difference (*error*) between the desired and the measured output;
- u is the *control signal*, also called *input variable* of the controlled plant.

In practice, a control system can be physically or logically divided into three subsystems, a sensory subsystem, a controller subsystem and a actuator subsystem.

[2]In control community, the controlled system is also widely called *process*. However, to avoid the name conflict with the *process* in the terminology of computer engineering, in this dissertation, the name *plant* will be mostly used.

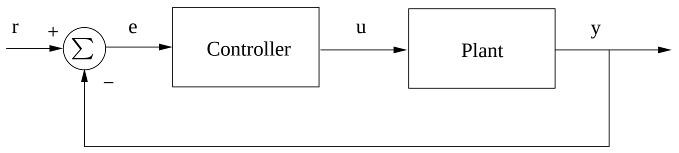


Figure 2.12: Diagram of feedback (closed-loop) control system.

The sensory subsystem measures the output of the controlled plant, sending the measurement signal to the controller subsystem. The controller produces control signal according to the measurement and sends it to the actuator subsystem. The actuator performs the action on the plant. Subsystems can be geographically sep-

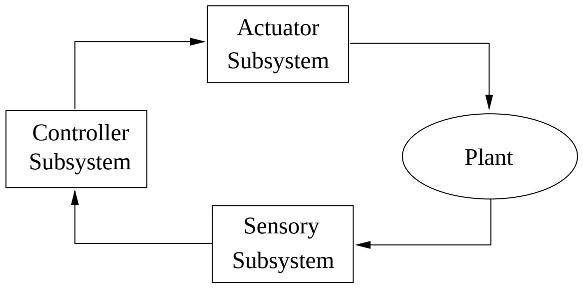


Figure 2.13: Subsystems in a feedback control system.

arated. For example, in *distributed control systems*, subsystems can be remotely connected through communication media (hence they are also called *networked control systems*). On the other hand, subsystems can also be grouped together, like in embedded systems.

There are two major problems considered in control theory, depending on different design concerns:

- The *servo problem* (or *tracking problem*), as depicted in Figure 2.12, concentrates on following the reference signal;
- The *regulation problem* mainly focus on making the system tolerant to external disturbances, e.g. measurement noise. The basic diagram of the problem is shown in Figure 2.14.

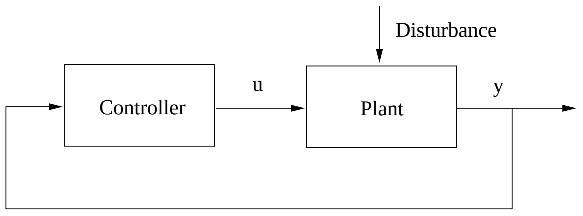


Figure 2.14: The regulation problem.

2.2.1 Control systems analysis

Model

Building up the mathematical models makes it possible to understand and analyze the complexity of the systems and perform control strategy. Because of the dynamic property, systems can be usually modeled by differential equations, utilizing physical laws, e.g. Newton laws, Euler-Lagrange laws, and Hamilton laws. By natural, all the physical systems are non-linear. However linear approximation is often possible within certain range of the system variables, i.e. assume small-signal conditions [CDHB04]. Moreover, if the system's response does not depend on the time at which the input is received, then the system is *time-invariant*. From a mathematical point of view, this means the coefficients of the differential equations are constants. In this work, we mainly consider *linear time-invariant (LTI) systems*.

With the system model in the form of differential equations, the system response can be obtained by solving the equations. However the linear time-invariant assumption allows to use *Laplace transform* to convert the model from *time domain* to *frequency domain*, and ease the difficulties of resolving differential equations. The *transfer function* model of a linear time-invariant system is then defined as the ratio of the Laplace transform of the output variable to the Laplace transform of the input variable, with all initial conditions assumed to be zero [CDHB04]. The transfer function model is also called *input-output model* because it clearly express the cause-effect relationship between system input and output. Figure 2.15 illustrates the relation between time domain and frequency domain and reveals the nature of Laplace transform. $\mathcal{L}(\cdot)$ means Laplace transform, and s is the Laplace variable which can be interpreted as the differential operator. Once the output in frequency domain $Y(s) = G(s) \cdot U(s)$ is calculated, the *inverse Laplace transform* can be performed to obtain the output in time domain, which is equivalent to the solution of the differential equation $f(t)$.

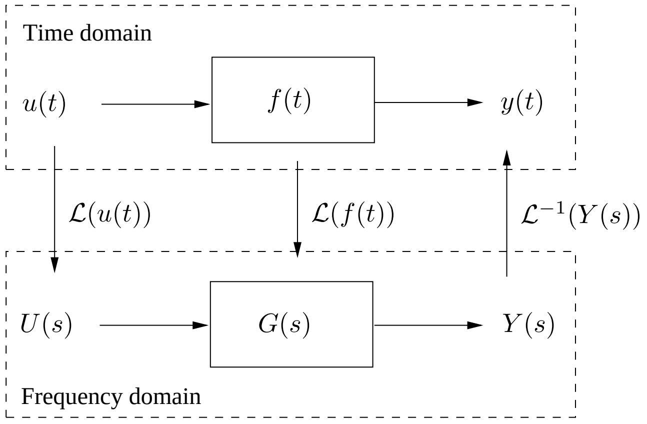


Figure 2.15: Laplace transform.

There exist numerous analysis methods and design principles for transfer function model. However it only fits for *single-input single-output (SISO)* systems. For a *multiple-input multiple-output (MIMO)* system, one must use the *state-space model,* which is essentially a group of first-order ordinary differential equations. It utilizes the states inside the system and describes their relations with input and output. Therefore the state-space model is sometimes called *internal model*, while in contrast the transfer function model is called *external model*. The general form of the state-space model is

$$\begin{aligned} \frac{dx}{dt} &= f(x,u) \\ y &= g(x,u) \end{aligned}$$

where $x = [x_1, x_2, \dots, x_n]$ is the *state vector* and n is the system order. Input u and output y are scalars (for SISO systems) or vectors (for MIMO systems). The state-space model of a linear time-invariant system can be expressed as

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{2.9}$$

where A is the state matrix, B is the input matrix, C is the output matrix, and D is the feedforward matrix[3]. The solution of Eq. (2.9) is given by

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\delta)}Bu(\delta)d\delta \tag{2.10}$$

$$y(t) = Ce^{At}x(0) + \int_0^t Ce^{A(t-\delta)}Bu(\delta)d\delta + Du(t) \tag{2.11}$$

Stability analysis

In the analysis and design of control systems, stability is the key issue of most importance. According to textbooks, the stability of a system is defined as follows [CDHB04]:

> A stable system is a dynamic system with a bounded response to a bounded input.

This is often referred to as *Bounded-input Bounded-output (BIBO)* stability.

Considerable attention has been devoted to the finding of various stability criteria in the evolution of control theory. A compendious description is given as follows:

- *Poles condition.* Defining the *characteristic equation* as the denominator polynomial of the system transfer function, the *poles* are the roots of the

[3] D is zero matrix when the system has no feedforward.

characteristic equation. A feedback system is stable if and only if the real parts of all the poles of the system transfer function are negative. The condition geometrically means all the poles should lie on the left side of the *s-plane*, as shown in Figure 2.16.

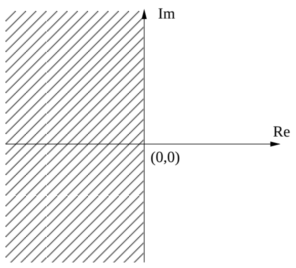


Figure 2.16: Stability region in the s-plane.

- *Routh-Hurwitz criterion.* This is the method for investigating the stability of a linear system by checking the coefficients of the characteristic equation. It allows to know the number of poles with positive real parts without actually computing their values.
- *Eigenvalues condition.* Each *eigenvalue* $\lambda_i(A)$ of the state matrix A corresponds to a root of the characteristic equation. Therefore, the system's stability is equivalent to that all the eigenvalues of the system have negative real parts.
- *Nyquist stability criterion.* By drawing the $G(i\omega)$ on a polar diagram as ω varies from 0 to ∞, the stability of the system can be investigated counting the number that the drawn contour encircles the (-1,0) point. This method is handy to determine the stability of the closed-loop system giving its open-loop model.
- *Lyapunov stability theorem.* The Lyapunov second theorem on stability uses a *Lyapunov function* $V(\cdot) : \mathbb{R}^n \to \mathbb{R}$ to in some sense act like an energy function of the system. The system is stable if the Lyapunov function decays over time, which can be visualized that the system loses energy and rests at some final state. Notice that the theorem is only sufficient condition, and is used on systems without input, i.e. autonomous systems.

The stability analysis discussed above is concerned with continuous-time domain (both time domain and frequency domain), while in Section 2.2.2 stability analysis for sampled control systems will be reviewed.

Control performance

The performance characterization of a control system is usually described by metrics in the transient response and the steady-state response to a unit step input.

Figure 2.17 depicts the response of a standard second order control system.

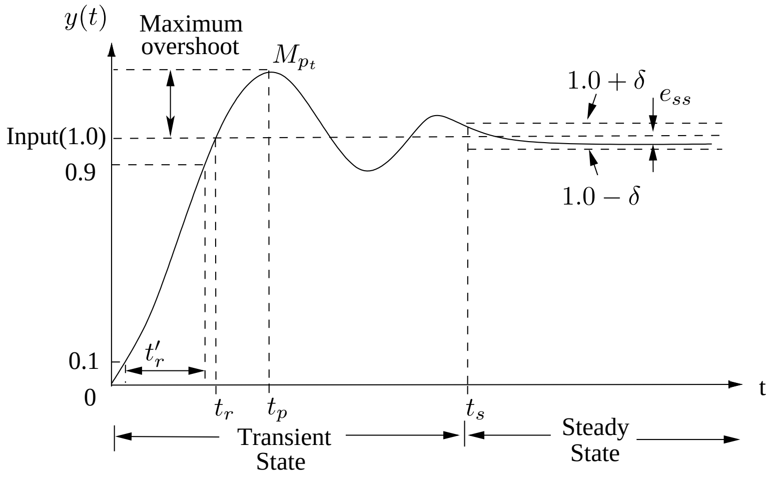


Figure 2.17: Unit step response of a standard second order control system.

As plotted, after the step input enters the system at time $t = 0$, the system enters the *transient state*, where the response of the system first rises, goes over the input, giving a overshoot, and then oscillates until it keeps in a small range around the input value, that is, when the system enters the *steady state*. This range is defined as $\pm\delta$ where δ is specified as a certain percentage of the input amplitude. Typical percentage values used in practice are 2% and 5%. The system is said to be *underdamped* when it oscillates to reach the steady state, while in contrast an *overdamped* system does not oscillate and hence has no overshoot.

The performance measures of the transient response and the steady-state response of the system are given by the following metrics:

- **Maximum overshoot** is the difference between the *peak value* M_{p_t} and the input amplitude;

- **Peak time** t_p is the time when the response reaches its maximum overshoot. Notice that for an overdamped system, t_p does not exist;

- **Rise time** t_r is defined as the time required for the response to rise from 0% to 100% of the final value for an underdamped system. For an overdamped system, the 10-90% rise time t_r' is used;

- **Settling time** t_s is the time required for the system to stay within the range $\pm\delta$, that is the finishing time of the transient state and the start time of the steady state;

- **Steady-state error** e_{ss} is the difference between the final value of the response and the input amplitude. This error exists due to the inner nature of the system.

Summarily speaking, the performance of the response in the transient state is described with two aspects:

- The *swiftness* is represented by the peak time and the rise time;
- The *closeness* (or *accuracy*) is represented by the maximum overshoot and the settling time.

And the performance of the response in the steady state is described with only the *closeness* (or *accuracy*), represented by the stead-state error. Control design requires a trade-off between the transient performance and the steady-state accuracy.

In modern control theory, *performance index* is used as a quantitative measure of the performance of a control system, which enables to make *optimal control* design based on mathematical computation rather than in an empirical way. The basic intention behind the performance index is similar to the performance metrics described above, that is, to consider both the swiftness and the accuracy of the response. Therefore, the performance index is expected to measures the error with respect to the desired response along the time. In [CDHB04], several such performance criteria are presented:

- *Integral of the squared error* $ISE = \int_0^{t_f} e^2(t)dt$
- *Integral of the absolute error* $IAE = \int_0^{t_f} |e(t)|dt$
- *Integral of the time-weighted squared error* $ITAE = \int_0^{t_f} t \cdot e^2(t)dt$
- *Integral of the time-weighted absolute error* $ITAE = \int_0^{t_f} t \cdot |e(t)|dt$

where $e(t)$ is the response error, and t_f is the final time for the performance measurement. Although in theory t_f should be ∞, it is usually substituted by a large enough time, e.g. some time beyond the rising time t_s. Notice that, the latter two criteria make emphasis on the errors occurring later in the response, thus treat the steady state more important than the transient state.

Some other performance criteria consider also the input variable $u(t)$, i.e. the output of the controller, and make a weighted combination with the error. This can be interpreted as the spent control energy. For instance, considering $u(t)$ into ISE (with w being the weighting factor) results in:

$$ISE' = \int_0^{t_f} \left(e^2(t) + w \cdot u(t)\right) dt$$

Let Q_1 and Q_2 be the weighting matrices, the performance index for state-space model can be expressed in the similar manner as:

$$J = \int_0^{t_f} \left(x^T Q_1 x + u^T Q_2 u\right) dt$$

which measures the system states variation and the expenditure of the control energy.

Controller design

Controller design is the action to make the system respond in the way such that stability and performance specifications are met. There are numerous techniques to design a controller among which a few are listed here:

- *PID controller* is a simple but powerful technique that has been widely used in industry. The main idea is to combine the proportional, integral, and derivative parts[4] in the controller to provide desired behavior of the closed-loop system.

- *Root-locus design* method draws a so-called root locus according to the changes in the system's feedback characteristics and other parameters, and shows how these changes influence the system poles.

- *Bode plot* depicts the magnitude curve and the phase curve of a system in frequency domain. It is also helpful for checking the *Gain margin* and *Phase margin* that are two important metrics to measure the relative stability[5].

- *Pole placement* allows to arbitrarily choose the location of the system poles to reach the desired transient dynamics and steady state. Being L the *state feedback gain vector*, the pole placement can be easily realized using *state feedback* where the control signal is produced in a linear feedback manner:

$$u = -L \cdot x \tag{2.12}$$

- *Optimal and stochastic control* has been a popular design tool for control systems. Optimal control means to design a controller that minimizes a specified cost function, usually a performance index. The stochastic control models the disturbances in the linear systems as random processes, and involves a quadratic cost function in the optimal design. When the disturbances are modeled as Gaussian processes, it is called the *Linear-Quadratic-Gaussian (LQG) control problem*, and the resulting controller is a *LQG controller* which is also in the form of state feedback.

2.2.2 Computer-controlled systems

In the previous section, the described control systems regard with continuous-time domain, which is mostly fundamental in the era of analog control. Nowadays, almost all the control systems have computing units within the control loop, as depicted in Figure 2.18 (taken from [ÅW97]).

Compared with the general closed-loop control system in Figure 2.12, in computer-controlled systems, the role of the controller is realized by a digital computer. The

[4]In some cases, only one or two terms are used, e.g. PI controller or PD controller.

[5]The relative stability answers the question of how stable is the system, rather than a simple yes-or-no question.

continuous-time signal $y(t)$ of the plant output is converted into digital signal by the analog-to-digital (A-D) converter and then delivered as a sequence of numbers $\{y(t_k)\}$ to the computer, where t_k is the sampling instant at which the conversion is done. The computer then handles the sampled signal, processes according to some algorithm, and generates a sequence of control signals $\{u(t_k)\}$. This sequence is converted to analog signal by a digital-to-analog (D-A) converter, usually being a zero-order-holder (ZOH) which keeps the control signal constant during two successive conversions. The operation events are synchronized using a real-time clock in the computer.

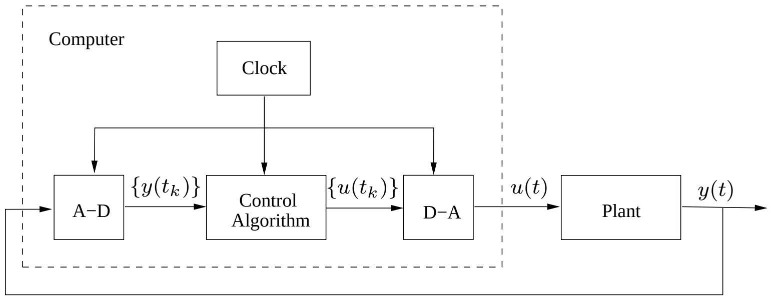


Figure 2.18: Diagram of a computer-controlled system.

In computer-controlled systems, both continuous-time signals and discrete-time (sampled) signals exit in the same control loop, leading to extra difficulties in analysis and design. However, there are basically two approaches that have successfully shown the capability of coping with the problem:

- *Discretization* of the continuous-time design means that the controller is designed in continuous-time domain and implemented by approximation using fast sampling. The approximation can be performed in several ways, such as Tustin's (or bilinear) approximation, Forward differences (Euler's method), and Backward differences. However, some approximation methods may lead to inaccurate mapping of the stability region [ÅW97]. Moreover, the fast sampling[6] required by the approximation may sometimes be too costly, especially in embedded systems with limited resources.

- *Sampled (Discrete-time) control theory* considers the system behavior at the sampling instants. The system is modeled by considering only the specific time instants and then can be analyzed and synthesized using similar approaches as in continuous-time domain, e.g. pole placement and optimal control design.

In sampled control theory, the sampling interval is usually assumed to be piecewise constant, and has to comply with the Nyquist-Shannon Sampling Theorem.

[6]It is suggested in[FPEN94] that the discretization yields reasonable results at sample rates of 20 times the natural frequency, and can be used with confidence for sample rates of 30 times the bandwidth or higher.

Besides, various rules of thumb suggest the choice of sampling periods depending on the dynamics of the control system and the desired control performance. For example, [ÅW97] suggests that the sampling period h can be chosen to give

$$\omega_n h \approx 0.1 - 0.6 \tag{2.13}$$

where ω_n is the desired natural frequency of the closed-loop system. Notice that, a long sampling interval saves resource consumption such as I/O operation and CPU usage, but may induce potential problems because the system evolves in open-loop between the sampling instants so that the disturbance can not be captured until the next sampling point.

Model

Difference equations are used to describe the input-output behavior of the discrete-time systems at the sampling instants, and play the same role in the analysis of discrete-time systems as the differential equations do with continuous-time systems. Additionally, *Z-transform* is the discrete-time analogy of the Laplace transform, and is devised to solve the linear difference equations.

State-space model remains its powerful capability in the discrete-time domain. Basically, the discrete-time state-space model comprise a group of first-order difference equations. The sampled version of Eq. (2.9) can be expressed by only considering the states at the sampling instants:

$$\begin{aligned} x(t_{k+1}) &= \Phi(t_{k+1}, t_k)x(t_k) + \Gamma(t_{k+1}, t_k)u(t_k) \\ y(t_k) &= Cx(t_k) + Du(t_k) \end{aligned} \tag{2.14}$$

where

$$\begin{aligned} \Phi(t_{k+1}, t_k) &= e^{A(t_{k+1}-t_k)} \\ \Gamma(t_{k+1}, t_k) &= \int_0^{t_{k+1}-t_k} e^{As} ds B \end{aligned}$$

For periodic sampling, the model becomes linear time-invariant by utilizing $t_k = k \cdot h$:

$$\begin{aligned} x(kh + h) &= \Phi(h)x(kh) + \Gamma(h)u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned} \tag{2.15}$$

where

$$\begin{aligned} \Phi(t) &= e^{At} \\ \Gamma(t) &= \int_0^{t} e^{As} ds B \end{aligned}$$

Stability analysis

A brief introduction of stability criteria in discrete-time domain is given bellow.

- *Poles/Eigenvalues condition.* The discrete-time system is stable if and only if all its poles are within the unit circle of the *z-plane*. Therefore the stability region of the left half plane of the s-plane (as shown in Figure 2.16) is mapped to the unit-circle in z-plane. The poles of a discrete-time system can be obtained by calculating the roots of the denominator of its *pulse-transfer function*. Besides, since each eigenvalue $\lambda_i(\Phi)$ of the matrix Φ corresponds to a pole, it is handy to acquire the poles with the state-space model.

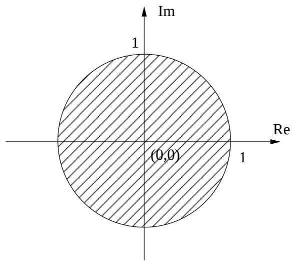


Figure 2.19: Stability region in the z-plane.

- *Spectral radius condition.* The spectral radius of a matrix is defined as the supremum among the absolute values of the eigenvalue, i.e. $\rho(A) = \max_i(|\lambda_i|)$. The discrete-time closed-loop system is stable if and only if the spectral radius of the closed-loop matrix ($\Phi_{cl} = \Phi - \Gamma L$, where L is the state feedback gain vector) is less than 1, i.e. $Stable \Leftrightarrow \rho(\Phi_{cl}) < 1$. This condition also allows to analyze the stability of a time-variant system. For example, in [MVFF01b], the spectral radius condition is used to perform the stability analysis for a control system with on-line compensated controller.

Other stability analysis methods [ÅW97] for discrete-time systems can be found to be similar to their continuous-time counterparts, including the Schur-Cohn-Jury's stability test, the Nyquist criterion, and the Lyapunov's Second theorem.

2.3 Real-time Control Integration

2.3.1 Control loop timing

In the traditional discrete-time control theory, the sampled version of the system model usually considers an equidistant sampling interval named as *sampling period*, as expressed in Eq. (2.15). Moreover, due to the time needed to compute the control algorithm, it can be incorporated in the model a computational latency, which is often assumed to be constant. This can be clarified by plotting the timing of the controller, shown in Figure 2.20. The plant output signal $y(t)$ is sampled at time instants $t_k = kh$ ($k = 0, 1, 2, \ldots$), separated by a constant sampling interval h. The sampled data $y(t_k)$ is sent to the controller, who calculates the control signal $u(t_k)$ and puts it to the ZOH. This process consumes a fixed amount of time δ, and therefore the latest active control signal is imposed on the plant δ time after the sampling.

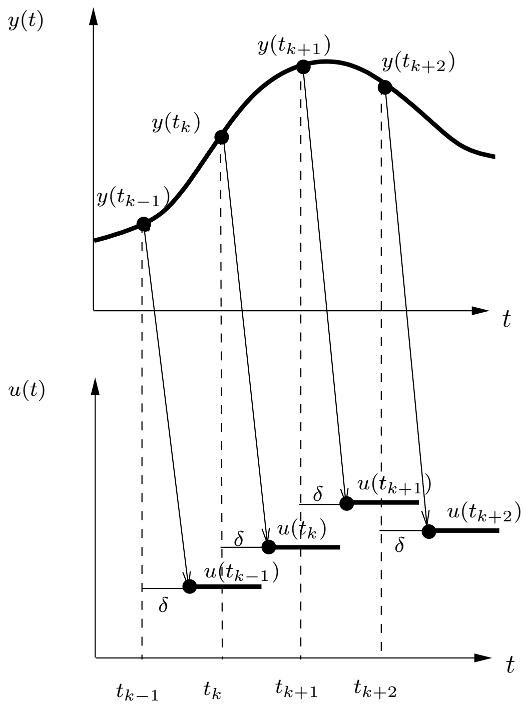


Figure 2.20: Ideal timing of a control task.

Assuming a constant delay less than the sampling period, i.e. $\delta < h$, the model of Eq. (2.15) is then extended to be

$$\begin{aligned} x_{k+1} &= \Phi(h)x_k + \Phi(h-\delta)\Gamma(\delta)u_{k-1} + \Gamma(h-\delta)u_k \\ y_k &= Cx_k \end{aligned} \tag{2.16}$$

Notice that a simplified notation is used, where the suffix k denotes time instant kh and hence x_k is equivalent to $x(kh)$. Plus, feedforward matrix D is assumed to be 0. The expression shows that in order to model the dynamics of the sampled system with constant delay, control signal of 1 sampling period before must be embodied into the equation. In other words, there is an extra state u_{k-1} in the extended state vector, and the state evolution equation of Eq. (2.16) can be reconstructed as

$$\begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} \Phi(h) & \Phi(h-\delta)\Gamma(\delta) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} \Gamma(h-\delta) \\ I \end{bmatrix} u_k \tag{2.17}$$

While longer delay exists, more extra states, i.e. previous control signals, must be taken into account. According to [ÅW97], if

$$\delta = (d-1)h + \delta' \qquad 0 < \delta' \leq h$$

where d is an integer, a number of d previous control signals are considered to be extra states in the expanded model.

The discrete-time state-space model (2.15) and its extended version (2.16) have shown that the timing issue can be addressed using the mathematical formulation. Therefore, the implementation should guarantee the timeliness assumed in the control design stage. In practice, real-time system theory is preferable to enforce such timing determinism.

There are several options to guarantee the timing assumed in the control design using real-time methodology. The off-line scheduling could be one of the candidates. In fact, the cyclic executive method is widely used in industry and served its original purpose. However, such static scheduling makes it arduous to modify and extend the already established application, and extremely difficult to adapt to resource availability and application changes. To overcome these issues, on-line scheduling is adopted by using real-time operating system support to fundamentally increase analyzability, feasibility, maintainability, and extensiblity.

Different choices of models exist for implementing a controller as a real-time task. For instance, sampling and actuation can be realized as interrupts, separated from the calculation part. However this solution increases the difficulty in analyzing schedulability and loses generality. Therefore, a naif control task model is often used in the real-time control systems. As shown in Figure 2.21, the plant of (2.16) is controlled by task τ_i. The equidistant sampling fashion is enforced by

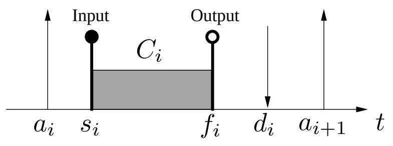


Figure 2.21: Naif control task model.

setting the task period $T_i = h$. Sampling and actuation occur at the beginning and termination of each job's execution, i.e. the start time s_i and the finishing time f_i, respectively. Task deadline $d_i = a_i + D_i$ is used to bound the delay. It is worth mentioning that, from the computer's point of view, the sampling is the input to the controller while the actuation is the output. Hence the input and output are inverse of the ones for the controlled plant, where the input variable is the control signal and the output variable is the sampled signal (see Section 2.2).

Notice that, if D_i is set equal to δ in the extend model of (2.16), then the task model can be used to precisely describe the sampled system's behavior (assuming constant computation time). However, this may cause significant impairment on the schedulability of the real-time system. On the other hand, if D_i is relaxed to be larger than δ, the actual delay becomes different from δ and inconstant during runtime. In other words, the naif task model brings timing uncertainty (temporal nondeterminism) into the control system.

In general, when a control task τ_i runs with other tasks on the same processor, it suffers interferences and experiences variable sampling period and inconstant delay, which is illustrated in Figure 2.22. Input and Output are abbreviated us-

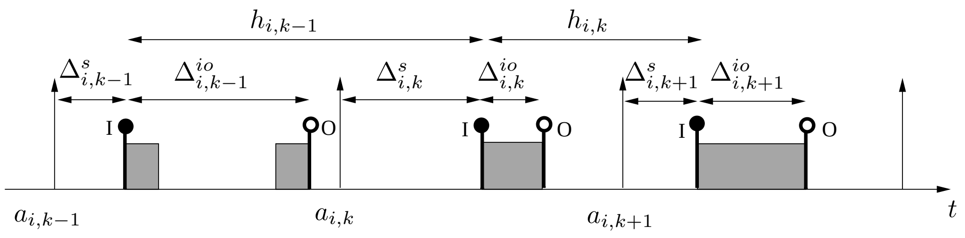


Figure 2.22: Timing uncertainty of a control task τ_i.

ing notation I and O. To characterize the timing of the control task, the *timing attributes* are commonly used and are described below.

Each job of the control task experiences two types of latency, defined as

- **Sampling delay** $\Delta^s_{i,k}$ is the latency from the ideal sampling instant to the actual sampling instant, that is the time between the arrival time $a_{i,k}$ and the start time $s_{i,k}$ of each job:

$$\Delta^s_{i,k} = s_{i,k} - a_{i,k} \tag{2.18}$$

- **Input-output delay** (*IO delay*) $\Delta^{io}_{i,k}$ is the latency from the sampling instant to the actuation instant, that is the time between the start time $s_{i,k}$ and the finishing time $f_{i,k}$ of each job:

$$\Delta^{io}_{i,k} = f_{i,k} - s_{i,k} \tag{2.19}$$

The sampling delay $\Delta^s_{i,k}$ is induced by the blocking after the control task is released, due to a running task with higher priority, and is variable from job to job. Similarly, the interferences (preemption or blocking on a mutual exclusive resource) from other tasks, as well as the varying computation time, make the input-output delay $\Delta^{io}_{i,k}$ inconstant. The variation of these two types of latency are denoted by

- **Sampling jitter** $\jmath^s_i$ is the maximum difference between $\Delta^s_{i,k}$ of all the jobs:

$$\jmath^s_i = \max_k \Delta^s_{i,k} - \min_k \Delta^s_{i,k} \tag{2.20}$$

- **Input-output jitter** (*IO jitter*) $\jmath^{io}_i$ is the maximum difference between $\Delta^{io}_{i,k}$ of all the jobs:

$$\jmath^{io}_i = \max_k \Delta^{io}_{i,k} - \min_k \Delta^{io}_{i,k} \tag{2.21}$$

Note that the sampling jitter leads to jitter in the nominal sampling period h_i. In fact, the actual sampling period $h_{i,k}$ is not invariable, clearly shown in Figure 2.22, whose variation is denoted by

- **Sampling period jitter** $\jmath_i^h$ is the maximum difference between all the actual sampling period $h_{i,k}$, which is quantified by

$$\jmath_i^h = \max_k h_{i,k} - \min_k h_{i,k} \tag{2.22}$$

  and according to [Cer03], it is upper bounded by

$$\jmath_i^h \leq 2\jmath_i^s$$

It has been widely acknowledged that delay and jitter have significant impact on the correct behavior of control systems, which implies that, if not properly taken into account, they may result in degradation of the control performance, and even lead to instability of the system [MVFF01a, CHL$^+$03]. Therefore, analysis of control loop timing and understanding of how the timing affects the control system are of great importance in real-time control co-design.

2.3.2 State of the art

In this section, some existing methods for real-time control co-design are reviewed. These methods are categorized into several tendencies based on [ÅCES00].

Task models

In Section 2.3.1, a naif control task model was introduced. This general task model facilitates to perform real-time analysis, e.g. schedulability check, but may not achieve the best control performance. Therefore several improved task models are proposed by different researchers to enhance the control performance during system runtime.

[Cer99] proposes to split the control task into two subtasks, where the first subtask calculates the control signal and the second subtask updates the internal states of the controller. In this way, the control signal can be output as soon as it is ready, reducing the latency in the control loop.

[CRA99] proposes to partition the control task into three parts: data acquisition, algorithm evaluation and action delivery, and the variable delay of the control activities is reduced by determining the minimum interval where the control action has to be allocated.

In [HHK03], the proposed task model uses two synchronization points, i.e. sampling and actuation, which are located at the release time of the current job and the next job. This method gives precise timing of the control task and thus allows to achieve better modeling and controller design. However, the introduction of synchronization points leads to several drawbacks, including the difficulty in schedulability analysis and the longer delay in the control loop.

The one-shot model presented in [LVM08] improves the above model where only one synchronization point is used at each actuation instant, while the sampling requires no synchronization point and is assumed to take place at the start time of each job. Therefore, the model allows the state-space controller to know the latency from the sampling instant to the actuation, and incorporate it into the system evolution equations.

Although these task models show the higher capability of predicting timing behavior and improving control performance, the naif task model is still suitable to use due to its generality. This dissertation will mostly use the naif task model.

Delay/Jitter Compensation

To cope with the problem of delay and jitter in real-time control applications, different techniques have been developed.

[NBW98] analyzes the performance and stability of real-time control systems with varying delays, and derives an optimal stochastic controller to compensate for jitter. The controller uses timestamps to track the sensor-to-controller and controller- to-actuator delays.

[LC02] uses a more realistic approach where the output jitter experienced in one period is compensated for in the next period. The resulting jitter-compensating controller can be viewed as a generalization of the well-known Smith predictor, and the design of the compensator does not require a full process model.

[MFFR01] presents a method to on-line compensate the control performance degradation caused by jitter. The compensation is achieved by adjusting the parameters of the controller at each job with the help of timing measurements provided by the real-time operating system. The stability analysis for such kind of controller is presented in [MVFF01b]. This kind of compensation method, when used on-line, introduces extra computation overhead into the system. If such overhead is significant, then controller parameters should be pre-calculated off-line and stored in a lookup table, which requires extra memory space.

Parameter selection

Parameter selection refers to the integrated approach of choosing task parameters to meet both control performance requirement and resource utilization requirement.

[SLSS96] presents an integrated approach where task frequencies (periods) are selected to comply with the schedulability constraints and an optimization problem is solved to minimize the control performance difference between the continuous-time design and the discrete-time implementation. This performance difference is approximated as an exponential function of the sampling frequency.

Instead of the period parameter, [RS00] uses the time slot length as the granularity of the schedule, and presents a method to decide the best off-line static cyclic schedule of several control tasks to optimize the overall system performance.

More related work will be reviewed in Chapter 4.

Feedback scheduling

When the control plant or the system workload is highly dynamic, it could be advantageous to adapt the task parameters on-line. In [SLS99, LSTS99, LSA$^+$00] and other similar works, feedback mechanism is employed to keep the real-time system acting with a stable behavior. This type of feedback scheduling is also known as *resource manager*.

However, the more control-related cases consider control tasks rather than regular real-time tasks in the system. Therefore, the feedback scheduler makes decision based on not only the system workload situation, e.g. deadline miss ratio, but also the status of the controllers implemented by the control tasks.

[EHÅ00] proposes to use a recursive optimization procedure to on-line change sampling periods of a group of control tasks to keep the system utilization at a stable level while maximizing the overall control performance. The feedback signal is the execution time change of each task.

[MLB$^+$04] presents a feedback-based resource management model that allows to allocate resources to control tasks as a function the current states of their controlled systems. It is shown that using this dynamic allocation mechanism based on the actual needs of the controllers, the available resources are well utilized to provide better control performance than using static resource allocation.

[HC05] presents a feedback scheduling strategy to dynamically adjust sampling rates for a set of LQ-controller tasks. The control performance is analytically expressed as a function of the sampling period and the state of controlled system, and is used for on-line sampling period adjustment.

Resource reservation

Some control applications may have highly variable execution times, such as visual tracking. In these cases, the controller normally executes with short computation times under most situations and only occasionally experiences the worst-case execution time. Therefore, the WCET assumption might lead to significant under-provision of computing resources, and hence is inefficient for performance optimization.

[CBS00] proposes to use the nominal computation times instead of the WCETs in optimizing sampling periods. Task overruns are handled by the presented Hard CBS (CBS^{hd}) algorithm.

[PAC$^+$00] suggests that certain amount of deadline misses may be tolerant due to the inherent robustness of the control systems. Therefore strict deadline constraints can be relaxed to enable higher sampling rates. The deadline miss ratio can be bounded by CBS if the probability distribution function of the execution times of the control task is known.

[CE03] presents the Control Server which gives small latency and jitter, and isolates timing misbehavior between unrelated tasks. A key property of the model is that both schedulability and control performance of a control task will depend on the reserved utilization factor only.

Event-driven control

Recent researches have shown that if the equidistant sampling period constraint is relaxed, computing resources usage may be reduced while the control performance is maintained. This leads to interesting topics in the integration of event-driven control and real-time systems theory. However, since this dissertation will focus on periodic control, only a brief review will be listed.

[JHC07] presents the analysis and performance evaluation of the event-based control of first-order stochastic systems. A minimum inter-event time is defined to treat the control tasks as sporadic tasks so that the system schedulability is guaranteed. The results indicate that the sporadic control can achieve better performance than periodic control in terms of reduced process state variance and control action frequency.

[DLCHZ07] proposes a self-triggered control task model which decides its next release time at each job execution to enforce upper bounds on the induced L_2 gain of a linear feedback control system. To ensure the schedulability of the system, the self-calculated release time is sent to the elastic scheduling algorithm [BAL98] which assigns the actual release time to the control task.

[VMB08] presents a framework to accommodate several existing event-driven control approaches and shows the schedulability analysis for a set of control-driven tasks using both Fixed Priority and Earliest Deadline First.

2.3.3 Analysis Tools

Owning to complex relations between control performance and timing attributes, as well as between timing attributes and implementation parameters [THÅ$^+$06], it might be intractable to express such relations in analytical way. However, several tools have been acknowledged to support the analysis of complicated nature of the real-time control problem.

Jitterbug

Jitterbug is a Matlab-based toolbox that allows the computation of a quadratic performance criterion for a linear control system under various timing conditions [LC02]. The toolbox is built on the LQG theory and jump linear systems.

In Jitterbug, a control system is built by a signal model and a timing model. The signal model includes a number of inter-connected continuous-time and discrete-time systems, in either state-space form or transfer function form, and are each associated with a continuous-time quadratic cost function for performance evaluation. The timing model consists of a group of timing nodes, each corresponding to zero or more discrete-time systems in the signal model. Timing nodes are connected so that a next node will be activated after the previous node is finished [7]. At each activation of a timing node, the corresponding discrete-time systems will be updated.

The timing model therefore is used to describe the timing behavior of the actual runtime of a real-time system. The first timing node can be activated in a periodic fashion (every h seconds) or aperiodic fashion to model the time-driven or event-driven controller. Between two timing nodes, a latency δ with a discrete-time probability density function can be specified to model the delay and jitter in runtime. Notice that in periodic systems, when the total delay exceeds the period h, the remaining timing nodes are skipped for activation. This models the behavior in hard real-time systems where control tasks must finish before the next sampling, however brings some limitation which will be mentioned in Chapter 4.

It is worth mentioning that Jitterbug toolbox also provides a convenient function to make LQG design. The *lqgdesign* function designs a discrete-time controller for a continuous-time LTI plant with a constant time delay and a continuous-time cost function (see Jitterbug manual [CL06] for more details).

TrueTime

TrueTime is a Matlab/Simulink-based simulator, which facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics [OHC07, CHL$^+$03].

A TrueTime simulation is constructed by connecting standard simulink blocks, which gives flexibility to easily build control systems. Besides, a few TrueTime-specific blocks are provided by the toolbox, including:

- *TrueTime kernel* is a block to simulate a real-time kernel. It was originally assumed for only uniprocessor, and starts to support simulating multiprocessor platform since TrueTime 2.0.

- *TrueTime networks* simulates medium access and packet transmission in a local area network, including a wired version and a wireless version. The possible network models are

  - For wired networks: CSMA/CD (e.g. Ethernet), CSMA/AMP (e.g. CAN), Round Robin (e.g. Token Bus), FDMA, TDMA (e.g. TTP), and Switched Ethernet.
  - For wireless networks: IEEE 802.11b/g (WLAN) and IEEE 802.15.4 (ZigBee).

- *TrueTime battery* is a block to mimic the battery charging and recharging.

[7]There is possibility to have alternative execution path, in the sense that the next activation timing node is picked among a group of candidates.

To customize the different configuration for these blocks, initialization scripts should be written. Moreover, executive scripts during simulation can also be specified. All these scripts can be written as either Matlab M-files or C++ codes. The former one gives ease of using the familiar Matlab syntax and APIs and requires no compilation, while the latter one increases the simulation speed.

The initialization scripts define the setup of the real-time kernel, as well as the networks, and create tasks, interrupt handlers, timers, events, monitors, etc for the simulation. The scheduling policy of the real-time kernel can be one of the predefined classic scheduling algorithms like EDF or RM, or it can be any user-defined priority function. The execution of the tasks and interrupt handlers, written in user scripts to perform jobs like I/O and control, are then scheduled according to the scheduling policy during the simulation.

Because of the seamless connection with Matlab/Simulink, TrueTime is considered to be a powerful tool to make extensive simulation, detailed analysis and system-wide real-time control co-design.

S.Ha.R.K

S.Ha.R.K (**S**oft and **Ha**rd **R**eal-time **K**ernel) is a highly configurable uniprocessor real-time kernel designed for supporting hard, soft, and non real-time applications on PC of x86 architecture. It includes device drivers for most common hardware, making it possible to easily interact with the environment. For example, it can be used on PC to act as a controller, with the help of I/O devices, and hence enables experiments on integrated real-time control systems.

The modular component-based interface for the specification of scheduling algorithms makes it extremely easy to utilize and evaluate existent or new scheduling policies. Moreover, the hierarchical structure of the scheduling modules further facilitates the system-level composition and interchangeability of multiple scheduling algorithms. This is illustrated in Figure 2.23. Each task is associated with a

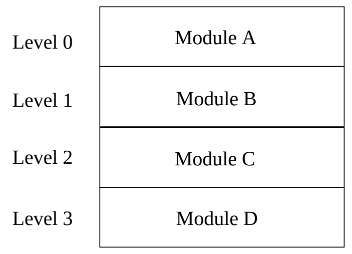


Figure 2.23: Hierarchical structure of scheduling modules.

scheduling module. Modules are ordered from top to bottom as levels. All the events of a task is scheduled by its associated module, in foreground of the tasks belonging to a lower-level module. In other words, each task have an extra fixed global priority specified by the index of the level at which its belonged module stays.

2.4 Conclusion

In this section, the background knowledge of real-time control integration has been introduced. In particular, basic concepts in the real-time systems have been described, and popular technologies have been presented, including scheduling policies, schedulability analysis, sensitivity analysis and resource reservation. A brief introduction of control systems has been given, as well as the discrete-time control theory for computer-controlled systems. The timing characterization for real-time control integration has been detailed. Finally, state of the art has been reviewed and several analysis tools have been described.

Chapter 3

Improved responsiveness using limited-preemption

3.1 Introduction

Limited-preemption EDF scheduling (LP-EDF) has been introduced by Baruah in [Bar05] to join the beneficial effects of both preemptive and non-preemptive scheduling. The main benefit of non-preemptive scheduling is indeed the reduced number of context switches, with a limited scheduling overhead due to cache misses and to the additional need of storing the state of a preempted task in order to safely retrieve it when the task will be resumed. On the other side, executing each task non-preemptively might lead to limited schedulability performances due to the large blocking imposed on tasks with smaller deadlines. With LP-EDF, instead, a task is executed non-preemptively as long as this does not cause the system to become unschedulable. When the task executed for the maximum allowed non-preemptive interval, the processor is surrendered to the ready task having earliest deadline, according to EDF. Pseudopolynomial complexity algorithms are presented in [Bar05, BB09a] to compute the durations of the maximum Non-Preemptive (NP) chunks for each task in the system. In this way, one can take advantage of the optimality of preemptive EDF with a reduced system overhead.

However, the benefits of limited preemption EDF are not limited to the smaller number of preemptions introduced in the system. This chapter exploits this technique to increase the responsiveness of a selected set of tasks, improving the control performance of a control system.

3.2 Related Work

In this section, we briefly remind the main results on non-preemptive and limited-preemption scheduling.

In [JSM91], Jeffay *et al.* proved that EDF is optimal even among non-preemptive work-conserving scheduling algorithms[1] for periodic or sporadic task sets. For these systems, a necessary and sufficient schedulability test with pseudo-polynomial complexity is provided. Moreover, it is shown that both the scheduling problem — find an algorithm that is able to schedule a feasible task set — and the feasibility problem — decide if a set of tasks can be scheduled without any deadline be missed — are NP-hard in the strong sense for concrete periodic task systems scheduled by non-preemptive algorithms[2].

Baruah and Chakraborty analyzed in [BC06] the schedulability of non-preemptive task sets under the recurring task model and showed that there exist polynomial time approximation algorithms for both preemptive and non-preemptive scheduling.

Mok and Poon presented in [MP05] sufficient conditions to guarantee the robustness (a.k.a. sustainability) of non-preemptive task systems, i.e., guaranteeing that the schedulability is not affected by the relaxation of one or more task timing requirements (like a decrease in the computation time or an increase in the period).

The idea of deferring preemptions until pre-determined points inside the task code has been first introduced by Burns in [Bur94]. An algorithm called Fixed Priority with Deferred Preemptions (FPDP) has been proposed, describing as well an associated response time analysis. However, a flaw in the analysis has been later corrected by Bril *et al.* in [BLV07].

With a similar idea, Baruah analyzed in [Bar05] the Limited Preemption EDF scheduling algorithm (LP-EDF). The maximum amount of time for which a task may execute non-preemptively, without missing any deadline, is computed. Differently from the model adopted in [Bur94, BLV07], there are no fixed preemption points, but the position of Non-Preempting Regions (NPR) may float inside the task code (provided it is shorter than the allowed length). The computation of the maximum NPR lengths has been later improved in [BB09a].

A response time analysis for preemptive EDF has been described by Spuri *et al.* in [SB96, SSRB98]. Such analysis has been extended to systems scheduled with non-preemptive EDF by George *et al.* in [GRS96]. Palencia and Gonzalez applied similar techniques for more general (distributed) task systems in [GH05].

The idea of exploiting non-preemptive scheduling to improve control performances has been adopted by Buttazzo and Cervin in [BC07], where non-preemptive EDF is used to reduce task jitter.

3.3 System Model

We will consider a set τ composed by n periodic and sporadic real-time tasks. Each task τ_i is defined by a worst-case execution requirement C_i, a relative deadline D_i

[1]A scheduling algorithm is work-conserving if the processor is never idled when a task is ready to execute. Note that EDF is not optimal among general non-preemptive schedulers (including non work-conserving ones).

[2]A concrete periodic task is a periodic task that comes with an assigned initial activation.

Figure 3.1: Placement of the final NP chunk of a task $\boldsymbol{\tau_i}$.

and a period, or minimum interarrival time, T_i (all parameters are assumed in the real numbers domain). Such a sporadic task generates an infinite sequence of jobs $\tau_{i,k}, k \in \mathbb{N}$, with the first job arriving at any time, and successive job-arrivals separated by at least T_i time units. Each job $\tau_{i,k}$ has an arrival time $a_{i,k}$, a finishing time $f_{i,k}$, and a deadline $d_{i,k} = a_{i,k} + D_i$. The starting time $s_{i,k}$ of job $\tau_{i,k}$ is the first time it is scheduled for execution.

We consider the scheduling of sporadic task systems upon a single processor, using the **Earliest Deadline First** (EDF) scheduling algorithm [LL73] with limited preemption (LP-EDF) [Bar05, BB09a]. For each task τ_i, we will assume to know in advance the maximum amount of time for which it can execute non-preemptively. Such value can be computed using the techniques described in [Bar05, BB09a], and will be denoted as Q_i. Note that $Q_i \leq C_i, \forall i$. Whenever $Q_i = C_i$, the task τ_i will always be executed non-preemptively.

3.3.1 Placement of NP chunks

In order to improve control performances for one or more tasks τ_i, we will place an NP region with length Q_i at the end of its worst-case execution, i.e., at time $C_i - Q_i$, as shown in Figure 3.1. This can be done considering the code executed by the task when it produces the largest worst-case execution time, and placing a preemptions disable command as close as possible to the instruction executed Q_i time-units before the end. When this is not possible, for instance because at time $C_i - Q_i$ the task is inside a loop, or it is calling a remote function which cannot be modified, the preemptions disable instruction is placed as soon as possible, resulting in a smaller non-preemptive region. Without losing generality, we assume Q_i to denote the effective length of such non-preemptive region.

3.3.2 Timing Attributes

To investigate the responsiveness of limited-preemption EDF, we will consider the timing attributes defined by Eq. (2.18)-Eq. (2.21) in Section 2.3.1. Besides, we also consider the response time as a measure of the responsiveness of a real-time task. The response time of a job $\tau_{i,k}$ is defined with Eq. (2.5) in Section 2.1.3. Here, the definition of its variation is given as:

- **Response Jitter** j_i^r is the maximum difference between $R_{i,k}$ of all the jobs:

$$j_i^r = \max_k R_{i,k} - \min_k R_{i,k} \tag{3.1}$$

Moreover, for each one of the job-based attributes $X_{i,k}$ (being X one of R (response time), Δ^s (sampling delay) and Δ^{io} (input-output delay)), we define the corresponding task-based average $\bar{X}_i$ values, taking the average among all jobs of a task τ_i:

$$\bar{X}_i = \lim_{m \to \infty} \frac{\sum_{k=1}^{m} X_{i,k}}{m} \tag{3.2}$$

3.4 Response Time Analysis

The worst-case response time of a task in a system scheduled with EDF has been computed by Spuri et al. in [SB96, SSRB98]. We briefly remind here the adopted technique.

Definition 1 (Deadline Busy Period)**.** *A deadline-d-busy period is an interval of continuous execution in which only instances with absolute deadline before d are scheduled*[3]*.*

Theorem 1 (from [SB96, SSRB98])**.** *The worst-case response time of a task τ_i is found in a deadline busy period in which all tasks but τ_i are released synchronously from the beginning of the deadline busy period, and at their maximum rate.*

3.4.1 Worst-Case Response Time of EDF

Exploiting Theorem 1, it is possible to compute the worst-case response time of each task τ_i, considering all deadline-$(a + D_i)$-busy periods for a meaningful set of possible release times a of jobs of τ_i, and taking the maximum response time among such jobs, as follows [SB96, SSRB98]:

1. The maximum busy period length L is found considering a situation in which all tasks are released synchronously and at their maximum rate. Therefore, L is the smallest positive value satisfying the following equation:

$$L = \sum_{\tau_i \in \tau} \left\lceil \frac{L}{T_i} \right\rceil C_i.$$

2. For each task τ_i, the maximum deadline-busy period L_i is found considering all tasks but τ_i synchronously released at time $t = 0$. An algorithm for the computation of all L_i values is presented in Figure 3.2.

3. The length $L_i(a)$ of the deadline-$(a + D_i)$-busy period of a job of task τ_i that arrives a time units after the synchronous arrival of all other tasks is the

[3]Note that a *busy period* (without referring to any particular deadline) is instead just an interval of continuous execution.

```
COMPUTE BUSY PERIOD LENGTHS($\tau$)
1  $A \doteq \{a < L \mid a = kT_j + D_j,\ 1 \le j \le n,\ k \in \mathbb{N} \cup \{0\}\}$
2  $L_{n+1} \leftarrow L$
3  for ($i = n$) down to $1$
4      repeat
5          $a \leftarrow \max\{\ell \in A \mid \ell \le L_{i+1} - C_i + D_i\} - D_i$
6      until ($L_i(a) > a$)
7      $L_i = L_i(a)$
8  return $L_1, \ldots, L_n$
```

Figure 3.2: Algorithm for the computation of the maximum deadline busy period lengths.

smallest positive value satisfying the following equation:

$$L_i(a) = \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i + \sum_{\substack{j \neq i \\ D_j \le a + D_i}} \min\left\{ \left\lceil \frac{L_i(a)}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} C_j. \tag{3.3}$$

4. The maximum response time of τ_i can be found as

$$R_i = \max_{a \in A_i}\{L_i(a) - a\}, \tag{3.4}$$

where

$$A_i \doteq \{a < L_i \mid a = kT_j + D_j - D_i,\ \forall j,\ k \in \mathbb{N} \cup \{0\}\}.$$

As proved in [GRS96], if the task set utilization is strictly lower than 1, L exists and is pseudopolynomial, so that the algorithm may converge in a pseudopolynomial number of steps. If instead the total utilization is 1, the maximum busy period length L is bounded by the least common multiple of the periods of the tasks, when such value exists. In that case, the complexity of the algorithm is exponential.

In the next section, we will show how to modify the algorithm in order to take into account non-preemptive regions.

3.4.2 Worst-Case Response Time of LP-EDF

George et al. presented in [GRS96] a method to compute the worst-case response time for task systems scheduled with non-preemptive EDF. The method takes into account the effect of priority inversion for the computation of the deadline busy period and can be used as well in analyzing the limited preemption EDF case.

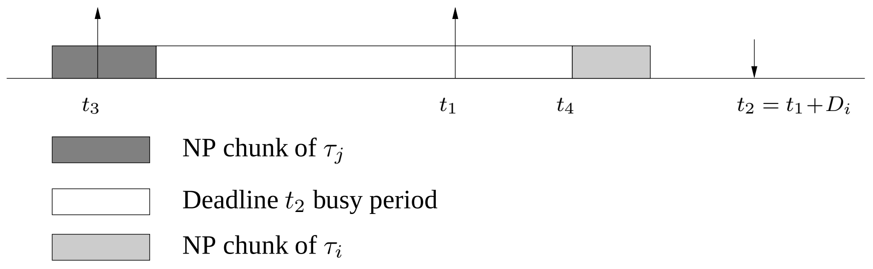


Figure 3.3: Worst-case response time scenario for Limited preemption EDF.

Theorem 2. *The worst-case response time of a task τ_i is found in a deadline busy period for τ_i in which:*

- *τ_i has a job released at time a (and possibly other jobs released before);*
- *all tasks with relative deadline smaller than or equal to $(a+D_i)$ are released synchronously at time $t = 0$ and at their maximum rate;*
- *a further task τ_j with relative deadline greater than $a + D_i$, if any, starts executing a non-preemptive region of length Q_j, an arbitrarily small amount of time before $t = 0$.*

Proof. We adapt the proof contained in [GRS96] to the case under consideration. Consider the scenario in Figure 3.3 where τ_i has a job with arrival time t_1 and absolute deadline $t_2 = t_1 + D_i$. Let t_4 be the start time of the NP chunk of τ_i, according to the rule defined in section 3.3.1. The actual execution time of the NP chunk is $\leq Q_i$. Finally let t_3 be the last time before or at t_1 such that there are no pending jobs with absolute deadlines before or at t_2.

By the choices made, t_3 must coincide with the release time of at least one job, and there cannot be idle time between t_3 and t_4. This means that the execution of τ_i's NP chunk of the job arrived at time t_1 is preceded by a busy period of those instances released between t_3 and t_4, and that have absolute deadlines before or at t_2, plus at most one other NP chunk released before t_3 with absolute deadline after t_2.

Consider now the scenario in which:

- all tasks but τ_i with relative deadline less than or equal to $(t_2 - t_3)$ are released from time $t = 0$ at their maximum rate;
- being $a = (t_1 – t_3)$, τ_i is released at time $\left(a - \left\lfloor \frac{a}{T_i} \right\rfloor T_i\right), \left(a - (\left\lfloor \frac{a}{T_i} \right\rfloor - 1)T_i\right), \ldots, a;$
- the task τ_j, if any, that attains the maximum values of $\max_{D_j > (t_2 - t_3)}\{Q_j\}$ is released an arbitrarily small amount of time before $t = 0$. That is, τ_j causes the worst possible priority inversion w.r.t. the absolute deadline $a + D_i$.

In the new scenario, the workload in the interval preceding the start time of the NP chunk of the considered instance of τ_i, released at time $a = t_1 - t_3$, cannot be less than in the previous scenario: the busy period preceding this NP chunk cannot be shorter, since it includes the worst-case priority inversion w.r.t the absolute deadline $a + D_i$, as well as the largest deadline-$(a + D_i)$-busy period preceding it. Therefore, τ_i's response time cannot diminish. □

Using Theorem 2, it is possible to adapt the algorithm described in Section 3.4.1 for the computation of the worst-case response time of a task τ_i to the limited preemption case. We will prove that we will only need to replace Equation (3.4) with

$$R_i = \max_{a \in A_i} \{L_i(a) - a + Q_i\}. \tag{3.5}$$

and Equation (3.3) with

$$\begin{aligned} L_i(a) = \max_{D_j > a + D_i} \{Q_j\} + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i - Q_i + \\ \sum_{\substack{j \neq i \\ D_j \leq a + D_i}} \min \left\{1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} C_j. \end{aligned} \tag{3.6}$$

Proof. Differently from the worst-case response time analysis for preemptive EDF, where the considered busy period ends at the finishing time of the reference job of τ_i, in LP-EDF we focus on the busy period *preceding* the execution of the final non-preemptive region of the reference job (time t_4 in Figure 3.3). Hence all jobs released before this time should be taken into account.

Consider the scenario described in the proof of Theorem 2. Let $L_i(a)$ be the length of the busy period starting at time $t = 0$ and preceding the start time of the last non-preemptive region of a job of τ_i released at time a. Let $\tau_{i,k}$ be such job. The response time of $\tau_{i,k}$ is given by $L_i(a) - a + Q_i$. Taking the maximum over all jobs released by τ_i, Equation (3.5) follows.

Now, to prove that the length $L_i(a)$ can be determined by finding the smallest solution of Equation (3.6), note that the first term on the RHS accounts for the maximum possible priority inversion w.r.t. the absolute deadline $a + D_i$. The second and third terms correspond to the time needed to execute the jobs of τ_i released before or at time a, excluding the final NP chunk of $\tau_{i,k}$. Finally, the last term represents the time needed to execute the jobs of tasks $\tau_{j \neq i}$, with absolute deadlines $\leq (a + D_i)$, that are released before the beginning of the last NP region of $\tau_{i,k}$. Note that this term is slightly different from the corresponding term used in Equation (3.3), to make sure that the last NP region of $\tau_{i,k}$ has already started[4]. □

[4]The analysis can be tightened adopting techniques used in [BLV07] for Fixed Priority scheduling. In particular, the term $\left(1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor\right)$ can be replaced by the tighter term $\left(\left\lceil \frac{L_i(a)}{T_j} \right\rceil\right)$, whenever there is a positive blocking term (i.e., if $\max_{D_j > a + D_i}\{Q_j\} > 0$). See [BLV07] for further details.

Observing Eq. (3.6), it is clear that the length of the busy period used to compute the worst-case response time of τ_i is not affected by NP regions of any task τ_j having $D_j \leq D_i$. In other words, the responsiveness of a task does not change if NP regions are introduced inside the code of one or more tasks having shorter deadlines.

By taking advantage of this observation, we will analyze in Section 3.5 the performances of the controller task having the largest relative deadline among tasks that have non-negligible NP regions.

3.5 Experimental Results

3.5.1 Experiment Setup

We consider a system with $n = 7$ hard real-time tasks. We will monitor the timing attributes listed in Section 3.3.2 for task τ_1, having period $T_1 = 50ms$ and execution time $C_1 = 5ms$ (and, therefore, utilization $U_1 = 0.1$). The other 6 tasks are generated using UUNIFAST algorithm [BB04], with period T_i uniformly distributed in $[10,\ 100]\ ms$ and utilization U_i chosen according to a 6-dimensional uniform distribution to reach the desired total utilization. No particular task ordering is assumed. For all tasks, including τ_1, D_i equals T_i.

The system utilization varies from 0.2 to 1 with steps of 0.1. For each utilization, 500 task sets are randomly generated. For each task set τ, three scheduling policies are tested:

- Fully preemptive Earliest Deadline First policy, denoted as EDF;
- Limited-preemption EDF, placing the largest possible non-preemptive regions (whose lengths Q_i are computed using the algorithm described in [Bar05]) at the end of the execution of each task; this policy will be denoted as LP-EDF;
- Limited-preemption EDF, placing the largest possible non-preemptive regions at the end of the execution of those tasks having a relative deadline $\leq D_1$, and scheduling the remaining ones with preemptive EDF; this policy will be denoted as LP-EDF*;

A schedule is generated for each one of the above policies, for a simulation length of 40 seconds, running in TrueTime [CHL$^+$03].

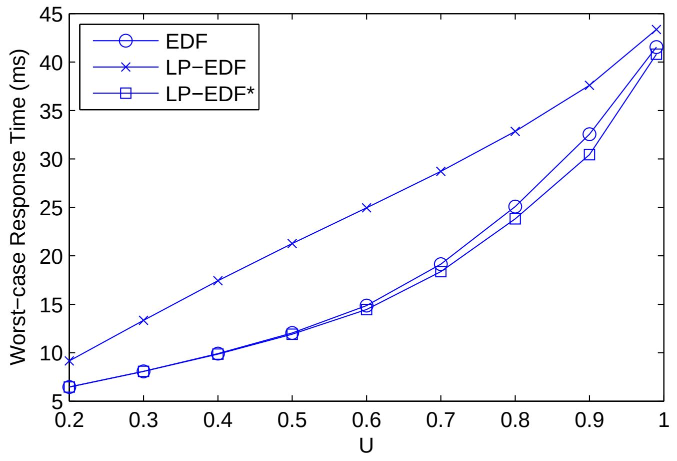
We will measure the average values defined in Section 3.3.2, approximating Equation (3.2) with the the following expression:

$$\bar{X}_i = \frac{\sum_{k=1}^{m_i} X_{i,k}}{m_i}, \tag{3.7}$$

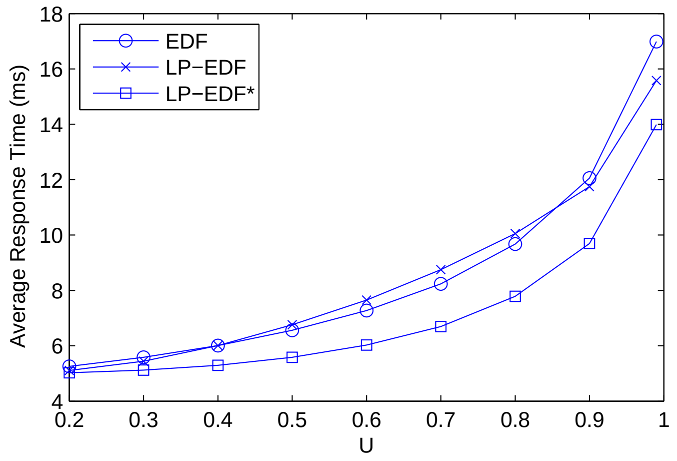
where m_i is the total number of jobs of τ_i generated during the $40s$ of simulation. Since the simulation time is very large, Equation (3.7) approximates well Equation (3.2).

3.5.2 Responsiveness Results

In Figure 3.4, we show the worst-case and average response times of task τ_1. The worst-case response time is computed with the method described in Section 3.4, while the average response time is derived as described above.



(a) Worst-case Response Time of τ_1



(b) Average Response Time of τ_1

Figure 3.4: Response Time of τ_1.

The results show that the smallest worst-case response times are obtained with LP-EDF*, thanks to the NP region of task τ_1. The largest worst-case response times are instead obtained with LP-EDF, due to the blocking imposed by the non-preemptive regions of lower-priority tasks. The lower subgraph shows that, with limited preemption scheduling, the system can achieve better average response time. More specifically, LP-EDF* gives always the best (shortest) average response time, which are from 20% to 30% smaller than the average response times of preemptive EDF. Under LP-EDF, instead, the average response time is comparable to the EDF case, being slightly shorter only when the system utilization is below 0.4, or above 0.85.

To highlight the improvement over preemptive EDF, we found it useful to introduce the **Relative Response Time Improvement** $\hat{R}_i^A$, defined as the difference between the average response time under preemptive EDF and under a particular scheduling policy A, divided by the task period, i.e.,

$$\hat{R}_i^A = \frac{\bar{R}_i^{\text{EDF}} - \bar{R}_i^A}{T_i}$$

where $\bar{R}_i^A$ is the average response time of τ_i under a policy A. Notice that $\hat{R}_i^A > 0$ means that the response time of τ_i is reduced (improved) under the policy A w.r.t. preemptive EDF.

The **Overall Relative Response Time Improvement** $\hat{R}$ is defined as the mean of $\hat{R}_i$ among all tasks τ_i:

$$\hat{R} = \frac{\sum_{\tau_i \in \tau} \hat{R}_i}{n}.$$

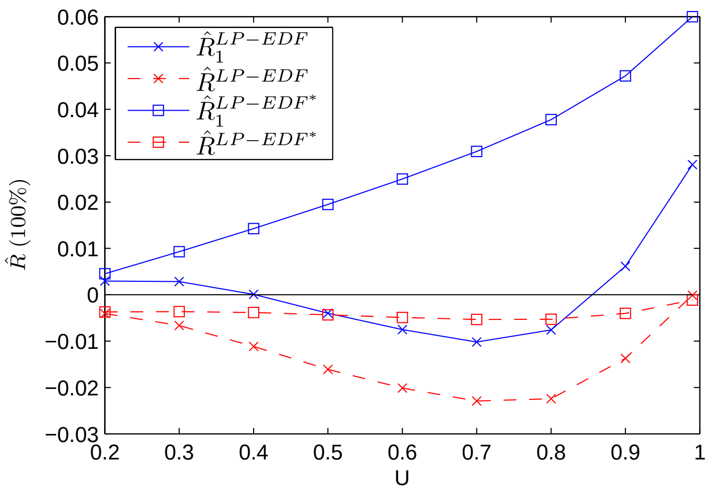
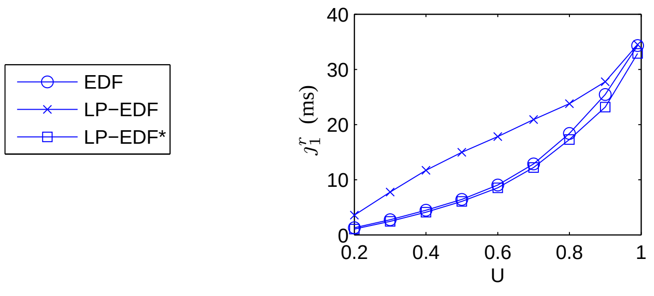


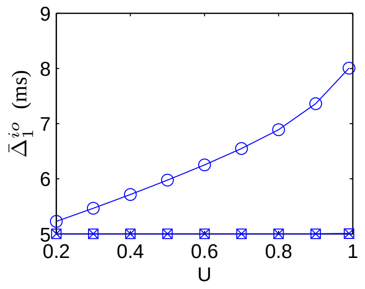
Figure 3.5: Relative Response Time Improvement ($\boldsymbol{U_1 = 0.1}$).

In Figure 3.5, the two solid curves for LP-EDF and LP-EDF* show that the average response time is improved using LP-EDF* and LP-EDF in highly loaded system, w.r.t the preemptive EDF case. Moreover, under LP-EDF*, τ_1 achieves *always* a much lower average response time, and $\hat{R}_1^{\text{LP-EDF}^*} > 0$ for all tested utilizations.

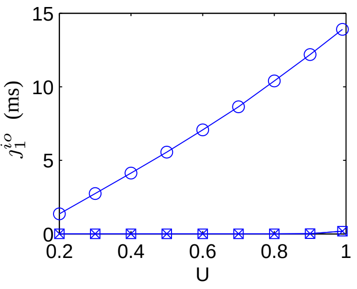
The dashed curves, however, show that the overall relative response time improvement $\hat{R}$ of the whole task set is negative, both for LP-EDF* and LP-EDF. This can be explained by the response-time "redistribution" that takes places when introducing NP regions: the average response times of shorter-period tasks are increased due to the extra blocking time from lower priority tasks; instead, the average response time of longer-period tasks are reduced, thanks to the non-preemptive execution of their NP regions. However, because of the smaller periods, the increase in the relative response time of shorter-period tasks is more significant than the reduction for longer-period tasks. Nonetheless, the negative effect is relatively small in LP-EDF*, since there are less priority inversions for shorter-period tasks.



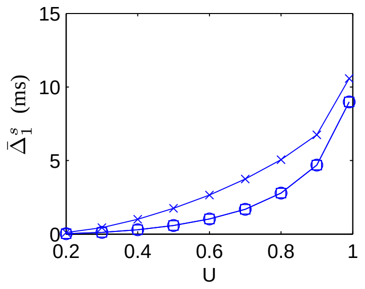
(a) Response jitter



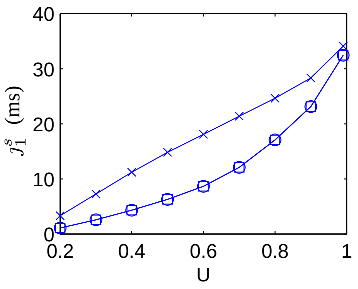
(b) Average IO Delay



(c) IO Jitter



(d) Average Start Delay



(e) Start jitter

Figure 3.6: Timing Attributes of τ_1 ($U_1 = 0.1$).

Figure 3.6 shows the timing attributes of τ_1. As depicted, both LP-EDF and LP-EDF* perform well in minimizing IO delay Δ_1^{io} and IO jitter j_1^{io}. It can be easily proved that the IO delay with limited preemptions is *always* smaller than with preemptive EDF, due to the smaller interference suffered by a task in its last chunk. Moreover, since once a task starts executing, it cannot be preempted by tasks with larger relative deadlines, the IO delay of τ_1 with LP-EDF is always identical to the IO delay with LP-EDF*, as shown in the figure. Both algorithms have IO delay Δ_1^{io} very close to $C_1 = 5ms$ (and IO jitter $\jmath_1^{io}$ close to zero), meaning that most of τ_1's jobs will be able to execute entirely non-preemptively.

Regarding the start delay Δ_1^s and the start jitter $\jmath_1^s$, LP-EDF* has the same performances as preemptive EDF. This can be explained by noting that no job $\tau_{1,k}$ of task τ_1 can be blocked by the NP regions of tasks having smaller relative deadlines, since either they have an absolute deadline earlier than $d_{1,k}$, or they arrive after $a_{1,k}$. Instead, both the start delay Δ_1^s and the start jitter $\jmath_1^s$ increase with LP-EDF, due to the blocking of lower priority tasks.

Finally, the response jitter $\jmath_1^r$ is influenced by both $\jmath_1^s$ and $\jmath_1^{io}$, being $R_{1,k} = \Delta_{1,k}^s + \Delta_{1,k}^{io}$. The smallest response jitter is obtained with LP-EDF*, while the larger response jitter of LP-EDF is due to its large start jitter.

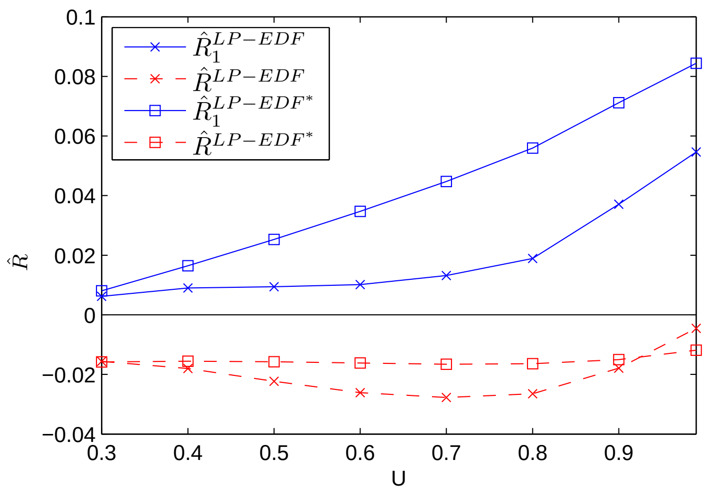
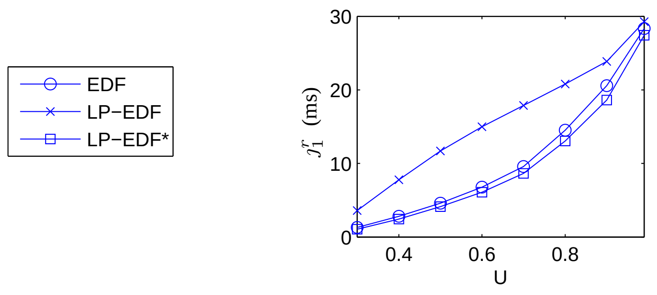
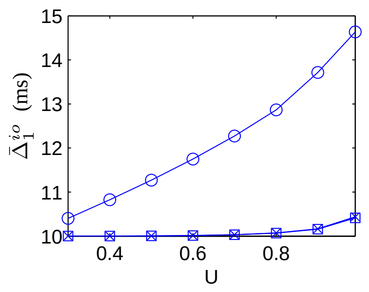


Figure 3.7: Relative Response Time Improvement ($\boldsymbol{U_1 = 0.2}$).

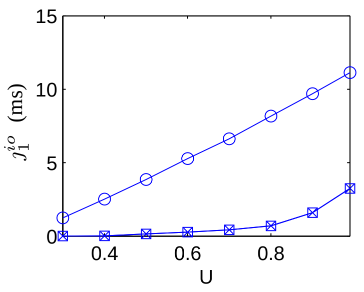
Similar results are obtained changing τ_1's execution time. Due to space reasons, we include here only the case with $C_1 = 10ms$ (and $U_1 = 0.2$). The results for $\hat{R}$ and the other timing attributes are shown, respectively, in Figure 3.7 and 3.8. Note that increasing τ_1's utilization, there is a more significant improvement (over preemptive EDF) in the average response time of τ_1 for both LP-EDF and LP-EDF*, as testified by the positive values of $\hat{R}_1^{\text{LP-EDF}}$ and $\hat{R}_1^{\text{LP-EDF}^*}$. Jitters and delays are similar to the previous case. Note that τ_1's IO delay and jitter are not always constant, but they increase for heavy loads, meaning that τ_1 is not always able to execute non-preemptively. Nevertheless, the values of Δ_1^{io} and $\jmath_1^{io}$ for LP-EDF and LP-EDF* are still significantly smaller than with EDF.



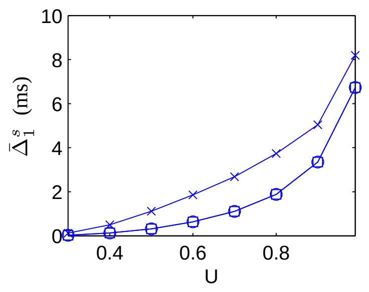
(a) Response jitter



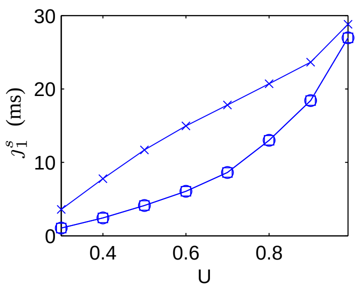
(b) Average IO Delay



(c) IO Jitter



(d) Average Start Delay



(e) Start jitter

Figure 3.8: Timing Attributes of $\tau_1(U_1 = 0.2)$.

3.5.3 Control Performance Results

When a controller is implemented as a hard real-time task running in a multi-threaded environment, the scheduling-induced delay and jitter affect assumptions like the constant sampling period and the null, or constant, input-output delay, degrading control performances [Mar02, BC07]. In general, a control task achieves better performance if it experiences smaller delay and jitter at runtime. To show how limited preemption scheduling can be exploited to increase the responsiveness and, accordingly, the performances of a controller tasks, we considered the following benchmark control system, inspired by the example shown in [BC07].

An inverted pendulum with natural frequency of $6\ rad/s$ is controlled by a

Linear Quadratic Gaussian (LQG) controller [ÅW97]. The state-space model of the inverted pendulum is:

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 36 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v$$
$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x + e$$

where v is a continuous-time Gaussian white-noise process with zero mean and variance 1, and e is a discrete-time Gaussian white-noise process with zero mean and variance 0.1.

A quadratic cost function J is provided to design the LQG controller as well as to evaluate the control performance.

$$J = E \lim_{t_p \to \infty} \frac{1}{t_p} \int_0^{t_p} \left(x^T \begin{bmatrix} 0 & 0 \\ 0 & 10 \end{bmatrix} x + u^2 \right) dt$$

where $[0, t_p]$ is the time span to be considered. Although from a theoretical point of view t_p should be ∞, in practice we could use a large enough value to evaluate the control performance. In our experiment, t_p is set to the simulation time, i.e., 40 seconds. Notice that the cost function is defined so as to minimize the state error and control energy. Therefore, a larger cost implies a worse control performance (see [BC07]).

The simulation setup is the same as in Section 3.5.1 with task τ_1 being the controller task. Hence, the sampling period of the controller is 50ms. Assuming sampling (input) and control signal actuation (output) happen, respectively, at the start time and at the finishing time of each job of τ_1, the lateness of the controller is equal the lateness of τ_1.

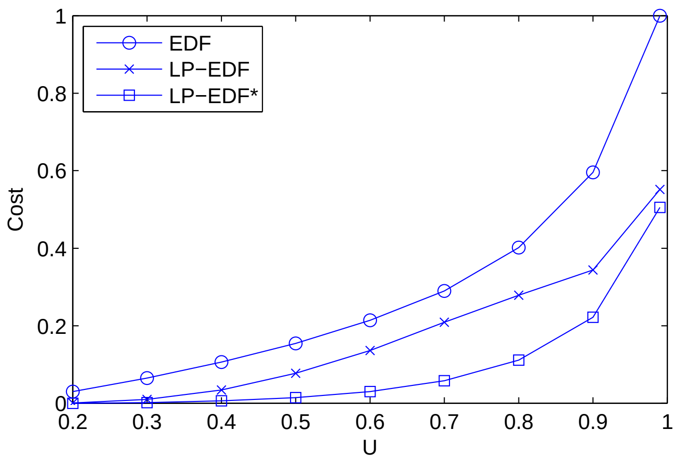


Figure 3.9: Control Performance of $\boldsymbol{\tau_1}$.

From Figure 3.9, we notice that by employing limited-preemption scheduling, the control performance improves (the cost is reduced) w.r.t. standard EDF, owing to the reduction of IO delay and jitter. The performance improvement of LP-EDF, however, is smaller than with LP-EDF* because of the negative effect on sampling delay and jitter. Nevertheless, the results demonstrate that the enhanced responsiveness obtained with limited preemption scheduling helps achieving better performances in real-time control systems.

3.6 Conclusion

The application of limited preemption EDF scheduling is proposed to improve the responsiveness of selected tasks in a uniprocessor real-time system. In particular, it is suggested to execute non-preemptively the last chunk of code of each control task, in order to improve the control performances. For each such task, an algorithm to compute the worst-case response time is provided, extending a previously proposed method for non-preemptive systems. The proposed policy is evaluated on a randomly generated task distribution, measuring average timing parameter that determine the performance of a control system. The simulations, together with an example case-study, showed the effectiveness of the proposed approach.

Chapter 4

Parameter Selection in an Integrated Framework

4.1 Introduction

As mentioned in Chapter 1, the typical approach adopted during the design of a control system is to separate performance requirements from architecture and implementation issues. In a first stage, the control law is designed assuming an ideal behavior of the computing system on which the controller executes, where tasks run smoothly on the processor without considering any kind of interference. This is equivalent of synthesizing a controller in the continuous time domain without delay. When computational resources are taken into account in the design, the limited processing power of the system is considered by assigning a fixed sampling rate to the controller, whereas other types of interference are cumulated by considering a fixed input-output delay in the control loop. In this case, a controller can either be discretized or directly designed in the discrete time domain using sampled-data control theory.

In a second stage, once performance requirements are ensured by the control laws, control loops are mapped into periodic tasks and schedulability analysis is performed to verify whether the timing constraints assumed by the control designer can be met. If so, the system is implemented, otherwise the control laws must be designed by assuming different sampling rates and/or delays, and the process must be repeated.

Even when timing constraints are verified through feasibility analysis (using predicted values), the actual system implementation may reveal overload conditions and longer delays that force further refinement steps in the design process, unless very pessimistic assumptions are considered on the system [BMV07]. Figure 4.1 illustrates the typical refinement process of the classical design methodology.

Such a separation of concerns facilitates both control design and implementation, allowing the system to be developed by teams with different expertise. In fact,

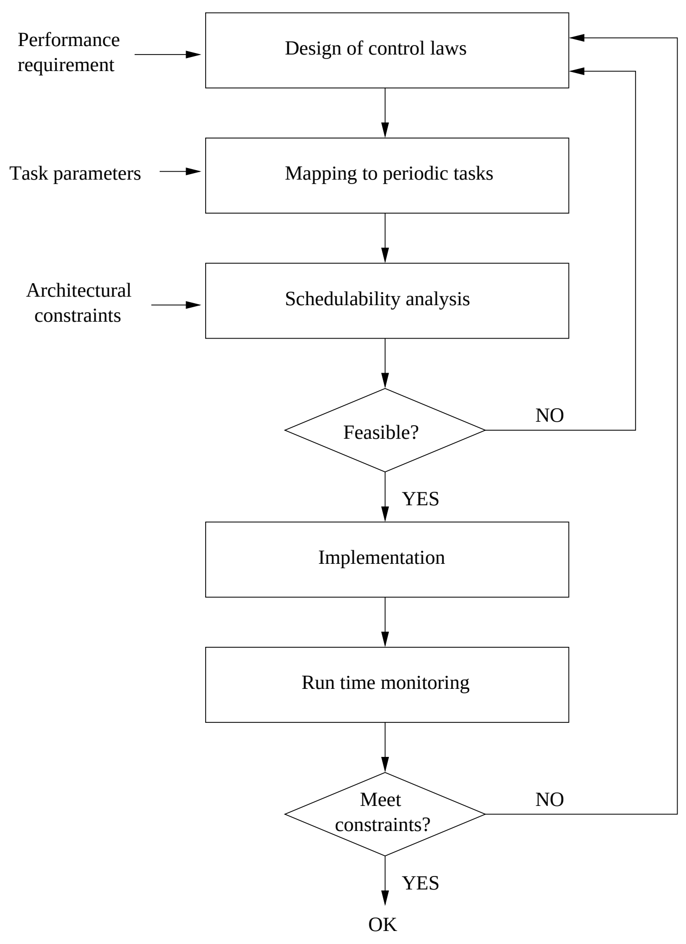


Figure 4.1: Typical design cycle of a real-time control system.

control experts can focus on system-level goals, such as stability, robustness, and control performance, whereas computer engineers can concentrate on task mapping, schedulability analysis, resource management and code generation to ensure a reliable support to the application [ÅCES00].

Unfortunately, however, such a repetitive design methodology has the following disadvantages:

- Long and expensive development. Since design is performed following a trial and error strategy, several refinement steps can be required to find a suitable solution, especially when computational resources are scarce and the application consists of several concurrent and interacting activities.

- Suboptimal performance. The myopic search in the space of solutions does not guarantee that the found solution leads to the best performance. A different setting of parameters could guarantee feasibility with a significant increase in the performance.

- Suboptimal use of the resources. Since resource constraints are not taken into account in the design process (except for verifying feasibility), a feasible solution does not guarantee optimal resource exploitation, which would be of crucial importance in embedded systems where resources are scarce. For instance, optimal resource usage would allow to minimize energy consumption while meeting performance requirements.

The major problem in such a design practice is that the assumptions made at the first stage of control design are difficult to meet in the implementation, unless delays are assumed equal to sampling periods [HHK01]. However, it has been shown [BC07] that, in most cases, a shorter and varying delay leads to a better performance than a fixed but longer delay. Sampled-data control theory usually assumes a negligible or at least constant input-output delay, whereas in resource constrained implementations (as the case of embedded systems and networked control systems) many concurrent tasks competing for computational resources may cause transient or permanent overload conditions, as well as introduce variable input-output latencies in control loops. Such non-deterministic effects can significantly degrade the overall system performance and possibly lead to the violation of some properties achieved during the control design phase, including system stability.

As a result, a trade-off between control performance and resources usage should be wisely considered during the whole design process. In particular, architecture constraints (as processing power, memory size, maximum power consumption) and operating system effects (as runtime overhead, blocking time, response time, intertask interference) should be properly modelled to possibly optimize the design towards a precise control objective.

In this chapter, an integrated approach to enhance the control performance of a system through proper selection of task periods and deadlines, under EDF scheduling. A general framework is proposed to extend Seto's method to optimize performance with respect to not only sampling periods but also other timing attributes. In particular, task deadlines are chosen to balance the scheduling-induced performance loss of each controller task exploiting the feasibility region in the space of EDF deadlines [BB09b]. Detailed simulations are also provided to demonstrate the usage of the proposed methodology and verify its effectiveness over other methods.

4.2 Related Work

To distribute the limited computing resources to different controller tasks, Seto et al. [SLSS96] proposed to formulate the real-time control co-design problem as an optimization problem, where the control performance index, expressed as a function of the sampling period, is constrained by the feasibility condition of the task set. By solving the optimization problem, the sampling period for each controller is computed to maximize the overall system performance. This methodology, further extended by many researches, is referred to as the *period selection problem*. Bini and Di Natale [BDN05] applied Seto's methodology to a set of controller tasks scheduled by Fixed Priorities.

To cope with the problem of delay and jitter in real-time control applications, different techniques have been developed. Nilsson [NBW98] analyzed the performance and stability of real-time control systems with varying delays, and derived an optimal stochastic controller to compensate for jitter. Cervin et al. [CLE$^+$04] introduced the concept of *jitter margin*, defined as the upper bound of the input-output jitter of a control task that guarantees the stability of the controlled system. Martí et al. [MFFR01] presented a method to online compensate the control performance degradation caused by jitter. Another approach for reducing delay and jitter is to use non-preemptive or limited-preemptive scheduling policies [BB09a, YBB09]. For example, Chapter 3 discussed the benefits of using EDF with limited preemptions to reduce input-output delay and jitter without impairing the schedulability of the task set.

Another widely adopted method to reduce delay and jitter is to limit the execution interval of each task by setting a proper relative deadline. Like *period selection*, this method can be referred to as *deadline selection*. Different algorithms for computing the minimum deadline have been proposed in the literature. Some methods [HBJK06, BRC06] allow to minimize the relative deadline of a single task at a time, following a given order. In this way, however, the first task in the sequence experiences the most significant deadline reduction, leaving little slack for the remaining tasks. A more uniform deadline reduction can be achieved by scaling all deadlines by the same factor [BRC06], but the improvement achieved in terms of delay and jitter is not significant and, in some cases, the schedule could even remain unchanged. Other methods [BBGL99, HB07] use binary search to reduce task relative deadlines as much as possible according to given reduction factors, while keeping the task set schedulable. These methods, however, are mainly focused on schedulability aspects and barely considered control issues; moreover, it is not clear how reduction factors can be assigned to tasks.

Different delay/jitter reduction methods have been discussed and compared in [BC07], where it is shown that the effectiveness of a particular method depends of the characteristic of the controlled system, although the deadline reduction approach is the simplest and most effective for most control systems.

Ryu and Hong [RH98] used a heuristic method to select periods and deadlines with respect to performance specification and schedulability constraints. The control performance was specified in terms of steady state error, overshoot, settling time, and rise time, which were expressed as functions of the sampling period and input-output latency. At each step of the heuristic method, the periods and deadlines were derived using the Period Calibration Method solving a nonlinear optimization problem. The optimization goal, however, was to minimize the utilization of the task set.

Kim [Kim98] suggested to express the control cost as a function of both periods and delays, where periods were found assuming that the delays were given. Then, the new delays were computed by simulating the schedule of all the tasks up to the hyperperiod, and iteratively the periods were updated assuming the new delay values. However, this method considered only fixed priorities and was extremely time consuming.

Palopoli et al. [PAC$^+$00] proposed to use resource reservation to serve control tasks as soft real-time threads. It was revealed that control tasks may tolerate a certain amount of deadline misses owing to their inherent robustness, therefore relaxing the hard timing constraints allows higher activation rates, which may lead to improved performance. However, no optimization was performed to select reservation parameters and only experimental results were presented.

Chantem et al. [CWLH08] proposed a heuristic search algorithm to find feasible period-deadline pairs, based on the assumption that task deadlines are piecewise first-order differentiable functions of their respective periods. However, this work mainly focused on schedulability issues.

Bini and Cervin [BC08] approximated the delays as a function of task periods and incorporated the delay consideration into the performance optimization, while the resource constraint remains to be the feasibility region with respect to task periods. This method only applies to fixed priority systems, because in dynamic priority systems delays are functions of both periods and deadlines.

4.3 System model

This work considers a set τ of n periodic real-time tasks that are executed on a uniprocessor system under the Earliest Deadline First (EDF) scheduling policy. The task set τ is logically divided into 2 subsets: one subset τ_{ctrl}, consisting of n_{ctrl} controller tasks that are each implemented using the naif task model described in Section 2.2.1, and another subset τ_{nctrl}, consisting of n_{nctrl} regular tasks that are not related to control. Each task τ_i is characterized by the following *scheduling parameters*:

C_i the worse-case execution time (WCET);

C_i^b the best-case execution time (BCET);

D_i^{min} the minimum allowed relative deadline;

D_i^{max} the maximum allowed relative deadline;

T_i^{min} the minimum allowed period;

T_i^{max} the maximum allowed period;

D_i the actual relative deadline, whose value has to be selected within the range $[D_i^{min}, D_i^{max}]$;

T_i the actual period, whose value has to be selected within the range $[T_i^{min}, T_i^{max}]$. For control tasks, T_i is set equal to the sampling period.

It is assumed that C_i, C_i^b, D_i^{min}, D_i^{max}, T_i^{min} and T_i^{max} are known, whereas T_i and D_i are the *design parameters* to be selected. Notice that, to derive more general results, relative deadlines are allowed to be less than, equal to, or greater than periods. In addition, the utilization U_i of each task can range within $[U_i^{min}, U_i^{max}]$, where $U_i^{min} = \frac{C_i^b}{T_i^{max}}$, and $U_i^{max} = \frac{C_i}{T_i^{min}}$. Similarly, U, U_{ctrl}, U_{nctrl} denote the total utilization of the whole task set ($U = \sum_{i=1}^{n} U_i$), the utilization of all the controller tasks ($U_{ctrl} = \sum_{\tau_i \in \tau_{ctrl}} U_i$), and the utilization of all the regular tasks ($U_{nctrl} = \sum_{\tau_i \in \tau_{nctrl}} U_i$), respectively.

4.4 The General Framework

4.4.1 Integrated Design approach

To avoid the repetitive design process, we propose to use a general framework that extends Seto's method [SLSS96] to achieve optimal performance and optimal resource usage. The extended framework considers not only the sampling periods but also other timing attributes. Figure 4.2 illustrates the basic idea of the proposed design methodology, whereas Figure 4.3 depicts a typical performance function in the space of the design parameters. The shadowed area denotes the feasible region where task parameters satisfy the required timing constrains. Notice that the optimal control performance must take such constraints into account and can only be achieved by wisely selecting the task parameters setting.

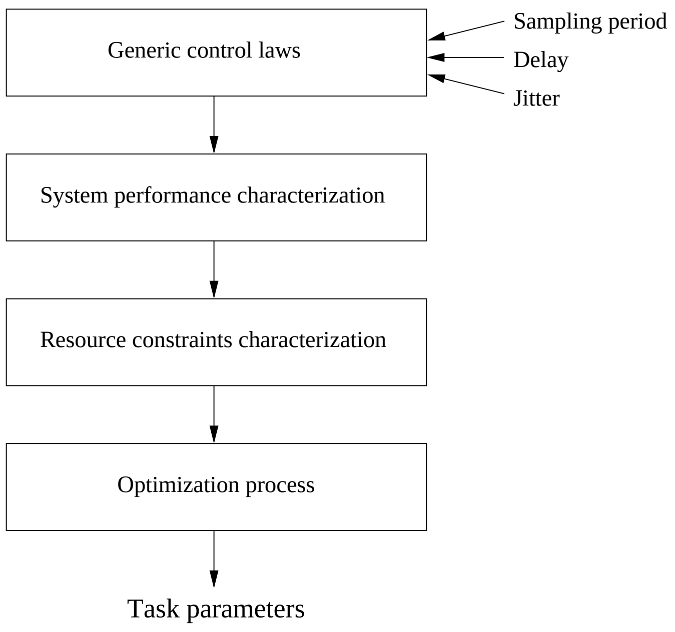


Figure 4.2: Proposed design methodology.

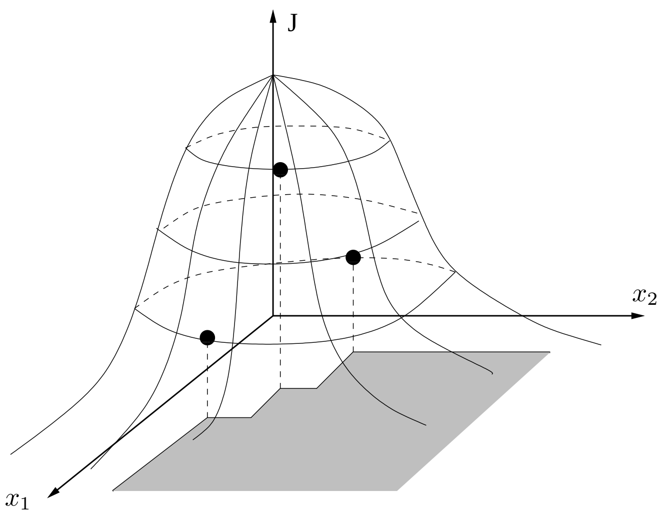


Figure 4.3: Relation between control performance and task parameters

4.4.2 The performance loss index

The primary goal of a control system is to meet stability and performance requirements, such as transient response and steady-state accuracy [BMV07]. Beyond such requirements, controller design attempts to minimize the system error, defined as the difference between the desired response and the actual response. The smaller the difference, the better the performance. Hence, performance criteria are mainly based on measures of the system error. Traditional criteria (reported in control text-books, e.g. [CDHB04]), such as IAE (Integral of the Absolute Error), ITAE (Integral of Time-weighted Absolute Error), ISE (Integral of Square Error) or ITSE (Integral of Time-weighted Square Error), provide quantitative measures of a control system response and are used to evaluate (and design) controllers.

More sophisticated performance criteria, mainly used in optimal control problems, account both for the system error and for the energy that is spent to accomplish the control objective. The higher the energy demanded by the controller, the higher the penalty paid in the performance criterion. The system error and control energy can be multiplied by a weight to balance their relative importance.

The performance index used in this work is the same as the one used in Linear Quadratic Gaussian (LQG) controller design (e.g. [ÅW97]). The performance of a control task is given by a quadratic cost function

$$J = \mathrm{E} \lim_{t_p \to \infty} \frac{1}{t_p} \int_0^{t_p} \left(x^T Q_1 x + u^T Q_2 u \right) dt, \tag{4.1}$$

where x is the state vector, u is the control signal vector, $[0, t_p]$ is the time span to be considered, and Q_1, Q_2 are weighting matrices. The performance J can be interpreted as the weighted sum of state errors and control energy. Higher values of J indicate larger deviation from the desired states or larger energy spent for control, which means worse control performance. For this reason, in the remainder of the paper, J is referred to as the *performance loss index*.

Previous work on period selection has considered the performance loss index as a function of the sampling period T,

$$J = J(T).$$

In most realistic cases, for a reasonable range of sampling intervals, the performance loss (4.1) is an increasing function of the sampling period. Cervin et al. [CEBÅ02] argued that the performance loss index can often be approximated by a linear function of the sampling period,

$$J \approx \alpha + \beta T,$$

or by a quadratic function of the sampling period,

$$J \approx \alpha + \beta T + \gamma T^2.$$

Delay and jitter in the control task execution can have a large impact on the control performance, especially if the sampling frequency is too low compared to the speed of the closed-loop system. It would hence be desirable to include the delay and jitter in the performance loss index. The relationship between these timing attributes and the resulting control performance is however very complex. The solution proposed in this work is to include the relative deadline D in the cost function:

$$J = J(T, D). \tag{4.2}$$

As will be shown in Section 4.5, the relative deadline D upper limits the amount of delay and jitter the controller can experience. Knowing T and D, it is hence possible to predict the worst-case performance degradation introduced by the scheduling. In general $J(T, D)$ is a nonlinear function. It is realistic to assume that it is an increasing function in both T and D, since the control performance typically degrades as the sampling period, delay, or jitter increases, as later shown in Figure 4.14 of Section 4.7.1.

4.4.3 The optimization problem

The *period selection* problem has received considerable attention in the real-time literature. It can be expressed as an optimization problem to find the best periods for the controller tasks that minimize the performance loss while guaranteeing the system schedulability. Such an optimization problem under EDF can be expressed as follows:

$$\begin{aligned}
&\min_{\{T_i\}} J = \sum_{\tau_i \in \tau_{ctrl}} J_i(T_i) \\
&s.t. \quad \sum_{\tau_i \in \tau_{ctrl}} \frac{C_i}{T_i} + \sum_{\tau_i \in \tau_{nctrl}} \frac{C_i}{T_i} \le 1
\end{aligned}$$

where the objective function is the sum of all the controller tasks' performance in dices, which are assumed to be function of the sampling period. For the constraints, the first equation relates the sampling periods with the task periods, while the second equation imposes the schedulability constraint for the given scheduling policy (EDF).

To take the impact of delay and jitter on control performance into account, the relative deadlines are included in the performance loss indexes and the optimization problem is generalized to

$$\begin{aligned} \min_{\{T_i,D_i\}} \; & J = \mathcal{F}_{\tau_i \in \tau_{ctrl}}(J_i(T_i, D_i)) \\ s.t. \quad & \{T_i, D_i\} \in \mathcal{S},\ \forall \tau_i \in \tau \end{aligned} \tag{4.3}$$

where $\mathcal{F} : \mathbb{R}^{n_{ctrl}} \rightarrow \mathbb{R}$ is a system-wide function used to combine the individual performance indices of control tasks into a global system performance index, and $\mathcal{S}$ is the set of resource constraints, i.e. schedulability conditions, imposed by the scheduling platform. The choice of function $\mathcal{F}$ depends on the user's interest and can be, for instance, a linear combination of all the individual performance loss indices, or the maximum among the performance loss indices.

4.5 Linking Task Parameters to Control Performance

This section explains how to derive the performance loss index given in Eq. (4.2) in a simulative or experimental fashion, describes the relation between control performance and scheduling parameters, and formalizes the optimization problem expressed by Eq. (4.3).

4.5.1 Characterization of the delay and jitter

Assuming that the task set is schedulable, each job will finish no later than its absolute deadline. This puts a limit on the amount of delay and jitter that a control task with period T_i and relative deadline D_i can experience.

Consider the worst-case scenario depicted in Figure 4.4. In this scenario, task τ_i releases 3 consecutive jobs, where job $\tau_{i,0}$ finishes with best-case execution time C_i^b, job $\tau_{i,1}$ starts at its release time and finishes at its deadline, and finally, job $\tau_{i,2}$ starts $D_i - C_i$ before its deadline to ensure that it will not cause an overrun. By analyzing the worse-case scenario, the following bounds on the delays can be derived:

$$\begin{aligned} \max \Delta_{i,k}^{io} &= \Delta_{i,1}^{io} \leq D_i \\ \min \Delta_{i,k}^{io} &= \Delta_{i,0}^{io} \geq C_i^b \\ \max \Delta_{i,k}^{s} &= \Delta_{i,2}^{s} \leq D_i - C_i \\ \min \Delta_{i,k}^{s} &= \Delta_{i,1}^{s} \geq 0 \end{aligned} \tag{4.4}$$

Also, the following relations on the jitter hold:

$$\begin{aligned} J_i^{io} &\le D_i - C_i^b \\ J_i^s &\le D_i - C_i \end{aligned} \tag{4.5}$$

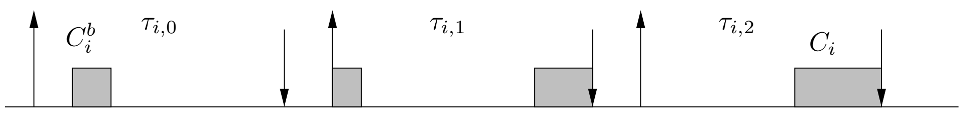


Figure 4.4: Worst-case scenario for delay and jitter.

Notice that the reported worst-case scenario must not necessarily take place in an actual schedule, since the interference on task τ_i depends on the scheduling parameters of other tasks as well. Therefore, the relations derived above represent only the lower or upper bounds of the actual delay and jitter.

The analysis above shows that a shorter relative deadline implies both shorter delays and less jitter, which should imply better control performance. How to find the actual performance loss index is treated next.

4.5.2 Performance loss index derivation

In most cases, it is impossible to evaluate the exact value of the performance loss index (4.1) for a controller executing in a real-time system. An execution of the real-time system will generate an infinite sequence of sampling and input-output delays, $\{\Delta^s_{i,0}, \Delta^{io}_{i,0}, \Delta^s_{i,1}, \Delta^{io}_{i,1}, \Delta^s_{i,2}, \Delta^{io}_{i,2}, \ldots\}$, for each control task τ_i. The delays are in general random and depend on the execution-time characteristics of the control algorithm and the preemption pattern created by the scheduling algorithm, which in turn depend on the execution of the other tasks in the system.

Using the bounds on the delay and jitter derived in the previous subsection, various approaches can be used to evaluate the performance loss index approximately:

- Taking a stochastic approach, one can assume that $\{\Delta^s_{i,j}, \Delta^{io}_{i,j}\}_{j=0}^{\infty}$ describe a sequence of independent two-dimensional uniform random variables with bounds given by (4.4). The performance index can then be evaluated numerically using a tool such as Jitterbug [LC02]. A limitation of Jitterbug, however, is that the maximum delay variation allowed is bounded by the sampling period. Hence, some cases where $D_i > T_i$ are not possible to evaluate.

- Taking a worst-case approach, one may try to evaluate the largest theoretically possible performance degradation given the delay bounds (4.4). For the case of pure input-output jitter, the jitter margin [CLE$^+$04] can be used. Unfortunately, however, no performance degradation theorem for mixed sampling jitter and input-output jitter exists today.

- A third option, which is advocated in this paper, is to do a quantitative analysis with respect to delay and jitter to determine which factor has the larger influence on the performance degradation. From previous experience, the worst case with respect to the bounds (4.4) is often achieved when $\Delta^s_{i,j} = 0$, $\Delta^{io}_{i,j} = D_i$, i.e., the worst control performance is typically obtained for zero jitter and a constant input-output delay of D_i. The quantitative analysis can be carried out using Jitterbug, simulation (using tools like TrueTime [HCAÅ06] or RTSim [CBLL98, PAC$^+$00]), or by experiments on the real system.

The last option is elaborated upon in the rest of this section.

4.5.3 Quantitative performance degradation analysis

As mentioned above, an approximative performance loss index for a control task can be derived in a simulative or experimental fashion. When the system model is not available or it is not accurate, the control performance can be directly monitored using a real-time kernel, like S.Ha.R.K [GAGB01], that allows to enforce desired and precise delays in task executions. The method presented here can be used on any real-time platform, either real or simulated, to derive the performance loss index of a single controller task at a time, as a function of configurable timing attributes.

The most intuitive solution to generate a sampling delay is to defer the start time of the job of the controller task by inserting a delay primitive before the input procedure. Similarly, the input-output delay can be introduced by inserting a delay primitive before the output procedure, as shown in Figure 4.5, where $\delta^s_{i,k}$ and $\delta^{io}_{i,k}$ represent the injected artificial sampling delay and the IO delay for each job $\tau_{i,k}$, respectively. Figure 4.6 illustrates this intuitive method. Notice that, assuming *Input* and *Output* operations consume negligible computation times, the actual input-output delay is $\Delta^{io}_{i,k} = \delta^{io}_{i,k} + C_i$, while the actual sampling delay is always equal to the artificial one, that is $\Delta^s_{i,k} = \delta^s_{i,k}$.

```
CONTROLLER-TASK()
1  Delay($\delta^s_{i,k}$)
2  $sampled\text{-}data$ ← Input()
3  $control\text{-}signal$ ← Calculation($sampled\text{-}data$)
4  Delay($\delta^{io}_{i,k}$)
5  Output($control\text{-}signal$)
```

Figure 4.5: Pseudocode for controller task τ_i with artificial delays.

A problem with this implementation is that, when deadlines are larger than periods, delays can be larger than expected, as depicted in Figure 4.7. In fact,

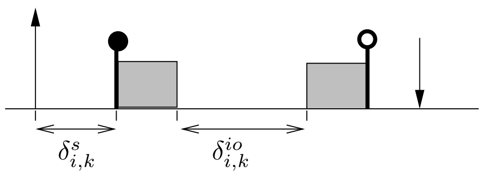


Figure 4.6: Inserting artificial delays.

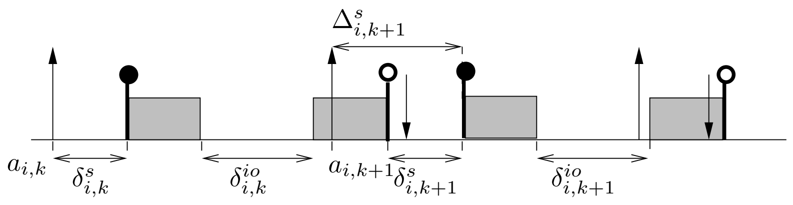


Figure 4.7: Problem when deadlines are larger than periods.

when the k^{th} job of task τ_i completes after the beginning of the next period, the actual sampling delay results to be higher than the specified $\delta^s_{i,k+1}$, and in particular equal to

$$\Delta^s_{i,k+1} = \delta^s_{i,k+1} + f_{i,k} - a_{i,k+1}.$$

To solve this problem, the controller task is split into three subtasks: a periodic subtask and two aperiodic subtasks, as illustrated in Figure 4.8. At the end of each job of the periodic subtask (*subtask1*), a system-level event is posted to activate the first aperiodic subtask (*subtask2*) after a given amount of time, equal to the specified sampling delay $\delta^s_{i,k}$. Such an aperiodic subtask performs *Input* and *Calculation* and then it posts another system-level event to activate the second aperiodic subtask (*subtask3*) after the specified input-output delay $\delta^{io}_{i,k}$. The second aperiodic subtask performs the *Output* and finishes the control job. The two aperiodic subtasks are scheduled with a lower priority with respect to the periodic task to ensure the proper activation sequence.

The timeline at the top of the figure shows the equivalent execution of the controller task with the proper enforced delays. It can be easily seen that, except for a negligible overhead due to the subtask activation, the specified sampling delay $\delta^s_{i,k}$ and input-output delay $\delta^{io}_{i,k}$ are not affected by the task finishing time. It is worth mentioning that the second aperiodic subtask is assigned a priority higher than that of the first aperiodic subtask, because the *Output* is less time consuming and should not be preempted by the execution of the first aperiodic subtask. Also notice that this approach allows generating tasks with arbitrary jitter as well, obtained by introducing random activation delays in the subtasks.

The pseudocode of the controller subtasks is listed in Figure 4.9, 4.10 and 4.11, where $\text{Post-Kernel-Event}(t, e)$ is a function that posts a system-level event e at time t, and t_{cur} is the current system time.

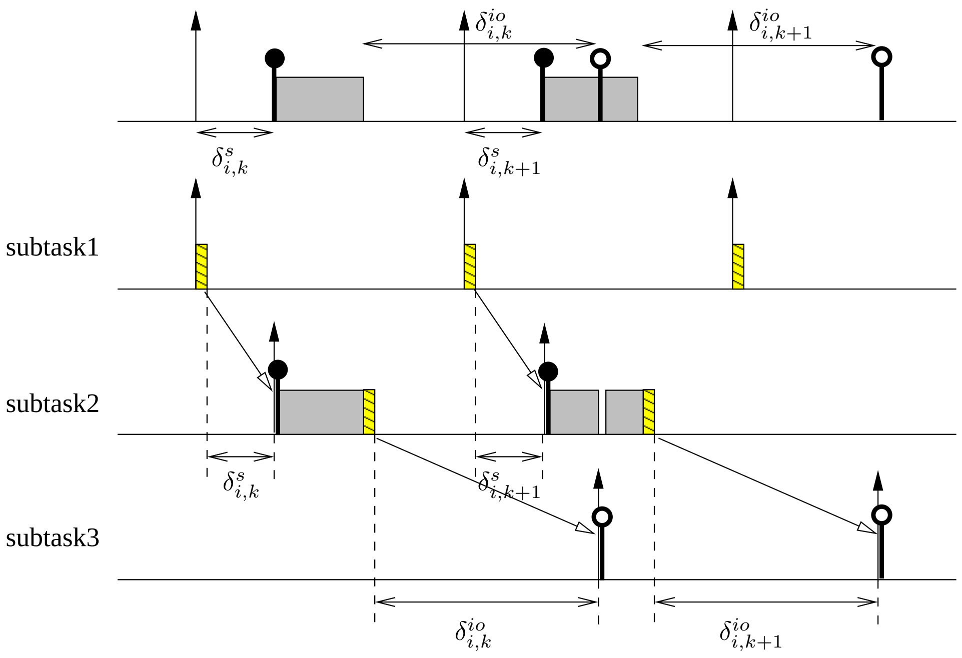


Figure 4.8: Sequence of subtasks to generate delays larger than periods.

```
SUBTASK1()
1  Post-Kernel-Event(
       $t_{cur}$ +$\delta^{s}_{i,k}$,
       $event\text{-}activate\text{-}subtask2$
   )
```

Figure 4.9: Pseudocode for subtask1 of τ_i.

```
SUBTASK2()
1  $sampled\text{-}data \leftarrow$ Input()
2  $control\text{-}signal \leftarrow$ Calculate($sampled\text{-}data$)
3  Post-Kernel-Event(
       $t_{cur}$ +$\delta^{io}_{i,k}$,
       $event\text{-}activate\text{-}subtask3$
   )
```

Figure 4.10: Pseudocode for subtask2 of τ_i.

4.5.4 Example of analysis results

As an example of the quantitative performance analysis, the LQG control of a double integrator process with the sampling interval $T = 0.02sec$ is studied. The

```
SUBTASK3()
1   Output(control-signal)
```

Figure 4.11: Pseudocode for subtask3 of τ_i.

performance loss as a function of the amount of sampling jitter, constant input-output delay, and input-output jitter is plotted in Figure 4.12. The figure makes

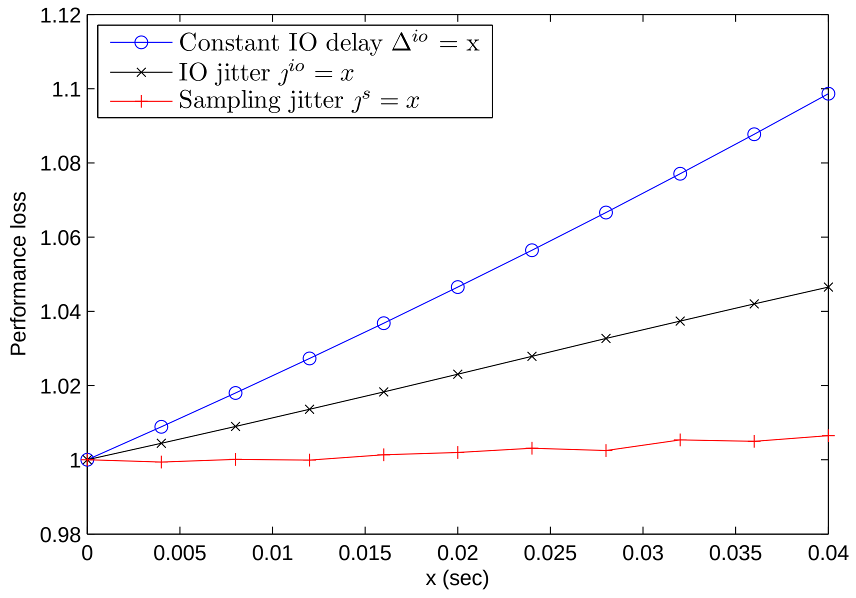


Figure 4.12: Comparison of the influence of delay and jitter for a double integrator with $T = 0.02sec$.

a comparison of the separate effects on control performance of different timing attributes Δ_i^{io} (constant), j_i^{io} and j_i^s. The values of x can be as large as twice the sampling period. Notice that the constant sampling delay is not considered in the comparison, since a task τ_i where all jobs have a constant sampling delay Δ_i^s is equivalent to a task with a release offset of Δ_i^s and sampling delay equal to 0 for all jobs. It is seen that, in this case, the IO delay is the most significant timing attribute influencing the control performance. Hence, the worst case respecting the bounds (4.4) occurs when $\Delta_i^{io} = D_i$ and $j_i^s = j^{io} = 0$.

4.6 Resource Constraints Characterization

Since the performance loss index is assumed to decrease as the period or the deadline of the controllers decrease, the solution of the design problem is to find the smallest values for T_i and D_i that guarantee schedulability.

To determine the feasible task parameters under EDF, the processor demand criterion proposed by Baruah et al. [BRH90] is used. According to this test, a task set is schedulable by EDF if and only if:

$$\begin{cases} \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \\ \forall t \in \mathsf{dlSet} \ \sum_{i=1}^{n} \max\left\{0, \left\lfloor \frac{t-D_i+T_i}{T_i} \right\rfloor \right\} C_i \leq t \end{cases} \tag{4.6}$$

where dlSet is an opportune subset of absolute deadlines.

Unfortunately this test does not provide a description of the feasible parameters that is well suited for maximizing the performance. In fact, since periods and deadlines appear within the floor operator, the shape of the boundary necessary to apply constrained optimization techniques (such as the Lagrange multipliers) is not easy to derive.

To overcome such a problem, the following two-step approach is adopted:

1. First, consider $D_i = T_i$ for all the tasks and find the periods that minimize the performance loss index, using the Liu and Layland necessary and sufficient test for EDF
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{4.7}$$
which is linear and it can be used in the optimization process [SLSS96].

2. Then, fix the task periods as derived in the previous step, relax the assumption $D_i = T_i$, and perform the optimization in the space of the feasible deadlines [BB09b].

Due to the regularity of the constraint expressed by Eq. (4.7), the first step can be made by applying standard convex optimization techniques. If the performance function conforms to a class of some special functions (such as linear, exponential or logarithmic) then a closed solution can also be found [SLSS96, AMMMA01].

The second step can be accomplished by exploiting the geometric properties of the space of feasible deadlines. Bini and Buttazzo [BB09b] proved that given the computation times $\mathbf{C} = (C_1, \ldots, C_n)$ and the periods $\mathbf{T} = (T_1, \ldots, T_n)$, the region of the feasible deadline can be expressed as follows:

$$\mathbb{S} = \bigcap_{\mathbf{k} \in \mathbb{N}^n} \bigcup_{i : k_i \neq 0} \{\mathbf{D} \in \mathbb{R}^n : D_i \geq \mathbf{k} \cdot \mathbf{C} - (k_i - 1) T_i\} \tag{4.8}$$

To clarify the geometry of the space of feasible deadlines we propose an example with two periodic tasks, whose parameters are $\mathbf{C} = (2, 6)$ and $\mathbf{T} = (4, 12)$.

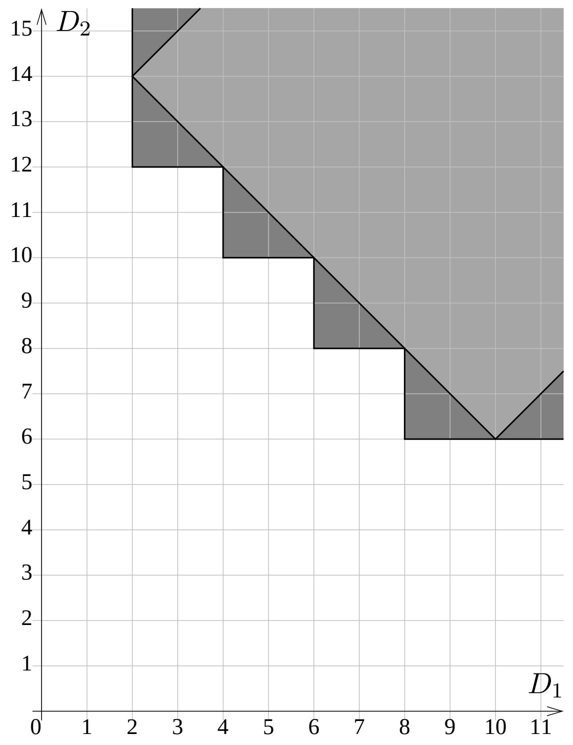


Figure 4.13: The region of feasible deadlines.

According to Eq. (4.8), the resulting space of feasible deadlines is illustrated in Figure 4.13.

Since the performance always improves as deadlines become smaller (i.e. $\frac{\partial J_i}{\partial D_i} \leq 0$), then all the corners of the region of the feasible deadlines are a local optima. An optimization routine should then test the performance value at all these local optima and select the best performing solution. In the example shown in Figure 4.13, local optima are in the set $\mathbb{S} = \{(8,6),(6,8),(4,10),(2,12)\}$.

Unfortunately, the cardinality of the set of local optima does not increase polynomially with the number of tasks, hence this method can be time consuming for large task sets. An alternative solution is to use a convex subregion of the exact space. In [BB09b], it is proved that if the following set of linear constraints are satisfied

$$\begin{cases} D_i - D_j \leq T_i & \forall i,j \\ D_j\,(1 - \sum_{i=1}^{n} U_i) + \sum_{i=1}^{n} U_i\, D_i \geq \sum_{i=1}^{n} C_i & \forall j \end{cases}$$

then the resulting deadline assignment is feasible. Notice that the number of the linear constraints is n^2. Moreover, if in the first step of the optimization procedure the periods are assigned such that the total utilization $\sum_i U_i$ reaches 1 (i.e. the computing resource is fully exploited), the convex constraint becomes

$$\begin{cases} D_i - D_j \leq T_i & \forall i, j \\ \sum_{i=1}^{n} U_i\, D_i \geq \sum_{i=1}^{n} C_i \end{cases} \tag{4.9}$$

whose region is delimited by $n \cdot (n-1) + 1$ linear constraints. In Figure 4.13 the convex subregion is depicted in light gray. Although Eq. (4.9) provides only a sufficient test, the convexity of the region allows implementing a very efficient algorithm for finding a deadline assignment.

In a system composed by both controller tasks and regular tasks, the number of constraints can be further reduced, if the deadlines of regular tasks are equal to the periods, i.e. $D_i = T_i, \forall \tau_i \in \tau_{nctrl}$.

From the first equation of Eq. (4.9), it follows that:

$$\begin{cases} D_i - D_j \leq T_i & \forall \tau_i, \tau_j \in \tau_{ctrl} \\ 0 \leq D_i \leq T_i + T_j & \forall \tau_i \in \tau_{ctrl}, \tau_j \in \tau_{nctrl} \end{cases}$$

From the second equation of Eq. (4.9), it follows that:

$$\begin{aligned} \sum_{\tau_i \in \tau_{ctrl}} C_i + \sum_{\tau_i \in \tau_{nctrl}} C_i &\leq \sum_{\tau_i \in \tau_{ctrl}} U_i\, D_i + \sum_{\tau_i \in \tau_{nctrl}} U_i\, D_i \\ &= \sum_{\tau_i \in \tau_{ctrl}} U_i\, D_i + \sum_{\tau_i \in \tau_{nctrl}} U_i\, T_i \\ &= \sum_{\tau_i \in \tau_{ctrl}} U_i\, D_i + \sum_{\tau_i \in \tau_{nctrl}} C_i \end{aligned}$$

Hence,

$$\sum_{\tau_i \in \tau_{ctrl}} U_i\, D_i \geq \sum_{\tau_i \in \tau_{ctrl}} C_i$$

Therefore Eq. (4.9) can be written as:

$$\begin{cases} D_i - D_j \leq T_i & \forall \tau_i, \tau_j \in \tau_{ctrl} \\ 0 \leq D_i \leq T_i + \min T_j & \forall \tau_i \in \tau_{ctrl}, \tau_j \in \tau_{nctrl} \\ \sum_{\tau_i \in \tau_{ctrl}} U_i\, D_i \geq \sum_{\tau_i \in \tau_{ctrl}} C_i \end{cases} \tag{4.10}$$

Notice that when $\tau_{nctrl} \neq \emptyset$, i.e. $n_{ctrl} < n$, the number of constraints is reduced to $n_{ctrl}^2 + n_{ctrl} + 1$.

4.7 Experimental results

This section illustrates how the proposed methodology can be used for selecting periods and deadlines in a system consisting of both controller tasks and regular tasks. The overall performance of the system is evaluated by simulating the runtime of the whole system scheduled by EDF on uniprocessor using TrueTime [HCAÅ06] in Matlab.

4.7.1 The control systems

Two types of plant have been considered with highly different dynamics to control. The first type, denoted as **Plant A**, is a double integrator with the following state-space model:

$$\begin{aligned} \frac{dx}{dt} &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v \\ y &= \begin{bmatrix} 0 & 1 \end{bmatrix} x + \sqrt{0.1} e. \end{aligned}$$

The cost function used for both LQG design [ÅW97] and control performance evaluation is

$$J = E \lim_{t_p \to \infty} \frac{1}{t_p} \int_0^{t_p} \left(x^T \begin{bmatrix} 0 & 0 \\ 0 & 10 \end{bmatrix} x + u^2 \right) dt.$$

The second type, denoted as **Plant B**, has the following state-space model:

$$\begin{aligned} \frac{dx}{dt} &= \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 35 \\ -61 \end{bmatrix} v \\ y &= \begin{bmatrix} 2 & 1 \end{bmatrix} x + e \end{aligned}$$

with its corresponding quadratic cost function

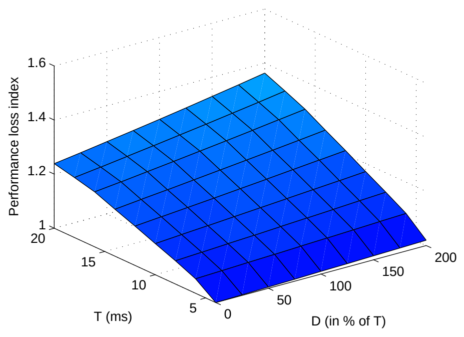
$$J = E \lim_{t_p \to \infty} \frac{1}{t_p} \int_0^{t_p} \left(x^T \begin{bmatrix} 700 & 20\sqrt{35} \\ 20\sqrt{35} & 20 \end{bmatrix} x + u^2 \right) dt.$$

This plant is a modification of the one investigated in [NBW98], where the LQG design results in a controller that is extremely sensitive to delay and jitter.

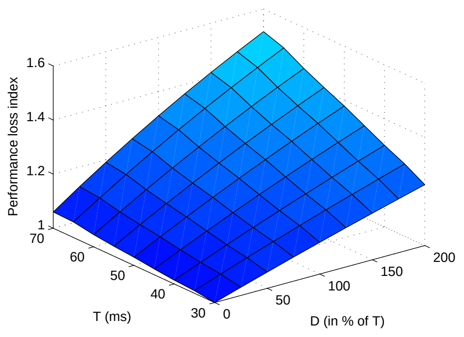
For all the plant models, v is a continuous-time zero-mean white noise process with unit intensity, and e is a discrete-time zero-mean white noise process with unit variance. In the cost function, $[0, t_p]$ is the time span to be considered. Although t_p should be ∞ in LQG design, when evaluating control performance, it is reasonable to use a suitable large value, which in this case was set to 50 seconds, also equal to the simulation time of the experiment.

The control performance loss index with respect to sampling period and relative deadline was derived for both types of plant. To obtain such an index, a performance derivation procedure using the method in Section 4.5.2 was set up in TrueTime. The adjustable ranges of sampling period are [4, 20] ms for **Plant A** and [30, 70] ms for **Plant B**, respectively. For both types of plant, the values of different timing attributes can be as large as twice the sampling period. The evaluated performance loss indices are plotted separately in Figure 4.14.

To facilitate the comparison of the performance between different plants, each performance loss index has been normalized so that the minimum performance value is 1. Figure 4.14a shows that **Plant A** is only sensitive to sampling period, and quite tolerant to relative deadline, especially when sampling period is small. On the contrary, Figure 4.14b shows that **Plant B** is much more sensitive to relative deadline than to sampling period.



(a) Performance loss index of **Plant A**.



(b) Performance loss index of **Plant B**.

Figure 4.14: Derived performance loss indices.

4.7.2 Experimental setup

The considered task set τ consists of $n = 7$ hard real-time tasks scheduled by EDF on a uniprocessor. Among those tasks, $n_{ctrl} = 3$ are controller tasks, conceptually grouped as τ_{ctrl}, and the rest are $n_{nctrl} = 4$ regular tasks denoted as τ_{nctrl}. The 3 controller tasks in τ_{ctrl} are labeled as τ_1, τ_2 and τ_3, where τ_1 and τ_2 control a **Plant A** type, whereas τ_3 controls a **Plant B** type. The derived performance loss indices of both types of plant are saved as 2-D lookup tables, which allows the optimization procedure to interpolate the cost value.

Let the variable ranges of task periods be the same as the ranges of sampling periods in the evaluation of the performance loss indices in Section 4.7.1, and similarly assume the WCET of each controller task is equal to 4 ms. Then, the maximum and minimum utilization of each task was obtained, as shown in Table 4.1.

Table 4.1: Summary of the controller tasks

Task	WCET(ms)	Period(ms)	Utilization(100%)
τ_1	4	$[4, 20]$	$[0.2, 1]$
τ_2	4	$[4, 20]$	$[0.2, 1]$
τ_3	4	$[30, 70]$	$[0.057, 0.133]$
U_{ctrl}			$[0.457, 2.133]$

Notice that the utilization of all controller tasks U_{ctrl} ranges from 45.7% to 213.3%, meaning that the controller tasks cannot be scheduled in EDF at their maximum sampling rates.

To investigate situations under different system loads, the utilization of all the controller tasks U_{ctrl} was fixed to 0.5 throughout the simulation, and the total utilization of the whole task set U was varied from 0.6 to 1 with a step of 0.1. The tasks within τ_{nctrl} were generated using the UUNIFAST algorithm [BB04], with computation time C_i uniformly distributed in $[1,\ 10]\ ms$ and utilization U_i chosen according to a 6-dimensional uniform distribution to reach $U_{nctrl} = U - U_{ctrl}$. For each U, $N = 100$ subsets of τ_{nctrl} were randomly generated.

4.7.3 Period and deadline selection

To select the scheduling parameters that optimize the overall control performance, the function $\mathcal{F}$ in Eq. (4.3) has been chosen as follows to form up a global performance loss index:

$$J = \sum_{i=1}^{3} w_i \cdot J_i(T_i, D_i)$$

where $w = [w_1, w_2, w_3]$ is a weight vector. For this case, all weights have been set to 1, meaning that all plants have the same importance.

As long as the utilization of all controller tasks U_{ctrl} is decided, period selection can be performed without consideration of any regular tasks, using the resource constraint of $\sum_{\tau_i \in \tau_{ctrl}} \frac{C_i}{T_i} \leq U_{ctrl}$, as the first step described in Section 4.6. By solving the optimization problem with deadlines equal to periods, the results shown in Table 4.2 were obtained.

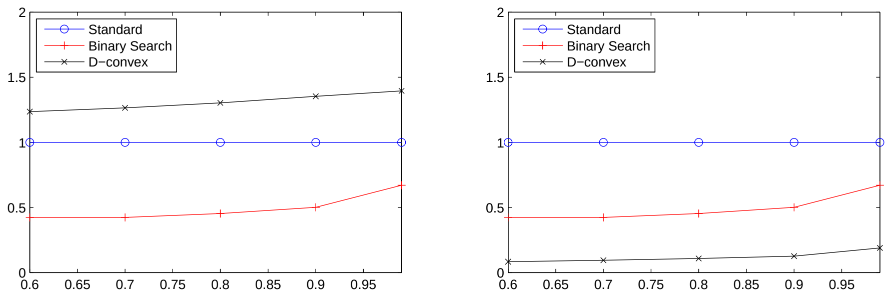
Table 4.2: Results of period selection

Task	Period(ms)	Utilization(100%)
τ_1	0.0189	0.212
τ_2	0.0189	0.212
τ_3	0.0528	0.076
U_{ctrl}		0.50

Once periods have been derived, deadline selection can then be performed in the deadline space. The advantages of the proposed method has been evaluated with respect to other two approaches under three different scenarios:

- *Standard*: Deadlines of tasks are equal to periods, therefore task deadlines are not utilized to limit delay and jitter.
- *Binary Search*: The deadlines of the three controller tasks are uniformly reduced by binary search. This method can be found in [BBGL99] or [HB07].
- *D-convex*: The deadlines of the three controller tasks are selected using the deadline convex space, as proposed in Section 4.6.

Notice that only the deadlines of the controller tasks are selected, while the deadlines of the regular tasks are equal to their periods, i.e. $\forall \tau_i \in \tau_{nctrl}, D_i = T_i$. The results of deadline selection are reported in Figure 4.15, which shows the average value of the ratio of the selected deadline D_i and the period T_i. Note that a ratio larger than 1 means that deadline is extended beyond the period. The ratios of τ_1 and τ_2 are the same due to the same performance loss index and the same weight, and thus reported in the same figure.



(a) Ratio $\frac{D_i}{T_i}$ of τ_1 and τ_2. (b) Ratio $\frac{D_i}{T_i}$ of τ_3.

Figure 4.15: Ratio of selected deadline and period $\frac{D_i}{T_i}$.

In both subfigures, the ratios under the *Standard* scenario stay at 1, whereas the ratios under *Binary Search* have the same value due to the uniform deadline reduction. However, as shown in Figure 4.15a, applying the *D-convex* method, the ratio of τ_1 and τ_2 becomes greater than 1, meaning that their deadlines are extended beyond their periods, to achieve a greater reduction of τ_3's deadline. Indeed, Figure 4.15b shows that, using *D-convex* method, τ_3's deadlines can be reduced more than under *Binary Search*.

The resulted control performance loss under the three considered cases is illustrated in Figure 4.16. As shown in the figure, under the *Standard* scenario and

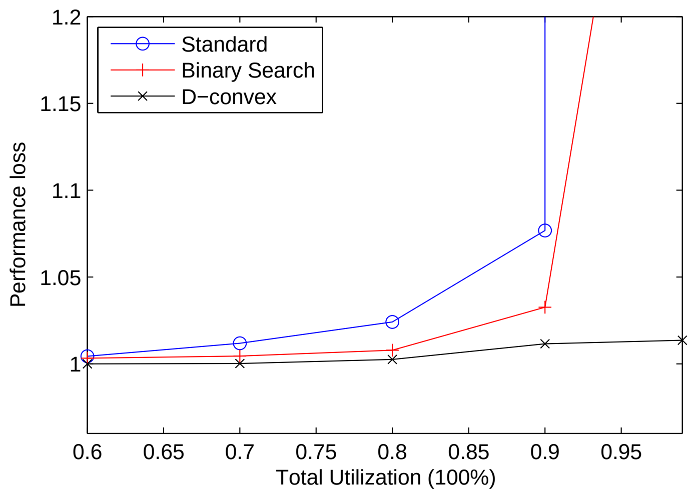


Figure 4.16: Control performance under different strategies.

Binary Search, when system load is high, the performance loss of the whole system tends to infinity. This means that, in a highly loaded system, the interference introduced on the execution of τ_3 leads to instability of the plant (*Plant B*). However, under scenario *D-convex*, the performance loss is kept at an acceptable level, even if the system is highly loaded. This is possible because the *D-convex* method allows a more aggressive reduction of τ_3 deadline, limiting its delay and jitter to maintain the stability.

A specific simulation was performed to compare the three methods in highly-loaded systems. All the conditions remain the same, while the total utilization is set to be 1. The result of performance over the simulation time is shown in Figure 4.17 which clearly demonstrates that using the *Standard* and *Binary Search* methods the system went unstable after certain time, while using the *D-convex* approach the system was stable and maintained a sufficiently good performance.

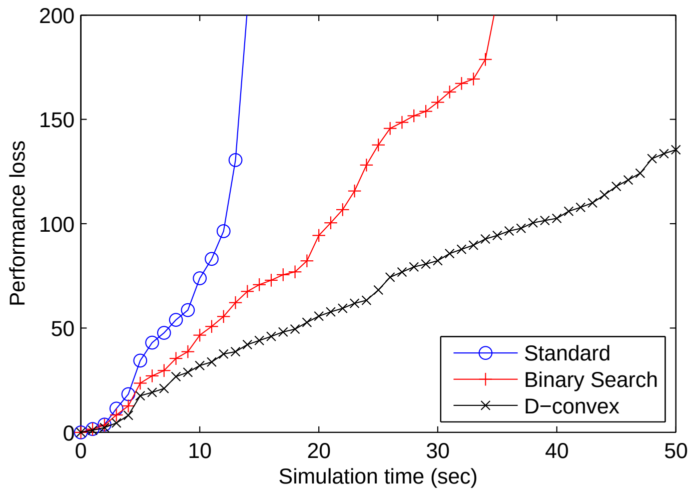
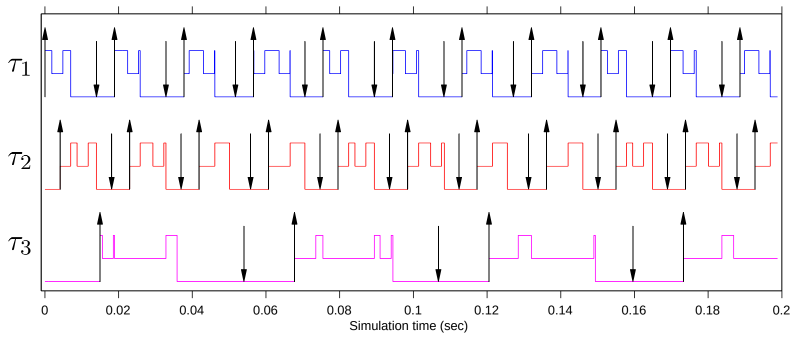


Figure 4.17: Control performance over time under different strategies.

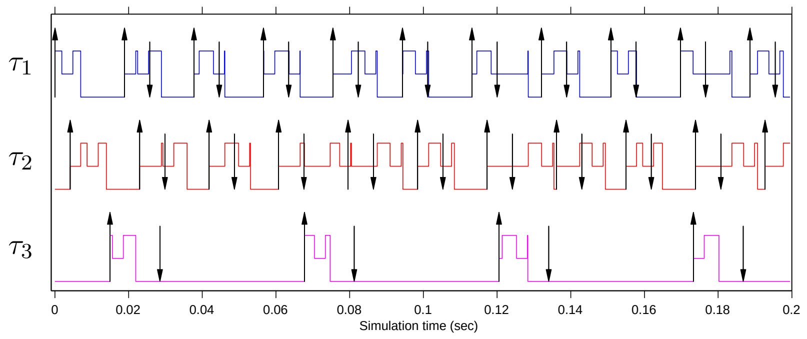
The schedules of the three controller tasks obtained in this simulation under *Binary Search* and *D-convex* are reported in Figure 4.18a and Figure 4.18b, respectively. The upwards arrows denote the arrival times and the downwards arrows indicate the deadlines. For each task, the 3 different levels of the step function mean (from top to bottom) running, ready and idle states of the task. Notice that the tasks are initially released with random offsets. It is shown that τ_3 experiences much less delay and jitter after using the proposed method for deadline selection.

4.8 Conclusion

In this chapter, the problem of task parameter selection for real-time controller tasks has been addressed. In particular, a general framework has been proposed to make integrated real-time control design that attempts to avoid the traditional



(a) Schedule (*Binary Search*).



(b) Schedule (*D-convex*).

Figure 4.18: Partial schedule.

repetitive design procedure and to achieve optimal performance and resource exploitation. A general method has been proposed to derive the control performance loss index in either a simulative or experimental way, with respect to various timing attributes, and arbitrary deadlines, which are allowed to be less than, equal to or greater than the periods. Task periods and deadlines were then selected by optimization upon the convex approximation of EDF deadline space, considering the delay and jitter effects on control performance.

Extensive simulations have been performed where a comparison was made between the proposed methodology and other methods. The results have shown that the proposed method managed to keep the performance loss at an acceptable level even in highly loaded systems which might lead to instability using other methods.

Chapter 5

Parallel control application on multiprocessor platform

5.1 Introduction

Multi-core architectures represent the next generation family of processors for providing an efficient solution to the problem of increasing the processing speed with a contained power dissipation. In fact, increasing the operating frequency of a single processor would cause serious heating problems and a considerable power consumption.

However, analyzing multi-core systems is not trivial, and the research community is still working to produce new theoretical results or extend the well established theory for uniprocessor systems developed in the last 30 years. Also, fully exploiting the computational power available in a multi-core platform requires new programming paradigms, which should allow expressing the intrinsic parallel structure of the applications in order to optimize the allocation of parallel execution flows to different cores.

Moreover, the complexity of modern embedded systems is growing continuously, and the software is often structured in a number of concurrent applications, each consisting of a set of tasks with various characteristics and constraints, and sharing the same resources. In such a scenario, isolating the temporal behavior of real-time applications is crucial to prevent a reciprocal interference among critical activities. As described in Section 2.1.5, temporal isolation can be achieved through *Resource Reservation* technique. When moving to multiprocessor systems, however, the meaning of reservations has to be revisited, and the research community just started to address this issue.

This chapter proposes a method for allocating a parallel real-time application, described as a set of tasks with time and precedence constraints, on a multi-core platform. To achieve modularity and simplify portability of applications on different multi-core platforms, we abstract the virtual platform by the Multi Supply Function (MSF) [BBB09]. The advantage of using the virtual platform MSF is that, if the hardware platform is replaced with another one with a different number of cores, the set of reservations does not need to be changed, and only the server mapping to physical processors has to be done. Also, to be independent of a particular reservation algorithm, a virtual processor reservation is expressed by a bounded-delay time partition, denoted by the pair (α, Δ), where α is the allocated bandwidth and Δ is the maximum service delay. This method, originally proposed by Mok et al. [MFC01], is general enough to express several types of resource reservation servers.

To better exploit the existing parallelism available in the computing platform, the application precedence graph is partitioned into a set of flows, each consisting of a subset of tasks to be sequentially executed on a virtual processor. For each flow, we determine its computational requirements and compute the minimum server bandwidth needed for executing it. Since the bandwidth requirements depend on the specific partition, the proposed method can be used to identify the partition that minimizes a given cost function (e.g., the overall bandwidth consumption or the maximum degree of parallelism).

A simple control application involving a ball-and-plate plant is used to exemplify the utilization of the proposed methodology in real-time control design. By exploiting the software parallelism of the controller and execute it on multiprocessor platform, smaller sampling period and control loop latency are possible to achieve than on a uniprocessor, hence leading to better control performance.

5.2 Related Work

The most natural abstraction of a multi-core platform is probably the uniform multiprocessor model proposed by Funk, Goossens and Baruah [FGB01], where a collection of sequential machines is abstracted by their speeds. In this paper, the authors also showed that a set of tasks scheduled by global EDF (with migrations) and requiring an overall bandwidth of 120% has higher chances to be successfully scheduled upon two virtual processors with bandwidth 100% and 20%, rather than on other two with the same bandwidth of 60%. However, when no task migration is allowed, packing the bandwidth into full reservations is not always the best approach. In fact, consider a periodic application Γ consisting of 5 tasks with computation times 1, 1, 5, 6, 6 and period equal to 10 (deadline = period). In this case, the bandwidth required by the application is $U_\Gamma = 190\%$, and a feasible schedule can be found using 3 reservations, equal to 80%, 60% and 50%. However, no feasible solution exists if the bandwidth is provided by two reservations equal to 100% and 90%.

Otero et al. [OPRS$^+$06] applied the resource reservation paradigm to interrelated resources (processor cycles, cache space, and memory access cycles) to achieve robust, flexible and cost-effective consumer products.

Shin et al. [SEL08] proposed a multiprocessor periodic resource model to describe the computational power supplied by a parallel machine. In their work, a resource is modeled using three parameters (P, Q, m), meaning that an overall budget Q is provided by at most m processors every period P.

Leontyev and Anderson [LA08] proposed a multiprocessor scheduling scheme for supporting hierarchical reservations (containers) that encapsulate hard and soft sporadic real-time tasks.

Very recently, Bini et al. [BBB09] proposed to abstract a set of m virtual processors by the set of the m supply functions [FM02, LB03, SL03] of each virtual processors. In this paper we borrow such an abstraction of a virtual multi-core platform.

In all these works, however, the application is modeled as a collection of sporadic tasks, and no precedence relations are taken into account.

A more accurate task model (generalized multiframe task) considering conditional execution flows, expressed by a Directed Acyclic Graph (DAG), has been proposed by Baruah et al. [BCGM99]. However, multiple branches outgoing from a node denote alternative execution flows rather than parallel computations.

The problem of managing real-time tasks with precedence relations was addressed by Chetto et al. [CSB90], who proposed a general methodology for assigning proper activation times and deadlines to each task in order to convert a precedence graph into timing constraints, with the objective of guaranteeing the schedulability under EDF. Their algorithm, however, is only valid for uniprocessor systems and does not consider the possibility of having parallel computations.

Partitioning and scheduling tasks with precedence constraints onto a multiprocessor system has been shown to be NP-Complete in general [Sar89], and various heuristic algorithms have been proposed in the literature to reduce the complexity [ACD74, ERL90, kKAA96], but their objective is to minimize the total completion time of the task set, rather than guaranteeing timing constraints under temporal isolation. One category of such algorithms, called List scheduling [ERL90, ACD74], is based on proper priority assignments to meet the application constraints. Another technique, called Critical Path Heuristics [Sar89, kKAA96], was developed to deal with non-negligible communication delays between tasks. The idea is to assigns weights to nodes to reflect their resource usage and to edges to reflect the cost of inter-processor communication, and then shorten the length of the Critical Path of a DAG by reducing the communication between tasks within a cluster.

Collette et al. [CCG08] proposed a model to express the parallelism of a code by characterizing all possible durations a computation would take on different number of processors. Schedulability is checked under global EDF, but no precedence relations are considered in the analysis.

Lee and Messerschmitt [LM87] developed a method to statically schedule synchronous data flow programs, on single or multiple processors. Precedence relations are considered in the model, but no deadline constraints are taken into account and temporal protection is not addressed.

Jayachandran and Abdelzaher [JA08] presented an elegant and effective algebra for composing the delay of applications modeled by DAGs and scheduled on distributed systems. However, they did not provide temporal isolation among applications.

Fisher and Baruah [FB09] derived near-optimal sufficient tests for determining whether a given collection of jobs with precedence constraints can feasibly meet all deadlines upon a specified multiprocessor platform under global EDF scheduling, so partitioning issues and resource reservations are not addressed.

5.3 System model and background

A real-time application is modelled as a set of tasks with given precedence constraints, specified as a Directed Acyclic Graph (DAG). An application is considered to be sporadic, meaning that it can be cyclically activated with a minimum interarrival period T and must be completed within a relative deadline D, which can be less than or equal to the period. Each task consists of a sequential portion of code with known worst-case execution time (WCET) C_i.

Note that the DAG represents a description of the application considering the maximum level of parallelism. This means that each task represents a sequential activity to be executed on a single core. Tasks can be preempted at any time and do not call blocking primitives during their execution. Figure 5.1 illustrates an example of DAG for an application consisting of five tasks, with execution times:

$$C_1 = 4, C_2 = 1, C_3 = 5, C_4 = 2, C_5 = 3.$$

The entire application starts at time $t = 0$ and is periodically activated with a period $T = 20$. We consider a relative deadline D equal to the period.

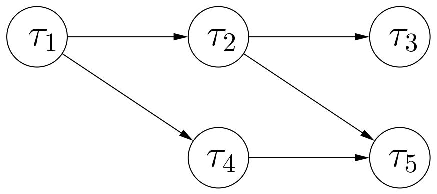

$$\begin{tikzcd}
\tau_1 \arrow[r] \arrow[dr] & \tau_2 \arrow[r] \arrow[dr] & \tau_3 \\
 & \tau_4 \arrow[r] & \tau_5
\end{tikzcd}$$

Figure 5.1: A sample application represented with a DAG.

To better illustrate the parallel execution of an application and identify the maximum number of required processors, we adopt a different description that visualizes the computation times of each task in the timeline, as in a Gantt chart. In such a diagram, denoted as the *timeline representation*, each task starts as soon as possible on the first available core, assuming as many cores as needed. For the application shown in Figure 5.1, the timeline representation is illustrated in Figure 5.2, where synchronization points coming from the precedence graph are represented by arrows.

An advantage of the timeline representation is that it clearly visualizes the intrinsic parallelism of the application, showing in each time slot the maximum number of cores needed to perform the required computation. This means that adding

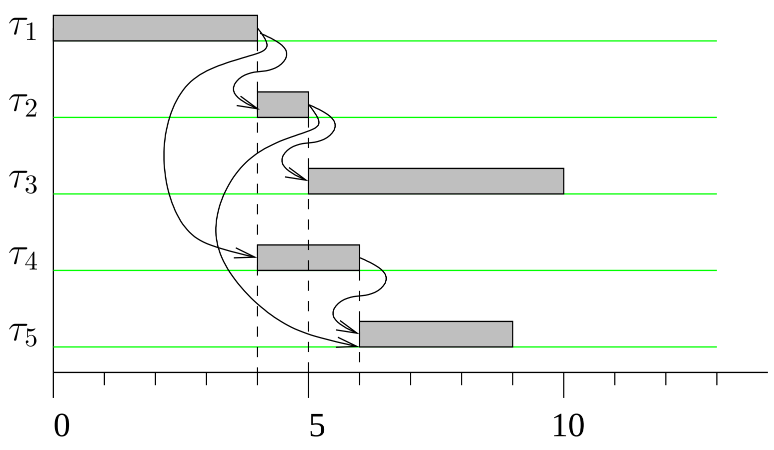


Figure 5.2: Timeline representation.

other cores will not reduce the overall response time, because the DAG already expresses the maximum level of parallelism.

5.3.1 Terminology and notation

First, to shorten the expressions, we may denote $\max\{0, x\}$ as $(x)_0$. Moreover, throughout the paper we adopt the following terminology and notation.

- **Application** Γ. It is a set of n tasks with given precedence relations expressed by a Directed Acyclic Graph (DAG). The application is sporadic, meaning that it is cyclically activated with a minimum interarrival time T and must complete within a given relative deadline D, which can be less than or equal to T. This allows asserting that only one instance of the application is running at any time.

- **Task** τ_i. It is a portion of code that cannot be parallelized and must be executed sequentially. τ_i can be preempted at any time and is characterized by a known worst-case execution time $C_i > 0$. τ_i is also assigned a deadline d_i and an activation time a_i relative to the activation of the first task of the application. The assignment of deadlines and activation times is investigated in Section 5.4.1. Tasks are scheduled by EDF.

- **Precedence relation** $\mathcal{R}$. It is formally defined as a partial ordering $\mathcal{R} \subseteq \Gamma \times \Gamma$. Notation $\tau_i \prec \tau_j$ denotes that τ_i is a *predecessor* of τ_j, meaning that τ_j cannot start executing before the completion of τ_i. Notation $\tau_i \rightarrow \tau_j$ denotes that τ_i is an *immediate predecessor* of τ_j, meaning that $\tau_i \prec \tau_j$ and

$$\tau_i \prec \tau_k \prec \tau_j \Rightarrow (\tau_k = \tau_i \text{ or } \tau_k = \tau_j).$$

- **Path** P. It is any subset of tasks $P \subseteq \Gamma$ that is totally ordered according to $\mathcal{R}$; i.e., $\forall \tau_i, \tau_j \in P$ either $\tau_i \prec \tau_j$ or $\tau_j \prec \tau_i$.

- **Execution time function** $C(\cdot)$. It is a function $C : \mathcal{P}(\Gamma) \to \mathbb{R}$ that, applied to any subset A of Γ, returns the total execution time of the tasks in A:

$$\forall A \subseteq \Gamma \quad C(A) \overset{\text{def}}{=} \sum_{\tau_i \in A} C_i.$$

- **Sequential Execution Time** C^s. It is the minimum time needed to complete the application on a uniprocessor, by serializing all tasks in the DAG. It is equal to the sum of all tasks computation times:

$$C^s \overset{\text{def}}{=} C(\Gamma).$$

  For the application illustrated in Figure 5.2, we have $C^s = 15$.

- **Parallel Execution Time** C^p. It is the minimum time needed to complete the application on a parallel architecture with an infinite number of cores. It is equal to

$$C^p \overset{\text{def}}{=} \max_{P \text{ is a path}} C(P). \tag{5.1}$$

  Notice that the application relative deadline cannot be less than C^p, otherwise it is missed even on an infinite number of cores. For the application illustrated in Figure 5.2, we have $C^p = 10$.

- **Critical path** (CP). It is a path P having $C(P) = C^p$.

- **Virtual processor** VP_k. It is an abstraction of a sequential machine achieved through a resource reservation mechanism characterized by a bandwidth $\alpha_k \leq 1$ and a maximum service delay $\Delta_k \geq 0$.

- **Flow** F_k. It is a subset of tasks $F_k \subseteq \Gamma$ allocated on virtual processor VP_k, which is dedicated to the execution of tasks in F_k only. An application Γ is partitioned into m flows.

- **Flow computation time** C_k^F. It is the cumulative computation time of the tasks in flow F_k:

$$C_k^F \overset{\text{def}}{=} C(F_k).$$

Dividing an application into parallel flows allows several options, from the extreme case of defining a single flow for the entire application (where no parallelism is exploited/necessary and all tasks are sequentially executed on a single core) to the case of having a flow per task (maximum parallelism). The way in which flows are defined may affect the total bandwidth required to execute the application. Hence, we now address the problem of finding the best partition of flows that minimizes the total bandwidth requirements.

Intuitively, grouping tasks into large flows improves schedulability, as long as each flow has a bandwidth less than or equal to one. To better explain each step of

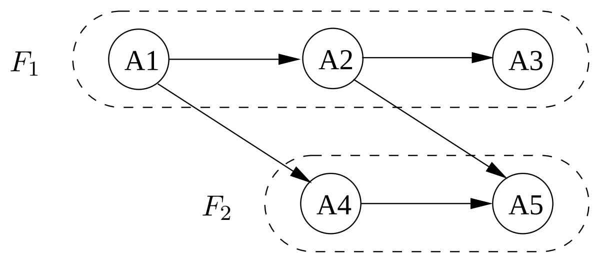


Figure 5.3: Parallel flows in which the application can be divided.

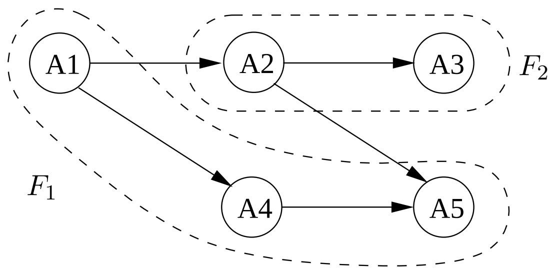


Figure 5.4: An alternate parallel flow selection.

the process, we consider a reference application consisting of five tasks, previously illustrated in Figure 5.1. For this example, we divide the application in two flows, as illustrated in Figure 5.3. Notice that there can be several ways for selecting flows in the same application. An alternative solution is shown in Figure 5.4.

5.3.2 Demand Bound Function

Since EDF is used as a scheduler, here we recall the concept of demand bound function that is used to estimate the amount of required computational resource. The processor demand of a task τ_i that has activation time a_i, computation time C_i, period T_i, and relative deadline d_i, in any interval $[t_1, t_2]$ is defined to be the amount of processing time $g_i(t_1, t_2)$ requested by those instances of τ_i activated in $[t_1, t_2]$ that must be completed in $[t_1, t_2]$. That is [BHR90],

$$g_i(t_1, t_2) \overset{\text{def}}{=} \left(\left\lfloor \frac{t_2 - a_i - d_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - a_i}{T_i} \right\rceil + 1 \right)_0 C_i.$$

The overall demand bound function of a subset of tasks $A \subseteq \Gamma$ is

$$h(A, t_1, t_2) \overset{\text{def}}{=} \sum_{\tau_i \in A} g_i(t_1, t_2)$$

where we made it depend on the beginning and the length of the interval.

As suggested by Rahni et al. [RGR08], we can use a more compact formulation of the demand bound function that depends only on the length t of the time interval

$[t_1, t_1 + t]$:

$$\mathsf{dbf}(A, t) \overset{\text{def}}{=} \max_{t_1} h(A, t_1, t_1 + t). \tag{5.2}$$

5.3.3 The (α, Δ) server

Mok et al. [MFC01] introduced the "bounded delay partition" to describe a reservation by two parameters only: a bandwidth α and a delay Δ. The bandwidth α measures the amount of resource that is assigned to the demanding application, whereas Δ represents the worst-case service delay.

Before introducing the α and Δ parameters, it is necessary to recall the concept of supply function [LB03, SL03], that represents the minimum amount of time that a generic virtual processor can provide in a given interval of time.

Definition 2 (Def. 9 in [MFC01], Th. 1 in [LB03], Eq. (6) in [EAL07])**.** *Given a virtual processor* VP_k*, its* supply function $Z_k(t)$ *is the minimum amount of time provided by the reservation in every time interval of length* $t \geq 0$*.*

The supply function can be defined for many kinds of reservations, as static time partitions [MFC01, FM02], periodic servers [LB03, SL03], or periodic servers with arbitrary deadline [EAL07]. For example, for the simple case of a periodic reservation that allocates Q units of time every period P, we have [LB03, SL03]:

$$Z(t) = \max\{0, t - P + Q - (k + 1)(P - Q), kQ\} \tag{5.3}$$

with $k = \left\lfloor \frac{t-P+Q}{P} \right\rfloor$.

Given the supply function, the bandwidth α and the delay Δ can be formally defined as follows.

Definition 3 (compare Def. 5 in [MFC01])**.** *Given* VP_k *with supply function* Z_k*, the* bandwidth α_k *of the virtual processor is defined as*

$$\alpha_k \overset{\text{def}}{=} \lim_{t \to \infty} \frac{Z_k(t)}{t}. \tag{5.4}$$

The Δ parameter provides a measure of the responsiveness, as proposed by Mok et al. [MFC01].

Definition 4 (compare Def. 14 in [MFC01])**.** *Given* VP_k *with supply function* Z_k *and bandwidth* α_k*, the* delay Δ_k *of the virtual processor is defined as*

$$\Delta_k \overset{\text{def}}{=} \sup_{t \geq 0} \left\{ t - \frac{Z_k(t)}{\alpha_k} \right\}. \tag{5.5}$$

Informally speaking, given a VP ν with bandwidth α_ν, the delay Δ_ν is the minimum horizontal displacement such that the line $\alpha_\nu(t-\Delta_\nu)$ is a lower bound of $Z_\nu(t)$.

Once the bandwidth and the delay are computed, the supply function of VP_k can be lower bounded as follows:

$$Z_k(t) \geq \alpha_k(t-\Delta_k)_0. \tag{5.6}$$

If the (α, Δ) server is implemented through a periodic server [LB03, SL03] that allocates a budget Q_k every period P_k, we have a bandwidth $\alpha_k = Q_k/P_k$ and a delay $\Delta_k = 2(P_k - Q_k)$. In practice, however, a portion of the processor bandwidth is wasted to perform context switches every time a server is executed. If σ is the runtime overhead required for a context switch, and P_k is the server period, the effective server bandwidth can be computed as:

$$B_k = \alpha_k + \frac{\sigma}{P_k}.$$

Expressing P_k as a function of α_k and Δ_k we have

$$P_k = \frac{\Delta_k}{2(1-\alpha_k)}.$$

Hence,

$$B_k = \alpha_k + 2\sigma\frac{1-\alpha_k}{\Delta_k}. \tag{5.7}$$

From previous results [SL03], we can state that a subset A is schedulable on the virtual processor characterized by bandwidth α and delay Δ, if and only if:

$$\forall t \geq 0 \quad \mathsf{dbf}(A,t) \leq \alpha(t-\Delta)_0. \tag{5.8}$$

5.4 Partitioning an application into flows

This section describes the method proposed in this paper to determine the optimal partition of an application into flows. A sample partition is depicted in Figure 5.5.

The possible partitions into flows are explored through a branch and bound search algorithm, whose details are given later in Section 5.4.3.

For a given partition (i.e., selection of flows), we first transform precedence relations into timing constraints by assigning suitable deadlines and activation times to each task, as illustrated in Section 5.4.1.

Once deadlines and activations are assigned, the overall computational requirement of each flow F_k is evaluated through its demand bound function and the parameters of the corresponding virtual processor VP_k are computed, as explained in Section 5.4.2.

Then, if the objective is to minimize the total bandwidth, the overall bandwidth required by the entire partition is computed by summing the bandwidths computed

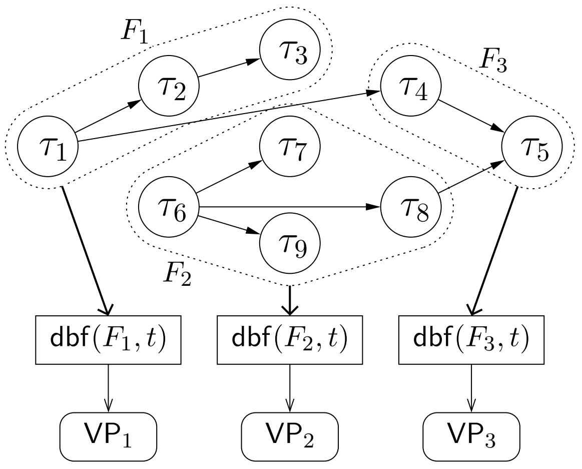


Figure 5.5: A sample partition into three flows.

for each flow using Equation (5.7) and, finally, the partition with the minimum bandwidth is determined as a result of the branch and bound search algorithm. A different metrics is also presented in Section 5.4.3 to minimize the maximum degree of parallelism.

5.4.1 Assigning deadlines and activations

Given a partition $\{F_1, \ldots, F_m\}$ of the application into m flows, activation times a_i and the deadlines d_i are assigned to all tasks to meet precedence relations and timing constraints. The assignment is performed according to a method originally proposed by Chetto-Silly-Bouchentouf [CSB90], adapted to work on multi-core systems and slightly modified to reduce the bandwidth requirements. The algorithm starts by assigning the application deadline D to all tasks without successors. Then, the algorithm proceeds by assigning the deadlines to a task τ_i for which all successors have been considered. The deadline assigned to such a task is

$$d_i = \min_{j:\tau_i \to \tau_j} (d_j - C_j) \tag{5.9}$$

The pseudo-code of the deadline assignment algorithm is illustrated in Figure 5.6.

For the application shown in Figure 5.1, considering that the overall deadline is $D = T = 20$, by applying the transformation algorithm, we get:

$$\begin{aligned}
d_3 &= 20\\
d_5 &= 20\\
d_2 &= \min(d_3 - C_3, d_5 - C_5) = \min(15, 17) = 15\\
d_4 &= d_5 - C_5 = 17\\
d_1 &= \min(d_2 - C_2, d_4 - C_4) = \min(14, 15) = 14.
\end{aligned}$$

```
ASSIGN DEADLINES($\tau$)
1 for all (nodes without successors) set $D_i = D$;
2 while (there exist nodes not set) {
3   select a task $\tau_k$ with all successors modified;
4   set $d_k = \min_{j:\tau_k \to \tau_j} (d_j - C_j)$;
5 }
```

Figure 5.6: The deadline assignment algorithm.

Activation times are set in a similar fashion, but we slightly modified the Chetto-Silly-Bouchentouf's algorithm to take into account that different flows can potentially execute in parallel on different cores. Clearly, τ_i cannot be activated before all its predecessors have finished.

Let τ_j be a predecessor of τ_i and let F_k be the flow τ_i belongs to. If $\tau_j \in F_k$, then the precedence constraint is already enforced by the deadline assignment given in Eq. (5.9). Hence, it is sufficient to make sure that τ_i is not activated earlier than τ_j. In general, we must ensure that

$$a_i \geq \max_{\tau_j \to \tau_i, \tau_j \in F_k} \{a_j\} \overset{\text{def}}{=} a_i^{\text{prec}}. \tag{5.10}$$

On the other hand, if $\tau_j \notin F_k$, we cannot assume that τ_j will be allocated on the same physical core as τ_i, thus we do not know its precise finishing time. Hence, τ_i cannot be activated before τ_j deadline, that is

$$a_i \geq \max_{\tau_j \to \tau_i, \tau_j \notin F_k} \{d_j\} \overset{\text{def}}{=} d_i^{\text{prec}}. \tag{5.11}$$

In general, a_i must satisfy both (5.11) and (5.10). Moreover a_i should be as early as possible so that the resulting demand bound function is minimized [BHR90]. Hence, we set

$$a_i = \max\{a_i^{\text{prec}}, d_i^{\text{prec}}\}. \tag{5.12}$$

The algorithm starts by assigning activation times to root nodes, i.e., tasks without predecessors. For such tasks, the activation time is set equal to the application activation time that we can assume to be zero, without loss of generality. Then, the algorithm proceeds by assigning activation times to a task for which all predecessors have been considered. Figure 5.7 illustrates the pseudo-code of the algorithm.

Indeed, the transformation algorithm proposed by Chetto, Silly, and Bouchentouf was designed to guarantee the precedence constraints, regardless of the processor demand. In fact it assigns deadlines as late as possible. However activations may coincide with some deadline as well. If an activation is too close to the corresponding deadline, then the demand bound function can become very large. To address this issue, in this work we propose an alternative deadline assignment that

```
ASSIGN ACTIVATION TIMES($\tau$)
1  for all (nodes without predecessors) set $a_i = 0$;
2  while (there exist nodes not set) {
3    select a task $\tau_k$ with all predecessors modified;
4    set $a_i = \max\{a_i^{\text{prec}}, d_i^{\text{prec}}\}$
5  }
```

Figure 5.7: The activation assignment algorithm.

reduces the processor demand of the flow by distributing tasks deadlines more uniformly along the time line. If C^p is the computation time of a critical path and U^p is defined as

$$U^p = \frac{C^p}{D}$$

we propose to assign task deadlines as follows:

$$d_i = \min_{j:\tau_i \to \tau_j} (d_j - C_j/U^p) \tag{5.13}$$

instead of according to Eq. (5.9).

The following lemma shows that such a deadline assignment is sound, in the sense that all relative deadlines are greater than the cumulative computation times of the preceding tasks in a path.

Lemma 1. *If each task τ_i of a path P is assigned a relative deadline*

$$d_i = \min_{j:\tau_i \to \tau_j} (d_j - C_j/U^p)$$

where $U^p = C^p/D$, then it is guaranteed that all the tasks in P have relative deadlines greater than the cumulative execution time of the preceding tasks, that is

$$d_i \geq \sum_{\tau_k \in P, \tau_k \prec \tau_i} C_k.$$

Proof. Given any node τ_i, let τ_{i+1}, $\tau_{i+2}, \ldots, \tau_L$ be the sequence of successors of τ_i such that τ_L is a leaf node (hence $d_L = D$) and

$$\forall j = i, \ldots, L-1 \quad d_j = d_{j+1} - C_{j+1}/U^p.$$

Then we have:

$$d_i = d_{i+1} - \frac{C_{i+1}}{U^p} = D - \frac{\sum_{j=i+1}^{L} C_j}{U^p}.$$

If P is a path including τ_i, $\tau_{i+1}, \ldots, \tau_L$, we can write:

$$d_i = D - \frac{C(P) - \sum_{j=1}^{i} C_j}{U^p} = D - \frac{C(P)}{U^p} + \frac{\sum_{j=1}^{i} C_j}{U^p}$$

and since $U^p = C^p/D$ we have

$$d_i = D - \frac{C(P)}{C^p}D + \frac{\sum_{j=1}^{i} C_j}{U^p}.$$

Since $C(P) \leq C^p$ for any P, and $C^p \leq D$, we have:

$$d_i \geq \frac{\sum_{j=1}^{i} C_j}{U^p} \geq \sum_{j=1}^{i} C_j.$$

Thus, the lemma follows. □

For the application shown in Figure 5.1, we have that:

$$\begin{aligned} a_1 &= 0 \\ D &= T = 20 \\ C^p &= 10 \\ C^s &= 15 \\ U^p &= \frac{C^p}{D} = 0.5 \end{aligned}$$

Hence, the proposed transformation algorithm (Eq. (5.13)) produces the following deadline assignment:

$$\begin{aligned} d_3 &= 20 \\ d_5 &= 20 \\ d_2 &= \min(20 - 5/0.5, 20 - 3/0.5) = \min(10, 14) = 10 \\ d_4 &= 20 - 3/0.5 = 14 \\ d_1 &= \min(10 - 1/0.5, 14 - 2/0.5) = \min(8, 10) = 8. \end{aligned}$$

If, for example, we select the flows $F_1 = \{\tau_1, \tau_2, \tau_3\}$ and $F_2 = \{\tau_4, \tau_5\}$, the activation times result to be:

$$\begin{aligned} a_1 &= 0 \\ a_2 &= 0 \\ a_3 &= 0 \\ a_4 &= d_1 = 8 \\ a_5 &= \max(a_4, d_2) = \max(8, 10) = 10 \end{aligned}$$

The demand bound functions of the two flows are reported in Figure 5.8 and Figure 5.9, respectively.

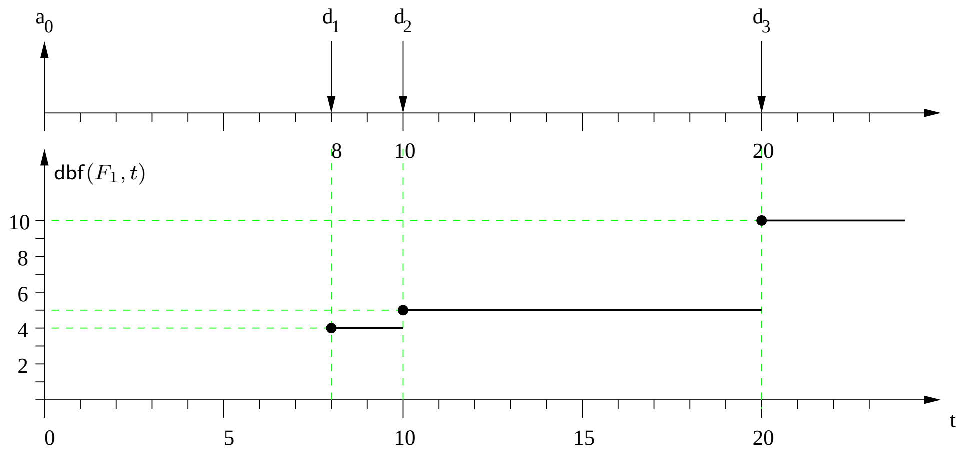


Figure 5.8: Demand bound function of flow F_1.

5.4.2 Bandwidth requirements for a flow

Once activation times and deadlines have been set for all tasks, each flow can be independently executed on different virtual processors under EDF, in isolation, ensuring that precedence constraints are met.

To determine the reservation parameters that guarantee the feasibility of the schedule, we need to characterize the computational requirement of each flow. By using the demand bound function defined in Equation (5.2) we have that a flow F is schedulable on the virtual processor VP characterized by bandwidth α and delay Δ if and only if:

$$\forall t \geq 0 \quad \mathsf{dbf}(F, t) \leq \alpha(t - \Delta)_0. \tag{5.14}$$

Now the problem is to select the (α, Δ) parameters among all possible pairs that satisfy Eq. (5.14). We propose to select the pair that minimizes the bandwidth B used by the virtual processor, as given by Eq. (5.7), which accounts for the cost of the server overhead. Hence, the best $(\alpha_, \Delta)$ pair is the solution of the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & \alpha + \varepsilon \frac{1 - \alpha}{\Delta} \\ \text{subject to} \quad & \mathsf{dbf}(F, t) \leq \alpha(t - \Delta)_0, \quad \forall t \geq 0 \end{aligned} \tag{5.15}$$

with $\varepsilon = 2\sigma$.

This problem have a very efficient solution that exploits the convexity of the domain and the quasiconvexity of the cost function (see Appendix A for the proof). More details of how to obtain the optimal (α, Δ) pair for a flow can be found in [BBW09].

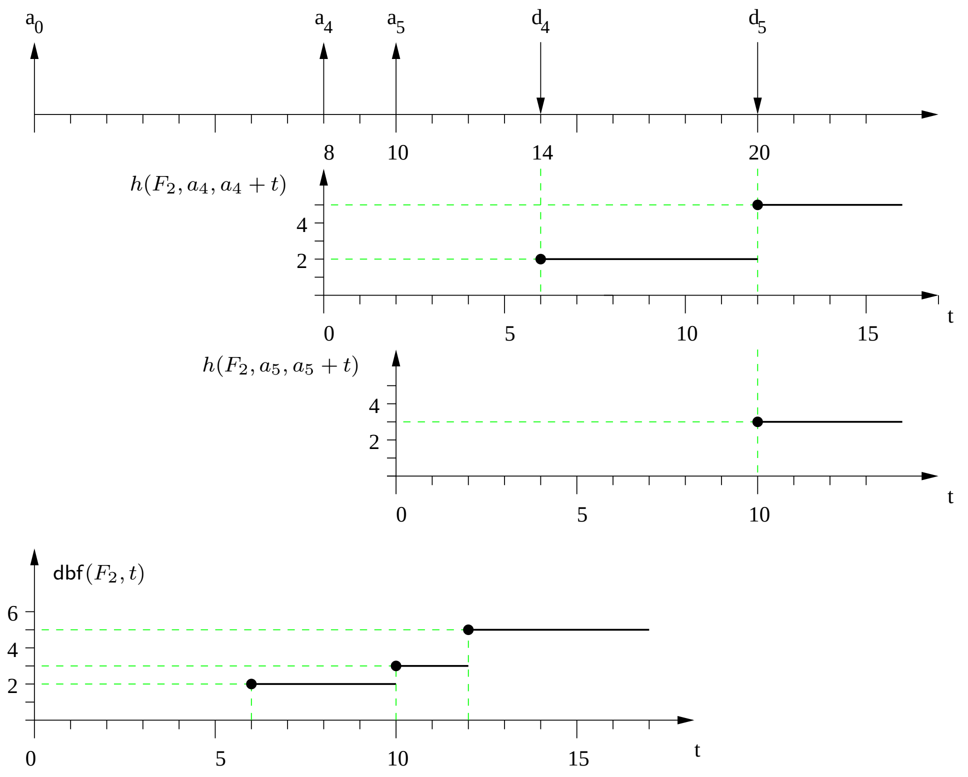


Figure 5.9: Demand bound function of flow F_2.

5.4.3 The branch and bound algorithm

This section illustrates the algorithm used for selecting the best partition of the application into flows. Two different objectives have been considered in the optimization procedure.

As a first optimization goal, we considered minimizing the overall bandwidth requirement of the selected flows, that is

$$B = \sum_{k=1}^{m} B_k = \sum_{k=1}^{m} \left(\alpha_k + 2\sigma \frac{1-\alpha_k}{\Delta_k} \right). \tag{5.16}$$

Clearly, the number m of flows has to be determined as well.

As a second optimization goal, we considered minimizing the maximum degree of parallelism, defined as

$$\beta = \max_{k=1,\dots,m} \frac{\sum_{i=k}^{m} B_i}{B_k}. \tag{5.17}$$

The selection of this metric is inspired by the global EDF test on uniform multiprocessors [FGB01]. In fact, in uniform multiprocessor scheduling, if $B_1 \geq B_2 \geq \ldots \geq B_m$ are the speeds of the processors, a platform with a low value of β has higher chance to schedule tasks due to the lower degree of fragmentation of the overall computing capacity[1].

To show the benefit of adopting the cost of Equation (5.17), let us consider a virtual platform with m identical processors, each providing $B_k = B/m$. While the cost according to Eq. (5.16) is B, hence independent of the number of virtual processors, the cost according to Eq. (5.17) is m. It follows that the minimization of β leads to the reduction of number of flows in which the application is partitioned. Nonetheless, the minimization of β also implicitly implies the selection of a partitioning with low overall bandwidth requirement B. In fact we have that

$$B = \sum_{i=1}^{m} B_i \leq \frac{\sum_{i=1}^{m} B_i}{B_1} \leq \max_{k=1,\dots,m} \frac{\sum_{i=k}^{m} B_i}{B_k} = \beta.$$

Hence β is also an upper bound of the overall bandwidth B, and a minimization of β leads indirectly to the selection of a low value of B as well. Later in Figure 5.14 we will show that in our experiments the difference between β and B is very small.

The search for the optimal flow partition is approached by using a branch and bound algorithm, which explores the possible partitions by generating a search tree as illustrated in Figure 5.10.

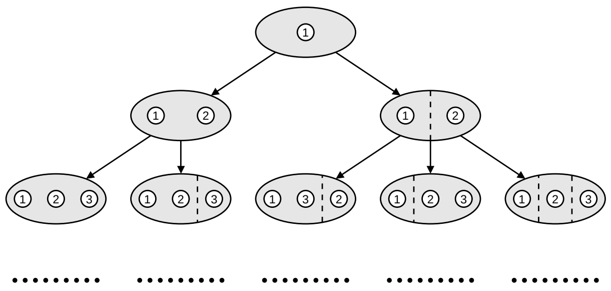


Figure 5.10: The search tree.

At the root level (level 1), task τ_1 is associated with flow F_1. At level 2, τ_2 is assigned either to the same flow F_1 (left branch) or to a newly created flow F_2 (right branch). In general, at each level i, task τ_i is assigned either to one of the existing flows, or to a new created flow. Hence, the depth of the tree is equal to the number n of tasks composing the application, whereas the number of leaves of the tree is equal to the number of all the possible partitions of a set of n members, given by the Bell Number b_n [Rot64], recursively computed by

$$b_{n+1} = \sum_{k=0}^{n} \binom{n}{k} b_k = \sum_{k=0}^{n} \frac{n!}{k!\,(n-k)!}\, b_k. \tag{5.18}$$

To reduce the average complexity of the search, we use some pruning conditions to cut unfeasible and redundant branches for improving the runtime behavior of the algorithm.

[1]Notice that in [FGB01] the authors use $\lambda = \beta - 1$ to express the parallelism of the platform.

We first observe that if, at some node, there is a flow F_k with bandwidth greater than one

$$B_k \geq \sum_{\tau_i \in F_k} \frac{C_i}{T} > 1 \tag{5.19}$$

then the schedule of the tasks in that flow is unfeasible, since

$$\sum_{\tau_i \in F_k} C_i > T \geq D. \tag{5.20}$$

Hence, whenever a node has a flow with bandwidth greater than one, we can prune the whole subtree, since no feasible partitioning can be found in the subtree. Moreover, the pruning efficiency can be further improved by allocating tasks by decreasing computation times, because this order allows pruning a subtree satisfying Eq. (5.19) at the highest possible level.

The following lemma provides a lower bound on the number of flows in any feasible partition:

Lemma 2. *In any feasible partitioning, the number of flows satisfies*

$$m \geq \left\lceil \frac{C^s}{D} \right\rceil. \tag{5.21}$$

Proof. In any feasible partitioning $\{F_1, \ldots, F_m\}$, we have

$$\frac{C(F_k)}{D} \leq 1. \tag{5.22}$$

Adding equations (5.22) for all the flows, we have

$$\frac{\sum_k C(F_k)}{D} = \frac{C^s}{D} \leq m$$

And since m is integer,

$$m \geq \left\lceil \frac{C^s}{D} \right\rceil.$$

□

Nonetheless, much of the complexity of the algorithm lies in the horizontal expansion of the tree: in fact, the search tree keeps adding possible new flows (at the rightmost branch) even when the number of flows is higher than the parallelism that can be possibly exploited by the application. Hence, we prune a subtree when the number of flows exceeds a given bound m_{max}. A tight value of m_{max} is not easy to find, hence we adopted the following heuristic value:

$$m_{\mathsf{max}} = \left\lceil \delta \frac{C^s}{D} \right\rceil \tag{5.23}$$

where $\delta \geq 1$ is a parameter for tuning the size of the search tree. A value of δ close to one allows a significant improvement in terms of execution time, but at the price of losing optimality. Larger values of δ permit reaching optimality with reasonable execution times. As illustrated in the next section, our simulation results show that the optimal solution is often achieved with $\delta \leq 2$.

5.5 Experimental results

To illustrate the effectiveness of the proposed search algorithm, in this section we present a number of experiments aimed at comparing the effectiveness of the produced solution (in terms of number of flows and required bandwidth), the efficacy of the pruning rules (in terms of reducing the number of steps) , and the advantage of the use in control applications.

5.5.1 Effectiveness evaluation

In a first experiment, we considered the application shown in Figure 5.5, consisting of $n = 9$ tasks with computation times $C_1 = 2$, $C_2 = 3$, $C_3 = 5$, $C_4 = 3$, $C_5 = 4$, $C_6 = 3$, $C_7 = 6$, $C_8 = 5$, and $C_9 = 6$. From the DAG of the application, it results that the sequential execution time is $C^s = 37$ and the parallel execution time is $C^p = 12$, corresponding to the critical path $P = \{\tau_6, \tau_8, \tau_5\}$. Notice that the ratio $\pi = C^s/C^p$ provides an indication of the maximum level of parallelism of the application. In this example, we have $\pi \simeq 3.08$. Clearly, when the application deadline D is less than C^p, the schedule is infeasible on any number of cores, whereas when $D = C^p = 12$, the number of cores cannot be less than 4 (see Lemma 2).

Figure 5.11 reports the number of flows of the optimal partition found by the algorithm as a function of the application deadline D (ranging from C^p to C^s), using the first optimization goal expressed by Eq. (5.16). The dashed line represents the theoretical bound given by Equation (5.21). Notice that the number of flows is equal to 4 when $D = C^p$ (meaning that the application needs 4 VP's to meet its deadline) and drops to 1 for $D \geq C^s$, meaning that the application can be completely hosted by 1 VP.

The corresponding bandwidth B acquired by the optimal partition (including the context switch overhead σ) is shown in Figure 5.12, for different value of σ. The figure also reports the minimum theoretical bound C^s/D (without overhead) and the worst-case bandwidth obtained by selecting one flow per task. Notice that the solution found by the algorithm is always very close to the ideal one and significantly better than the worst-case curve.

Considering the second optimization goal, expressed with the cost function reported by Eq. (5.17), Figure 5.13 reports the optimal β achieved by the search algorithm, as a function of the application deadline, for different values of σ.

The difference between the bandwidth achieved by the second and the first optimization goal is reported in Figure 5.14. Notice that, such a difference is never less than 0, since the first optimization goal aims at minimizing the total bandwidth. However, the bandwidth loss resulted from the second method was never larger than 0.12.

To test the runtime behavior of the search algorithm and the efficiency of the pruning rule, we ran another experiment with a fully parallel application (i.e., no precedence relations) with random computation times, generated with uniform dis-

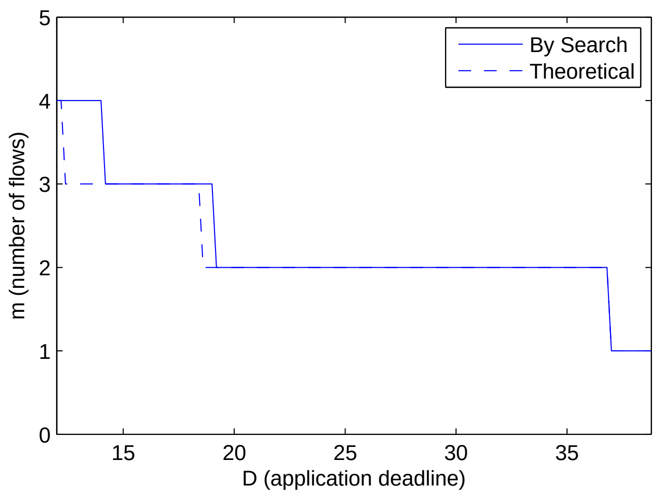


Figure 5.11: Number of flows as a function of the application deadline.

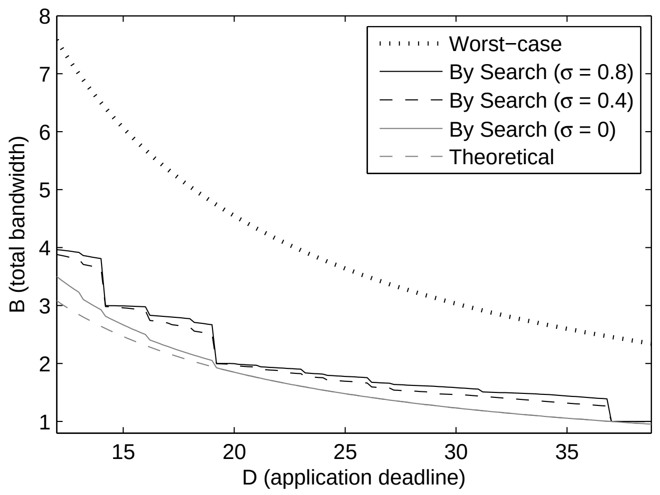


Figure 5.12: Total bandwidth as a function of the application deadline.

tribution in [1,10]. The application deadline was set between C^p and C^s, with a value $D = (C^p + C^s)/2$. The runtime behavior of the algorithm was monitored by counting the number of steps for reaching a solution, as a function of the number of tasks, for different values of the pruning parameter δ. The results of this experiment are shown in Figure 5.15, which clearly shows that a considerable amount of steps are saved when small values of δ are used. It is worth mentioning that using a small value of δ results in negligible bandwidth loss. Intuitively, this can be justified by considering that a high number of flows often requires a high total B.

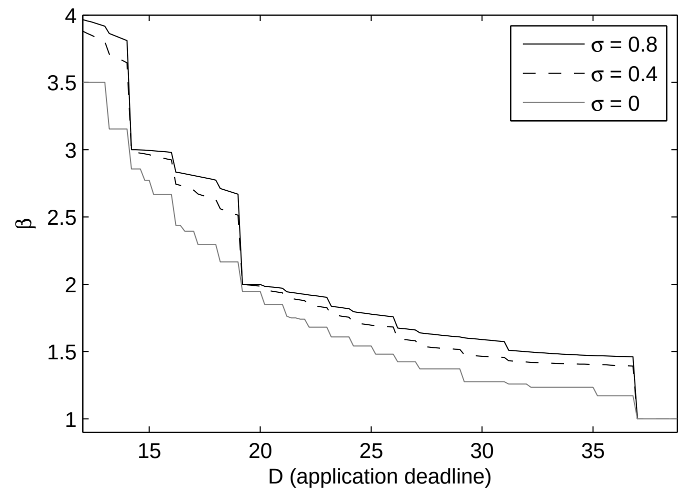


Figure 5.13: β as a function of the application deadline.

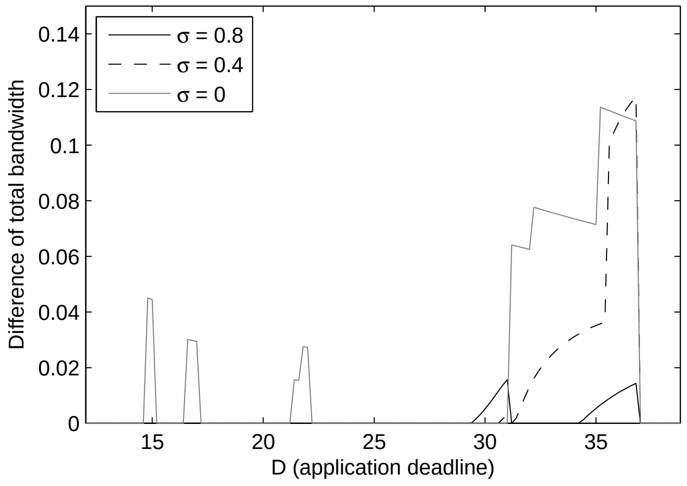


Figure 5.14: Bandwidth loss resulted from minimizing β.

5.5.2 A Control Example

The considered controlled plant is a ball-and-beam system, where the plate is tilted around two axes (X-axis and Y-axis) that are mutually perpendicular. On each axis, the rotation of the plate around that axis is actuated by a servo motor. Therefore, it can be viewed as a ball-and-beam system on each axis, which gives the possible parallelism of the controller. Moreover, according to works on task splitting [Cer99], controller can be in general split into two parts: *Calculate Output* and *Update State*. The *Calculate Output* part takes charge of producing the control signal, while the *Update State* part updates the states of the controller and makes

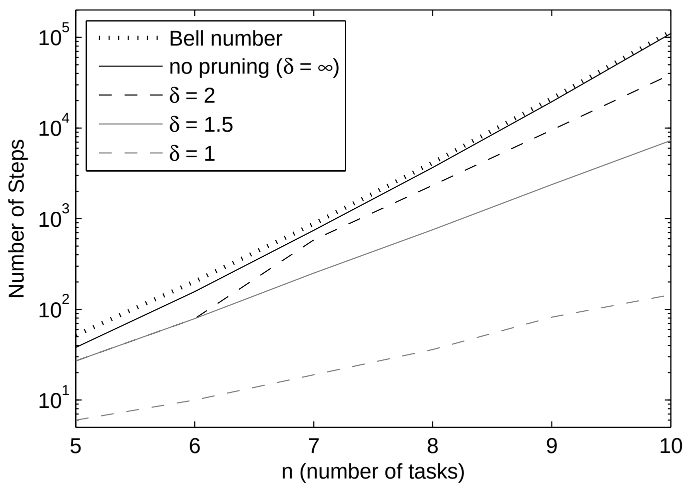


Figure 5.15: Runtime of the algorithm as a function of n.

any other operation. Hence, further parallelism can be exploited.

The state-space model of each ball-and-beam plant is given by

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ K \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x + 0.1e$$

where $K = 20$ is the factor due to physical modeling, v and e are Gaussian white-noise process with zero mean and unit variance. One LQG controller is designed for each ball-and-beam system, according to the associated cost function:

$$J = E \lim_{t_p \to \infty} \frac{1}{t_p} \int_0^{t_p} \left(x^T \begin{bmatrix} 10 & 0 \\ 0 & 0 \end{bmatrix} x + u^2 \right) dt \tag{5.24}$$

The control application Γ_{bap} [2] is described with the precedence graph in Figure 5.16. The tasks *Cal-X* and *Cal-Y* denote the *Calculate Part* of the algorithm for each ball-and-beam control, while tasks *Up-X* and *Up-Y* denote the *Update State* part to update the controller states. It is assumed that the application has other 2 objects: data logging and LCD monitoring, whose software code can both be parallelized, resulting in tasks *Log1-4* and *LCD1-4*, respectively. The indices and WCETs of all the tasks are shown in Table 5.1. The sequential execution time C^s is $36ms$, and the parallel execution time C^p is $16ms$.

Assume there is another task τ_{13} running on the same platform, with a predicted WCET $2ms$ and period $8ms$, giving the required bandwidth of $U_{13} = 0.25$. To show the benefits of using resource reservation, τ_{13} may misbehave in its execution time after $t = 25sec$ during the simulation, giving a possible execution time in

[2]Abbreviation bap is short for **b**all **a**nd **p**late.

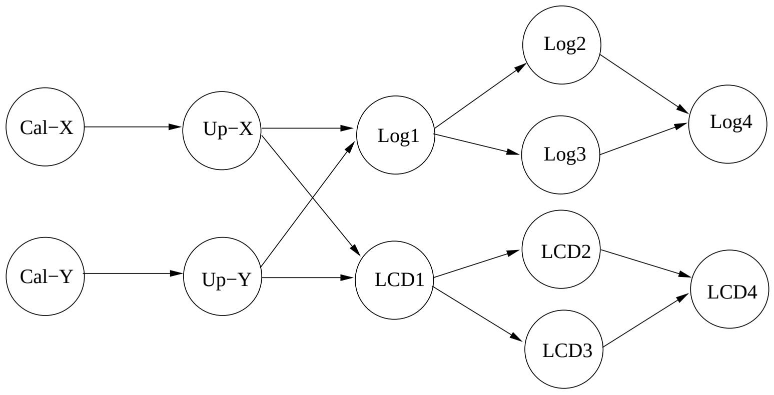


Figure 5.16: The precedence graph of the ball-and-plate control application.

Table 5.1: The tasks in the ball-and-plate application.

Task	Label	WCET(ms)	Task	Label	WCET(ms)
τ_1	Cal-X	2	τ_2	Up-X	3
τ_3	Cal-X	2	τ_4	Up-X	3
τ_5	Log1	3	τ_6	Log2	2
τ_7	Log3	4	τ_8	Log4	4
τ_9	LCD1	4	τ_{10}	LCD2	3
τ_{11}	LCD3	5	τ_{12}	LCD4	1

$[2,\ 4]ms$. This disturbance in the real-time scheduling system will cause the off-line guarantee of the resource given to Γ_{bap} become insufficient to fulfill its timing constraints, if resource reservation mechanism is not utilized.

Three experiments are performed, whose different scenarios are described as below:

- **Uniprocessor** On uniprocessor platform, Γ_{bap} has to be executed sequentially, in the topological order of the nodes in the precedence graph. To output the control signals as soon as they are ready, the order shown in Figure 5.17 is assumed. Task *Cal-X* and *Cal-Y* are firstly executed, and the calculated control signal for each axis is output when the corresponding task is finished. Then *Up-X* and *Up-Y* are executed, and task *LCD1-4* and *Log1-4* will later run in topological order in accordance to their precedence relations.

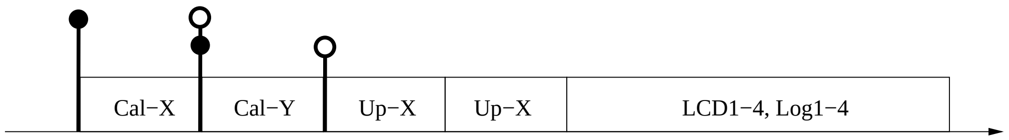


Figure 5.17: Sequential execution of the ball-and-plate application on uniprocessor.

Knowing the required bandwidth for τ_{13} is $U_{13} = 0.25$, the available CPU bandwidth left for the ball-and-plate application is $U_{\Gamma_{bap}} = 1 - U_{13} = 0.75$, if EDF scheduling policy is used. That means, the maximum achievable sampling rate for Γ_{bap} is

$$f_{\Gamma_{bap}} = \frac{U_{\Gamma_{bap}}}{C^s} = \frac{0.75}{36} = 0.02083Hz$$

which is equivalent to a sampling period of $48ms$.

- **Uniprocessor + Disturbance** All the setup is precisely the same as the first simulation, except for that τ_{13} will misbehave after $t = 25sec$.
- **Multiprocessor + Disturbance** The proposed method in the chapter is used. The deadline and period of Γ_{bap} are chosen to be equally $25ms$. Assuming the context switch overhead $\sigma = 0.1ms$, the optimal partitioning with respect to minimizing the maximum parallelism is shown in Figure 5.18. The

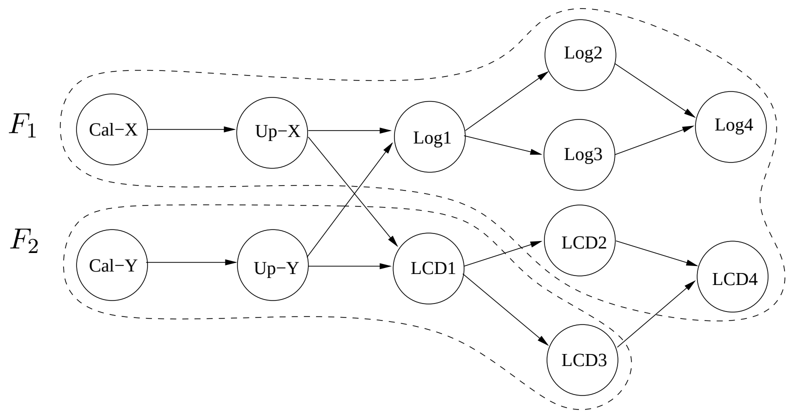


Figure 5.18: The optimal partition of Γ_{bap}.

results of deadline and arrival time assignment of tasks are summarized in Appendix B. The (α, Δ) pair of Flow F_1 is (0.997037, 0.136985) and the one of Flow F_2 is (0.727272, 0.375000), which leads to CBS server parameters (Q, P) of (23.047, 23.116) and (0.500, 0.688), respectively. A multiprocessor platform with two symmetric CPUs is used, where F_1 is assigned to CPU1, and F_2 is assigned to CPU2.

Same disturbance from τ_{13} take place, but this time τ_{13} is put in a CBS with $Q_s = 2ms$ and $P_s = 8ms$, which is then allocated to run on CPU2. Therefore, total utilization of CPU2 is

$$U_{cpu2} = \frac{0.5}{0.688} + \frac{0.002}{0.008} = 0.977 < 1$$

which is schedulable using EDF.

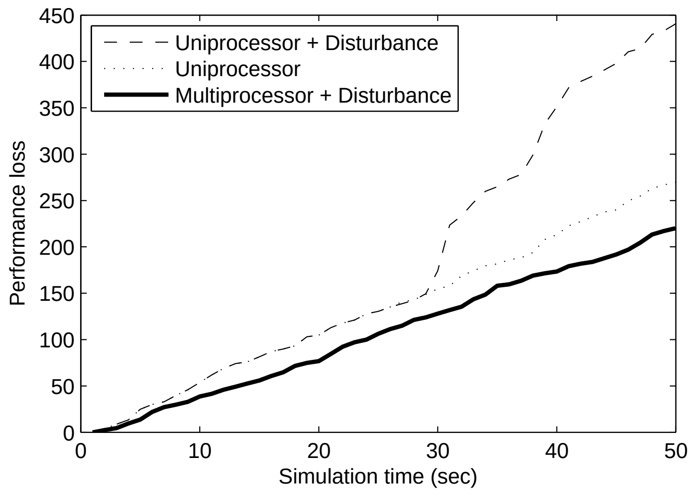


Figure 5.19: Control performance comparison.

All three experiments run in TrueTime 2.0 with $50sec$, and the control performance is evaluated using Eq. (5.24). The results is shown in Figure 5.19. It clearly shows that due to the disturbance brought in by task τ_{13}, the performance significantly degrades after $t = 25sec$. However, the resource reservation mechanism used in the 3rd experiment manages to isolate the incorrect timing behavior of τ_{13}. A more interesting fact reported in the figure is that the curve of the 3rd experiment is lower than the one from the 1st experiment, which means utilizing the proposed method to exploit the parallelism in the controller software enables the real-time control system to have a faster sampling rate and short end-to-end latency, hence resulting in better performance.

Notice that, in the experiment only simple state feedback controllers are used for the *Calculate Output* parts *Cal-X* and *Cal-Y*. We believe that the control performance improvement should be largely increased if more sophisticated controllers are involved so that our proposed method can take advantage of possibly more parallelism in the control algorithm.

5.6 Conclusion

The chapter presented a general methodology for allocating a parallel real-time application to a multi-core platform in a way that is independent of the number of physical cores available in the hardware architecture. Independency is achieved through the concept of virtual processor, which abstracts a resource reservation mechanism by means of two parameters, α (the bandwidth) and Δ (the maximum service delay).

An algorithm was developed to automatically partition the application into flows, meeting the specified timing constraints and minimizing either the overall required bandwidth B or the maximum degree of parallelism β. The computational requirements of each flow were derived through the processor demand criterion, after defining intermediate activation times and deadlines for each task, properly selected to satisfy precedence relations and timing constraints.

The employment of the method in control scenario was illustrated by an example with a ball-and-plate system. The experimental results show that the multiprocessor platform allows the control application to execute with certain degree of parallelism such that faster sampling and shorter end-to-end latency is achieved. The control performance is improved even when disturbance exist in the scheduling system, owning to the resource reservation.

Chapter 6

Conclusion

6.1 Summary

In this dissertation, the analysis and design for real-time control systems in resource-constrained platform is discussed. After reviewing the background knowledge and existing state-of-the-art techniques, several new methods have been presented to enhance the current technology for real-time controller design.

Limited-preemption scheduling is investigated for its advantages in improving task responsiveness and control performance. It has been shown in previous research that using non-preemptive EDF scheduling, the input-output delays of control tasks are minimized to their WCET, thus improving the control performance. However, the achievement of better control performance is paid by impairing the feasibility of the task set, because of the property of the non-preemptive scheduling. In other words, non-preemptive EDF forces a reduction of the total resource utilization. Increasing the responsiveness of a control task results in smaller delay and jitter, thus improving its control performance. A trade-off can be made by using limited-preemption, where the feasibility of the task set is maintained while the responsiveness of certain tasks can be increased. By selectively applying limited-preemption to control tasks in dynamic priority systems (EDF), their response times, as well as input-output delay/jitter, are reduced and the corresponding control performance are improved.

In resource-constrained systems, the interference generated by the concurrent execution of multiple controller tasks leads to extra delay and jitter, which degrade control performance and may even jeopardize the stability of the controlled system. A general methodology has been presented which integrates control issues and real-time schedulability analysis to improve the control performance in embedded systems with time and resource constraints. The performance increase is achieved by properly selecting task periods and deadlines under feasibility constraints.

A full exploitation of the computational power available in a multi-core platform requires the software to be specified in terms of parallel execution flows. At the same time, modern embedded systems often consist of more parallel applications with timing requirements, concurrently executing on the same platform and sharing common resources. To prevent reciprocal interference among critical activities, a resource reservation mechanism is highly desired in the kernel to achieve temporal isolation. In this dissertation, a general methodology is proposed for partitioning the total computing power available on a multi-core platform into a set of virtual processors, which provide a powerful abstraction to allocate applications independently of the physical platform. The application, described as a set of tasks with precedence relations expressed by a directed acyclic graph, is automatically partitioned into a set of subgraphs that are selected to minimize either the overall bandwidth consumption or the maximum degree of parallelism. The effectiveness of the proposed method for real-time control systems is illustrated by experiment on a ball-and-plate control system.

6.2 Future Work

The maturity of the techniques in designing real-time control systems requires continuous attention and efforts from both control community and scheduling community. Regarding the presented methods in this dissertation, several extensions are possible.

- **Utilization of limited preemption.** In the presented method in Chapter 3, the non-preemptive chunk is proposed to be placed at the end of each control task, which lead to a significant decrease of input-output delay. However, it would be interesting to investigate the placement of the NP chunk when a control task is divided into separated parts like *Calculate Output* and *Update States*. It can be envisioned that if the *Calculate Output* part is completely non-preemptive, then the input-output delay is minimized, which could lead to even better performance.

- **Parameter Selection.** The proposed method for period and deadline selection in Chapter 4 uses a 2-step procedure, which basically can not give the optimal solution. To reach the optimal control performance, the feasible region of both period and deadline is required. However, the finding of such region is not trivial and has only been studied with one single task in [BRC09].

- **Multiprocessor platform.** The investigation of real-time control design problem on multiprocessor platform has just received attention recently. It would be interesting to see what is the effect on control performance from different scheduling policies on multiprocessor platform. For instance, would global scheduling policies benefit the real-time control systems, or would partitioned scheduling policies be preferable? Regarding to the proposed method in Chapter 5, a planned work is to investigate the relation between (α, Δ) parameter and the control performance, and to involve the performance in the searching of the optimal partitioning.

Appendix A

Proof of the quasiconvexity of optimization problem (5.15)

Since the dbf is a step function, it is enough to ensure that Eq. (5.14) is verified at all the steps. If schedP is the set of time instants where the dbf has a step, then Eq. (5.14) can be equivalently ensured by

$$\forall t \in \mathsf{schedP} \quad \mathsf{dbf}(t) \leq \alpha(t - \Delta)_0.$$

Accordingly, the minimization problem (5.15) can the be simplified to

$$\begin{array}{rl} \text{minimize} & \alpha + \varepsilon \dfrac{1-\alpha}{\Delta} \\ \text{subject to} & \mathsf{dbf}(F, t) \leq \alpha(t-\Delta)_0, \quad \forall t \in \mathsf{schedP} \end{array} \tag{A.1}$$

Such an optimization problem be solved very efficiently, thanks to the good properties of both the constraint and the cost function. We first prove the convexity of the constraint.

Lemma 3. *Given $t, w > 0$, let $D(t, w)$ be defined as*

$$D(t, w) = \{(\alpha, \Delta) \in \mathbb{R}^2 : \alpha(t - \Delta) \geq w, \alpha \geq 0\}$$

then $D(t, w)$ is convex.

Proof. We start observing that

$$\alpha(t - \Delta) \geq w \geq 0 \;\Rightarrow\; t - \Delta \geq 0 \tag{A.2}$$

because $\alpha \geq 0$. To prove the convexity of $D(t, w)$ we use the property that

$$\{(x, y) : f(x) \leq y\} \text{ is convex} \Leftrightarrow \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \geq 0 \tag{A.3}$$

In fact we have

$$D(t, w) = \left\{ \frac{w}{t - \Delta} \leq \alpha \right\}$$

Now

$$\frac{\mathrm{d}^2}{\mathrm{d}\Delta^2}\frac{w}{t-\Delta} = \frac{2w}{(t-\Delta)^3} \geq 0$$

because of Eq. (A.2). Hence from the property of Eq. (A.3), the Lemma follows. □

Figure A.1 shows examples of the domains $D(t, w)$.

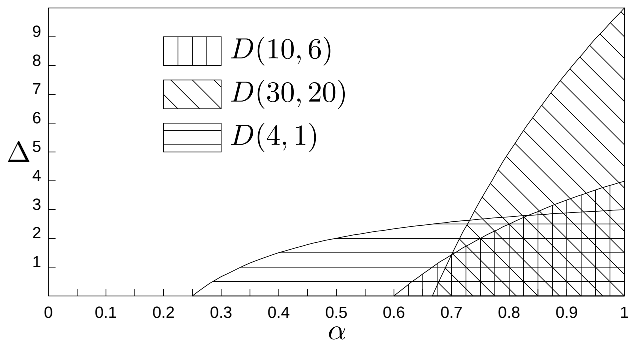


Figure A.1: Examples of the regions $D(t, w)$.

Regarding the properties of the cost function, we first recall the following definition.

Definition 5 (Section 3.4 in [BV04]). *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is called* quasiconvex *if its domain and all its sublevel sets* $S_B = \{x \in \mathbf{dom}f : f(x) \leq B\}$*, for* $B \in \mathbb{R}$*, are convex.*

Notice that convexity implies quasiconvexity, but the viceversa is not true [BV04]. We then have the following result.

Lemma 4. *The function* $g : [0, 1] \times (0, +\infty) \to \mathbb{R}$

$$g(\alpha, \Delta) = \alpha + \varepsilon\frac{1-\alpha}{\Delta}$$

is quasiconvex.

Proof. We first notice that the domain of g, that is $G = [0, 1] \times (0, +\infty)$ is convex. From the definition of quasiconvexity we have to prove that all the level sets

$$S_B = \left\{(\alpha, \Delta) \in G : \alpha + \varepsilon\frac{1-\alpha}{\Delta} \leq B\right\} \tag{A.4}$$

are convex (see Figure A.2 for graphical representation). Since B is interpreted as the overall bandwidth used by the reservation, we only need to prove this for $B \leq 1$. Since $B \geq \alpha$ and $\Delta \geq 0$, we have that:

$$\alpha + \varepsilon\frac{1-\alpha}{\Delta} \leq B \quad \Leftrightarrow \quad \varepsilon\frac{1-\alpha}{B-\alpha} \leq \Delta$$

and from the property of Eq. (A.3),

$$\left\{(\alpha,\Delta) : \varepsilon\frac{1-\alpha}{B-\alpha} \leq \Delta\right\} \Leftrightarrow \frac{\mathrm{d}^2}{\mathrm{d}\alpha^2}\frac{1-\alpha}{B-\alpha} \geq 0$$

since $k > 0$.

We have

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{1-\alpha}{B-\alpha} = \frac{-1(B-\alpha)+(1-\alpha)}{(B-\alpha)^2} = \frac{1-v}{(B-\alpha)^2}$$
$$\frac{\mathrm{d}^2}{\mathrm{d}\alpha^2}\frac{1-\alpha}{B-\alpha} = 2\frac{1-B}{(B-\alpha)^3}$$

that is greater than or equal to zero, because $B \leq 1$ and $\alpha \leq B$. This proves the convexity of the level sets S_B and the quasiconvexity of g as required. □

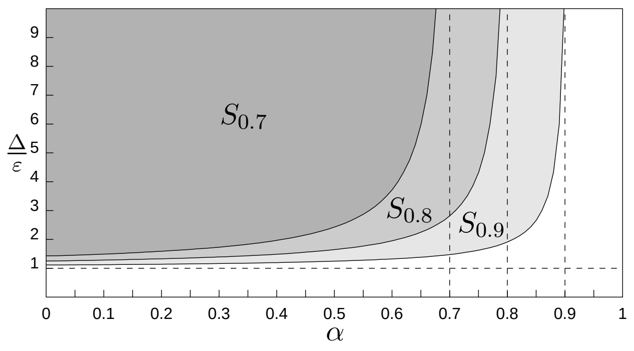


Figure A.2: Examples of the regions S_B.

Since the cost function of the problem of Eq. (A.1) is quasiconvex (from Lemma 4) and the feasibility region is the intersection of convex regions (from Lemma 3), then the minimization problem is a standard quasiconvex optimization problem [BV04], which can be solved very efficiently by standard techniques.

Appendix B

Deadline and activation assignment for Γ_{bap}

Table B.1: The D, a assignment for tasks in Γ_{bap}.

Task	Label	$a_i(ms)$	$D_i(ms)$	Task	Label	$a_i(ms)$	$D_i(ms)$
τ_1	Cal-X	0	3.13	τ_2	Up-X	0	7.81
τ_3	Cal-X	0	3.13	τ_4	Up-X	0	7.81
τ_5	Log1	3.13	12.50	τ_6	Log2	7.81	18.75
τ_7	Log3	7.81	18.75	τ_8	Log4	7.81	25.00
τ_9	LCD1	7.81	15.63	τ_{10}	LCD2	15.63	23.44
τ_{11}	LCD3	7.81	23.44	τ_{12}	LCD4	23.44	25.00

Bibliography

[AB98] Luca Abeni and Giorgio Buttazzo, *Integrating multimedia applications in hard real-time systems*, Proceedings of the 19th IEEE Real-Time Systems Symposium (Madrid, Spain), December 1998, pp. 4–13.

[ACD74] Thomas L. Adam, K. M. Chandy, and J. R. Dickson, *A comparison of list schedules for parallel processing systems*, Communications of the ACM **17** (1974), no. 12, 685–690.

[ÅCES00] Karl-Erik Årzén, Anton Cervin, Johan Eker, and Lui Sha, *An introduction to control and scheduling co-design*, Proceedings of the 39th IEEE Conference on Decision and Control (Sydney, Australia), December 2000.

[AMMMA01] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alarez, *Optimal reward-based scheduling for periodic real-time tasks*, IEEE Transactions on Computers **50** (2001), no. 2, 111–130.

[ÅW97] Karl Johan Åström and Björn Wittenmark, *Computer-controlled systems: Theory and design*, 3rd ed., Prentice Hall, 1997.

[BAL98] Giorgio C. Buttazzo, Luca Abeni, and Giuseppe Lipari, *Elastic task model for adaptive rate control*, Proceedings of the 19th IEEE Real Time System Symposium (Madrid, Spain), December 1998.

[Bar05] Sanjoy Baruah, *The limited-preemption uniprocessor scheduling of sporadic task systems*, Proceedings of the EuroMicro Conference on Real-Time Systems (Palma de Mallorca, Balearic Islands, Spain), IEEE Computer Society Press, July 2005, pp. 137–144.

[BB04] Enrico Bini and Giorgio C. Buttazzo, *Biasing effects in schedulability measures*, Proceedings of the 16th Euromicro Conference on Real-Time Systems (Catania, Italy), jun 2004, pp. 196–203.

[BB09a] Marko Bertogna and Sanjoy Baruah, *Uniprocessor scheduling of sporadic task systems under preemption constraints*, IEEE Transactions on Computers (2009), In submission. Available for download at http://retis.sssup.it/ marko/publi.html.

[BB09b] Enrico Bini and Giorgio Buttazzo, *The space of edf deadlines: the exact region and a convex approximation*, Real-Time Systems **41** (2009), no. 1, 27–51.

[BBB03] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe M. Buttazzo, *Rate monotonic scheduling: The hyperbolic bound*, IEEE Transactions on Computers **52** (2003), no. 7, 933–942.

[BBB09] Enrico Bini, Giorgio C. Buttazzo, and Marko Bertogna, *The multy supply function abstraction for multiprocessors*, Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (Beijing, China), August 2009, pp. 294–302.

[BBGL99] Sanjoy Baruah, Giorgio Buttazzo, Sergey Gorinsky, and Giuseppe Lipari, *Scheduling periodic task systems to minimize output jitter*, Real-Time Computing Systems and Applications, International Workshop on **0** (1999), 62.

[BBW09] Enrico Bini, Giorgio Buttazzo, and Yifan Wu, *Selecting the minimum consumed bandwidth of an EDF task set*, Proceedings of the 2nd Workshop on Compositional Real-Time Systems (Washington DC, USA), December 2009.

[BC06] Sanjoy Baruah and Samarjit Chakraborty, *Schedulability analysis of non-preemptive recurring real-time tasks*, International Workshop on Parallel and Distributed Real-Time Systems (IPDPS) (Rhodes, Greece), April 2006.

[BC07] Giorgio Buttazzo and Anton Cervin, *Comparative assessment and evaluation of jitter control methods*, Proc. of the 15th International Conference on Real-Time and Network Systems (RTNS2007) (March 29-30, 2007), 137–144.

[BC08] Enrico Bini and Anton Cervin, *Delay-aware period assignment in control systems*, Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS 2008) (Barcelona, Spain), December 2008, pp. 291–300.

[BCGM99] Sanjoy K. Baruah, Deji Chen, Sergey Gorinsky, and Aloysius K. Mok, *Generalized multiframe tasks*, Real-Time Systems **17** (1999), no. 1, 5–22.

[BDN05] Enrico Bini and Marco Di Natale, *Optimal task rate selection in fixed priority systems*, Proceedings of the 26th IEEE Real-Time Systems Symposium (Miami (FL), U.S.A.), December 2005, pp. 399–409.

[BDNB06] Enrico Bini, Marco Di Natale, and Giorgio C. Buttazzo, *Sensitivity analysis for fixed-priority real-time systems*, Proceedings of the 18th Euromicro Conference on Real-Time Systems (Dresden, Germany), July 2006.

[BHR90] Sanjoy K. Baruah, Rodney Howell, and Louis Rosier, *Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor*, Real-Time Systems **2** (1990), 301–324.

[Bla96] Jacek Blazewicz, *Scheduling in computer and manufacturing systems*, Springer-Verlag, 1996.

[BLV07] Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh, *Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited*, ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems, 2007, pp. 269–279.

[BMV07] Giorgio Buttazzo, Paul Martí, and Manel Velasco, *Quality-of-control management in overloaded real-time systems*, IEEE Transactions on Computers **56** (2007), no. 2, 253–266.

[BRC06] Patricia Balbastre, Ismael Ripoll, and Alfons Crespo, *Optimal deadline assignment for periodic real-time tasks in dynamic priority systems*, Proceedings of the 18th Euromicro Conference on Real-Time Systems (Dresden, Germany), July 2006, pp. 65–74.

[BRC09] ______, *Period sensitivity analysis and d-p domain feasibility region in dynamic priority systems*, Journal of Systems and Software **82** (2009), no. 7, 1098–1111.

[BRH90] Sanjoy K. Baruah, Louis E. Rosier, and R. R. Howell, *Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor*, Real-Time Systems **2** (1990), 301–324.

[BS88] T.P. Baker and A. Shaw, *The cyclic executive model and ada*, Real-Time Systems Symposium, 1988., Proceedings., Dec 1988, pp. 120–129.

[Bur94] A. Burns, *Preemptive priority based scheduling: An appropriate engineering approach.*, In S. Son, editor, Advances in Real-Time Systems (1994), 225–248.

[But97] Giorgio C. Buttazzo, *Hard real-time computing systems: Predictable scheduling algorithms and applications*, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[BV04] Stephen Boyd and Lieven Vandenberghe, *Convex optimization*, Cambridge University Press, 2004.

[CBLL98] A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari, *Simulation and tracing of hybrid task sets on distributed systems*, Proceedings of the 5th International Conference of Real-Time Computing Systems and Applications, IEEE, October 1998, pp. 303–310.

[CBS00] Marco Caccamo, Giorgio Buttazzo, and Lui Sha, *Elastic feedback control*, Proceedings of the 12th Euromicro Conference on Real-Time Systems (Stockholm, Sweden), June 2000, pp. 121–128.

[CCG08] Sébastien Collette, Liliana Cucu, and Joël Goossens, *Integrating job parallelism in real-time scheduling theory*, Information Processing Letters **106** (2008), no. 5, 180–187.

[CDHB04] Richard C. Dorf and Robert H. Bishop, *Modern control systems*, tenth ed., Prentice Hall, 2004.

[CE03] A. Cervin and J. Eker, *The control server: a computational model for real-time control tasks*, Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on, July 2003, pp. 113–120.

[CEBÅ02] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén, *Feedback-feedforward scheduling of control tasks*, Real-Time Systems **23** (2002), no. 1–2, 25–53.

[Cer99] Anton Cervin, *Improved scheduling of control tasks*, Proceedings of the 11th Euromicro Conference on Real-Time Systems (York, UK), June 1999, pp. 4–10.

[Cer03] ______, *Integrated control and real-time scheduling*, Ph.D. thesis, Department of Automatic Control, Lund University, Sweden, April 2003.

[CHL$^+$03] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén, *How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime*, IEEE Control Systems Magazine **23** (2003), no. 3, 16–30.

[CL06] Anton Cervin and Bo Lincoln, *Jittrbug 1.21 Reference Manual*, February 2006.

[CLE$^+$04] Anton Cervin, Bo Lincoln, Johan Eker, Karl-Erik Årzén, and Giorgio Buttazzo, *The jitter margin and its application in the design of real-time control systems*, Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (Göteborg, Sweden), August 2004, Best paper award.

[CRA99] A. Crespo, I. Ripoll, and P. Albertos, *Reducing delays in rt control: the control action interval*, Proceedings of the 14th IFAC World Congress (Beijing, China), vol. 5, 1999, pp. 257–262.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf, *Dynamic scheduling of real-time tasks under precedence constraints*, Real-Time Systems **2** (1990), no. 3, 181–194.

[CWLH08] Thidapat Chantem, Xiaofeng Wang, M. D. Lemmon, and X. Sharon Hu, *Period and deadline selection for schedulability in real-time systems*, ECRTS ’08: Proceedings of the 2008 Euromicro Conference on Real-Time Systems (2008), 168–177.

[Der74] M. L. Dertouzos, *Control robotics: The procedural control of physical processes*, Information Processing **74** (1974).

[DLCHZ07] Michael D. Lemmon, Thidapat Chantem, Xiaobo Sharon Hu, and Matthew Zyskowski, *On self-triggered full-information h-infinity controllers*, Proceedings 10th International Conference on Hybrid Systems: Computation and Control(HSCC 2007) (Pisa, Italy), apr 2007, pp. 371–384.

[EAL07] Arvind Easwaran, Madhukar Anand, and Insup Lee, *Compositional analysis framework using EDP resource models*, Proceedings of the 28$^{\text{th}}$ IEEE International Real-Time Systems Symposium (Tucson, AZ, USA), 2007, pp. 129–138.

[EHÅ00] Johan Eker, Per Hagander, and Karl-Erik Årzén, *A feedback scheduler for real-time controller tasks*, Control Engineering Practice **8** (2000), no. 12, 1369–1378.

[ERL90] Hesham El-Rewini and T. G. Lewis, *Scheduling parallel program tasks onto arbitrary target machines*, Journal of Parallel and Distributed Computing **9** (1990), no. 2, 138–153.

[FB09] Nathan Fisher and Sanjoy Baruah, *The feasibility of general task systems with precedence constraints on multiprocessor platforms*, Real-Time Systems **41** (2009), no. 1, 1–26.

[FGB01] Shelby Funk, Joël Goossens, and Sanjoy Baruah, *On-line scheduling on uniform multiprocessors*, Proceedings of the 22$^{\text{nd}}$ IEEE Real-Time Systems Symposium (London, United Kingdom), December 2001, pp. 183–192.

[FM02] Xiang Feng and Aloysius K. Mok, *A model of hierarchical real-time virtual resources*, Proceedings of the 23$^{\text{rd}}$ IEEE Real-Time Systems Symposium (Austin, TX, U.S.A.), December 2002, pp. 26–35.

[FPEN94] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini, *Feedback control of dynamic systems*, 3rd ed., Addison-Wesley Publishing Company, Inc., Boston, MA, USA, 1994.

[GAGB01] Paolo Gai, Luca Abeni, Massimiliano Giorgi, and Giorgio Buttazzo, *A new kernel approach for modular real-time systems development*, Proceedings of the 13th Euromicro Conference on Real-Time Systems (Delft, The Nederlands), June 2001, pp. 199–206.

[GH05] José Carlos Palencia Gutiérrez and Michael González Harbour, *Response time analysis of EDF distributed real-time systems*, Journal of Embedded Computing **1** (2005), no. 2, 225–237.

[GJ79] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of np-completeness*, W.H. Freeman and Company, 1979.

[GRS96] Laurent George, Nicolas Rivierre, and Marco Spuri, *Preemptive and non-preemptive real-time uniprocessor scheduling*, Tech. Report RR-2966, INRIA: Institut National de Recherche en Informatique et en Automatique, 1996.

[HB07] Hoai Hoang and Giorgio Buttazzo, *Reducing delay and jitter in software control systems*, Proc. of the 15th International Conference on Real-Time and Network Systems (RTNS2007) (Nancy, France), March 2007.

[HBJK06] Hoai Hoang, Giorgio Buttazzo, Magnus Jonsson, and Stefan Karlsson, *Computing the minimum edf feasible deadline in periodic systems*, Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (Sydney, Australia), August 2006, pp. 125–134.

[HC05] Dan Henriksson and Anton Cervin, *Optimal on-line sampling period assignment for real-time control tasks based on plant state information*, Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005 (Seville, Spain), December 2005.

[HCAÅ06] Dan Henriksson, Anton Cervin, Martin Andersson, and Karl-Erik Årzén, *TrueTime: Simulation of networked computer control systems*, Proceedings of the 2nd IFAC Conference on Analysis and Design of Hybrid Systems (Alghero, Italy), June 2006, Invited talk.

[HHK01] Thomas A. Henzinger, Benjamin Horowitz, and Christoph Meyer Kirsch, *Embedded control systems development with giotto*, Proceedings of The Workshop on Languages, Compilers, and Tools for

Embedded Systems (Snow Bird (UT), U.S.A.), June 2001, pp. 64–72.

[HHK03] T.A. Henzinger, B. Horowitz, and C.M. Kirsch, *Giotto: a time-triggered language for embedded programming*, Proceedings of the IEEE **91** (2003), no. 1, 84–99.

[JA08] Praveen Jayachandran and Tarek Abdelzaher, *Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems*, Proceedings of the 29th IEEE Real-Time Systems Symposium (Barcelona, Spain), December 2008, pp. 259–269.

[JHC07] Erik Johannesson, Toivo Henningsson, and Anton Cervin, *Sporadic control of first-order linear stochastic systems*, Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control (Pisa, Italy), April 2007.

[JP86] Mathai Joseph and Paritosh K. Pandya, *Finding response times in a real-time system*, The Computer Journal **29** (1986), no. 5, 390–395.

[JSM91] K. Jeffay, D. Stanat, and C. Martel, *On non-preemptive scheduling of periodic and sporadic tasks*, Proceedings of the 12th Real-Time Systems Symposium (San Antonio, Texas), IEEE Computer Society Press, December 1991, pp. 129–139.

[Kim98] Byung Kook Kim, *Task scheduling with feedback latency for real-time control systems*, Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (Hiroshima, Japan), October 1998, pp. 37–41.

[kKAA96] Yu kwong Kwok, Ishfaq Ahmad, and Ishfaq Ahmad, *Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors*, IEEE Transactions on Parallel and Distributed Systems **7** (1996), 506–521.

[LA08] Hennadiy Leontyev and James H. Anderson, *A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees*, Proceedings of the 20th Euromicro Conference on Real-Time Systems (Prague, Czech Republic), July 2008, pp. 191–200.

[LB03] Giuseppe Lipari and Enrico Bini, *Resource partitioning among real-time applications*, Proceedings of the 15th Euromicro Conference on Real-Time Systems (Porto, Portugal), July 2003, pp. 151–158.

[LC02] Bo Lincoln and Anton Cervin, *Jitterbug: A tool for analysis of real-time control performance*, Proceedings of the 41^{st} IEEE Conference on Decision and Control (Las Vegas, NV U.S.A.), December 2002.

[LL73] C. L. Liu and James Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the ACM **20** (1973), no. 1, 46–61.

[LM87] Edward Ashford Lee and David G. Messerschmitt, *Static scheduling of synchronous data flow programs for digital signal processing*, IEEE Transactions on Computers **36** (1987), no. 1, 24–35.

[LSA$^+$00] Chenyang Lu, J.A. Stankovic, T.F. Abdelzaher, Gang Tao, S.H. Son, and M. Marley, *Performance specifications and metrics for adaptive real-time systems*, Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE, 2000, pp. 13–23.

[LSTS99] C. Lu, J. Stankovic, G. Tao, and S. H. Son, *Design and evaluation of a feedback control EDF scheduling algorithm*, Proceedings of the Real-Time Systems Symposium (Phoenix, AZ), IEEE Computer Society Press, December 1999.

[LVM08] Camilo Lozoya, Manel Velasco, and Pau Martí, *The one-shot task model for robust real-time embedded control systems*, IEEE Transactions on Industrial Informatics **4** (2008), no. 3, 164–174.

[LW82] Joseph Y.-T. Leung and J. Whitehead, *On the complexity of fixed-priority scheduling of periodic real-time tasks*, Performance Evaluation **2** (1982), no. 4, 237–250.

[Mar02] Pau Martí, *Analysis and design of real-time control systems with varying control timing constraints*, Ph.D. thesis, Automatic Control Department, Technical University of Catalonia, July 2002.

[MFC01] Aloysius K. Mok, Xiang Feng, and Deji Chen, *Resource partition for real-time systems*, Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (Taipei, Taiwan), May 2001, pp. 75–84.

[MFFR01] P. Martí, J.M. Fuertes, G. Fohler, and K. Ramamritham, *Jitter compensation for real-time control systems*, Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE, Dec. 2001, pp. 39–48.

[MLB$^+$04] Pau Martí, Caixue Lin, Scott A. Brandt, Manuel Velasco, and Josep M. Fuertes, *Optimal state feedback based resource allocation for resource-constrained control tasks*, Proceedings of the 25^{th}

IEEE Real-Time Systems Symposium (Lisbon, Portugal), December 2004, pp. 161–172.

[MP05] Aloysius K. Mok and Wing-Chi Poon, *Non-preemptive robustness under reduced system load*, RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2005, pp. 200–209.

[MVFF01a] Pau Martí, Ricard Villà, Josep M. Fuertes, and Gerhard Fohler, *On real-time control tasks schedulability*, European Control Conference (ECC01) (Porto, Portugal), Sept. 2001, pp. 2227–2232.

[MVFF01b] ______, *Stability of on-line compensated real-time scheduled control tasks*, IFAC Conference on New Technologies for Computer Control (Hong Kong), 2001.

[NBW98] Johan Nilsson, Bo Bernhardsson, and Björb Wittenmark, *Stochastic analysis and control of real-time systems with random time delays*, Automatica **34** (1998), no. 1, 57–64.

[OHC07] Martin Ohlin, Dan Henriksson, and Anton Cervin, *TrueTime 1.5—Reference Manual*, January 2007.

[OPRS$^+$06] Clara Otero Pérez, Martijn Rutten, Liesbeth Steffens, Jos van Eijndhoven, and Paul Stravers, *Resource reservations in shared-memory multiprocessor SoCs*, Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices (Book Series Philips Research, ed.), Springer, Netherlands, 2006, pp. 109–137.

[PAC$^+$00] Luigi Palopoli, Luca Abeni, Fabio Conticelli, Marco Di Natale, and Giorgio Buttazzo, *Real-time control system analysis: An integrated approach*, Proceedings of the 25st IEEE Real-Time Systems Symposium (Orlando,Florida,U.S.A.), Dec 2000.

[RGR08] Ahmed Rahni, Emmanuel Grolleau, and Michael Richard, *Feasibility analysis of non-concrete real-time transactions with edf assignment priority*, Proceedings of the 16th conference on Real-Time and Network Systems (Rennes, France), October 2008, pp. 109–117.

[RH98] Minsoo Ryu and Seongsoo Hong, *Toward automatic synthesis of schedulable real-time controllers*, Integrated Computer-Aided Engineering **5** (1998), no. 3, 261–277.

[Rot64] Giancarlo Rota, *The number of partitions of a set*, American Mathematical Monthly **71** (1964), no. 5, 498–504.

[RS00] H. Rehbinder and M. Sanfridson, *Integration of off-line scheduling and optimal control*, Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on, 2000, pp. 137–143.

[Sar89] V. Sarkar, *Partitioning and scheduling parallel programs for execution on multiprocessors*, MIT Press, 1989.

[SB96] Marco Spuri and Giorgio C. Buttazzo, *Scheduling aperiodic tasks in dynamic priority systems*, Journal of Real-Time Systems **10** (1996), no. 2, 179–210.

[SEL08] Insik Shin, Arvind Easwaran, and Insup Lee, *Hierarchical scheduling framework for virtual clustering multiprocessors*, Proceedings of the 20th Euromicro Conference on Real-Time Systems (Prague, Czech Republic), July 2008, pp. 181–190.

[SL03] Insik Shin and Insup Lee, *Periodic resource model for compositional real-time guarantees*, Proceedings of the 24th Real-Time Systems Symposium (Cancun, Mexico), December 2003, pp. 2–13.

[SLS99] J. A. Stankovic, C. Lu, and S. H. Son, *The case for feedback control in real-time scheduling*, Proceedings of the IEEE Euromicro Conference on Real-Time (York, U.K.), June 1999.

[SLSS96] Danbing Seto, John P. Lehoczky, Lui Sha, and Kang G. Shin, *On task schedulability in real-time control systems*, Proceedings of the 17th IEEE Real-Time Systems Symposium (Washington (DC), U.S.A.), December 1996, pp. 13–21.

[SRL90] Lui Sha, R. Rajkumar, and J. P. Lehoczky, *Priority inheritance protocols: An approach to real-time synchronization*, IEEE Transactions on Computers **39** (1990), no. 9, 1175–1185.

[SSRB98] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo, *Deadline scheduling for real-time systems: EDF and related algorithms*, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park Norwell, MA 02061, USA, 1998.

[Sta88] J.A. Stankovic, *Real-time computing systems: The next generation*, Tutorial on Hard Real-Time Systems (J. Stankovic and K. Ramamritham, eds.), IEEE Computer Society Press, 1988.

[THÅ$^+$06] Martin Törngren, Dan Henriksson, Karl-Erik Årzén, Anton Cervin, and Zdenek Hanzalek, *Tools supporting the co-design of control systems and their real-time implementation; current status and future directions*, 2006 IEEE International Symposium on Computer-Aided Control Systems Design (Munich, Germany), October 2006.

[VMB08] Manel Velasco, Pau Martí, and Enrico Bini, *Control-driven tasks: Modeling and analysis*, Proceedings of the 29th IEEE Real-Time Systems Symposium (Barcelona, Spain), December 2008.

[YBB09] Gang Yao, Giorgio Buttazzo, and Marko Bertogna, *Bounding the maximum length of non-preemptive regions under fixed priority scheduling*, Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009) (Beijing, China), Aug 2009.