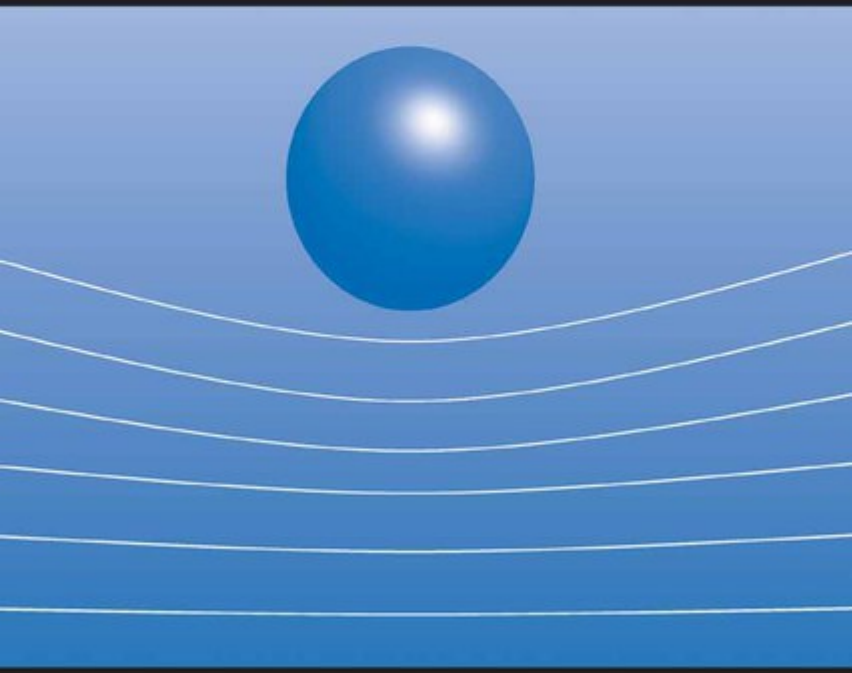
SERIES IN COMPUTER SCIENCE

Soft Real-Time Systems

Predictability vs. Efficiency

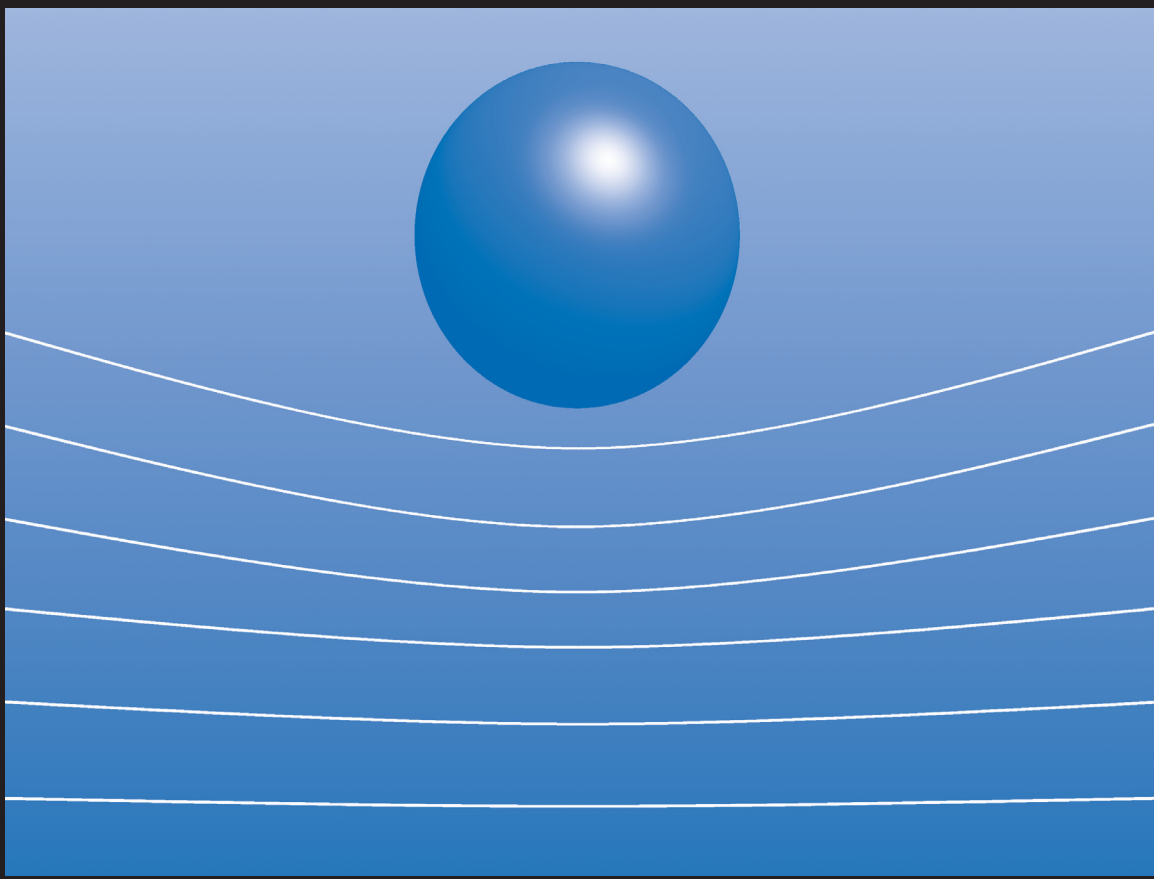


Giorgio Buttazzo, Giuseppe Lipari,
Luca Abeni, and Marco Caccamo

SERIES IN COMPUTER SCIENCE

Soft Real-Time Systems

Predictability vs. Efficiency



Giorgio Buttazzo, Giuseppe Lipari,
Luca Abeni, and Marco Caccamo

Soft Real-Time Systems

Predictability vs. Efficiency

SERIES IN COMPUTER SCIENCE

Series Editor: Rami G. Melhem

University of Pittsburgh
Pittsburgh, Pennsylvania

DYNAMIC RECONFIGURATION
Architectures and Algorithms
Ramachandran Vaidyanathan and Jerry L. Trahan

ENGINEERING ELECTRONIC NEGOTIATIONS
A Guide to Electronic Negotiation Technologies for the Design and Implementation of Next-Generation Electronic Markets—Future Silkroads of eCommerce
Michael Ströbel

HIERARCHICAL SCHEDULING IN PARALLEL AND CLUSTER SYSTEMS
Sivarama Dandamudi

MOBILE IP
Present State and Future
Abdul Sakib Mondal

NEAREST NEIGHBOR SEARCH
A Database Perspective
Apostolos N. Papadopoulos and Yannis Manolopoulos

OBJECT-ORIENTED DISCRETE-EVENT SIMULATION WITH JAVA
A Practical Introduction
José M. Garrido

A PARALLEL ALGORITHM SYNTHESIS PROCEDURE FOR HIGH-PERFORMANCE COMPUTER ARCHITECTURES
Ian N. Dunn and Gerard G. L. Meyer

POWER AWARE COMPUTING
Edited by Robert Graybill and Rami Melhem

SOFT REAL-TIME SYSTEMS
Predictability vs. Efficiency
Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo

THE STRUCTURAL THEORY OF PROBABILITY
New Ideas from Computer Science on the Ancient Problem of Probability Interpretation
Paolo Rocchi

Soft Real-Time Systems

Predictability vs. Efficiency

Giorgio Buttazzo
University of Pavia
Pavia, Italy

Giuseppe Lipari
Scuola Superiore Sant'Anna
Pisa, Italy

Luca Abeni
MBI Group
Pisa, Italy

Marco Caccamo
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA

Springer

ISBN 0-387-23701-1

Printed in the United States of America. (BS/DH)

9 8 7 6 5 4 3 2 1 SPIN 1136648

springeronline.com

CONTENTS

PREFACE

Real-time systems technology, traditionally developed for safety-critical systems, has recently been extended to support novel application domains, including multimedia systems, monitoring apparatuses, telecommunication networks, mobile robotics, virtual reality, and interactive computer games. Such systems are referred to as *soft* real-time systems, because they are often characterized by a highly dynamic behavior and flexible timing requirements. In such systems, missing a deadline does not cause catastrophic consequences on the environment, but only a performance degradation, often evaluated through some quality of service parameter.

Providing an appropriate support at the operating system level to such emerging applications is not trivial. In fact, whereas general purpose operating systems are not predictable enough for guaranteeing the required performance, the classical hard real-time design paradigm, based on worst-case assumptions and static resource allocation, would be too inefficient in this context, causing a waste of the available resources and increasing the overall system cost. For this reason, new methodologies have been investigated for achieving more flexibility in handling task sets with dynamic behavior, as well as higher efficiency in resource exploitation.

This book illustrates the typical characteristics of soft real-time applications and presents some recent methodologies proposed in the literature to support this kind of applications.

Chapter 1 introduces the basic terminology and concepts used in the book and clearly illustrates the main characteristics that distinguish soft real-time computing from other types of computation.

Chapter 2 is devoted to overload management techniques, which are essential in dynamic systems where the computational requirements are highly variable and cannot be predicted in advance.

Chapter 3 introduces the concept of temporal protection, a mechanism for isolating the temporal behavior of a task to prevent reciprocal interference with the other system activities.

Chapter 4 deals with the problem of executing several independent multi-thread applications in the same machine, presenting some methodologies to partition the processor into several virtual slower processors, in such a way that each application can be independently guaranteed from each other.

Chapter 5 presents a number of synchronization protocols for limiting blocking times when mutually exclusive resources are shared among hard and soft tasks.

Chapter 6 describes resource reclaiming techniques, which enhance resource exploitation when the actual resource usage of a task is different than the amount allocated off line. These techniques basically provide a method for reassigning the unused resources to the most demanding tasks.

Chapter 7 treats the issue of quality of service management. It is addressed through an adequate formulation that univocally maps subjective aspects (such as the perceived quality that may depend on the user) to objective values expressed by a real number.

Chapter 8 presents some feedback-based approach to real-time scheduling, useful to adapt the behavior of a real-time system to the actual workload conditions, in highly dynamic environments.

Chapter 9 addresses the problem of performing a probabilistic analysis of real-time task sets, with the aim of providing a relaxed form of guarantee for those real-time systems with highly variable execution behavior. The objective of the analysis is to derive for each task a probability to meet its deadline or, in general, to complete its execution within a given interval of time.

Acknowledgments

This work is the result of several years of research activity in the field of real-time systems. The majority of the material presented in this book is taken from research papers and has been elaborated to be presented in a simplified form and with a uniform structure. Though this book carries the names of four authors, it has been positively influenced by a number of people who gave a substantial contribution in this emerging field. The authors would like to acknowledge Enrico Bini, for his insightful discussions on schedulability analysis, Paolo Gai for his valuable work on kernel design and algorithms implementation, and Luigi Palopoli for his contribution on integrating real-time and control issues. Finally, we would like to thank the Kluwer editorial staff for the support we received during the preparation of the manuscript.

1

INTRODUCTION

In this chapter we explain the reasons why soft real-time computing is being deeply investigated during the last years for supporting a set of application domains for which the hard real-time approach is not suited. Examples of such application domains include multimedia systems, monitoring apparatuses, robotic systems, real-time graphics, interactive games, and virtual reality.

To better understand the difference between classical hard real-time applications and soft real-time applications, we first introduce some basic terminology that will be used throughout the book, then we present the classical design approach used for hard real-time systems, and then describe the characteristics of some soft real-time application. Hence, we identify the major problems that a hard real-time approach can cause in these systems and finally we derives a set of features that a soft real-time system should have in order to provide efficient support for these kind of applications.

1.1 BASIC TERMINOLOGY

In the common sense, a real-time system is a system that reacts to an event within a limited amount of time. So, for example, in a web page reporting the state of a Formula 1 race, we say that the race state is reported in real-time if the car positions are updated "as soon as" there is a change. In this particular case, the expression "as soon as" does not have a precise meaning and typically refers to intervals of a few seconds.

When a computer is used to control a physical device (e.g., a mobile robot), the time needed by the processor to react to events in the environment may significantly affect the overall system's performance. In the example of a mobile robot system, a correct maneuver performed too late could cause serious problems to the system and/or the

environment. For instance, if the robot is running at a certain speed and an obstacle is detected along the robot path, the action of pressing the brakes or changing the robot trajectory should be performed within a maximum delay (which depends on the obstacle distance and on the robot speed), otherwise the robot could not be able to avoid the obstacle, thus incurring in a crash.

Keeping the previous example in mind, a real-time system can be more precisely defined as a computing system in which computational activities must be performed within predefined timing constraints. Hence, the performance of a real-time system depends not only on the functional correctness of the results of computations, but also on the time at which such results are produced.

The word *real* indicates that the system time (that is, the time represented inside the computing system) should always be synchronized with the external time reference with which all time intervals in the environment are measured.

1.1.1 TIMING PARAMETERS

A real-time system is usually modeled as a set of concurrent *tasks*. Each task represents a computational activity that needs to be performed according to a set of constraints. The most significant timing parameters that are typically defined on a real-time computational activity are listed below.

- **Release time** r_i: is the time at which a task becomes ready for execution; it is also referred to as *arrival time* and denoted by a_i;
- **Start time** s_i: is the time at which a task starts its execution for the first time;
- **Computation time** C_i: is the time necessary to the processor for executing the task without interruption;
- **Finishing time** f_i: is the time at which a task finishes its execution;
- **Response time** R_i: is the time elapsed from the task release time and its finishing time ($R_i = f_i - r_i$);
- **Absolute deadline** d_i: is the time before which a task should be completed;
- **Relative deadline** D_i: is the time, relative to the release time, before which a task should be completed ($D_i = d_i - r_i$);

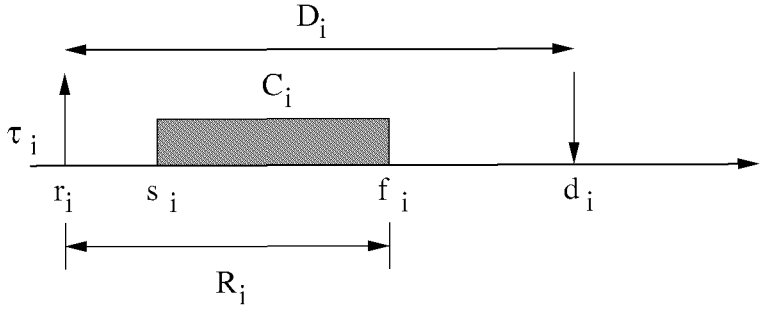


Figure 1.1 Typical timing parameters of a real-time task.

Such parameters are schematically illustrated in Figure 1.1, where the release time is represented by an up arrow and the absolute deadline is represented by a down arrow.

Other parameters that are usually defined on a task are:

- **Slack time** or **Laxity**: denotes the interval between the finishing time and the absolute deadline of a task ($slack_i = d_i - f_i$); it represents the maximum time a task can be delayed to still finish within its deadline;
- **Lateness** L_i: $L_i = f_i - d_i$ represents the completion delay of a task with respect to its deadline; note that if a task completes before its deadline, its lateness is negative;
- **Tardiness** or *Exceeding time* E_i: $E_i = max(0, L_i)$ is the time a task stays active after its deadline.

If the same computational activity needs to be executed several times on different data, then a task is characterized as a sequence of multiple instances, or *jobs*. In general, a task τ_i is modeled as a (finite or infinite) stream of jobs, $\tau_{i,j}$, $(j = 1, 2, \ldots)$, each characterized by a release time $r_{i,j}$, an execution time $c_{i,j}$, a finishing time $f_{i,j}$, and an absolute deadline $d_{i,j}$.

1.1.2 TYPICAL TASK MODELS

Depending on the timing requirements defined on a computation, tasks are classified into four basic categories: hard, firm, soft, and non real time.

A task τ_i is said to be *hard* if all its jobs have to complete within their deadline ($\forall j \;\; f_{i,j} \leq d_{i,j}$), otherwise a critical failure may occur in the system.

A task is said to be *firm* if only a limited number of jobs are allowed to miss their deadline. In [KS95], Koren and Shasha defined a firm task model in which only one job every S is allowed to miss its deadline. When a job misses its deadline, it is aborted and the next $S-1$ jobs must be guaranteed to complete within their deadlines. A slightly different firm model, proposed by Hamdaoui and Ramanathan in [HR95], allows specifying tasks in which at least k jobs every m must meet their deadlines.

A task is said to be *soft* if the value of the produced result gracefully degrades with its response time. For some applications, there is no deadline associated with soft computations. In this case, the objective of the system is to reduce their response times as much as possible. In other cases, a *soft deadline* can be associated with each job, meaning that the job should complete before its deadline to achieve its best performance. However, if a soft deadline is missed, the system keeps working at a degraded level of performance. To precisely evaluate the performance degradation caused by a soft deadline miss, a performance value function can be associated with each soft task, as described in Chapter 2.

Finally, a task is said to be *non real time* if the value of the produced result does not depend on the completion time of its computation.

1.1.3 ACTIVATION MODES

In a computer controlled system, a computational activity can either be activated by a timer at predefined time instants (time-triggered activation) or by the occurrence of a specific event (event-triggered activation).

When jobs activations are triggered by time and are separated by a fixed interval of time, the task is said to be *periodic*. More precisely, a periodic task τ_i is a time-triggered task in which the first job $\tau_{i,1}$ is activated at time Φ_i, called the task phase, and each subsequent job $\tau_{i,j+1}$ is activated at time $r_{i,j+1} = r_{i,j} + T_i$, where T_i is the task period. If D_i is the relative deadline associated with each job, the absolute deadline of job $\tau_{i,j}$ can be computed as:

$$\begin{cases} r_{i,j} = \Phi_i + (j-1)T_i \\ d_{i,j} = r_{i,j} + D_i \end{cases}$$

If job activation times are not regular, the task is said to be *aperiodic*. More precisely, an aperiodic task τ_i is a task in which the activation time of job $\tau_{i,k+1}$ is greater than or equal to that of its previous job $\tau_{i,k}$. That is, $r_{i,k+1} \geq r_{i,k}$.

If there is a minimum separation time between successive jobs of an aperiodic task, the task is said to be *sporadic*. More precisely, a sporadic task τ_i is a task in which the difference between the activation times of any two adjacent jobs $\tau_{i,k}$ and $\tau_{i,k+1}$ is greater than or equal to T_i. That is, $r_{i,k+1} \geq r_{i,k} + T_i$. The T_i parameter is called the *minimum interarrival time*.

1.1.4 PROCESSOR WORKLOAD AND BANDWIDTH

For a general purpose computing system, the processor workload depends on the amount of computation required in a unit of time. In a system characterized by aperiodic tasks, the average load $\overline{\rho}$ is computed as the product of the average computation time $\overline{C}$ requested by tasks and the average arrival rate λ:

$$\overline{\rho} = \overline{C}\lambda.$$

In a real-time system, however, the processor load also depends on tasks' timing constraints. The same set of tasks with given computation requirements and arrival patterns will cause a higher load if it has to be executed with more stringent timing constraints.

To measure the load of a real-time system in a given interval of time, Baruah, Howell and Rosier [BMR90] introduced the concept of *processor demand*, defined as follows:

Definition 1.1 *The processor demand $g(t_1, t_2)$ in an interval of time $[t_1, t_2]$ is the amount of computation that has been released at or after t_1 and must be completed within t_2.*

Hence, the processor demand $g_i(t_1, t_2)$ of task τ_i is equal to the computation time requested by those jobs whose arrival times and deadlines are within $[t_1, t_2]$. That is:

$$g_i(t_1, t_2) = \sum_{r_{i,j} \geq t_1, d_{i,j} \leq t_2} c_{i,j}$$

For example, given the set of jobs illustrated in Figure 1.2, the processor demand in the interval $[t_a, t_b]$ is given by the sum of computation times denoted with dark gray, that is, those jobs that arrived at or after t_a and have deadlines at or before t_b.

The total processor demand $g(t_1, t_2)$ of a task set in an interval of time $[t_1, t_2]$ is equal to the sum of the individual demands of each task. That is,

$$g(t_1, t_2) = \sum_{i=1}^{n} g_i(t_1, t_2)$$