Adaptive Resource Reservation

PhilosophiæDoctor (Ph.D.) Thesis

Luca Santinelli

Retis Lab

Scuola Superiore Sant'Anna

December 2010

Commission

1. Reviewer: Prof. Giorgio Buttazzo
Retis Lab, Scuola Superiore Sant'Anna

2. Reviewer: Prof. Lothar Thiele
TIK, ETH Zurich

3. Reviewer: Researcher Liliana Cucu-Grosjean
Trio Team, INRIA Nancy-Grand Est

Abstract

Many real-time systems consist of sets of applications that need to be executed in isolation to limit their reciprocal interference. Temporal isolation can be effectively achieved through resource reservations, which is a scheduling technique to partition computational resources into a set of "virtual" processors, with reduced but dedicated bandwidth. Moreover, each application is typically designed to work in different operating modes, each characterized by different functionality and resource demands. When applications and modes are activated online depending on user or environmental conditions, the system workload becomes highly dynamic and resource reservations need to be reconfigured at runtime to comply with the new conditions.

The problem of finding the optimal reservation parameters that satisfy the application requirements in each operating mode has been deeply investigated in the real-time literature. Similarly, there are solutions that allow performing safe mode transitions, but without reservations. Surprisingly, the problem of resource reconfigurations for systems with reservations has not been addressed yet.

This thesis investigates methods for analyzing and reconfiguring resource reservation under different abstraction models. Adaptive mechanisms are studied and developed to offer a novel framework for scheduling dynamic real-time applications under dynamic resource reservation mechanisms. Optimal or suboptimal solutions to the resource reservation problem can be requested depending on the system condition and the necessary online applicability. Results are illustrated by examples, simulations, and case studies.

To my family

How do you pick up the threads of an old life? How do you go on, when in your heart you begin to understand there is no going back. There are some things that time can not mend. Some hurts that go too deep... that have taken hold.

Acknowledgements

This work would not have been possible without the contribution of many people, to whom I would now take the opportunity to thank. Foremost, I thank Prof. Giorgio Buttazzo, my considerate adviser. His confidence in our work compelled me to persist and his unique ability of solving problems challenged me to do the same. His personality commends my admiration.

My gratitude goes also to Prof. Giuseppe Lipari. He has been the ideal research group director during these years capable to instill motivation and coordinate activities. I thank him for his gentle and effective leading style. He helped me finding my own research style.

In the same breath, I want to thank Dr. Enrico Bini, Dr. Marko Bertogna, Dr. Mauro Marinoni and Dr. Fabio Checconi because they have offered me support and time to confront with their enlightening researching style everyone from different perspectives. Our long discussion have passed on me the value of investigating every possible aspect to obtain valuable results. I also want to thank Dr. Paolo Pagano which has offered to me the opportunity to "meet" the Wireless Sensor Networks together with the possibility to do research in that field.

The whole Retis Lab, and in a bigger context the CEIIC, provided an excellent environment for my professional and personal growth. I thank all those who contributed to it. Foremost my colleagues at the Retis Lab for their friendship, spirit, and time.

Some of the work in this thesis stems from the period in which I visited the TIK laboratory at the ETH University in Zurich led by Prof. Lothar Thiele. I want to thank him for providing me that opportunity and mostly because he motivated me with his daily example and his capacity of proposing and investigating interesting research topics. I also thank all the people I met there for their friendship and support.

New interests and enthusiasm, professional as well as cultural, has been awaken in me during my visit to the research center of INRIA Nancy-Grand Est. I am grateful to Dr. Liliana Cucu-Grosjean for the advise she gave (and still is giving) to me and the enlightening discussion about a probabilistic vision of real-time. I take this opportunity to thank her for inviting me and to salute the people I met there.

I would like to express my gratitude to my co-authors for their collaboration and guidance. I really enjoyed working with you.

Geographically, more or less distant, many people honored me with their friendship for already frighteningly many years. It means very much to me. Thus, I thank my friends for the smile and the confidence they have always offered me. Mostly, I am grateful to my family for the mind set and values they have passed to me. They have always supported me in any of my choices, I know it will be like this always.

Luca Santinelli

Pisa, December 2010

Contents

Chapter 1

Introduction

Embedded systems are special-purpose information processing systems that are closely integrated into their environment. An embedded system is typically dedicated to a specific application domain. The knowledge about this domain and the systems environment are used to develop customized and optimized system designs. That makes the embedded systems also reactive systems that are in continuous interaction with their physical environment to which they are connected through sensors an actuators. Consequently they must execute at a pace determined by their environment. This results in many embedded systems that have to meet timing constraints, i. e. they must react to stimuli within a time interval dictated by the environment itself. Such real-time constraints if not met, they could result in a impermissible failure of the system, or a degradation of the Quality of Service (QoS) the system provides. So to speak, a real-time systems behavior depends not only on the functional correctness of its inputs but also on the time on which results are produced.

Nowadays, real-time systems are composed by many applications sharing resources and executing concurrently. That further increase the complexity of real-time systems and requires ad hoc countermeasures for guaranteeing the application properties. The research on real-time systems is mature to a point that many existing works have been proposed to tackle with the scenario in which a set of independently developed applications are scheduled upon a computing platform.

Reservation-based resource partitioning, Resource Reservation (RR) for short, is an emerging paradigm for resource management in embedded systems with timing requirements. In particular, resource reservations mechanisms allow the operating system to monitor and enforce application resource usage and timing requirements, obtaining temporal isolation between real-time applications. This property ensures that the temporal behavior of applications does not depend on the behavior of any other, and is as powerful as spatial protection in systems with separate address-space protection. In other words, resource reservation mechanisms are used as a general framework for accessing time-multiplexed resources, i.e. the computing resource, the communication resource, etc.. Indeed, the goal of the resource management is to guarantee availability of required resources to applications so they can rely on them. This implies that applications are to some degree isolated from the behavior of other applications on the same system.

Dynamic environments where applications may be added and removed on-line, together with more application domains, require adaptive real-time embedded systems that can change their functionality and resource demand over time. Examples are applications characterized by multiple execution modes, each consisting of a specific task set and a specific workload requirement. For these systems, the feasibility of the schedule has to be guaranteed not only within each individual mode, but also during mode transitions.

Dynamic domains ask for real-time embedded systems that can adapt their behavior at run-time by changing their operating mode. Depending on the load of the system the resources should be distributed in a flexible manner among the applications, and the applications should adapt their algorithms accordingly, so that the best possible quality of service for the given resources is achieved (29; 87; 152). According to each mode change, the static resource reservation paradigm may not be suitable anymore in case of resource demands of changing applications. In such scenarios, reconfigurations are needed for changing the resource reservations during runtime and achieve better resource allocations.

The reservation of the computational resource need to be reconfigured dynamically to adapt the resource reservations and reflect the changes in the system or its environment. Such reconfigurations need to be performed online without jeopardizing the schedulability. Thus, in such systems it is not only necessary to guarantee timing constraints in every operating mode, but also during the transition between different modes. It is therefore essential to develop appropriate resource reconfiguration criteria and algorithms to manage the criticality of the transition phase.

To resume, adaptive real-time systems have to be able to adjust any of their internal strategies in response to changes in the resource availability and resource demands to keep the system performance at an acceptable level.

1.1 Real-Time

Real-time systems serve application requests with stringent timing constraints; applications that are composed by tasks which exploits the basic functionalities of the system. Since the main issue of real-time systems is predictability, scheduling algorithms are required because they allow to analyze a priori the feasibility of the system that is establishing whether the timing constraints are going to be met or there are potential failure.

In general, to define a scheduling problem it is required to specify a set of n tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$, a set of m processors $P = \{P_1, \ldots P_m\}$ and a set of g types of resources $r = \{r_1, \ldots, r_g\}$. Moreover, precedence relations among tasks can be specified through a directed acyclic graph and timing constraints can be associated with each task. In this context, scheduling means assigning processors from P and resources from r to tasks from Γ in order to complete all tasks under the imposed constraints.

Since any real-time computing has to be predictable, a task τ_i is characterized by a relative deadline D_i (relative to the task activation), which is the maximum time window within which the task must complete its execution.

A real-time task is generally placed into one of the following broad categories based upon its deadline (36; 118; 148). If meeting a given task deadline is critical to the operation of the system, then the task deadline is considered to be hard and the task is *hard* real-time. If it is desirable to meet a task deadline but occasionally missing the deadline can be tolerated, then the deadline is considered to be soft and consequently the tasks are *soft* real-time. Tasks with no timing constraints at all are simply named *non-real* time tasks.

A task can also be classified in accordance to its arrival pattern. Tasks with regular arrival times are called *periodic*. A common use of periodic tasks is to process sensor data and update the current state of the real-time system on a regular basis. Periodic tasks, typically used in control and signal processing applications, have hard deadlines. Tasks with irregular arrival times are *aperiodic* which are mainly used to handle the processing requirements of random events such as operator requests. An aperiodic task typically has a soft deadline, while those aperiodic tasks that have hard deadlines are called *sporadic* tasks.

Each task τ_i consists of a sequence of jobs that need to receive a certain amount of execution time. In order to meet the mandatory deadlines of hard real-time tasks, it is necessary to define upper bounds on the worst-case execution times (WCET) C_i of such tasks. Soft real-time tasks are treated as hard real-time task, so it is required a WCET in order to exploit their analysis. For aperiodic and non real-time tasks (tasks with no timing constraints at all), instead, there is no need to specify worst-case parameters, since there are no hard deadlines. Although, we apply C_i to them too in order to make our analysis more general and take into account also those tasks.

Furthermore, every periodic or sporadic task τ_i is characterized by a period T_i which is the exact task period or its minimum inter-arrival time in case of periodic or sporadic task representation, respectively. Aperiodic tasks does not assume any pseudo-periodicity in their arrivals. The ratio among the worst case execution time and the task period $\frac{C_i}{T_i}$, is known as the task utilization factor describing the fraction of the processor spent in the execution of task τ_i, (106). The processor utilization factor of the whole task set $U = \sum_{i \in \Gamma} \frac{C_i}{T_i}$ provides a measure of the computational load on the CPU due to the task set Γ.

The initial offset O_i of a periodic task τ_i is the instant of the first activation of the task. Every successive activation is a multiple of the task period plus the initial offset. However, even if a periodic task is activated as some time t its release time (i.e. the time from which it can start executing) may be delayed due to the precedence constraint. In fact, a task belonging to a group may start only after it has been activated and the preceding task in the group has completed execution. In case of aperiodic or sporadic tasks the initial offset is meaningless. Nevertheless, we keep it in the task model in order to be as much generic as possible.

In this dissertation we consider an application Γ as a set of n periodic or sporadic hard or soft real-time tasks, $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$. Every task τ_i is characterized by the set of parameters describe before, so it can be represented by the tuple (O_i, C_i, T_i, D_i), $\tau_i = (O_i, C_i, T_i, D_i)$.

Real-time systems require also *scheduling policies* that reflect the timeliness constraints of real-time tasks. Schedulers produce a schedule for a given set of tasks, and if

a task set can be scheduled to meet given pre-conditions the task set is termed feasible. A typical pre-condition for hard real-time periodic tasks is that they should always meet their deadlines, which means terminating the task execution before the deadline. An optimal scheduler is then able to produce a feasible schedule for all feasible tasks sets conforming to a given pre-condition. For a particular process set an optimal schedule is the best possible schedule according to some pre-defined criteria. Typically a scheduler is optimal if it can schedule all task sets that other schedules can (36; 151).

To meet the timing constraints of the system, a scheduler must coordinate the use of all system resources by using a set of real-time scheduling algorithms in order to

1. guarantee that tasks with hard timing constraints will always meet their deadlines;

2. provide fast average response times for tasks with soft deadlines (aperiodic tasks);

3. ensure scheduling stability under transient overload.

The three main approaches used to schedule a real-time task system are a) Clock-Driven, b) Processor-Sharing, and c) Priority-Driven. Although this dissertation focuses on priority-driven approaches, it marginally considers the other two cases. Few papers detail the work that has been done related to clock-driven and processor-sharing scheduling.

The priority-driven approach can be divided into dynamic-priority (DP)(19; 39; 151) and fixed priority (FP) (96; 98; 106). The priority is a number associated with a task and used to establish an order of precedence among tasks competing for a common resource. For FP scheduling, the tasks and their jobs are prioritized a priori. Once a job of a task is active into the system, the priority of the job is set to the pre-defined priority of the task it belongs to. Priorities are assigned to tasks before execution and do not change over time. As an example, the rate monotonic (RM) scheduling algorithm is a simple rule that assigns priorities to tasks to their request rate, (106). Specifically, tasks with higher request rates (that is with shorter periods) have higher priorities. Since period are constant, RM is a fixed priority assignment.

DP scheduling assigns the priority to the tasks at run-time, the highest priority is given to the task occurrence according to the actual conditions. The earliest deadline first (EDF) is a example of dynamic scheduling policy that selects tasks according to their absolute deadlines. Specifically, tasks with earlier deadlines are executed at higher priorities. Since the absolute deadline of a periodic task τ_i depend on the current j-th instance as $d_{i,j} = O_i + (j-1)T_i + D_i$, EDF is a dynamic priority assignment (at task level), although the priority of each job is fixed.

For both FP and DP scheduling, the system executes the incomplete tasks with the highest priority. If there are more than one event with the highest priority, we break ties by applying the first-come-first-serve (FCFS) strategy. With respect to fixed priority assignments, dynamic scheduling algorithms are characterized by higher schedulability bounds which allows the processor to be better utilized.

1.2 The Resource Reservation Framework

Until recent years, the study of real-time scheduling problems has been primarily concerned with allocating dedicated resources to serve a set of application programs which are characterized by a real-time system model. Since the first real-time system model was introduced by Liu and Layland in 1973 (106), there have been many variations proposed to model real-time systems, e.g., the sporadic model (118), the pinwheel model (76). The schedulability analysis of these models always assumes that the resource to be allocated is made available at a uniform rate and accessible exclusively by the tasks under consideration.

Todays system requires resource sharing and that poses new difficulties. If predictability is the main goal of a real-time system, traditional real-time scheduling theory can be successfully used to verify the feasibility of the schedule under worst-case scenarios. However, when efficiency becomes relevant and when the worst-case parameters of the tasks are too pessimistic or unknown, the hard real-time approach presents some problems. In particular, task overruns can cause temporary or permanent overload conditions that may degrade system performance in an unpredictable fashion. *When executing different real-time applications on a single processor system, one problem is how to compose these applications and guarantee that their timing requirements are not violated*. Recent study (126) has shown that the use of resource reservation techniques can bring advantages in terms of control performance. Designing the system for the worst-case guarantees absence of deadline misses, but can impose a low control loop rate. On the other hand, designing the system for the average case allows increasing the control loop rate, and thus improving the control performance as average, but can lead to deadline misses of critical activities. Resource reservation techniques permit to calibrate the resource usage for the different activities. For example, in a complex control system with multi rate sensor acquisition that uses a video camera for pattern recognition, the most critical low-level control loop can be considered as a hard activity, which should be assigned an amount of resource equal to its worst-case requirement. The less critical activities, like image acquisition and recognition, can be assigned a fraction of the resource equal to their average case conditions, thus increasing their rate, (2; 126). The *resource reservation* is a mechanism for partitioning a resource among tasks, so that each application is assigned a fraction of the resource according to a predefined strategy, (68; 121).

A considerable amount of work has been recently addressed to the analysis of resource reservation mechanisms for achieving temporal protection in real-time systems. The concept of reservations was originally introduced by Mercer, Savage and Tokuda (117), and later formalized by Rajkumar et al. (131) as a generic kernel mechanism to allocate a fraction of a computational resource to a set of tasks. Feng and Mok (64) adopted resource reservations for achieving hierarchical partitioning of computational resources.

The key idea behind the concept of hierarchical scheduling is to use two levels of schedulers: a) global scheduler that allocates the processor to a partition (the active partition); b) local schedulers that elect tasks to run among the ready tasks in the active partition. There is a single global scheduler in a computer node and a local

scheduler for every single partition. This leads to view each application as accessing a virtual resource that operate at a fraction of the rate of the physical resource shared by the application, (60; 107; 121). Informally, a partition is a collection of intervals during which the computation resource is made available to the application scheduled on the partition. So the notion of *virtual resource* is introduced for abstracting resource sharing by application task that are subject to different timing requirements (120). Sharing is enforced by partitioning schemes that time-multiplexes the resource among the different applications.

The shared resource is partitioned into real-time virtual resources by a resource-level scheduler such that each real-time virtual resource is accessible only by an individual application task group; tasks within the same task group are scheduled by an application level scheduler that is specialized to the real-time requirements of the tasks in the group. Through this model, partitions on each level are scheduled as if they had access to a dedicated resource and there is minimal interference between neighboring partition levels resulting in a hierarchical scheduling scheme. The hierarchical partitioning of a computational resource is well known in literature, (120; 121). With that it is possible to compose schedulers at arbitrary levels of the hierarchy. It is possible to analyze and guarantee the schedulability of an application on a partition as well as to address the reverse problem: given an application, scheduled by a local fixed priority scheduler, how to select the best resource partition for the given application.

For real-time systems, there has been a growing attention to compositional analysis of hierarchical scheduling frameworks (113; 160). Traditionally, this analysis has been achieved by using interfaces that characterize the resource supply necessary to schedule components. Mok and Feng proposed the bounded delay resource partition model for a hierarchical scheduling framework (64), and Shin and Lee (141; 142; 144) addressed the interface generation problem for these models. Similarly, there are studies on the component abstraction problem using periodic resource models (68; 134; 135). When these techniques are used for hierarchical frameworks with conditional task models, complexity of interface generation depends on the utilization of the task set (i.e., schedulability checking of conditional task models depends on utilization (16)).

Within a partitioned scenario, the concept of server become relevant. A server is a real-time system element dedicated to the management of a shared resource and it is implemented as a kernel process. *A possible way of composing applications is through the resource reservation approach where each application is handled by a dedicated server that is assigned a fraction of the processor.* Using this approach, the system can be seen as a multi-level hierarchical scheduler where servers are used to isolate the temporal behavior of real-time applications through resource reservations (116).

Resource reservation is typically implemented through a server mechanism, which allocates a budget Q every period P to the served application. Several service algorithms have been proposed in the literature, both under fixed priority systems, like the Polling Servers (PS) (36), the Deferrable Server (DS) (97) and the Sporadic Server (SS) (147), and under EDF, like the Dynamic Sporadic Server (DSS) (109) and the Constant Bandwidth Server (CBS) (5).

Another common problem is the coexistence of hard periodic tasks and soft aperiodic tasks. Even such a problem can be solved by a resource reservation strategy,

which assigns each soft task a maximum resource bandwidth, calculated using the mean execution time and the desired activation period, in order to increase CPU utilization. If a task needs more than its reserved bandwidth, it may slow down its execution, but it will not jeopardize the schedulability of the hard real-time tasks. Once isolated the effects of task overloads, hard tasks can be guaranteed using classical schedulability analysis.

Real-time servers (36) are used for both scheduling soft and non real-time tasks together with other instances having hard timing requirements (95; 147; 148), and also providing temporal isolation among different applications in a hierarchical environment (53; 63; 110; 116). From the computational resource point of view the two problem formerly listed, are the same, such as different applications need to share a common computing platform, without interfering each other. By encapsulating each application in a particular real-time server, it is possible to enforce the desired isolation among the various system elements. When a particular server is selected for execution by the high-level scheduler, the corresponding application is executed. The internal scheduler of the selected application will then decide which instance to execute. Note that an application could also select to execute another (lower-level) server, increasing the hierarchical structure depth.

The servers mechanisms enforces the bandwidth reservation and the resource reservation strategies because servers are abstract entities used by a scheduler to reserve a fraction of CPU-time to a particular instance. In general a server S_k is characterized by the period of the reservation P_{S_k} and by the reserved execution time per period Q_{S_k}; we define $U_{S_k} = Q_{S_k}/P_{S_k}$ the fraction of CPU-time reserved by server S_k, also called utilization factor. In addition, each server maintains its own internal variables that are updated by the scheduler depending on the server rules. One of these variables is the server priority, so that servers are inserted in the priority queue of the scheduler. When a server is selected by the scheduler to execute, the corresponding instance is executed and the server budget is accordingly decremented.

In the following, we assume each application Γ_k be scheduled by means of a dedicated server S_k. Since an application may as well be composed of a single task, the above mentioned bandwidth reservation mechanism for soft and non real-time tasks can be described with the same model.

Along this dissertation, we will refer to resource as the computational resource if not specified otherwise.

In real-time operating systems, servers are a specific kernel processes that help scheduling mechanism that handles aperiodic requests as soon as possible, while preserving hard periodic tasks from missing their deadlines. The classical server classification distinguishes between fixed priority and dynamic-priority servers, depending on the scheduling policy used to schedule them. Among fixed priority servers, Deferrable Server (DS) (156) and Sporadic Server (SS) (147) are the most well known techniques that preserve their capacity when no request is pending upon the invocation of a server. Spuri et al. (109) presented a survey of dynamic priority servers that can efficiently work under EDF, such as the Dynamic Sporadic Server (DSS) (109) and the Constant Bandwidth Server (CBS) (5).

Server models can also be broadly classified into *event-driven servers*: the servers are driven by the application requirements. The sporadic servers, the dynamic servers, the dynamic sporadic servers and constant bandwidth servers are typical examples. There exist also *time-triggered servers*. Those are the servers which resource supply is driven by a predefined timing pattern that depends only on the server properties. An example is the Time Division Multiple Access (TDMA) server where the resource is periodically partitioned (167). In particular, a TDMA server assigns time slots to its applications that repeat each cycle. The time-triggered architectures play an increasingly important role in large distributed embedded systems as described in (75; 170). Mainly, time-triggered servers offer high predictability with enormous benefits to the analysis of real-time systems, as we will see later on.

1.3 Dynamic Real-Time Systems

Complex real-time systems are dynamic which consists in having several behaviors and is described by a set of functionalities that are carried out by different parameter settings. Those systems are characterized by different operational modes, designed to achieve different functionalities or to respond to changes in the environment, (123; 128; 133; 145). Each mode specifies functional and not functional characteristics and consists of specific computational demands, resource requirements and resource availabilities.

A typical example is that of an aircraft control system where it can be distinguished *landing*, *take off* and *normal cruise* modes, each with a different general objective. The current operating mode of the aircraft depends on the particular phase it is executing.

For generic embedded systems many could be the operating modes composing the system. For example:

- the *initialization* mode which is the startup phase where the different hardware devices and software modules are initialized before they could be operative. All the systems virtually have this phase.
- The *low power* mode where the system optimizes the power supply by minimizing the number of running activities or the execution frequency in order to reduce power consumption.
- The *high quality* mode where the system executes all the requests at the maximum resource available in order to provide the highest quality of service and no constrains from the energy consumption.

As a consequence, the overall computational load and the allocated resources may change over time depending on the operational mode selected for the system. For example, adding a new task into the system at run-time may result in a reduction of the computational resources allocated to the other tasks. Other examples comes form the energy consumption policy where to save energy, the system may be required to discard some of its functionalities and redistribute the resources among its components at run-time.

A change in the system state, e.g., from start-up to normal, or from normal to energy saving, or from normal to shut-down, may also require re-allocating the computational resources among the tasks composing the application. Changing one or more parameters in the system components at run-time must also be considered a *mode change*, because it affects the system load and hence modifies the timing behavior of the application and the system itself.

The mode change is required to cope with the changing conditions so that the system can work properly with the new requirements. To change the operating mode it is necessary to switch form a set of parameters to another and this circumstance introduces a transient stage where it is not well defined what are the condition and the parameters of the system along that phase. Such an uncertainty has to be investigated to make the system predictable even during the mode changes. A mode change is initiated whenever the system detects a change either in the environment or in the internal state.

Multi-moded real-time systems require a more accurate analysis with respect to classical single-mode systems, because of the criticality of mode transitions. In fact, there are situations in which, although timing constrains can be guaranteed to be met within each individual mode, in steady state conditions, deadlines can still be missed during mode transitions. It is therefore essential to analyze the system in order to guarantee feasibility not only within each mode, but also cross modes.

If a resource reservation approach (5; 116) is adopted in the system to achieve temporal isolation between different application components, then a mode transition may also require changing reservation parameters, e.g., to re-distribute the available resources whenever a new component is dynamically activated. In this situation, achieving predictability means not only providing guarantee *between components* (i.e., at the reservation level) before, after, and during mode transitions, but also *within each component*, before, after, and during mode transitions.

Whereas a server manages an application by supplying the resource it requires, adaptive applications must rely on adaptive servers to meet their changing resource requirements. Changing a reservation means changing the corresponding server parameters (its budget and/or its period) to adapt the resource provisioning to the new requirements demanded by the system. The server reconfigurations need to be performed online without jeopardizing schedulability. It is therefore essential to develop appropriate resource reconfiguration criteria and algorithms to manage the criticality of the transition phase.

In most of the cases already studied are the application running on a real-time systems that are changing due to the triggering events. Thus, the running task set is modified passing from an old version to a new one by deleting some tasks and releasing some new tasks. In this case, the transient introduced may cause temporal overload conditions where both old-mode and new-mode tasks are concurrently executed.

The problem of timing analysis across mode changes has been addressed in the real-time literature under different assumptions and system models (128; 139; 153; 163). For instance, Fohler (65) investigated the problem of mode changes in the context of pre run-time scheduled hard real-time systems, where a table-driven schedule is constructed for each operational mode and an appropriate time must be selected to

start a new mode and avoid deadline misses. Crespo et al. (133) presented a survey of mode change protocols for uniprocessor systems under fixed-priority scheduling and proposed a new protocol along with their own schedulability analysis. Guangming (71) computed the earliest time at which a new task can be safely added to the system under the Earliest Deadline First (EDF) scheduling, without jeopardizing the feasibility of the task set. The underline idea behind such solutions is to wait for a certain amount time before changing the schedule, identifying a safe time instant where the new mode can be activated without causing deadline misses.

All of these results address the problem of performing mode transitions in applications without violating their schedulability. None of them considers how to *change resource reservations* online without violating applications schedulability which is the goal of this paper.

Together with the applications, any other element of real-time system may change at run-time. The parameters of servers, schedulers (hence the scheduling policy) and more other can change as a consequence of either external or internal changes, (130). Furthermore, whereas a server manages an application by supplying the resource it requires, adaptive applications must rely on adaptive servers to meet their changing resource requirements.

Fixed reservations paradigms are not appropriate to achieve the desired performance with applications in which the computational demand is highly variable. To cope with such dynamic systems, Buttazzo et al. (33) proposed an elastic scheduling methodology for adapting the rates of a periodic task set to different workload scenarios, without affecting the system schedulability. Abeni et al. (3) presented a framework for dynamically allocating the CPU to tasks whose execution times are not known a priori. Adaptive reservation techniques based on feedback scheduling have also been investigated by the same authors (4). All of these frameworks are only suitable for soft real-time systems. Abeni and Buttazzo (5) introduced a bandwidth reservation mechanism (the constant bandwidth server) that allows real-time tasks to execute in dynamic environments under a temporal protection mechanism, so that the server never exceeds a predefined bandwidth, independently on the actual requests of the server tasks.

Despite the amount of works done so far, the classical server paradigms and models do not allow adaptations to changing conditions. To the best of our knowledge, however, none of the proposed reservation mechanisms has been analyzed to predict the timing behavior of the served application during a reconfiguration process. Clearly, a safe approach could be to delay the mode change at the next idle time in the system, as done in the FRESCOR framework (52). However, the delay cold be too long and it is highly unlikely that the idle time occurs at the same time for all applications.

Recently, some mechanisms have been proposed to dynamically change the server models at run-time. For instance, de Olivera et al. (59) addressed the problem of dynamically reconfiguring reservation parameters, offering support for multi-moded and adaptive real-time applications. Valls et al. (165) presented an adaptation protocol based on the definition of a contract model for filtering peaks in resource demands. However, in both frameworks no schedulability guarantee is provided during reconfigurations. The FRESCOR project (73) has proposed mode change protocols for sporadic

servers, but they are not as general as the results presented in this paper, which can cope with arbitrary activation patterns.

Finally, in (155) Stoimenov et al. have tackled with the problem of adaptive resource reservation mechanisms in case of TDMA servers. Resource provisioning guarantees are investigated during the mode changes of the TDMA server paradigm.

1.4 Aim of the Thesis

In this dissertation we aim to support the claim that *dynamic real-time systems require adaptive resource reservation mechanisms*. In doing that, we will propose an effective component-based view of real-time systems to actively support system analysis through interface-based design methodologies. We will apply alternative approaches of modeling component of real-time systems such as tasks, schedulers and resource reservation mechanisms. We will also present examples to support the need of adaptive mechanisms. in case of complex and dynamic real-time systems. Those examples will outline scenarios and conditions where the classical (and static) resource reservation mechanisms are improper because unable to cope with changing conditions. Moreover, we will analyze the requirements and the costs of adaptive resource mechanisms in terms of the real-time guarantees that have to be offered to the system in any of its working condition. Besides, solutions to the adaptivity problem will be provided in terms of adaptive resource reservation mechanisms that effectively manages with the changing requirements of the systems.

1.5 Overview of the Dissertation

The thesis is composed by Chapter 2 where the most common real-time models are presented and detailed, including the real-time calculus framework. Feasibility and schedulability concepts are outlined together with a component-based view of real-time systems. Those are the basic elements of the analysis framework we are proposing. Chapter 3 describes examples of scheduling strategies applied to the problem of energy aware scheduling. Both real-time calculus and the classical real-time analysis are applied to propose efficient solutions to the problem. Resource reservation mechanisms are showed through those examples as well as the need for adaptive solutions in case of complex systems. Chapter 4 presents a survey and an analysis of resource reservation mechanisms. Among all the possible schemes, some reservation mechanisms are deeply investigated defining their resource provisioning guarantees to their real-time applications. Chapter 5 describes the real-time analysis in case of dynamic real-time systems. First, the case of multi-moded applications is presented and studied. The well known solutions are briefly mentioned while new solutions are detailed. Second, it is addressed the problem of multi-moded resource reservation mechanisms. Server and their resource provisioning mechanisms are investigated in case of mode changes, and in particular during the transition stage. Finally, Chapter 6 shows two significative examples of adaptive resource reservation mechanisms. One in case of distributed

systems and the other in case of adaptive servers. Conclusions and future works are then proposed in the last chapter of the dissertation.

Chapter 2

Real-Time System Modeling

The environment and the system models describe how the system is being used by the environment: how often will system functions be called, how much data is provided as input to the system, and how much data is generated by the system back to its environment. Real-time systems have functional and mainly temporal requirements that have to be represented in the model itself.

A real-time system applies computational resources to execute tasks and let them perform their job. All the system resources have to be considered, and the resource models have to provide information about the properties of the computing and communication resources that are available within a system, such as processor speed, communication bus bandwidth and storage amount. On the other side, tasks, scheduling mechanisms, server mechanisms and other elements composing real-time systems, apply the resources in order to execute and by that, to define the functional aspects of the systems.

The temporal requirements of the systems and the tasks composing the systems have to be fulfilled. Hence, the real-time modeling supports the analysis of real-time systems providing guarantees on the predictable behavior if the whole system.

2.1 Real-Time Analysis

The real-time theory models the elements of a real-time system and their requirements. It also applies the abstraction of those elements to scheduling and feasibility analysis frameworks in order to guarantee hard real-time or simply the quality of service requirements for complex systems.

The schedulability of a system depends on the scheduling policy and most of all on the available resource. Therefore, both in case of fixed priority and dynamic-priority scheduling the resource required to execute applications and the resource provided by the system are compared in order to derive feasibility conditions.

In the rest of the dissertation we will make use of t to indicate time instant, while γ is used to represent time intervals, because some of the analysis criteria refer to time instants, while others consider intervals.

2.1.1 Fixed-Priority Scheduling

In fixed priority scheduling any task has assigned a priority that the scheduler applies to order the task set and let at any time the highest priority runnable task actually run. There is a unique priority associated with each task, and all the jobs generated by a task have assigned this priority.

The absolute priority can be assigned in many ways, i.e. the rate monotonic (RM) policy where the deadline is assigned according to the period of the tasks (96), or deadline monotonic (DM) scheduling algorithm (98), which assigns priorities to tasks in inverse order of their relative deadline parameters. Without loss of generality, we assume that the tasks are indexed in decreasing order of priority: jobs of task τ_i have priority over the jobs of task τ_j for all i, j such that $1 \leq i < j \leq n$.

Definition 2.1.1 (Response Time)**.** *The response time R is the time (measured from the release time of a task) at which the task instance is completed.*

Audisley et al. have developed a necessary and sufficient method in order to compute the interference received by the task τ_i from its higher priority tasks, (8; 9). The basic idea is that in oder to compute the largest response time of τ_i, R_i, then the interference I_i has to be computed in the interval $[0, R_i]$. The interference is given by high priority tasks, $I_i = \sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil$, and the response time results $R_i = C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil C_j$. In order to get a result from the last equation and provide the maximum response time for a task, an iterative solution is obtained by

$$R_i^{(s)} = C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i^{(s-1)}}{T_j} \rceil C_j, \tag{2.1}$$

and the iteration ends whenever two consecutive R_i, (R_i^{s-1}, R_i^s) with the same value or $R_i^s > D_i$. Equation 2.1 is solved with a fixed point algorithm.

Definition 2.1.2 (Level-i Workload)**.** *The worst-case workload $w_i(t)$ of the i highest priority tasks in $[0, t]$ (level-i workload) is the total time the processor is i-busy in $[0, t]$.*

Given the i-th task, the cumulative workload on the processor made by that task and its high priority tasks (96) is provided by

$$w_i(t) = \sum_{j=1}^{i} = \lceil \frac{t}{Tj} \rceil C_j,$$

or equivalently

$$w_i(t) = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j, \tag{2.2}$$

where $hp(i)$ is the set of high priority tasks with respect to τ_i. It describes the total amount of time the processor is busy serving task τ_i and its high priority tasks.

Lehoczky et al. in (96) came out with the schedulability criteria for rate monotonic scheduling algorithms. by comparing the workload ad the total available resource task by task.

Theorem 2.1.3 (Workload RM Schedulability)**.** *Given the task set* $\Gamma = \{\tau_1, \ldots, \tau_n\}$

- τ_i *can be scheduled for all task phasing using the rate monotonic algorithm iff*

$$L_i = min_{t \in s_i} \frac{w_i(t)}{t} \leq 1,$$

where the elements of s_i *are the scheduling points for task* i, $s_i = \{kT_j \mid j = 1, \ldots, i;\ k = 1, \ldots, \lfloor \frac{T_i}{T_j} \rfloor\}$.

- *The entire task set* Γ *is schedulable for all task phasing iff*

$$L = max_{1 \leq i \leq n} L_i \leq 1.$$

The set of all schedulability point as been refined in order to reduce the complexity if such schedulability condition (25; 96).

2.1.2 Dynamic-Priority Scheduling

Under EDF, the analysis of periodic tasks with deadline less than the period can be carried out using the processor demand criterion (PDC) as introduced by Baruah et al. (21). In general, the processor demand of a task τ_i in an interval $[t_1, t_2]$ is the amount of processing time $g_i(t_1, t_2)$ requested by the instances of τ_i activated and that must be completed in that interval, that is $g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1, d_{i,k} \leq t_2} C_i$. By $r_{i,k}$ it is intended the release time of the k-th instance of task i, and $d_{i,k}$ denotes the absolute deadline of such a job. For the whole task set Γ, the processor demand in $[t_1, t_2]$, $g(t_1, t_2)$ is given by the sum of the processing time of the tasks composing the task set. The feasibility of the task set is guaranteed if and only if in any interval of time the processor demand does not exceed the available time, which is

$$\forall t_1, t_2 \quad g(t_1, t_2) = \sum_{i \in \Gamma} g_i(t_1, t_2) \leq (t_2 - t_1).$$

From (21) it comes the following schedulability criterion.

Theorem 2.1.4 (Processor Demand Criterion)**.** *A set of synchronous periodic tasks with relative deadline less than or equal to periods can be scheduled under EDF iff*

$$\forall t \in \mathcal{D} \quad \sum_{i=1}^{n} \lfloor \frac{L + T_i - D_i}{T_i} \rfloor \leq t,$$

where $\mathcal{D} = \{d_k \mid d_k \leq min(t^*, H)\}$, *and* $t^* = \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U}$; d_k *are the deadlines of the tasks.*

2.1.3 The Demand Function

The computational demand of a task set can be precisely described by the *demand bound function* (dbf), introduced by Baruah et al. (21) to express the total computation that must be executed by the processor in each interval of time when tasks are scheduled by EDF. For any given periodic task τ_i activated at time $t = 0$, its demand bound function $\mathsf{dbf}_i(t)$ in any interval $[0, t]$ is given by

$$\mathsf{dbf}_i(t) = \max\left\{0, \left(\left\lfloor \frac{t - D_i}{T_i} + 1 \right\rfloor C_i\right)\right\}.$$

Hence, the computational demand of a task set Γ of periodic tasks synchronously activated at time $t = 0$, can be computed as the sum of the individual demand bound functions of each task, that is:

$$\mathsf{dbf}_\Gamma(t) = \sum_{\tau_i \in \Gamma} \mathsf{dbf}_i(t).$$

By definition, the demand bound function provides an upper bound of the resource requested by the task set in each interval of time. The $\mathsf{dbf} = g$ once it has been fixed the first extreme of the interval. In case of FP scheduling policies is the concept of workload (96) to be applied in order to verify the schedulability, so it exploits the resource requirements of single task or task sets. For a task τ_i the workload $w_i(t)$ is the total amount of time the processor is busy to serve τ_i and its high priority tasks in any interval length of t. By extension, $w_0(t) = 0$ for all t, as from Equation 2.2. The workload to be applied in the schedulability conditions representing the equivalent of the demand bound function in case of FP.

2.1.4 Server Mechanisms

From the real-time guarantee point of view it is required that the aperiodic tasks do not interfere with the schedulability of the periodic ones. Server mechanisms allow to obtain those guarantees through server processes that manage resource request from aperiodic tasks. Most of those processes can be modeled by a periodic process which assigns its resource to its tasks every period, (142). Figure 2.1 show the typical server usage: to isolate application execution. The servers are described with parameters like the utilization factors U, periods T and more others.

Although with different peculiarities, a lot of servers can be modeled as periodic servers because they guarantee to provide Q (and no more than Q) unit of time each period P. Among the periodic servers we include the polling servers (36), the deferrable servers (97; 156) and the sporadic servers (147) which are classified as fixed priority scheduling periodic server; while among the dynamic priority periodic servers we recall the constant bandwidth servers (2).

Definition 2.1.5 (Periodic Server)**.** *A periodic server S is characterized by two parameters (Q, P) where Q is the maximum budget (or server capacity), and P is the server period. A server must guarantee that Q units of time are allocated in each period P to the served application, with $Q \leq P$.*

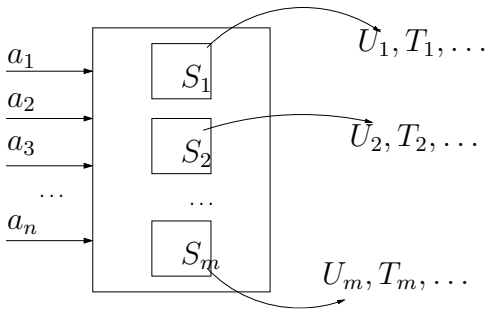


Figure 2.1: Server mechanisms providing isolation among the applications a.

2.1.5 The Supply Function

To derive a consistent definition of resource supply, we first introduce some definitions that are common in reservation-based scheduling theory (61; 101; 102; 104), even though presented with different terminology.

Definition 2.1.6 (Partition)**.** *A partition $\mathcal{P} \subseteq R$ is a countable union of non-overlapping intervals*

$$\mathcal{P} = \bigcup_{i \in N} [a_i, b_i) \quad a_i < b_i \leq a_{i+1}.$$

The characteristic function $I_{\mathcal{P}} \ : \ R \rightarrow 0, 1$ of the partition $\mathcal{P}$ is defined as $I_{\mathcal{P}}(t)$ having values in $0, 1$. If $I_{\mathcal{P}}(t) = 1$ then the resource is allocated to the partition at time t. A partition is periodic if it exist $per > 0$ such that $I_{\mathcal{P}}(t) = I_{\mathcal{P}}(t + per)$.

We distinguish between static partitions and dynamic partitions. A static allocation mechanism pre-computes the partitions off-line, and, at run-time, a dispatch mechanism will make use of a simple table to allocate the resource. On the other hand, a dynamic resource allocation mechanism uses some rule for dynamically allocating the resource (for example allocating Q time units every period P). Therefore, a dynamic algorithm may produce different partitions every time it is executed, depending on the arrival times and execution times of the application tasks. Moreover, these partitions are not necessarily periodic.

For a given partition, we define the minimum amount of time that is available to the application in every interval of length γ.

Definition 2.1.7 (Supply Function)**.** *Given a partition $\mathcal{P}$, we define the supply function $Z_{\mathcal{P}}(\gamma)$ as the minimum amount of time provided by the partition in every time interval of length $\gamma \geq 0$, that is*

$$Z_{\mathcal{P}}(\gamma) = min_{t_0 \geq 0} \int_{t_0}^{t_0+\gamma} I_{\mathcal{P}}(x)dx.$$

If the partition is static, the previous equation can be readily used to compute the supply function. However, if the partition is dynamic then it is not known in advance when the time will be allocated; just at runtime it wil be known. To extend

the definition of supply function also to servers allocating time by dynamic partitions, we refer to (121) and (108) by introducing the following definitions.

Definition 2.1.8 (Set of Partitions)**.** *Given a reservation S, we define $legal(R)$ as the set of partitions $\mathcal{P}$ that can be generated by the reservation S.*

Notice that if the server S allocates statically the time by a static partition $\mathcal{P}$, then $legal(S)$ is constituted by the unique element P. We now generalize the supply function to any server.

Definition 2.1.9 (Minimum Supply Function)**.** *Given a server S, its supply function $Z_S(\gamma)$ is the minimum amount of time provided by the server S in every time interval of length $[0, t)$,*

$$Z_S(\gamma) = min_{P \in legal(S)} Z_P(\gamma).$$

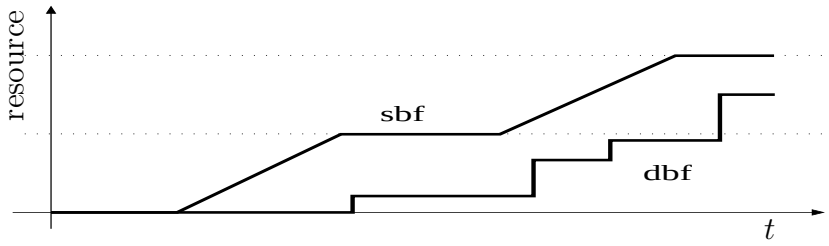


Figure 2.2: Demand bound function and supply bound function.

Given a partition $\mathcal{P}(\gamma)$, Bini et al. (68) defined the *supply bound function* $\mathsf{sbf}_{\mathcal{P}}(\gamma)$ of the partition $\mathcal{P}$, as the minimum amount of time provided by the partition in every interval of length $\gamma \geq 0$. That is,

$$\mathsf{sbf}_{\mathcal{P}}(\gamma) = min_{t_0 \geq \gamma} \int_{t_0}^{t_0+\gamma} R(x)dx.$$

The sbf is defined in the interval domain and gives the minimum amount of resource available in any interval of time. Hence, the supply bound function represents a lower bound of the actual resource provided by a server.

As an example, for a periodic model $S = (Q_S, P_S)$, its supply bound function $\mathsf{sbf}_S(\gamma)$ is defined to compute the minimum resource supply for every interval length γ as follows:

$$\mathsf{sbf}_S(\gamma) = \begin{cases} \gamma(k+1)(P_S Q_S) & \text{if } \gamma \in [(k+1)P_S 2Q_S, (k+1)P_S Q_S] \\ (k-1)Q_S & \text{otherwise,} \end{cases}$$

from (143).

2.1.5.1 Linear Approximation: $(slope, \Delta)$ model

The supply function $Z_S(\gamma)$ fully describes the computation time (the resource) provided by the server to any time consuming entity requiring it. However it is sometimes convenient to extract the most significant features from the supply function $Z_S(\gamma)$. A convenient abstraction of server is based on the only concepts of *bandwidth slope* and *delay* Δ. Note that for the slope we used *slope* in place of the α used in the original model to avoid clashes with the other symbols used in the dissertation. In the rest of the thesis we refer to *slope* to indicate the slope of a curve.

The bandwidth *slope* is the average slope of $Z_S(\gamma)$, formally defined as:

$$slope = lim_{\gamma \to \infty} \frac{Z_S(\gamma)}{\gamma} \tag{2.3}$$

The computation of the value of Δ requires some more efforts. Informally speaking, once we have computed *slope*, the delay Δ is the minimum horizontal displacement such that $slope(\gamma - \Delta)$ is a lower bound of $Z_S(\gamma)$. Formally:

$$\Delta = \inf\{d \geq 0 \ : \ \forall \gamma \geq t \quad Z_S(\gamma) \geq slope(\gamma - d)\} \tag{2.4}$$

It can be noticed that this abstraction is very simple, since it is constituted by only two parameters: the bandwidth and the delay. The advantage of a simple abstraction is that it can be placed on top of very different server mechanisms. On the other hand the price of simplicity is paid in terms of tightness: a more detailed description of the time provided by a server would allow a tighter usage of resources.

The $(slope, \Delta)$ representation provides a linear approximation to the resource supply sbf of a server which is called *bounded-delay function* (bdf) with

$$\forall \gamma \quad \mathsf{bdf}(\gamma) \leq \mathsf{sbf}(\gamma).$$

and

$$\mathsf{bdf}(\gamma) = \begin{cases} slope(t - \Delta) & \text{if } t \geq \Delta \\ 0 & \text{otherwise} \end{cases}$$

with $slope = lim_{\gamma \to \inf} \frac{\mathsf{sbf}(\gamma)}{t}$ and $\Delta = inf\{q \mid slope\gamma + q \leq \mathsf{sbf}(\gamma) \ \forall \gamma\}$.

The resource provided by a reservation server can also be described by the bounded-delay function (68; 103; 141) characterized by the tuple $(slope, \Delta)$, where *slope* is the resource provisioning rate of the server and Δ is longest interval with no resource provisioning.

The bounded-delay function of a resource partition R, $\mathsf{bdf}_R = (slope, \Delta)$ is defined as a linear approximation of the resource provisioning, and it holds

$$\mathsf{bdf}_R(t) \leq \mathsf{sbf}_R(t) \quad \forall t.$$

In the worst-case PS, DS, SS and CBS servers have the same resource supply bound function which is the one from the periodic server. The worst-case resource supply of a periodic server is provided whenever the resource is allocated at the beginning of

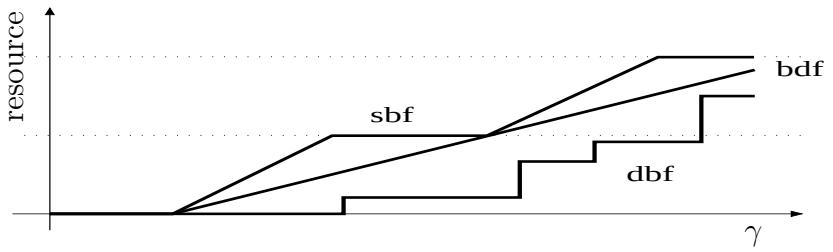


Figure 2.3: Demand bound function, supply bound function, and its bounded delay approximation in the interval domain.

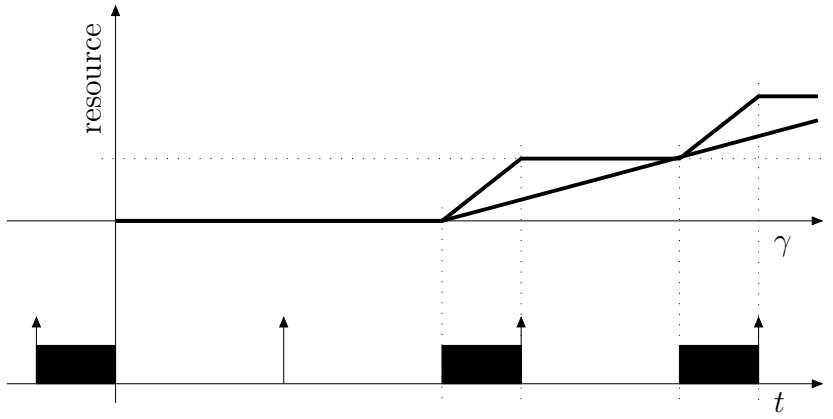


Figure 2.4: Supply bound function and bounded delay function for a periodic server in the interval domain t and scheduling in the time domain.

the first period P and at the end of all the other periods, see Figure 2.4 for details. In the interval domain, at most there is an interval of $2(P-Q)$ where no resource is provided, while after $2(P-Q)$ the server supplies the resource at a constante rate of $\frac{Q}{P}$ alternating the resource provisioning for an interval of Q and the resource holding for rest of the period $(P-Q)$.

The sbf of a periodic server, as the worst-case resource supply in time interval $[0,t) = \gamma$, is

$$\mathsf{sbf}_S(t) = \max\{0, (k-1)Q, t-(k+1)(P-Q)\} \tag{2.5}$$

with $k = \lceil \frac{t-(P-Q)}{P} \rceil$.
Such a curve is the shifted version of $\mathsf{sbf}_S(\gamma) = \left\lfloor \frac{\gamma}{P} \right\rfloor Q + \gamma - \left\lfloor \frac{\gamma}{P} \right\rfloor P - (P-Q)^+$ by $P-Q$. In case of periodic resource reservation mechanisms (periodic servers) $S = (Q_S, P_S)$, the resource supply bound can be approximated with the bounded-delay function

$$\mathsf{bdf}_S(t) = \begin{cases} slope(\gamma - \Delta) & \text{if } \gamma \geq \Delta \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

with *slope* and Δ that can be computed as

$$slope = \frac{Q_S}{P_S}$$

and

$$\Delta = 2(P_S - Q_S),$$

as illustrated in Figure 2.4.

Resource Bounded Delay Approximation The demand bound function dbf can be bounded with a bounded-delay linear approximation $(slope, \Delta)$ as well as the resource supply sbf. In that case $slope$ is the slope of the linear bound, while Δ is the offset of such a curve. By using

$$slope \stackrel{def}{=} \lim_{\gamma \to \infty} \frac{\mathsf{dbf}(\gamma)}{\gamma}, \tag{2.7}$$

$$\Delta \stackrel{def}{=} \sup_{\gamma \geq 0} \left\{ \gamma - \frac{\mathsf{dbf}(\gamma)}{slope} \right\}, \tag{2.8}$$

it is possible to prove that $\mathsf{dbf}(\gamma)$ is bounded by

$$\min\{0, slope \cdot \gamma + \Delta\} \geq \mathsf{dbf}(\gamma). \tag{2.9}$$

The resulting curve is the minimum upper bound of the demand bound function.

The level-i workload w_i can be approximated in a linear form applying the same reasoning as the previous one.

$$slope \stackrel{def}{=} \lim_{\gamma \to \infty} \frac{w_i(\gamma)}{\gamma}, \tag{2.10}$$

$$\Delta \stackrel{def}{=} \sup_{\gamma \geq 0} \{w_i - slope \cdot \gamma\} \tag{2.11}$$

It is possible to prove that $w_i(\gamma)$ is bounded by

$$\min\{0, slope \cdot \gamma + \Delta\} \geq w_i(\gamma). \tag{2.12}$$

2.1.6 Classical Feasibility Analysis

In general, a resource R is said to satisfy the resource demand of τ_i, In other words it is schedulable, if its resource supply satisfies its resource demand, $\mathsf{dbf}_i(t) \leq \mathsf{sbf}_R(t)$. The taks is then schedulable. With the dbf and sbf modeling the former scheduling criteria can be generalized in order to have what follows. Using the functions defined above, the EDF schedulability of a task set Γ within a reservation R can be guaranteed if and only if:

$$\forall t \quad \mathsf{dbf}_\Gamma(t) \leq \mathsf{sbf}_R(t). \tag{2.13}$$

Note that schedulability can also be checked using the linear bounded-delay function bdf, but, due to the approximation involved, the condition becomes only sufficient. An example of demand bound function, supply bound function and its bounded delay

approximation is illustrated in Figure 2.3. In that example the applications is feasible under EDF, since in each interval of time the amount of resource always exceeds the processor demand of the application.

In case of fixed priority scheduling, a task set Γ is schedulable within a reservation R if and only if

$$\forall i \quad \exists t \ \in \ schedP_i \quad w_i(t) \leq \mathsf{sbf}_R(t), \tag{2.14}$$

where $schedP_i$ is the set of testing points where the schedulability has to be checked (25).

With the bounded-delay approximation, the schedulability condition become only sufficient.

Proposition 2.1.10 (EDF Schedulability)**.** *The EDF schedulability of a task set* Γ *within a reservation* R *can be guaranteed if*

$$\forall t \quad \mathsf{dbf}_\Gamma(t) \leq \mathsf{bdf}_R(t). \tag{2.15}$$

Proposition 2.1.11 (FP Schedulability)**.** *A task set* Γ *is schedulable within a reservation* R *with FP scheduling policies if*

$$\forall i \quad \exists t \ \in \ schedP_i \quad w_i(t) \leq \mathsf{bdf}_R(t), \tag{2.16}$$

2.2 Real-Time Calculus

The increasing complexity of real-time and embedded systems has prompted the need for modeling and analysis techniques that go beyond those traditionally studied in the real-time systems literature. In this context, the Real-Time Calculus (RTC) framework proposed in (43; 159) and subsequently extended in (160; 170) is targeted towards analyzing heterogeneous real-time systems that process various types of streaming data. RTC is a worst-case analysis framework for real-time system based on the Network Calculus, (93). The main strength of RTC is a count-based abstraction, where arrival patterns of event streams are specified as constraints on the number of events that may arrive over any specified time interval. A collection of such constraints for different interval lengths are captured as curves which denote upper and lower bounds on the event arrival process. The service availability of computational resources is also specified in a similar fashion.

In an embedded system, an incoming event stream is typically processed by a sequence of tasks and system elements, that is the reason why event streams are associated to tasks activations and so tasks.

Arrival Curve A trace of an event stream can conveniently be described by means of a differential arrival function $R[s,t)$ that denotes the sum of events that arrive in the time interval $[s,t)$ with $s \leq \gamma < t$, with $R[s,s) = 0$, and with $s,t \in \mathbb{R}$. We make use of the cumulative arrival function $R(\gamma)$ that is defined as $R(\gamma) = R[0,\gamma)$ for all $\gamma \geq 0$. While any arrival function R always describes one concrete trace of an event stream, a tuple $\overline{a}(\gamma) = [\overline{a}^u(\gamma), \overline{\alpha}^l(\gamma)]$ of upper and lower arrival curves provides an event stream model, representing all the possible traces of an event stream. For this, the upper arrival curve $\overline{\alpha}^u(\gamma)$ provides an upper bound on the number of events that are seen on the event stream in any time interval of length γ, and analogously, the lower arrival curve $\overline{\alpha}^l(\gamma)$ provides a lower bound on the number of events in a time interval γ. In other words, in any time interval of length γ there will always arrive at least $\overline{\alpha}^l(\gamma)$ and at most $\overline{\alpha}^u(\gamma)$ events on an event stream that is then modeled by $\overline{\alpha}(\gamma)$.

Definition 2.2.1 (Arrival Curve)**.** *Let $R[s,t)$ denote the number of events that arrive on an event stream in the time interval $s \leq \gamma < t$. Then, R, α^u and α^l are related to each other by the following inequality*

$$\overline{\alpha}^l(t-s) \leq R[s,t) \leq \overline{\alpha}^u(t-s), \quad \forall t \geq s \geq 0$$

with $\overline{\alpha}^l(0) = \overline{\alpha}^u(0) = 0$.

The concept of arrival curves unifies many other common timing models of event streams. For example, a periodic event stream with period T can be modeled by a set of step functions where $\bar{\alpha}^u(\gamma) = \left\lfloor \frac{\gamma}{T} \right\rfloor + 1$ and $\bar{\alpha}^l(\gamma) = \left\lfloor \frac{\gamma}{T} \right\rfloor$. For a sporadic event stream with minimal inter arrival distance T and maximal inter arrival distance T', the upper and lower arrival curve is $\bar{\alpha}^u(\gamma) = \left\lfloor \frac{\gamma}{T} \right\rfloor + 1$, $\bar{\alpha}^l(\gamma) = \left\lfloor \frac{\gamma}{T'} \right\rfloor$, respectively. Moreover, for an event stream with period T, jitter J, and minimal inter arrival distance d, the upper arrival curve is $\bar{\alpha}^u(\gamma) = \min\{\left\lceil \frac{\gamma+J}{T} \right\rceil, \left\lceil \frac{\gamma}{d} \right\rceil\}$. Fig. 2.6 illustrates arrival curves for the above cases. For details, please refer to (159). Event-based arrival curves can be converted to workload-based arrival curves α by scaling with the best-case/worst-case execution demand of events. In this dissertation, we make use use of the workload-based interpretation and assume that each event has a fixed execution demand. More general concepts for characterizing these units are discussed in (115).

Service Curve Analogously to the differential arrival function $R[s,t)$ that is used to describe a concrete trace of an event stream, the concrete availability of a computation or communication resource (any resource in real-time systems) can be described by a differential service function $C[s,t)$ with $s,t \in \mathbb{R}$, where $C[s,t)$ denotes the sum of available resource units, e. g. processor cycles or transmittable bits on a bus, in the time interval $s \leq \gamma < t$, with $C[s,s) = 0$. Sometimes, we will also use the cumulative service function $C(\gamma)$ that is defined as $C(\gamma) = C[0,\gamma)$ for all $\gamma \geq 0$. The tuple $\beta(\gamma) = [\beta^u(\gamma), \beta^l(\gamma)]$ of upper and lower service curves provides a resource model. The upper service curve $\beta^u(\gamma)$ provides an upper bound on the available resources in any time interval of length γ, and the lower service curve $\beta^l(\gamma)$ provides a lower bound on the available resources in a time interval γ. And in other words again, in any time interval of length γ there will always be at least $\beta^l(\gamma)$ and at most $\beta^u(\gamma)$ resource capacity available on a resource that is modeled by $\beta(\gamma)$.

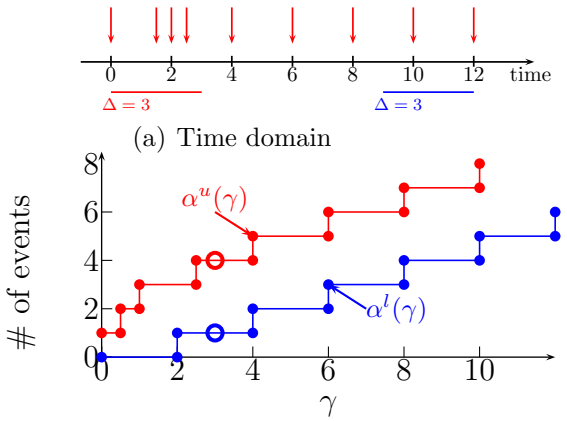


Figure 2.5: Graphical representation of a event stream with the trace and its curve in the interval domain.

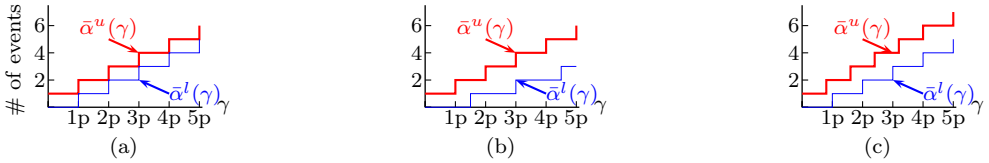


Figure 2.6: Examples for arrival curves, where (a) periodic events with period p, (b) events with minimal inter-arrival distance p and maximal inter-arrival distance $p' = 1.5p$, and (c) events with period p, jitter $j = p$, and minimal inter-arrival distance $d = 0.8p$.

Definition 2.2.2 (Service Curve)**.** *Let $C[s,t)$ denote the number of processing or communication cycles available from a resource over the time interval $s \leq \gamma < t$. Then C, β^u and β^l are related by the following inequality*

$$b^l(t-s) \leq C[s,t) \leq b^u(t-s), \quad \forall t \geq s \geq 0$$

with $b^l(0) = b^u(0) = 0$.

With the service curve abstraction $\beta(\gamma)$ it is possible to model any resource supply in the interval domain, including the bandwidth provisioning in WSNs or the computational resource provided by a processor to tasks or server.

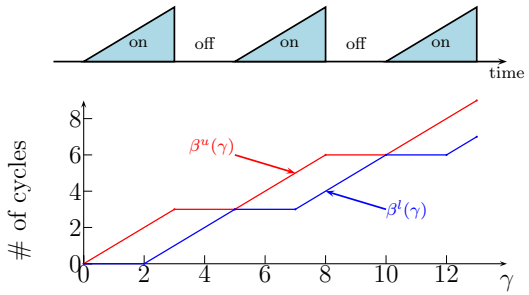


Figure 2.7: Graphical representation of a periodic service provisioning with the resource provisioning trace and its curve in the interval domain.

Together with the input curves, there are other important curve for a real-time element: the output curves, which are.

- the *Output Arrival Curve*. The outgoing arrival curves are those curves resulting from the processing a component served by a service $\beta(\gamma)$ does on its input arrivals $\alpha(\gamma)$, in any interval γ.

$$\begin{aligned}\alpha'^{l}(\gamma) &= min\{\beta^{l}(\gamma), \\ &\quad \inf_{0\leq\mu,\gamma}\{\sup_{\lambda\geq 0}\{\alpha^{l}(\mu+\lambda)-\beta^{u}(\lambda)\}+\beta^{l}(\gamma-\mu)\}\} \\ \alpha'^{u}(\gamma) &= min\{\beta_{i}^{u}(\gamma), \\ &\quad \sup_{\lambda>0}\{\inf_{0\leq\ \mu\leq\lambda+t}\{\alpha^{u}(\mu)+\beta^{u}(\lambda+t-\mu)\}-\beta^{l}(\lambda)\}\}\end{aligned}$$

- The *Remaining Service*. If an event stream with arrival curve $\alpha(\gamma)$ in any interval γ is processed by an abstract component on a resource with availability $\beta(\gamma)$, then the remaining resources that are not consumed by the abstract component can be bounded by the service curve:

$$\begin{aligned}\beta'^{l}(\gamma) &= \sup_{0\leq\lambda\leq\gamma}\{\beta^{l}(\lambda)-\alpha^{u}(\lambda)\} \\ \beta'^{u}(t) &= max\{\inf_{\lambda\geq\gamma}\{\beta^{u}(\lambda)-\alpha^{l}(\lambda)\},0\}\end{aligned}$$

In RTC important notions like delay and backlog define schedulability conditions.

- *Delay*: The maximum delay d_{max} experienced by an event on an event stream with arrival curve $\alpha(\gamma)$ that is processed on an element with service curve $\beta(\gamma)$, is bounded by:

$$d_{max} \leq sup_{\lambda\geq 0} inf\{\gamma \geq 0 \mid \alpha^{u}(\lambda) \leq \beta^{l}(\lambda+\gamma)\} \tag{2.17}$$

$$\stackrel{def}{=} Del(\alpha^{u},\beta^{l}) \tag{2.18}$$

  The delay is then the amount of time that an application has to wait in order to have the necessary amount of resource available and then execute properly. It can be seen as the application worst-case response time and the it can be applied to schedulability purposes.

- *Backlog*: The maximum number of backlogged events from the stream α that is waiting to be processed is given by the inequality

$$b_{max} \leq sup_{t\geq 0}\{\alpha^{u}(t)-\beta^{l}(t)\} \stackrel{def}{=} Buf(\alpha^{u},\beta^{l}). \tag{2.19}$$

In the performance model of a system, various performance measures can be computed analytically. For instance, for an FP component the maximum delay d_{max} experienced by an event is bounded by the previous relationship (Equation 6.19), and

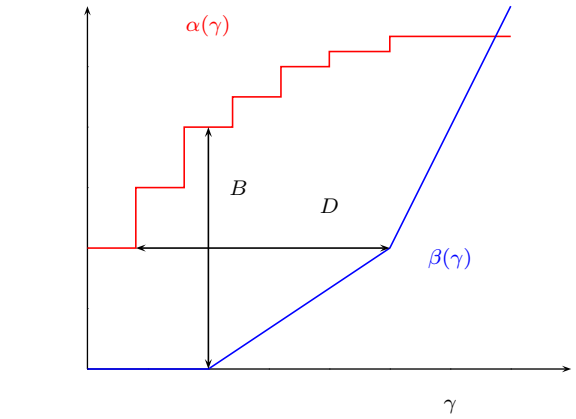


Figure 2.8: Graphical representation of delay and backlog and the horizontal and vertical distance respectively.

when processed by a sequence of components, the total end-to-end delay experienced by an event is bounded by

$$d_{max} \leq Del(\alpha^u, \beta_1^l \otimes \beta_2^l \otimes \ldots \otimes \beta_n^l)$$

(43; 93). Similarly, the maximum buffer space b_{max} required to buffer an event stream in front of such an FP component is bounded by Equation 2.19. When the buffers of consecutive components access the same shared memory, the total buffer space is bounded by

$$b_{max} \leq Buf(\alpha^u, \beta_1^l \otimes \beta_2^l \otimes \ldots \beta_n^l)$$

In case of EDF an order among the tasks and their execution cannot be decide a priori. They are scheduled with dynamic priority rules, so the FP analysis does not apply to those schedulability mechanisms. The schedulability analysis is then carried out composing the arrivals of the scheduling component $\alpha = \sum_{\tau_i \in \Gamma} \alpha_i$ in order to obtain the total resource demand such which is required by that scheduling element. The compositional rules will be clarified later on.

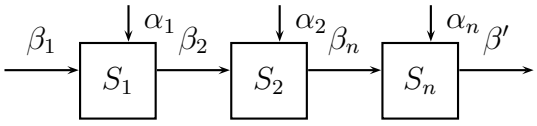


Figure 2.9: An example for fixed-priority scheduling

2.3 Component and Interface-Based Real-Time Systems

Recent trends depict real-time systems as component-base systems where the application tasks or dedicated HW/SW component models provide information about the

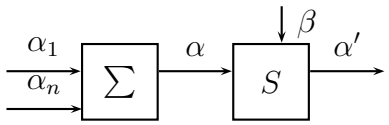


Figure 2.10: An example for earliest deadline first scheduling

processing semantics that are used to execute the various application tasks or to run the dedicated HW/SW components.

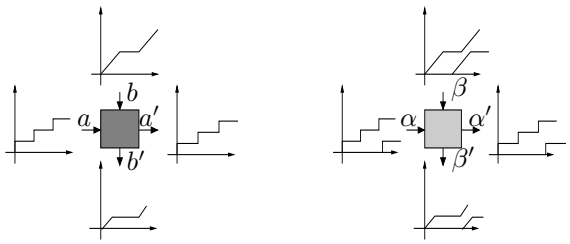


Figure 2.11: Generic component with input and output event streams and its RTC abstraction with arrival and service curves.

Imagine the simple component depicted in Figure 2.11. The component has two input variables p_1 and p_2, and one output variable p_3. What the component does is further described by a component description that could be expressed by the formula $p_3 = p_2 - p_1$. To put this component in a context, lets suppose that it is the abstraction of a concrete system component, and we want to analyze real-time properties of the concrete component. We could then interpret this component as follows: input variable p_1 describes the resource demand of an arriving event stream, while p_2 describes the resources that are available to process the arriving event stream. Output variable p_3 would then describe the resources that remained unused after processing the resource demand p_1.

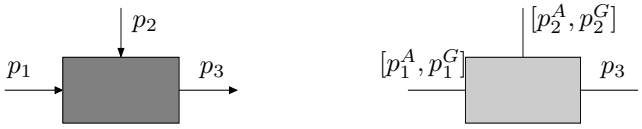


Figure 2.12: Generic component and its interface abstraction.

In contrast to the component description, that models what the component actually does, the component interface describes how the component can be used. A well designed component interface has to provide enough information to decide whether two or more components can work together properly in a system.

In our scenario, a real-time system is made by components which implement functional services with temporal requirements, expressed by a Real-Time Interface (RTI) (160; 166). Figure 2.12 depicts an example of a component and its component interface abstraction. To model RTI we make use of an approach similar to the Real-Time

Calculus formerly introduced, where a Real-Time Interface of a generic network component has input and output variables related to event streams (arrivals) and resource availability (services). The definition of Real-Time Interfaces follows the principles of Interface-based Design as described by de Alfaro and Henzinger in (57), and more recently in (58). Whereas more recent results relate to state-based interface languages such as interface automata and extensions towards the use of resources, the Real-Time Interfaces are based on stateless assume/guarantee (A/G) interfaces, see (57). We make use of the assume/guarantee paradigm to describe real-time components because closer to the classical real-time theory. In this section, we introduce some underlying principles of interfaces and Interface-based Design on an example of an simple imaginary component for real-time system design. Note that interfaces and all interface-related terms in this section are formally defined in (57) and (58).

Figure 2.12 depicts a component and its assume/guarantee interface. Although not depicted explicitly, this interface also has the two input variables p_1 and p_2 and the output variable p_3. The component interface puts a constraint on the environment through a predicate ϕ^I on its input variables: the environment is expected to provide inputs that satisfy ϕ^I. In return, the interface communicates to the environment a constraint ϕ^O on its output variables: it guarantees to provide only outputs that satisfy ϕ^O.

If our interface gets connected to other interfaces, these will provide output guarantees on their output variables, e.g. $a \leq \hat{p_1}^G$ and $b \geq \hat{p_2}^G$, and input assumptions on their input variables, e.g. $c \geq \hat{c}^A$. The interface relations $\hat{p_3}^G = \hat{p_2}^G - \hat{p_1}^G$, $\hat{p_1}^A = \hat{p_2}^G - \hat{p_3}^A$ and $\hat{p_2}^A = \hat{p_1}^G + \hat{p_3}^A$, then bring these different input assumptions and output guarantees into relation.

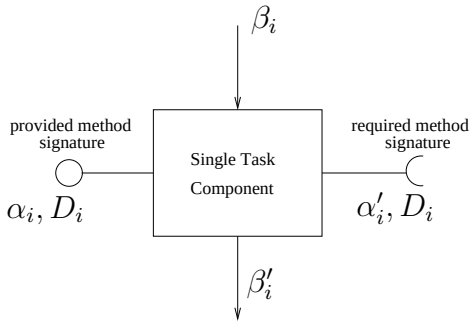


Figure 2.13: Real-time component.

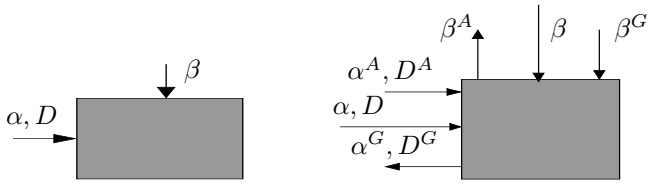


Figure 2.14: Generic real-time component and its assume/guarantee interface abstraction.

For interface-based design, we first need a set of interfaces, one for every component in the system we want to design. As with the components themselves, interfaces can then be composed into bigger interfaces by interconnecting an output of one interface U to an input of another interface V. The composed interface is then denoted as $U||V$. This composition is however only possible, if the two interfaces are semantically compatible. And if they are compatible, it is guaranteed that the two components belonging to the two interfaces can work together properly in the real system. Two interfaces U and V are semantically compatible if whenever one interface provides inputs to the other interface, then the output guarantee of the former implies the input assumption of the latter. In the closed case, where all inputs of U are outputs of V , and vice versa, U and V are compatible if the closed formula $\phi_U^O \wedge \phi_V^O \Rightarrow \phi_U^I \wedge \phi_V^I$ is true. In the open case on the other hand, where some inputs of U and V are left free, this formula has free input variables. U and V are then compatible if the above formula is satisfiable. This formula is then the input assumption $\phi_{U||V}^I$ of the composite interface $U||V$, as it encodes the weakest condition on the environment of $U||V$ to make U and V work together properly.

Interface-based design supports incremental design of systems, because interfaces can be composed one-by-one in any order. During the composition, the assumptions on the free input variables are getting increasingly tight, and if we eventually succeed to compose all component interfaces of a complete system, we are guaranteed that all components in the system work together properly.

The component composition can be derived from the electronic theory. Indeed, it is possible to distinguish among *serial* and *parallel* composition of real-time components; such a classification depends on the resource.

- In case of serial composition the same resource is passed from one component to the next one; just the residual resource is then passed to the next component.
- In case of parallel composition, different resources are passed to different components. The components process the same inputs because the output of one component is the input of the next one.

Figure 2.15 gives an example of the two possible composition schemes.

Besides composition, it is important to apply refinement with interfaces. Refinement of interfaces is very similar to sub-typing of classes in object oriented (OO) programming: a refinement of an interface must accept all inputs that the original interface accepts, and it may produce only outputs that the original interface specification allows. Hence, to refine an assume/guarantee interface, the input assumption can be weakened, and the output guarantee can be strengthened. This definition ensures that compatible interfaces can always be refined independently and still remain compatible. We then say that interface-based design supports independent implementability. In practice, this allows to outsource the implementation of different system components, or to replace existing implementations of sub-systems with different or new implementations.

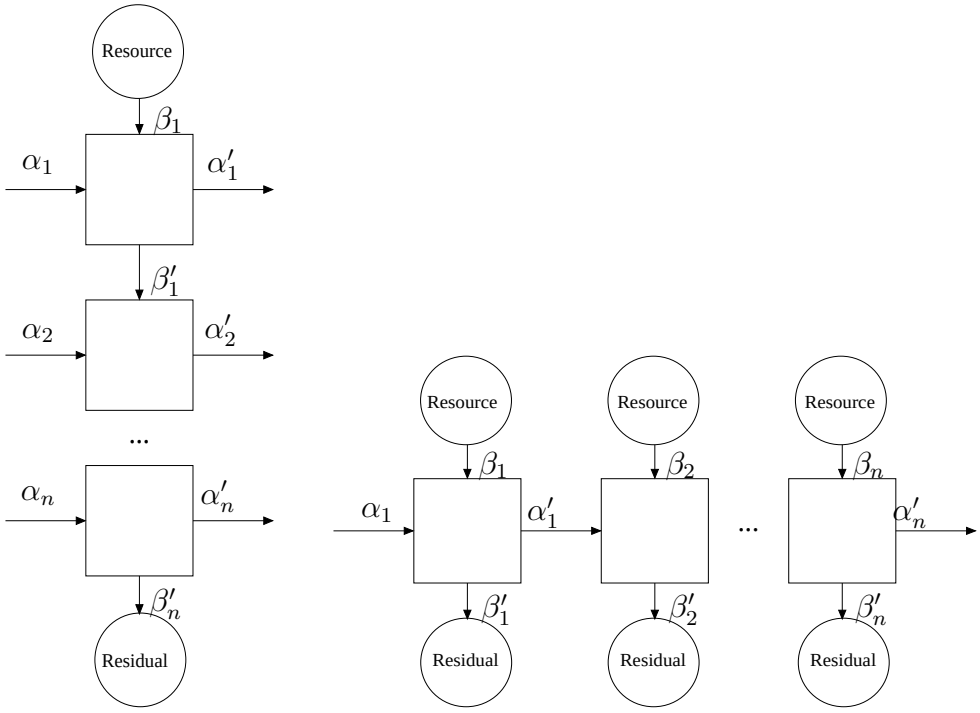


Figure 2.15: Series and parallel composition of real-time components.

2.3.1 Abstract Components and Real-Time Interfaces

In this dissertation, we model HW/SW real-time components by using abstract components as depicted in Figure 2.14. An event stream, represented by the arrival curve $\alpha(\gamma)$ with associated the maximum event delay D, triggers the component. A fully preemptable and independent tasks is instantiated at every event arrival and is processed greedily while being restricted by the resource availability, that is represented by the service curve $\beta(\gamma)$. Resources that are not consumed by the component are represented by the service curve $\beta(\gamma)$. Following the results from network calculus and real-time calculus, the following relations can be derived that describe such an abstract component.

With abstract components as defined above, scheduling policies on a resource can be expressed by the way the abstract resources $\beta(\gamma)$ are distributed among the different abstract components. For example, consider preemptive fixed priority scheduling: an abstract component A with the highest priority may use all resources, whereas an abstract component B with the second highest priority only gets resources that were not consumed by A. This is modeled by using the remaining service $\beta_A(\gamma)$ from A as input to B. For more details see (44).

A real-time interface may have input and output variables related to event streams (arrival variables) and resource availability (service variables). The output guarantee on an arrival variable contains the bounds $\alpha^G(\gamma)$ and D^G and the output predicate ϕ^O guarantees $\alpha(\gamma) \leq \alpha^G(\gamma)$, and $D \geq D^G$. The input assumption on the other hand contains the bounds $\alpha^A(\gamma)$ and D^A and the input predicate ϕ^I reflects the assumption

that $\alpha(\gamma) \leq \alpha^A(\gamma)$ and $D \geq D^A$. The value of a service variable consists of a service curve $\beta(\gamma)$. The output guarantee on a service variable contains the bound $\beta^G(\gamma)$, and the output predicate ϕ^O guarantees $\beta(\gamma) \geq \beta^G(\gamma)$. The input assumption contains the bound $\beta^A(\gamma)$ and the input predicate ϕ^I reflects the assumption $\beta(\gamma) \geq \beta^A(\gamma)$ for all γ.

In order to determine whether two Real-Time Interfaces are compatible, we need to check that $\phi^O \Rightarrow \phi^I$ is true for all connections. Two Real-Time Interfaces that are connected only via a single arrival variable are compatible if

$$(D^A \leq D^G) \wedge (\alpha^A(\gamma) \geq \hat{\alpha}^G(\gamma)) \;\; \forall\, \gamma \geq 0$$

and when connected only via a service connection they are compatible if

$$\beta^A(\gamma) \leq \beta^G(\gamma) \;\; \forall\, \gamma \geq 0$$

From this, we can generalize that two Real-Time Interfaces are compatible if the former two conditions are true for all internal arrival and service connections respectively, and if the input predicates of all open input variables are still satisfiable.

It is then possible to define composability among the components in terms of service and arrivals.

2.3.2 Real-Time Composability

Any real-time component consists of a real-time workload and a scheduling policy for the workload it manages. The analysis of such systems can be done compositionally using interfaces that abstract the timing requirements of components.

The Real-Time Interfaces that we introduce in this chapter not only expose enough information to decide on composability and compatibility with other component interfaces, but in addition they also change their assumptions and guarantees, following principles of constraint propagation. Which means that, in the real-time domain, the concept of *composability* can be translated into *schedulability*, and verified at composition time. Components are composable if their composition is schedulable which means that the resulting system is schedulable.

Recently, the concept of demand bound functions $\mathsf{dbf}(\gamma)$ and supply bound functions $\mathsf{sbf}(\gamma)$ got about in the area of compositional scheduling, see e.g. (119; 141; 143) or (144). With these functions, a component i is considered to be schedulable, if $\forall\, \gamma \;\; \mathsf{dbf}_i(\gamma) \leq \mathsf{sbf}_i(\gamma)$, where dbf_i is the resource demanded by the component and sbf_i is the resource demand the component receives. This concept also exists in the theory of Real-Time Interfaces, where the bound $\beta^A(\gamma)$ of the input assumption on a service connection can be interpreted as a demand bound function $\mathsf{dbf}(\gamma)$, and the bound $\beta^G(\gamma)$ of the output guarantee on a service connection can be interpreted as a supply bound function $\mathsf{sbf}(\gamma)$. Then, the compatibility requirement on a service connection equals the above described schedulability requirement $\beta^A(\gamma) \leq \beta^G(\gamma)$. This way, in terms of resource the service request never exceeds the service provisioning.

A schedulable/composable component is the one depicted in Figure 2.17. Using the RTC and its curve modeling it is possible to derive schedulability conditions similar to the ones of the classical real-time analysis.

The resource demand of a task i is the the resource amount the task request on order to be scheduled properly, hence not missing its deadline. It is also the minimum resource a scheduling component has to provide in order to schedule its workload. It is defined from the arrival curve of the task by considering its deadline D, as $\alpha_i^d = \alpha_i(\gamma - D_i)$. Given a task set Γ, the total resource demand is $\alpha_\Gamma^d \equiv \alpha^d = \sum_{i\in\Gamma} \alpha_i^d = \sum_i \alpha_i(\gamma - D_i)$. Arrival curves, demand curves and service curves can be bounded by bounded-delay functions in the former definition.

The EDF schedulability of a task set Γ within a reservation R can be guaranteed if and only if:

$$\forall t \quad \alpha_\Gamma^d(t) \leq \beta_R(t). \tag{2.20}$$

Note that schedulability can also be checked using the linear bounded-delay function bdf, but, due to the approximation, the condition becomes only sufficient. An example of demand bound function, supply bound function and its bounded delay approximation is illustrated in Figure 2.3, where the applications is feasible under EDF, since in each interval of time the amount of resource always exceeds the processor demand of the application.

In case of fixed priority scheduling the schedulability applies the level-i workload of Equation 2.2. Each task contribution to the level-i workload is given by the arrival curve of the task itself; for the i-task $w_i(\gamma) = \alpha(\gamma) + \sum_{hp(i)} \alpha_j(\gamma)$. A task set Γ is schedulable within a reservation R if and only if

$$\forall i \quad \exists \gamma \ \in \ schedP_i \quad w_i(\gamma) \leq \mathsf{sbf}_R(\gamma), \tag{2.21}$$

where $schedP_i$ is the set of testing points where the schedulability has to be checked (24). With the bounded-delay approximation, the schedulability condition become only sufficient. The workload curve can be bounded by bounded-delay functions as well as all the other curves.

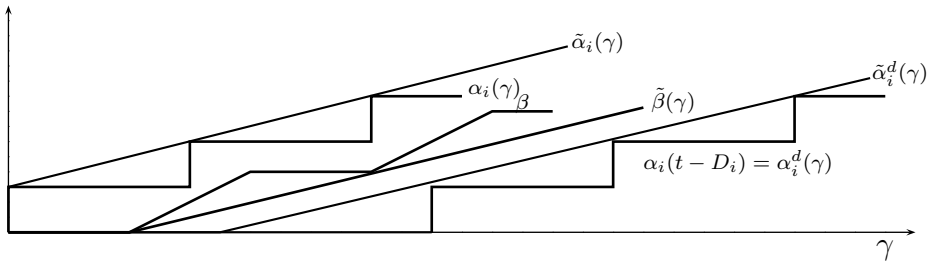


Figure 2.16: Arrival and demand curves of the i-th task together with the service curve and their bounded-delay approximations.

Linear bounding eases the curve representation but sensibly reduces the accuracy of the analysis.

Workload Bounding The arrival curve $\alpha(\gamma)$ can be bounded (upper bounded) with a linear curve with the *slope* and the initial offset of such a curve Δ. The initial offset is required in order to bound possible bursts that can happen to the task arrival, as in case of aperiodic tasks.

$$slope \stackrel{def}{=} \lim_{\gamma \to \infty} \frac{\alpha(\gamma)}{\gamma}, \tag{2.22}$$

$$\sigma \stackrel{def}{=} \sup_{\gamma \geq 0} \{\alpha - slope \cdot \gamma\}, \tag{2.23}$$

it is possible to prove that $\alpha(\gamma)$ is bounded by:

$$\min\{0, slope \cdot \gamma + \sigma\} \geq \alpha(\gamma). \tag{2.24}$$

The arrival curve is exactly the task workload mentioned by Lehoczky et al. in (96), while the linear bounding of the arrival curve is described by the tuple $(slope, \sigma)$ as bounded-delay functions where *slope* and σ are defined as the Equation 2.22 and Equation 2.23.

Resource Bounding The linear bounding of the service and demand curves are again bounded-delay functions with slope and initial delay, defined as

$$slope \stackrel{def}{=} \lim_{t \to \infty} \frac{\beta(\gamma)}{\gamma}, \tag{2.25}$$

$$\Delta \stackrel{def}{=} \sup_{t \geq 0} \left\{ \gamma - \frac{\beta(\gamma)}{slope} \right\}, \tag{2.26}$$

respectively. The service curve has to be lower bounded by bounded-delay functions

$$\tilde{\beta} = max\{slope, \Delta \mid slope \cdot (\gamma - \Delta) \leq \beta(\gamma)\}$$

Even $\alpha^d(\gamma)$ can be bounded by the delay-bound function $\tilde{\alpha}^d$ modeled with the tuple $(slope, \Delta)$ which is composed by the slope and the maximum delay of such a curve, R and Δ respectively. The demand curve represents the minimal resource required to schedule the event stream. For schedulability reason that will be detailed later, the demand curve has to be upper bounded by bounded-delay curves

$$\tilde{\alpha}^d = min\{slope, \Delta \mid slope(\gamma - \Delta) \geq \alpha^d(\gamma)\}.$$

Figure 2.16 depicts the RTC curves and their linear approximations.

Using the delay concept, Disequation 6.19, it is possible to define the schedulability of task sets which depends on the scheduling policy applied. Those conditions are equivalent to the previous ones.

Theorem 2.3.1 (EDF Delay Schedulability)**.** *Given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ with each task τ_i described with α_i and a timing constraint D_i (α_i^d), served with a resource*

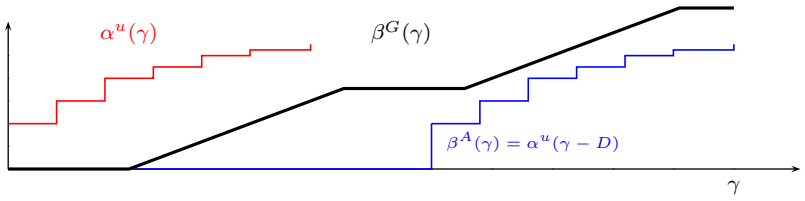


Figure 2.17: A schedulable application.

β. Γ is schedulable under an EDF scheduling policy if and only if the cumulative delay experienced by the task set in each task is less than or equal to 0,

$$d \leq 0, \tag{2.27}$$

with the delay bounded considering the cumulative demand curve $\alpha^d = \sum_{\tau_i \in \Gamma} \alpha_i^d$ and the service available

$$d \leq \sup_{t \geq 0} \{\inf\{\gamma \geq 0 \mid \alpha^d(t) \leq \beta(t+\gamma)\}\}.$$

Proof. The demonstration comes from the processor demand criterion (PDC) (21). The processor demand of a task set is represented by the cumulative demand curve $\alpha^d = \sum_{\tau_i \in \Gamma} \alpha_i^d$, and the task set is schedulable iff the demand is less than or equal to the service available, Disequation 2.15. Which means that the delay among the curves

$$d \leq \sup_{t \geq 0} \{\inf\{\gamma \geq 0 \mid \alpha^d(t) \leq \beta(t+\gamma)\}\},$$

has to be less than or equal to 0. That conclude the demonstration. □

Theorem 2.3.2 (FP Delay Schedulability)**.** *Given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ served with a resource β, and each task τ_i described with α_i and a timing constraint D_i. Γ is schedulable under a fixed priority scheduling policy if and only if the delay experienced by each task is less than or equal to its deadline,*

$$\forall i \quad d_i \leq D_i, \tag{2.28}$$

with the delay bounded considering the level-i workload and the service available

$$d_i \leq \sup_{t \geq 0} \{\inf\{\gamma \geq 0 \mid w_i(t) \leq \beta(t+\gamma)\}\}.$$

Proof. The demonstration comes from the Response Time Analysis (RTA) (96). The response time of a task τ_i is obtained by considering the level-i workload $w_i = \sum_{j \in hp(i)} \alpha_i$. Such a response time has to be less than or equal to the task deadline D_i. The delay, computed using the level-i workload is the response time of task τ_i

$$d_i \leq \sup_{t \geq 0} \{\inf\{\gamma \geq 0 \mid w_i(t) \leq \beta(t+\gamma)\}\}.$$

According to the RTA, every task must have a response time (or delay) less than or equal to its deadline. That conclude the demonstration. □

The same theorems can be carried out (as sufficient condition only) by applying the bounded delay approximations of the curves.

Chapter 3

Resource Reservation and Schedulability Analysis

In this chapter there are presented examples about real-time scheduling. Those examples refers either to the classical real-time representation or the RTC one. It is the case of real-time theory applied to specific scenarios.

Mainly, the examples tackle with the energy aware scheduling problem where the scheduling takes into account energy issues in order to reduce the energy consumption of embedded systems. Thus, the real-time analysis couple the classical scheduling and energy constraints improving the efficiency of the devices. In the following we will show the real-time analysis (deterministic - the classical one - and the non deterministic - the RTC) applied to solve such problem.

Examples and related papers allow us to show the effective of the modeling framework that the real-time community consistently applies.

3.1 Power Management for Hard Real-Time Systems

Power dissipation has been an important design issue in a wide range of computer systems in the past decade. Power management with energy efficiency considerations is not only useful for mobile devices for the improvement on operating duration but also helpful for server systems for the reduction of power bills.

Two major sources of power consumption of a CMOS circuit are dynamic power consumption due to switching activities and static power consumption mainly due to leakage current (84). For micrometer-scale semiconductor technology, dynamic power dominates the power consumption of a processor. However, as modern VLSI technology is scaling down to deep sub-micron domain, the tremendous amount of transistors integrated within a chip consumes significantly more static power. The leakage current that originates in the dramatic increase in both sub-threshold current and gate-oxide leakage current is projected to account for as much as 50 percentage of the total power dissipation for high-end processors in 90 *nm* technologies (11). The ITRS expects static

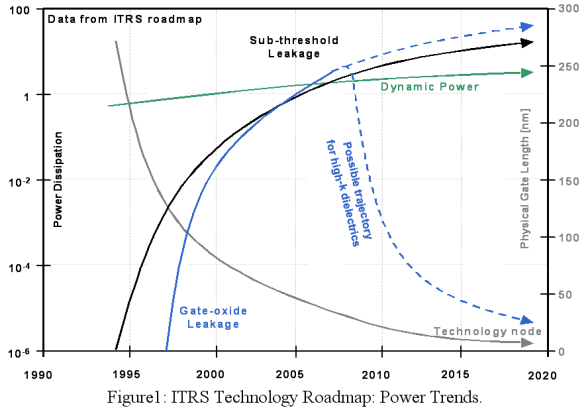


Figure 3.1: Energy consumption trends form ITRS.

power in the future will be much greater than their calculated value due to variability and temperature effects (83), as depicted in Figure 3.1.

It has been shown in (7) that for server systems the electricity cost remains significant even if servers do not always operate with the maximum power consumption. For mobile devices, ITRS reports the slow growth of energy density of batteries lacks far behind the tremendous increase of demands (83). Because of these facts, power consumption becomes one of the first-class design concerns for modern computer systems.

The dynamic voltage scaling (DVS) technique was introduced to reduce the dynamic energy consumption by trading the performance and the computational resource provided for energy savings. For DVS processors, a higher supply voltage, generally, leads not only to a higher execution speed/frequency but also higher power consumption. As a result, DVS scheduling algorithms, e.g., (12; 175; 176), tend to execute events as slowly as possible, without any violation of timing constraints. On the other hand, dynamic power management (DPM) can be applied to control the change of system mode to consume less leakage power, e.g., to a sleep mode. For DVS systems with non-negligible leakage power consumption, to minimize the energy consumption for execution, there is a *critical speed*, in which executing at any speed lower than the critical speed consumes more energy than at the critical speed (46; 84). However, returning from the sleep mode has timing and energy overheads, due to the wakeup/shutdown of the processor and data fetch in the register/cache. For example, the Transmeta processor in 70nm technology has $483\mu J$ energy overhead and less than 2 msec timing overhead (84).

For non-DVS systems with the sleep mode, Baptiste (15) proposes an algorithm based on dynamic programming to control when to turn on/off the system for aperiodic real-time events with the same execution time. For multiple low-power modes, Augustine et al. (10) determine the mode that the processor should enter for aperiodic real-time events and propose a competitive algorithm for on-line use. Swaminathan et al. (157; 158) explore dynamic power management of real-time events in controlling shutting down and waking up system devices for energy efficiency. To aggregate the idle time for energy reduction, Shrivastava et al. (146) propose a framework for code transformations.

Leakage-aware scheduling has also been recently explored on DVS platforms, such as (45; 46; 84; 85; 86; 94). In particular, researches in (46; 84; 94) propose energy-efficient scheduling on a processor by procrastination scheduling to control when to turn off the processor. Jejurikar and Gupta (86) then further consider real-time events that might complete earlier than their worst-case estimation. Fixed-priority scheduling is also considered by Jejurikar and Gupta (85) and Chen and Kuo (45). For uniprocessor scheduling of aperiodic real-time events, Irani et al. (81) propose a 3-approximation algorithm for the minimization of energy consumption. Niu and Quan (124) apply similar procrastination strategies for periodic real-time events with leakage considerations. The basic idea behind the above results is to execute at some speed (mostly at the critical speed) and control the procrastination of the real-time events as long as possible so that the idle interval is long enough to reduce the energy consumption.

In the following examples we show how to reduce the energy consumption of embedded devices while satisfying the real-time or quality of service (QoS) constraints. We consider systems (or devices) with active, standby, and sleep modes with different power consumptions. Similar to the approaches in (45; 84; 85; 86), for systems with DVS capability, we assume that the execution of events is at the critical speed and explore the energy reduction by applying DPM for reducing the energy consumption for idling.

Most of the above approaches require either precise information of event arrivals, such as systems with periodic real-time events (45; 46; 84; 85; 86; 94) or aperiodic real-time events with known arrival time (10; 15; 81). However, in practical, the precise information of event arrival time might not be known in advance since the arrival time depends on many factors. When the precise information of event arrivals is unknown, to our best knowledge, the only known approach is to apply the on-line algorithms proposed by Irani et al. (81) and Augustine et al. (10) to control when to turn on the system. However, since the on-line algorithms in (10; 81) greedily stay in the sleep mode as long as possible without referring to incoming events in the near future, the resulting schedule might make an event miss its deadline.

To model such irregular events, we make use of real-time calculus in order to characterize events with arrival curves by evaluating how often system functions will be called, how much data is provided as input to the system, and how much data is generated by the system back to its environment. Therefore, schedulability analysis can be done based on the curve abstraction, which are the arrival curves for the event streams and the service curves for the resource. For scheduling event streams based on Real-Time Calculus in DVS systems under buffer constraints, Maxiaguine et al. (114) develop adaptive algorithms to control the execution speed dynamically at periodic intervals of predefined length without exploiting the possibility for finding the optimal interval length with respect to the power consumption.

Distinct from the on-line adaptive algorithms in (114), to reduce the run-time overhead for determining when to perform mode changes, first we propose first off-line algorithms to derived optimal or approximated solutions to periodic power management for controlling when to change the system mode periodically. The light run-time overhead of the periodic power management schemes is very suitable for devices that only have limited power on computation. For scheduling one event stream under the

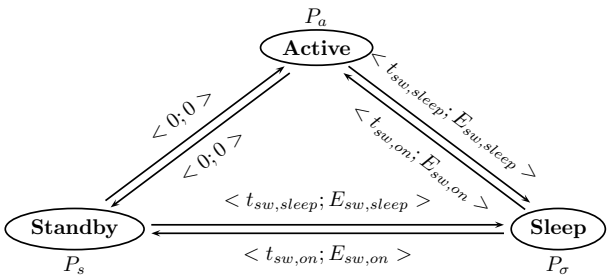


Figure 3.2: Example for state transit, where the tuple one each transit is the timing overhead and energy overhead.

real-time or QoS constraints, we develop an approach to derive optimal solutions and another to derive approximated solutions with lower complexity. Then, by applying the Modular Performance Analysis (173), we extend the developed approaches to cope with multiple event streams. To demonstrate the performance of the proposed approaches, several case studies are explored, in which the results reveal the effectiveness of our approaches.

Hardware Model We consider a system (or a device) that has three power consumption modes, including `active, standby,` and `sleep` modes. The power consumption in the sleep mode is P_σ. To serve an event, the system must be in the active mode with power consumption P_a, in which $P_a > P_\sigma$. Once there is no event to serve, the system can enter the sleep mode. However, switching from the sleep mode to the active mode takes time, denoted by $t_{sw,on}$, and requires additional energy overhead, denoted by $E_{sw,on}$. To prevent the system from frequent mode switches, the system can also stay in the standby mode. The power consumption P_s in the standby mode, by definition, is less than P_a and is more than P_σ. In (77; 78; 79), we assume that switching between the standby mode and the active mode has negligible overhead, compared to the other switches, which is the same as the assumption in (174; 178). Moreover, switching from the active (also standby) mode to the sleep mode takes time, denoted by $t_{sw,sleep}$, and requires additional energy overhead, denoted by $E_{sw,sleep}$. Figure 3.2 illustrates the state diagram of these three modes. For simplicity, once the system issues a mode switch from one mode to another mode, we assume that the power consumption of the system before the system enters the new mode is P_σ.

The model is based on curves, service and arrival curves as interfaces of real-time scheduling components.

Scheduling Policies For scheduling event streams, we consider the fixed priority scheduling and the earliest-deadline-first scheduling. For FP scheduling, event streams are prioritized a priori. Once an event of an event stream arrives to the system, the priority of the event is set to the pre-defined priority of the event stream. For EDF scheduling, the highest priority is given to the event with the earliest deadline. For both FP and EDF scheduling, the system executes the incomplete event with the highest priority. If there are more than one event with the highest priority, we break ties

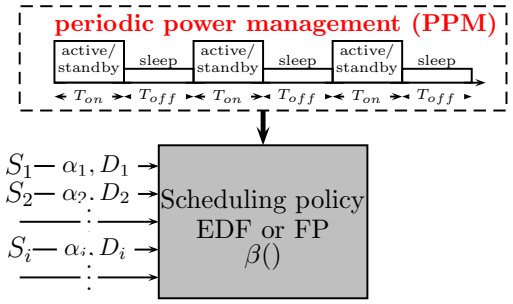


Figure 3.3: The abstract model of the periodic power management problem.

by applying the first-come-first-serve (FCFS) strategy. Therefore, events in the same event stream will be executed in the FCFS manner.

3.1.1 Periodic Power Management

We explore how to efficiently and effectively minimize the energy consumption to serve a set of event streams $\mathcal{S}$ under the real-time or QoS requirements. Of course, one could determine when to transit between modes to reduce the energy consumption dynamically, but the periodic approach represents the first strep toward adaptive solution to be applied n-line.

In this dissertation we recall a *Periodic Power Management schemes* (PPM), introduced in (78). That model is abstractly illustrated in Figure 3.3, in which the power management is done by analyzing the arrival curves of event streams $\mathcal{S}$ statically. Specifically, the periodic power management schemes first decide the period $T = T_{on} + T_{off}$ for power management, then switch the system to the standby mode for T_{on} time units, following by T_{off} time units in the sleep mode. Therefore, given a time interval L, where $L \gg T$ and $\frac{L}{T}$ is an integer, suppose that $\gamma_i(L)$ is the number of events of event stream S_i served in interval L. If all the served events finish in time interval L, the energy consumption $E(L, T_{on}, T_{off})$ by applying the PPM is

$$\begin{aligned} E(L, T_{on}, T_{off}) &= \frac{L}{T_{on} + T_{off}}(E_{sw,on} + E_{sw,sleep}) \\ &\quad + \frac{L \cdot T_{on}}{T_{on} + T_{off}} P_s + \frac{L \cdot T_{off}}{T_{on} + T_{off}} P_\sigma \\ &\quad + \sum_{S_i \in \mathcal{S}} c_i \cdot \gamma_i(L)(P_a - P_s) \\ &= \frac{L \cdot E_{sw}}{T_{on} + T_{off}} + \frac{L \cdot T_{on}(P_s - P_\sigma)}{T_{on} + T_{off}} \\ &\quad + L \cdot P_\sigma + \sum_{S_i \in \mathcal{S}} c_i \cdot \gamma_i(L)(P_a - P_s) \end{aligned}$$

where E_{sw} is $E_{sw,on} + E_{sw,sleep}$ for brevity.

As a result, for an L that is sufficiently large, without changing the scheduling policy, the minimization of energy consumption $E(L, T_{on}, T_{off})$ is to find T_{on} and T_{off} such that the *average idle power consumption* $P(T_{on}, T_{off})$

$$P(T_{on}, T_{off}) = \frac{E_{sw} + T_{on}(P_s - P_\sigma)}{T_{on} + T_{off}} \tag{3.1}$$

is minimized. The problem considered in (78) is that given a set of event streams $\mathcal{S}$ under the real-time or QoS requirements, the objective of the studied problem is to find a periodic power management characterized by T_{on} and T_{off} that minimizes the average standby power consumption, in which the response time of any event of event stream S_i in $\mathcal{S}$ must be no more than D_i.

3.1.2 One Event Stream

For event streams described by real-time calculus, one can apply real-time interface (160) to verify whether a system can provide guarantee output service $\beta^G(\gamma)$. Correspondingly, to guarantee that all events in one event stream can be processed while respecting all timing constraints, the event stream demands a service bound $\beta^A(\gamma)$. To satisfy the required deadline for the i-th event stream D_i, $\beta^A(\gamma)$ can be computed as $\beta^A(\gamma) = \alpha_i^u(\gamma - D_i)$. To check the schedulability of event stream S_1 in the system, the following predicate has to be true:

$$\beta^G(\gamma) \geq \beta_i^A(\gamma), \qquad \forall \gamma \geq 0$$

For PPM with specified T_{on} and T_{off}, the guarantee service of the system can be refined as:

$$\beta^G(\gamma) = \max\left(\left\lfloor \frac{\gamma}{T_{on} + T_{off}} \right\rfloor \cdot T_{on}, \right. \\ \left. \gamma - \left\lceil \frac{\gamma}{T_{on} + T_{off}} \right\rceil \cdot T_{off}\right) \tag{3.2}$$

For the rest of this section, we are going to present our schemes to find the pair $(T_{on}, T_{off}) \in \mathbb{R}^+ \times \mathbb{R}^+$ to minimize the average idle power consumption such that the service constraint $\beta^G(\gamma)$ is satisfied, and, hence, all events in stream $\mathcal{S}_\rangle$ have response time shorter than the timing constraint D_i.

Reviewing the formulation of the average idle power consumption $P(T_{on}, T_{off}) = \frac{E_{sw}+T_{on}(P_s-P_\sigma)}{T_{on}+T_{off}}$, there are two cases. (1) If $\frac{E_{sw}}{P_s-P_\delta} \geq T_{off}$, we know that $P(T_{on}, T_{off})$ is minimized when T_{on} is set to ∞. (2) If $\frac{E_{sw}}{P_s-P_\delta} < T_{off}$, the minimal T_{on} under the service constraint $\beta^G(\gamma)$ minimizes the average idle power consumption $P(T_{on}, T_{off})$. In this sense, $\frac{E_{sw}}{P_s-P_\delta}$, can be seen as the break-even time of the system. Our approaches proposed in (78) paper are based on (1) the finding of the minimal T_{on} under the service constraint $\beta^G(\gamma)$, provided that T_{off} is given, and (2) the exploration of the best T_{off}. One could also derive solutions in another direction by searching the best T_{off} for a specified T_{on} along with the exploration on T_{on}, but the procedure would be more complicated.

3.1.2.1 Finding the Minimal T_{on}

By the service guarantee curve β^G in (3.2), the service demand curve $\beta^A = \alpha_1^u(\gamma - D_1)$, and the schedulability definition in (3.7), the minimal T_{on} to fulfill the schedulability requirement in terms of a given T_{off} can be defined as:

$$T_{on}^{\min} = \min\big\{T_{on} : \beta^G(\gamma) \geq \beta^A(\gamma),\, \forall\gamma \geq 0\big\}. \tag{3.3}$$

To our best knowledge, there is no explicit form to compute T_{on}^{min}. Furthermore, due to the complex shape of the arrival curves, exhaustive testing of (3.7) is the only way to determine the minimum from all possible T_{on}.

Instead of calculating the exact $T_{on}^{\min}$, we propose an alternative approach, namely bounded-delay approximation, to find an approximated minimal $\tilde{T}_{on}$. The bounded-delay approach, on one hand, can reduce the computational complexity of finding the minimal $\tilde{T}_{on}$, and, on the other hand, can provide means to solve the PPM problem efficiently.

The basic idea of the proposed approach is to compute a minimal *bounded-delay function* $\beta^{A'}(\gamma)$, then derive the minimal T_{on} based on $\beta^{A'}$. A bounded-delay function $\mathsf{bdf}(\gamma, slope, T_{off})$, defined by the slope *slope* and the bounded-delay T_{off} for interval length γ, is $\max\{0, slope \cdot (\gamma - T_{off})\}$.

For a given bounded-delay function with slope *slope* and bounded-delay T_{off}, we can construct a PPM with

$$\tilde{T}_{on} = \frac{slope \cdot T_{off}}{1 - slope}$$

such that the resulting service curve of the PPM is no less than the minimal $\mathsf{bdf}(\gamma, slope, T_{off})$ for any $\gamma \geq 0$. Figure 3.8 illustrates an example to derive $\tilde{T}_{on}$. From above definitions, we can have the following lem.

Lemma 3.1.1. *For specified $T_{off} > 0$ and $0 < slope \leq 1$:*
(1) If $\mathsf{bdf}(\gamma, slope, T_{off}) \geq \alpha_1^u(\gamma - D_1)$, then, for any $slope' > slope$, $\mathsf{bdf}(\gamma, slope', T_{off}) \geq \alpha_1^u(\gamma - D_1)$.
(2) If $\mathsf{bdf}(\gamma, slope, T_{off}) < \alpha_1^u(\gamma - D_1)$, then, for any $slope' < slope$, $\mathsf{bdf}(\gamma, slope', T_{off}) < \alpha_1^u(\gamma - D_1)$.

Proof. This is simply based on the construction of the bounded delay function. □

By (3.4) and Lemma 3.1.1, finding the minimum *slope*, namely $slope_{\min, T_{off}}$, under the constraint of the service demand β^A, is equivalent to the derivation of the minimal $\tilde{T}_{on}$ in the bounded-delay approximation, where

$$slope_{\min, T_{off}} = \inf\{slope : \mathsf{bdf}(\gamma, slope, T_{off}) \geq \alpha_1^u(\gamma - D_1), \forall\gamma \geq 0\}.$$

Now we can formally define $\tilde{T}_{on}$ as following.

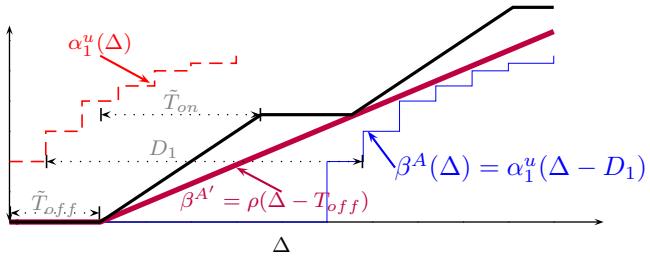


Figure 3.4: An example for the bounded delay approximation, in which only part of the upper arrival curve $\alpha_1^u(\Delta)$ is presented for simplicity.

Definition 3.1.2. *The minimal T_{on} acquired from the bounded-delay approximation is a function of T_{off}:*

$$\begin{aligned}\tilde{T}_{on} &= \frac{T_{off} \cdot slope_{\min, T_{off}}}{1 - slope_{\min, T_{off}}} \\ &\stackrel{def}{=} f(T_{off})\end{aligned} \tag{3.4}$$

To compute $slope_{\min, T_{off}}$, based on Lemma 3.1.1, we can simply apply binary search of *slope* in the range of $[0, 1]$. Suppose that there are n possible values of *slope*, the number of exploration required to derive $slope_{\min, T_{off}}$ is $O(\log n)$. Compared to the search of optimal $T_{on}^{\min}$ with respect to a given T_{off} in Equation 3.3 with $O(n)$ explorations of possible combinations, the binary search greatly improves the running time, since verifying whether $\beta^G(\gamma) \geq \beta^A(\gamma)$ for all $\gamma \geq 0$ is time-consuming.

Moreover, the derived $\tilde{T}_{on}$ has certain nice property in the following lem, which will be used to improve the time complexity for searching the optimal PPM.

Lemma 3.1.3. *Given a β^A, the function $f(T_{off})$ defined in Equation 3.4 is strictly increasing and $\frac{T_{off}}{f(T_{off})} > \frac{(1+\epsilon)T_{off}}{f((1+\epsilon)T_{off})}$ for any $\epsilon > 0$.*

Proof. From the definition of f, one has $slope_{\min, T_{off}} < slope_{\min, (1+\epsilon)T_{off}}$ and thereby $f(T_{off}) < f((1+\epsilon)T_{off})$, which proves the property for strict increase. Because $slope_{\min, T_{off}} < slope_{min, (1+\epsilon)T_{off}}$, we can derive $\frac{1}{1+\frac{T_{off}}{f(T_{off})}} < \frac{1}{1+\frac{(1+\epsilon)T_{off}}{f((1+\epsilon)T_{off})}}$, then, $\frac{T_{off}}{f(T_{off})} > \frac{(1+\epsilon)T_{off}}{f((1+\epsilon)T_{off})}$. □

Before presenting how to search the optimal T_{off}, we will first discuss about the feasible region of T_{off}. Intuitively, if T_{off} is smaller than the break-even time, i.e., $\frac{E_{sw}}{P_s - P_\delta}$, turning the system to the sleep mode consumes more energy than the energy overhead E_{sw} for mode switching. The sleep mode thereby introduces additional energy consumption. Therefore, for searching the optimal T_{off}, the region $\left[0, \frac{E_{sw}}{P_s - P_\delta}\right]$ can be safely discarded. Moreover, as T_{off} must also satisfy the timing overhead for mode switches, we also know that T_{off} must be no less than t_{sw}, where $t_{sw} = t_{sw,sleep} + t_{sw,on}$.

There is also an upper bound for T_{off}. On one hand, T_{off} should be smaller than $D_1 - c_1$. Otherwise, no event can be finished before its deadline. On the other hand, as the system provides no service when it is off, that imposes a maximum service

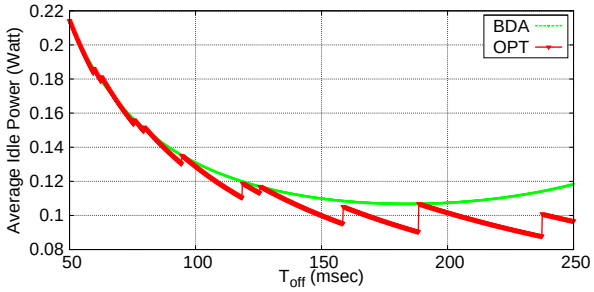


Figure 3.5: The relation of the minimal average idle power consumption and T_{off} for the OPT and BDA approaches. The stream and device are S_1 and IBM Microdrive in Table 3.1 and 3.2 of Section 3.1.4.9, respectively.

$\beta_{\top}^{G}(\gamma) = \max\{0, \gamma - T_{off}\}$. According to real-time interface in Equation 3.7, we know that predicate

$$\beta_{\top}^{G}(\gamma) = \max\{0, \gamma - T_{off}\} \geq \beta_1^A(\gamma) = \alpha_1(\gamma - D_1) \tag{3.5}$$

has to be true to satisfy the timing constraint. By inverting Equation 3.5, we can compute the maximum T_{off} as

$$T_{off}^{max} = \max\left\{T_{off} : \beta_{\top}^{G}(\gamma) \geq \beta_1^A(\gamma),\ \forall\gamma \geq 0\right\}.$$

In summary, to find an optimal PPM, the feasible region of $T_{off} \in [T_{off}^{l},\ T_{off}^{r}]$ can be bounded as follows:

$$T_{off}^{l} = \max\left\{t_{sw},\ \frac{E_{sw}}{P_S - P_\delta}\right\}$$
$$T_{off}^{r} = \min\left\{D_1 - c_1,\ T_{off}^{max}\right\}$$

3.1.2.2 Optimal and Approximated PPMs

We now present how to apply the results in Sections 3.1.2.1 for deriving T_{on} and T_{off} to minimize the average idle power consumption $P(T_{on}, T_{off})$. Depending on the bounded-delay approximation and the derivation of minimum T_{on} with respect to a given T_{off}, we will present two approaches, namely BDA and OPT, to derive T_{on} and T_{off}. To show the difference between these two schemes, for a given T_{off}, Figure 3.5 presents an example for illustrating the irregular pattern of the minimum average idle power consumption by solving Equation 3.3 and the convexity of the average idle power consumption by applying bounded-delay approximation.

Approach OPT As shown in Figure 3.5, the minimal average idle power consumption in the feasible region of T_{off} is irregular. Therefore, to find the optimal solution, we have to explore all the feasible solution space of T_{off}. Suppose that there are m values of T_{off} within the region $[T_{off}^{l},\ T_{off}^{r}]$, the complexity of the overall algorithm thereby is $O(n \cdot m)$. The pseudo-code of the OPT scheme is shown in Algorithm 1.

Algorithm 1 OPT

Input: α_1, D_1, T^l_{off}, T^r_{off}, ϵ, $P_{min} = \infty$
Output: T'_{on}, T'_{off}
1: **for** $T_{off} = T^l_{off}$ to T^r_{off} step ϵ **do**
2: exhaustively find T^{min}_{on} by testing (3.3)
3: **if** $P(T^{min}_{on}, T_{off}) < P_{min}$ **then**
4: $T'_{on} \leftarrow T^{min}_{on}$; $T'_{off} \leftarrow T_{off}$
5: $P_{min} \leftarrow P(T^{min}_{on}, T_{off})$
6: **end if**
7: **end for**

Approach BDA We first show the convexity of the objective function $P(T_{on}, T_{off})$ acquired from the application of bounded-delay approximated $\tilde{T}_{on}$ defined in Definition 3.1.2.

Theorem 3.1.4. *Using the bounded-delay algorithm approach to compute $\tilde{T}_{on} = f(T_{off})$ as depicted in (3.4), $P(\tilde{T}_{on}, T_{off}) = P(f(T_{off}), T_{off})$ is a convex function.*

Proof. The objective function $P(\tilde{T}_{on}, T_{off})$ can be split into two parts: $\frac{E_{sw}}{T_{on}+T_{off}}$ and $(P_s + P_\delta) \cdot \frac{T_{on}}{T_{on}+T_{off}}$. For the first part $\frac{E_{sw}}{T_{on}+T_{off}} = \frac{E_{sw}}{f(T_{off})+T_{off}}$, $f(T_{off}) + T_{off}$ is strictly increasing according to Lemma 3.1.3. Therefore $\frac{E_{sw}}{T_{on}+T_{off}}$ is a monotonically decreasing convex function. For the second part $(P_s + P_\delta) \cdot \frac{T_{on}}{T_{on}+T_{off}} = \frac{P_s+P_\delta}{1+\frac{T_{off}}{f(T_{off})}}$, according to Lemma 3.1.3, we know that $\frac{1}{1+\frac{T_{off}}{f(T_{off})}}$ is monotonically increasing and is a convex function as well. As a linear combination of convex functions is a still a convex function, the original function $P(\tilde{T}_{on}, T_{off})$ is a convex function of T_{off}. □

Based on the result from Thm. 3.1.4, exhaustive search for every T_{off} is not necessary. For instance, the complexity is reduced to $O(\log n \cdot \log m)$ by applying a bisection search for the feasible region of T_{off}. The pseudo code of the algorithm is described in the Algorithm 2.

3.1.3 Multiple Event Streams

The developed periodic power management method has been extended next to the case of multiple event streams. Due to space limitation, we will focus our discussions on fixed priority scheduling, and at the end of this section, we will briefly show how to handle earliest deadline first scheduling.

Suppose that there are n event streams in $\mathcal{S}$, where $n \geq 2$. For FP scheduling, without loss of generality, we order the event streams $S_1, S_2, \ldots, S_n$ according to their priorities, where the priority of event stream S_i is higher than that of S_k when $k > i$. Suppose that $\beta^l_1(\gamma)$ is the lower service curve of the system. By real-time calculus (159), we know that the remaining lower service curve $\beta'_1(\gamma)$ after serving event stream S_1 is

Algorithm 2 BDA

Input: α_1, D_1, T^l_{off}, T^r_{off}, ϵ
Output: T'_{on}, T'_{off}

```
if T^r_off − T^l_off < ε then
    if P(T^l_off, f(T^l_off)) < P(T^r_off, f(T^r_off)) then
        return {T'_on ← f(T^l_off); T'_off ← T^l_off}
    else
        return {T'_on ← f(T^r_off); T'_off ← T^r_off}
    end if
end if
slope_l ← P'(T^l_off, f(T^l_off))        ▷ P' is the derivative of P with respective to T_off
slope_m ← P'( (T^l_off+T^r_off)/2, f((T^l_off+T^r_off)/2) )
if slope_l · slope_m > 0 then
    T^l_off ← T^m_off
else
    T^r_off ← T^m_off
end if
recursively call BDA with the new T^l_off and T^r_off
```

$\sup_{0\le\lambda\le\gamma}\{\beta^l_1(\lambda)-\alpha^u(\lambda)\}$. In FP scheduling, the remaining service will be used to serve the other event streams, in which $\beta'_1(\gamma)$ is the available service curve of event stream S_2. For example, as illustrated in Figure 3.6 for three event streams, for FP scheduling, the schedulability analysis can be decomposed as three components by using the remaining service curve left by higher priority streams.

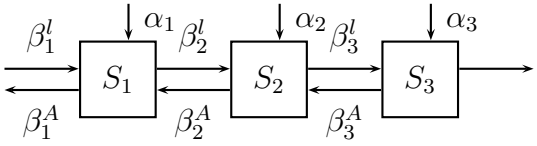


Figure 3.6: An example for fixed-priority scheduling

Therefore, to guarantee the satisfaction of timing constraint for event stream S_n, as shown in Section 3.1.2, the service provided to stream S_N must be at least $\beta^A_n(\gamma) = \alpha^u_n(\gamma - D_N)$. This also implies that the remaining service curve after serving streams $S_1, S_2, \ldots, S_{n-1}$ must be at least $\beta^A_n(\gamma)$. To derive the service bound $\beta^A_1(\gamma)$, we have to compute the service bounds $\beta^A_{N-1}(\gamma), \beta^A_{N-2}(\gamma), \ldots, \beta^A_2(\gamma)$, sequentially. Suppose that $\beta^A_k(\gamma)$ has been derived, we can apply the following equation to derive $\beta^\sharp_{k-1}(\gamma)$ so that the remaining service curve is guaranteed to be no less than $\beta^A_k(\gamma)$ if $\beta^l_{k-1}(\gamma)$ is no less than $\beta^\sharp_{k-1}(\gamma)$:

$$\beta^\sharp_{k-1}(\gamma) = \beta^A_k(\gamma-\lambda) + \alpha^u_{k-1}(\gamma-\lambda)$$
$$\text{where } \lambda = \sup\{\tau : \beta^A_k(\gamma-\tau) = \beta^A_k(\gamma)\}.$$

To guarantee the timing constraint of event stream S_{k-1}, we also know that $\beta^l_{k-1}(\gamma)$ must be no less than $\alpha^u_{k-1}(\gamma - D_{k-1})$. Therefore, we know that

$$\beta^A_{k-1}(\gamma) = \max\{\beta^{\sharp}_{k-1}(\gamma), \alpha^u_{k-1}(\gamma - D_{k-1})\}. \tag{3.6}$$

By applying 3.6 for $k = n-1, n-2, \ldots, 2$, we can then derive the lower service curve, i.e., $\beta^A_1(\gamma)$, that the system must provide for satisfying the timing constraints. Then, the bounded delay approximation (BDA) scheme and the optimal periodic power management (OPT) scheme can be applied to minimize the average idle power consumption $P(T_{on}, T_{off})$ by setting $\beta^A(\gamma)$ to $\beta^A_1(\gamma)$ derived above.

For EDF scheduling, as shown in (171), the service bound $\beta^A(\gamma)$ is simply $\sum_{S_i \in \mathcal{S}} \alpha^u_i(\gamma - D_i)$, coming from the composition of all the demands of the tasks composing the task set. The cumulative demand is the minimum resource requirements in order to have the deadlines satisfied. With dynamic scheduling algorithms a punctual analysis like the fixed priority one is not possible, so the only solution is to cumulate the contribution of the tasks and carrying on a cumulated analysis for all the streams composing the system.

For FCFS, the service bound $\beta^A(\gamma)$ is $\sum_{S_i \in \mathcal{S}} \alpha^u_i(\gamma - D_{\min})$, where $D_{\min}$ is the minimum relative deadline of the event streams in $\mathcal{S}$.

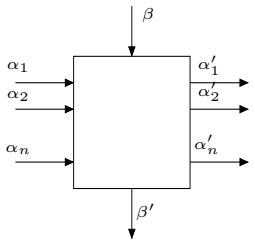


Figure 3.7: An example of EDF scheduling.

3.1.4 Adaptive Energy Aware Scheduling

In (77; 79) we explore how to efficiently do dynamic power management to reduce static power consumption while satisfying real-time constraints. Again, we consider a system that consists of a device with `active`, `standby`, and `sleep` modes with different power consumptions and a controller that decides when to change the power modes of the device. Intuitively, the device can be switched off to the sleep mode to reduce the power consumption when it becomes idle and switched on again to `active` mode upon the arrival of an event. These switching operations, however, need more careful consideration. On the one hand, the sleep period of the device after switching-off should be long enough to recuperate mode-switch overheads. On the other hand, when to activate the device is even more involved due to the possible burstiness of future event arrivals. For every switching-on operation, sufficient time has to be reserved to serve the possible burstiness of future events in order to prevent deadline violation of events or overflow of system backlog.

To resolve these two concerns, we propose online algorithms that are applicable for the controller. We apply real-time calculus (159) to predict future event arrival and real-time interface (161) for the schedulability analysis. Specifically, we try to be optimistic to handle events only when they really arrive. Our algorithms adaptively predict the next moment for mode switch by considering both historical and future event arrivals, and procrastinate the buffered and future events as late as possible without violating the timing and backlog constraints for the given event streams. To demonstrate the performance of the proposed approach, several case studies are explored, in which the results reveal the effectiveness of our approach.

3.1.4.1 Real-Time Calculus Routines

To compute a safe interval for putting the device to sleep, Real-Time Calculus (159) and real-time interface (169) are applied. Within this context, the device is said to provide guaranteed output service $\beta^G(\gamma)$. Correspondingly, a stream S_i assumes to request service demand $\beta^A(\gamma)$. To obtain a feasible scheduling of stream S_i on the device, the condition

$$\beta^G(\gamma) \geq \beta^A(\gamma),\ \forall\gamma \geq 0$$

has to be fulfilled. In this section, we present how to construct proper service guarantee and demand.

3.1.4.2 Bounded Delay

A service curve $\beta(\gamma)$ can be constructed as a bounded delay function.

Definition 3.1.5 (Bounded Delay Service Curve)**.** *A bounded delay service curve is defined as no service provided for at most τ units of time:*

$$\mathsf{bdf}(\gamma,\, \tau) \stackrel{def}{=} \max\big\{0,\, (\gamma - \tau)\big\},\ \forall\gamma \geq 0$$

Definition 3.1.6 (Longest Feasible Sleep Interval)**.** *The longest feasible sleep interval τ^* with respect to a given service demand $\beta^A(\gamma)$ is thereby defined as:*

$$\tau^* = \max\big\{\tau : \mathsf{bdf}(\gamma, \tau) \geq \beta^A(\gamma),\ \forall\gamma \geq 0\big\} \tag{3.7}$$

Definition 3.1.7 (Deadline Service Demand)**.** *Suppose an event stream S_i with relative deadline D_i. To satisfy the required relative deadline D_i, the minimum service demand of stream S_i is*

$$\beta^\flat(\gamma) \stackrel{def}{=} \alpha_i^u(\gamma - D_i) \tag{3.8}$$

Definition 3.1.8 (Backlog-size Service Demand)**.** *Suppose a system has a backlog of size Q_i to buffer un-processed events for stream S_i. To prevent backlog overflow, the minimum service demand of stream S_i is*

$$\beta^{\dagger}(\gamma) \stackrel{def}{=} \alpha_i^u(\gamma) - w_i \cdot Q_i \tag{3.9}$$

Combining both deadline and backlog service demands, the τ^* in (3.7) can be refined as

$$\tau^* = \max\{\tau : \mathsf{bdf}(\gamma, \tau) \geq \max\{\beta^{\flat}(\gamma),\ \beta^{\dagger}(\gamma)\}\} \tag{3.10}$$

Figure 3.8 depicts an example from the above analysis for single event stream. Based on above definitions, we state the following lem.

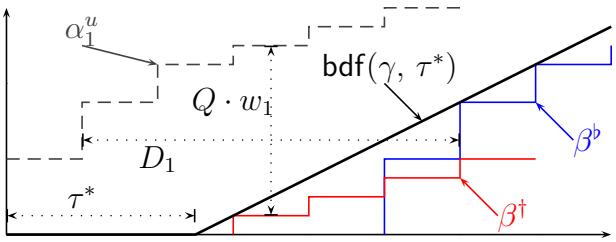


Figure 3.8: An example for the bounded delay function for event stream S_1, in which only part of the upper arrival curve $\alpha_1^u(\gamma)$ is presented for simplicity.

Lemma 3.1.9. Given τ^* computed from Equation 3.10 which is larger than T_{BET}, at any time instance t when the device is active and no event to process, it is feasible to deactivate the device for $[t,\ t+\tau^*)$ interval without violating the deadline and backlog-size requirements of for any event in stream S_i.

Proof. The service demand $\beta^{\flat}$ in 3.8 is constructed as horizontally right shifting α_i^u with D_i distance, which represents the tightest bound that guarantees the deadline constraint. Similarly, the service demand $\beta^{\dagger}$ in 3.9 obtained by vertically shifting α_i^u down by $w_i \cdot Q$ distance and defines the tightest bound that prevents backlog overflow. The bounded delay function $\mathsf{bdf}(\gamma,\ \tau^*)$ constructs a β^G that bounds both $\beta^{\flat}$ and $\beta^{\dagger}$, thus fulfills (3.7). □

3.1.4.3 Future Prediction with Historical Information and Backlogged Demand

As the scheduling decision is made online and is depended on the actual arrivals of events, we keep track of event arrivals in the past as a history. Because the total amount of arrived events in any time interval is constrained by the corresponding arrival curves, one can predict the future event arrivals based on a certain length of historical information of event arrivals in the recent past. If a burstiness has been observed recently, for instance, it can be foreseen that sparse events will arrival in the near future. To make use of the historical information, we define the history curve.

Definition 3.1.10 (History Curve)**.** *Suppose t is the current time and $R_i(t)$ is the accumulated number of events of stream S_i in interval $[0, t)$. γ^h is the length of the history window of which controller can maintain, i.e., historical information for only γ^h time units is recorded. At time t the history curve for stream S_i is define as*

$$H_i(\gamma, t) \stackrel{def}{=} \begin{cases} R_i(t) - R_i(t-\gamma), & \text{if } \gamma \leq \gamma^h, \\ R_i(t) - R_i(t-\gamma^h), & \text{otherwise} \end{cases}$$

The maximal future event arrivals in the near future from time t to $t+\gamma$, denoted as $\bar{\alpha}_i^u(\gamma,\, t)$, is

$$\bar{\alpha}_i^u(\gamma, t) = \inf_{\lambda \geq 0} \left\{ \bar{\alpha}_i^u(\gamma + \lambda) - H_i(\lambda, t) \right\}$$

Analogously, we can preciously define the service demand of those backlogged events. We denote the set of unfinished events of S_i in the backlog at time t as $E_i(t)$. Note that although the absolute deadline $D_{i,\,j}$ for event $e_{i,\,j} \in E_i(t)$ does not change, the relative deadline is not D_i anymore. It varies according to relative distance from t.

Definition 3.1.11 (Backlogged Demand)**.** *Suppose that events in $E_i(t)$ are indexed as $e_{i,1},\, e_{i,2},\, \ldots,\, e_{i,|E_i(t)|}$ from the earliest deadline to the latest. a backlog demand curve for time t and stream S_i is defined as*

$$B_i(\gamma,\, t) \stackrel{def}{=} w_i \cdot \begin{cases} (j-1), & D_{i,\,j} - t < \gamma \leq D_{i,\,j+1} - t, \\ |E_i(t)|, & \gamma > D_{i,\,|E_i(t)|} - t, \end{cases} \tag{3.11}$$

in which $D_{i,0}$ is defined as t for brevity.

3.1.4.4 Basic Algorithms for Single Stream

In general, we have to decide when to turn the device to active mode to serve events from sleep mode, and when to turn it back to sleep mode in order to reduce static power. Therefore, we have to deal with *deactivation scheduling decisions* and *activation scheduling decisions* to switch safely and effectively. An overview of our approach is illustrated in Figure 3.9.

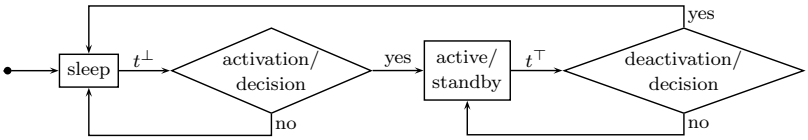


Figure 3.9: The control flow of our approach.

For deactivation scheduling decisions, when the device is in active mode and there is no event in the backlog, we have to decide whether the device has to change to sleep mode instantly or it should remain active/idle for a while for serving the next incoming event (We assume the device switches between active and standby modes automatically). For brevity, for the rest of this chapter, time instants for deactivation scheduling decisions are denoted by $t^{\top}$.

After the device is switched to sleep mode, it has to be switched active again for event processing. The activation scheduling decision is evaluated at the time instance upon the arrival of an event or expiration of the sleep interval that the controller previously set. Our algorithms have to decide whether the device has to change to active mode instantly to serve events, or it should remain in sleep mode for a while to aggregate more events for processing and prevent unnecessary mode switches. For brevity, for the rest of this chapter, time instants for activation scheduling decisions are denoted by $t^{\perp}$.

In this section, we present our approach to minimize static power. The assumption of our approach is that the scheduling decision is always feasible if the device provides full service all the time, i.e., the device never turns to sleep mode. For simplicity, we consider only single event stream case, says S_1. We first present how to deal with deactivation of scheduling decisions and then propose two methods for the scheduling decisions of the activation. The solutions to multiple event streams are presented in the subsequent two sections.

Deactivation Algorithm The History-Aware Deactivation (HAD) algorithm analyzes whether the device should be turned to sleep mode from active mode. The principle is to switch the device only when energy saving is possible. One obvious fact is that as long as there are events in the system backlog, the device can be kept busy in active mode and no static power is introduced. In order to reduce static power, the deactivation decision thereby makes sense only when the device is in active or standby mode while there is no new arrival of events as well as no event in system backlog. Suppose that $t^{\top}$ is such a time instant. Turning the device instantly at time $t^{\top}$ to sleep mode, however, does not always help still. The reason is that we pay overheads for mode switching. In the case there are events arriving in the very near future, the device has to be activated again to process these events. If the energy saving obtained from the short sleep period cannot counteract this switching overhead, the mode switching only introduces additional energy consumption. Therefore, the idea is firstly to compute the maximal possible sleep interval τ^* and check this τ^* is sufficient to cover the break-even time. Specifically, we calculate the arrival curve $\hat{\alpha}_1^u(\gamma, t^{\top})$ at time $t^{\top}$ by 3.11 and refine the service demands in Equation 3.8 and 3.9 as

$$\beta^{\flat}(\gamma) = \alpha_1^u(\gamma - D_1, t^{\top}) \tag{3.12}$$

$$\beta^{\dagger}(\gamma) = \alpha_1^u(\gamma, t^{\top}) - Q \cdot w_1 \tag{3.13}$$

By applying Equation 3.12 and 3.13 to Equation 3.10, the maximal sleep interval τ^* is computed, which means turning the device to sleep mode at time $t^{\top}$ and switch it active again at time $t^{\top} + \tau^*$ will not cause constraint violations. If τ^* is larger than T_{BET}, we turn the device to the sleep mode at time $t^{\top}$; otherwise, we do not turn the device to the sleep mode. The pseudo code of the algorithm is depicted in Algorithm 3. The algorithm leads to the following theorem:

Theorem 3.1.12. *The HAD algorithm guarantee a feasible scheduling upon a deactivation decision at any time $t^{\top}$ for one event-stream system, if the device provides full service from time $t^{\top} + \tau^*$ on.*

Algorithm 3 HAD deactivation

procedure *at time instance* $t_w^{\perp}$*:*
1: compute τ^* of 3.10 by $\beta^{\flat}$ and $\beta^{\dagger}$ in 3.12 and 3.13;
2: **if** $\tau^* > T_{BET}$ **then**
3: deactivate the device;
4: **end if**

Proof. We prove this theorem by contradiction. At any time instance $t^{\top}$ at which the HAD algorithm decides to deactivate the device, the latest activation time to prevent constraint violations is $t^{\top} + \tau^*$. Suppose at a later time $t^{\top} + \lambda$, the deadline of an event which comes within the interval $[t^{\top}, t^{\top} + \lambda)$ is missed. We denote the number of events arrived within this interval as u. Because of the deadline missing, the service demand $u \cdot w_1$ in this interval is larger than our constructed service supply $\mathsf{bdf}(\lambda, \tau^*)$ which actually bounds the service demand of the maximum number of events that can arrival, i.e., $w_1 \cdot \bar{\alpha}_1^u(\lambda, t^{\top})$. The inequality $u > \bar{\alpha}_1^u(\lambda, t^{\top})$ contradicts to the definition in Equation 3.11. Therefore, the theorem holds. □

Once the device is in sleep mode, the controller needs to switch the device to active mode later to process events. How to decide the actual switch moment needs more consideration. On the one hand, it is preferable to aggregate as many events as possible for each switch operation to not only reduce the standby period but also minimize the number of switch operations. On the other hand, the real-time constraints of the aggregated and future events need to be guaranteed. Furthermore, a polling mechanism is not desirable which will overload the controller. In this section, we present two algorithms, namely worst-case-greedy (WCG) algorithm and event-driven-greedy (EDG) algorithm, for the activation scheduling decisions. The difference between these two algorithms is the following:

- Algorithm WCG conservatively assumes worst-case event arrivals and predicts the earliest switch moment. If the worst case does not occur when the predicted time comes, a new prediction is conducted and the switch decision is deferred to a later moment.
- Algorithm EDG optimistically considers the least event arrivals and predicts the latest time for mode switch. Upon the arrival of each new event before the predicted time, the decision is reevaluated and it is shifted earlier if necessary.

Therefore, the time instance $t^{\perp}$ for the activation scheduling decisions can be evaluated at either event arrivals or the predicted activation time. We identify these two cases by *event arrival* and *wake-up alarm arrival*, at time instant $t_e^{\perp}$ and $t_w^{\perp}$, respectively.

Worst-Case-Greedy (WCG) Activation The WCG algorithm works in a time-triggered manner. It reacts to each wake-up alarm and performs two tasks: a) check whether the device has to be switched to active mode for the current alarm; b) if not, determine a new wake-up alarm. In the case that worst-case burst happens before the wake-up alarm $t_w^{\perp}$, i.e., our previous prediction is correct, the mode switch has to be carried on at the current wake-up alarm. If the actual arrivals of events are less than the worst case, switching the device at the current $t_w^{\perp}$ will reserve too much service than actual needs. The device can stay in the sleep mode for a longer period. To evaluate the activation decision and predict the next wake-up alarm, we again apply the bounded-delay function. The deadline service demand $\beta^{\flat}$ and the backlog-size service demand $\beta^{\dagger}$ are refined as

$$\beta^{\flat}(\gamma) = \alpha_1^u(\gamma - D_1,\, t_w^{\perp}) + w_1 \cdot B_1(\gamma,\, t_w^{\perp}) \tag{3.14}$$

$$\beta^{\dagger}(\gamma) = \alpha_1^u(\gamma, t_w^{\perp}) - \big(Q - |\boldsymbol{E}(t^{\perp})|\big) \cdot w_1 \tag{3.15}$$

Besides the history-refined worst-case event arrival $\alpha_1^u(\gamma - D_1,\, t_w^{\perp})$ at the current wake-up alarm $t_w^{\perp}$, the deadline service demand $\beta^{\flat}$ needs to consider those events already stored in the system backlog, i.e., $B_1(\gamma,\, t_w^{\perp})$ defined in Equation 3.11. Similarly, the current size of the available backlog is the original size subtracted by the number of backlogged events, i.e., $|\boldsymbol{E}(t^{\perp})|$ defined in previously. Using Equations 3.14 and 3.15, the next sleep interval τ^* is computed. If $\tau^* > 0$, the next wake-up alarm is set to time $t_w^{\perp} + \tau^*$. Otherwise, the device is switched to active mode. The pseudo code of the WCG algorithm is depicted in Algorithm 4. The constructed $\beta^{\flat}$ in Equation 3.14

Algorithm 4 WCG activation

procedure *event arrival at time* $t_e^{\perp}$*:*
1: do nothing;
procedure *wake-up alarm arrival at time* $t_w^{\perp}$*:*
1: compute τ^* of (3.10) with $\beta^{\flat}$ and $\beta^{\dagger}$ by 3.14 and 3.15;
2: **if** $\tau^* > 0$ **then**
3: new wake-up alarm at time $t_w^{\perp} \leftarrow t_w^{\perp} + \tau^*$;
4: **else**
5: wakeup the device;
6: **end if**

bounds the future arrival events from $t_w^{\perp}$ on and the $\beta^{\dagger}$ in 3.15 guarantees the backlog constraint, which lead to following theorem:

Theorem 3.1.13. *The WCG algorithm guarantees a feasible scheduling upon an activation decision at any wake-up alarm* $t_w^{\perp}$ *for one event-stream system, if the device provides full service from time* $t_w^{\perp}$ *on.*

We omit the proof due to the similarity to Theorem 3.1.12. The WCG algorithm is effective in the sense that it greedily extends the sleep period as long as the device is schedulable. It is efficient as well when the event arrival pattern is close to the worst-case scenario, where the reevaluation of the wakeup alarm does not take place often. Furthermore, the number of reevaluation is bounded by $\alpha_1^u(D_1 - w_1)$.

The last problem is where the first wake-up alarm is. There are two possibilities: a) at the time the first arrived event after the device is deactivated, and b) at the deactivation time instance $t^{\top} + \tau^*$ (τ^* is computed by 3.12 and 3.13) in the HAD algorithm). Both approaches are effective. For consistency, we adopt the second approach, such that the WCG algorithm is purely time-driven.

Event-Driven-Greedy (EDG) Activation In contrast to the WCG algorithm which predicts the earliest wake-up alarm $t_w^{\perp}$, the EDG algorithm predicts the latest one. It computes the latest moment by assuming the least event arrivals in the near future. Unlike the WCG algorithm where the evaluation of the activation scheduling decisions takes place upon each wake-up alarm arrives, the decision here is refined upon event arrivals. At time $t_{e_{1,i}}^{\perp}$ when an event $e_{1,i}$ arrives, when is the corresponding latest wake-up alarm $t_w^{\perp}$ is not that obvious. One intuitive guess is $t_{e_{1,i}}^{\perp} + D_1 - w_1$. This time instance is however too optimistic except for the first event $e_{1,1}$ after the device is deactivated. Our EDG algorithm works in the following manner. For the first arrived event $e_{1,1}$, the wake-up alarm is set to $t_{e_{1,1}}^{\perp} + D_1 - w_1$. For any subsequent event $e_{1,i}$, wake-up alarm is set to the smaller value of the previous $t_w^{\perp}$ and $t_w^{\perp} - (w_1 - (t_{e_{1,i}}^{\perp} - t_{e_{1,i-1}}^{\perp}))$. This new $t_w^{\perp}$ is not yet always a feasible activation time instance. If τ^* computed from this time instance is not larger than 0, the activation is set to an earlier time, i.e., the earliest activation time as if worst-case event arrival happen at $t_{e_{1,1}}^{\perp}$. For an event $e_{1,i}$ arrived at time $t_{e_{1,i}}^{\perp}$, the service demand for the newly computed wake-up alarm $t_w^{\perp}$ includes a) the possible burst from $t_w^{\perp}$ on, which is bounded by $\bar{\alpha}_1^u(\gamma,\, t_w^{\perp})$, b) the backlog until $t_w^{\perp}$, and c) the estimated least event arrival between $[t_{e_{1,i}}^{\perp},\, t_w^{\perp})$, constrained by $\bar{\alpha}_1^l(\gamma)$. To compute a precise $\bar{\alpha}_1^u(\gamma,\, t_w^{\perp})$, we first revise the historical information $H_1(\gamma,\, t_w^{\perp})$ by advancing the time from $t_{e_{1,i}}^{\perp}$ to $t_w^{\perp}$ to include those events which definitely have to come between $[t_{e_{1,i}}^{\perp},\, t_w^{\perp})$. We denote such a trace as $H'(\gamma, t_w^{\perp})$:

$$H_1'(\gamma, t_w^{\perp}) = \begin{cases} \bar{\alpha}_1^l(\epsilon) - \bar{\alpha}_1^l(\epsilon - \gamma), & \text{if } \gamma < \epsilon, \\ H_1(\gamma, t_{e_{1,i}}^{\perp}) + \bar{\alpha}_1^l(\epsilon), & \text{if } \epsilon < \gamma < \gamma^h - \epsilon, \\ H_1(\gamma^h - \epsilon, t_{e_{1,i}}^{\perp}) + \bar{\alpha}_1^l(\epsilon), & \text{otherwise,} \end{cases} \tag{3.16}$$

where $\epsilon = t_w^{\perp} - t_{e_{1,i}}^{\perp}$ for abbreviation. The H_1' can be seen as the concatenation of the historical information H_1 until $t_{e_{1,i}}^{\perp}$ and the time inversion of $\bar{\alpha}_1^l$ in the interval $[0, \epsilon)$. The worst-case arrival curve after time $t_w^{\perp}$ with the new historical information H_1' is

$$\bar{\alpha}_1^u(\gamma, t_w^{\perp}) = \inf_{\lambda \geq 0} \left\{ \bar{\alpha}_1^u(\gamma + \lambda) - H_1'(\lambda, t_w^{\perp}) \right\}$$

and $\alpha_1^u(\gamma, t_w^{\perp}) = w_1 \cdot \bar{\alpha}_1^u(\gamma, t_w^{\perp})$. The corresponding backlog demand curve that encapsulates the estimated least arrival events within the interval $[t_{e_{1,i}}^{\perp},\, t_w^{\perp})$ is

$$B_1'(\gamma,\, t_w^{\perp}) = \begin{cases} j - 1, & D_{1,\,j} - t_w^{\perp} < \gamma \leq D_{1,\,j+1} - t_w^{\perp}; \\ \mathcal{E}, & \gamma > D_{1,\,\mathcal{E}} - t_w^{\perp}, \end{cases}$$

where $\mathcal{E} = |E_1(t_{e_{1,i}}^{\perp})| + \bar{\alpha}_1^l(\epsilon)$, $\epsilon = t_w^{\perp} - t_{e_{1,i}}^{\perp}$, and $D_{1,\,0}$ is defined as $t_{e_{1,i}}^{\perp}$. With the refined historical information and backlog demand, the two service demands $\beta^{\flat}$ and $\beta^{\dagger}$ can be

refined as

$$\beta^{\flat}(\gamma) = \alpha_1^u(\gamma - D_1,\, t_w^{\perp}) + w_1 \cdot B_1'(\gamma,\, t_w^{\perp}) \tag{3.17}$$

$$\beta^{\dagger}(\gamma) = \alpha_1^u(\gamma, t_w^{\perp}) - \big(Q - |\boldsymbol{E}(t^{\perp})| - \bar{\alpha}_1^l(\epsilon)\big) \cdot w_1 \tag{3.18}$$

By applying Equation 3.17 and 3.18, the sleep interval τ^* in 3.10 is computed for event $e_{1,i}$. If $\tau^* > 0$, the wakeup alarm is valid. Otherwise, the new wakeup alarm is set to an earlier moment. The pseudo code of the algorithm is depicted in Algorithm 5.

Algorithm 5 EDG activation

procedure *event arrival at time* $t_{e_{1,i}}^{\perp}$*:*

1: $t_w^{\perp} \leftarrow (t_{e_{1,i}}^{\perp} - t_{e_{1,i-1}}^{\perp} > w_1)?t_w^{\perp} : t_w^{\perp} - (w_1 - (t_{e_{1,i}}^{\perp} - t_{e_{1,i-1}}^{\perp})$

2: calculate τ^* at time $t_w^{\perp}$ by 3.16–3.18;

3: **if** $\tau^* \leq 0$ **then**

4: $\quad t_w^{\perp} \leftarrow t_{e_{1,1}}^{\perp} + \tau^{\perp}$, where $\tau^{\perp}$ computed from 3.8 – 3.10

5: **end if**

procedure *wake-up alarm arrival at time* $t_w^{\perp}$*:*

1: activate the device;

Theorem 3.1.14. *The EDG algorithm guarantees a feasible scheduling upon an activation decision at any wakeup alarm* $t_w^{\perp}$ *for single event-stream system, if the device provides full service from time* $t_w^{\perp}$ *on.*

Proof. We differentiate the mode switch decisions into two categories: decisions by hitting Line 4 of the Algorithm 5 or without. In the case of without hitting, the feasibility of the decision is guaranteed by Equations 3.16–3.18 where the events actually arrived before time $t_w^{\perp}$ and the potential burstiness after are considered in each evaluation. In the case of hitting, $t_{e_{1,1}}^{\perp} + \tau^{\perp}$ is always feasible since it assumes the worst-case event arrivals happen at the time instance of the arrival of the first event after the device is switched to sleep mode. □

The activation decision in Line 4 is pessimistic. It is possible to refine $t_w^{\perp}$ when $\tau^* \leq 0$ happens, instead of setting the prediction back to $t_{e_{1,1}}^{\perp} + \tau^{\perp}$. However, such refinement demands more computation. The EDG algorithm is designed for scenarios where events come sparsely and the worst case seldom occurs. In such cases, the algorithm is effective because Line 4 will seldom be hit. The algorithm is efficient as well as the number of reevaluations is small which is approximately equal to the number of events actually arrived. Furthermore, $\tau^{\perp}$ can be computed offline as it is a constant given the specification of the stream.

3.1.4.5 Solutions for Multiple Streams

To tackle multiple event streams, the essential problem is to compute the total service demand for the stream set itself. Without loss of generality, we assume a stream set $\mathcal{S}$ with n event streams, where $n \geq 2$. The total service demand for $\mathcal{S}$ depends on

not only the service demand of individual streams but also the scheduling policy as well as the system backlog organization. To compute the total service of $\mathcal{S}$, real-time interface theory (169) is applied. In particular, two preemptive scheduling, i.e., earliest-deadline-first and fixed priority policies are considered for the resource sharing. With respect to the backlog organization, two different scenarios, i.e., distributed and global backlog, are investigated. In the case of distributed backlog, each event stream S_i owns its private backlog with size Q_i. Buffering more than Q_i events for S_i incurs a backlog overflow and causes a controller failure. In the case of global, all event streams in $\mathcal{S}$ share the same backlog.

In this section, we present solutions only for the EDG algorithm. The solutions for the HAD and WCG algorithms are omitted due to similarity and the limited space. Note that the refinements of the history curve and backlog demand in Equation 3.16 and 3.17 can be applied to individual stream, denoted as H'_i and B'_i for briefness.

3.1.4.6 FP Scheduling with Distributed Backlog

Unlike a system with one event stream where the bounded delay is applied directly to the computed service demand of the event stream, we compute first the individual service demand of every stream, denoted as β_i^A, then derive the total service demand of the set $\mathcal{S}$, denoted as β_{total}^A. With the computed β_{total}^A, the bounded delay is applied to calculate the feasible sleep interval τ^*.

Without loss of generality, the event streams $S_1, S_2, \ldots, S_n$ in $\mathcal{S}$ are ordered according to their priorities. The priority of stream S_i is higher than that of S_k when $k > i$. Event streams can thereby be modeled as an ordered chain according to their priorities and a lower priority stream can only make use of the resource left from a higher priority stream. To compute the service demand of a higher priority stream, a backward approach is applied by considering the service demand from the directly lower priority stream, as shown in Figure 3.10.

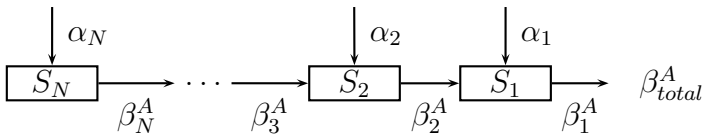


Figure 3.10: The computation flow for total service demand for the FP scheduling with distributed backlog.

For the activation scheduling decision of the arrival of an event $e_{1,i}$, the service demand of stream S_n at time $t_w^\perp$ is computed as

$$\beta_n^A(\gamma,\, t_w^\perp) = \max\{\beta_n^\flat(\gamma,\, t_w^\perp),\, \beta_n^\dagger(\gamma,\, t_w^\perp)\}, \text{where}$$
$$\beta_n^\flat(\gamma,\, t_w^\perp) = \alpha_n^u(\gamma - D_N,\, t_w^\perp) + B'_n(\gamma,\, t_w^\perp)$$
$$\beta_N^\dagger(\gamma,\, t_w^\perp) = \alpha_N^u(\gamma,\, t_w^\perp) - \left(Q_n - |E_n(t_w^\perp)| - \bar{\alpha}_n^l(t_w^\perp - t_{e_{1,i}}^\perp)\right) \cdot w_n$$
$$\alpha_n^u(\gamma,\, t_w^\perp) = w_n \cdot \left(\inf_{\lambda \geq 0} \left\{\bar{\alpha}_n^u(\gamma + \lambda) - H_N(\lambda, t_w^\perp)\right\}\right)$$

To derive β_1^A, we have to compute the service bounds $\beta_{n-1}^A, \beta_{N-2}^A, \ldots, \beta_2^A$, sequentially. Suppose that β_k^A has been derived, the resource constraint is that the remaining service curve should be guaranteed to be no less than β_k^A, i.e.,

$$\beta_{k-1}^{\sharp}(\gamma) \geq \inf\left\{\beta : \beta_k^A(\gamma,\, t_w^{\perp}) = \sup_{0\leq\lambda\leq\gamma}\left\{\beta(\lambda) - \alpha_{k-1}^u(\lambda,\, t_w^{\perp})\right\}\right\} \tag{3.19}$$

By inverting the former equation we can derive $\beta_{k-1}^{\sharp}$ as:

$$\begin{aligned}\beta_{k-1}^{\sharp}(\gamma) &= \beta_k^A(\gamma-\lambda) + \alpha_{k-1}^u(\gamma-\lambda,\, t_w^{\perp}) \\ &\text{where } \lambda = \sup\left\{\tau : \beta_k^A(\gamma-\tau,\, t_w^{\perp}) = \beta_k^A(\gamma,\, t_w^{\perp})\right\}\end{aligned}$$

To guarantee the timing constraint of event stream S_{k-1}, we also know that β_{k-1}^A must be no less than its own demand. Therefore, we know that

$$\beta_{k-1}^A(\gamma) = \max\left\{\beta_{k-1}^{\sharp}(\gamma),\, \beta_{k-1}^{\flat}(\gamma,\, t_w^{\perp}),\, \beta_{k-1}^{\dagger}(\gamma,\, t_w^{\perp})\right\} \tag{3.20}$$

where

$$\beta_{k-1}^{\flat}(\gamma,\, t_w^{\perp}) = \alpha_{k-1}^u(\gamma - D_{k-1},\, t_w^{\perp}) + B'_{k-1}(\gamma,\, t_w^{\perp}) \tag{3.21}$$

$$\beta_{k-1}^{\dagger}(\gamma,\, t_w^{\perp}) = \alpha_{k-1}^u(\gamma,\, t_w^{\perp}) - \left(Q_{k-1} - |E_{k-1}(t_w^{\perp})| - \bar{\alpha}_{k-1}^l(t_w^{\perp} - t_{e_{1,i}}^{\perp})\right)\cdot w_{k-1}$$

$$\alpha_{k-1}^u(\gamma,\, t_w^{\perp}) = w_{k-1}\cdot\left(\inf_{\lambda\geq 0}\left\{\bar{\alpha}_{k-1}^u(\gamma+\lambda) - H_{k-1}(\lambda, t_w^{\perp})\right\}\right) \tag{3.22}$$

By applying (3.20) for $k = n-1, n-2, \ldots, 2$, the service demand β_1^A of stream S_1 is derived.

Based on this approach, the computed service demand for the highest priority stream S_1 can be also seen as the total service demand β_{total}^A for stream set $\mathcal{S}$ under the fixed priority scheduling. Therefore, the timing as well as backlog constraints for all streams in $\mathcal{S}$ can be guaranteed by the sleep interval τ^* with which $\mathsf{bdf}(\gamma,\, \tau^*)$ bounds β_1^A:

$$\tau^* = \max\left\{\tau : \mathsf{bdf}(\gamma,\tau) \geq \beta_1^A(\gamma),\, \forall\gamma \geq 0\right\} \tag{3.23}$$

3.1.4.7 EDF Scheduling with Distributed Backlog

For earliest-deadline-first scheduling, the total service demand β_{total}^A for all n streams can be bounded by the sum of their service demands. The β_{total}^A computed in this manner, however, is not sufficient to guarantee the backlog constraint of any stream in $\mathcal{S}$. When an event of a stream S_j is happened to have the latest deadline, events in any stream of $\mathcal{S} \setminus \{S_j\}$ will be assigned a higher priority. S_j will suffer from backlog overflow.

To compute a correct service demand which satisfies the backlog constraint for stream S_j, S_j has to be considered as the lowest priority. Similar back-forward approach

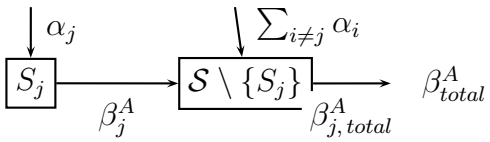


Figure 3.11: The computation flow for the total service demand for the EDF scheduling with distributed backlog.

is applied, as shown in Figure 3.11. Instead of tracing back stepwise, the service demand needed for higher-priority streams is the sum of all streams from $\mathcal{S}\setminus\{S_j\}$. Again, we present the revision of the EDG algorithm as an example. The service $\beta^\sharp_j$ to guarantee the lowest priority stream S_j should be more than the demand β^A_j of S_j, i.e.,

$$\beta^\sharp_j(\gamma) \geq \inf\left\{\beta : \beta^A_j(\gamma,\, t^\perp_w) = \sup_{0\leq\lambda\leq\gamma}\{\beta(\lambda) - \sum_{i\neq j}^{n}\alpha^u_i(\lambda,\, t^\perp_w)\}\right\} \tag{3.24}$$

By inverting (3.24), we can derive $\beta^\sharp_j(\gamma)$ as:

$$\begin{aligned}\beta^\sharp_j(\gamma) &= \beta^A_j(\gamma-\lambda,\, t^\perp_w) + \sum_{i\neq j}^{N}\alpha^u_i(\gamma-\lambda,\, t^\perp_w)\\ &\text{where } \lambda = \sup\left\{\tau : \beta^A_j(\gamma-\tau,\, t^\perp_w) = \beta^A_j(\gamma,\, t^\perp_w)\right\},\ \text{and}\\ \beta^A_j(\gamma,\, t^\perp_w) &= \max\{\beta^\flat_j(\gamma,\, t^\perp_w),\, \beta^\dagger_j(\gamma,\, t^\perp_w)\}\end{aligned}$$

where $\beta^\flat_j$ and $\beta^\dagger_j$ are from (3.21) and (3.22). To guarantee the timing constraint of all higher-priority streams, we also know that $\beta^A_{j,total}$ must be no less than the demand of $\mathcal{S}\setminus\{S_j\}$ as well. Therefore, we know that at time $t^\perp_w$,

$$\beta^A_{j,total}(\gamma) = \max\left\{\beta^\sharp_j(\gamma),\, \sum_{i\neq j}^{N}\beta^A_i(\gamma,\, t^\perp_w)\right\} \tag{3.25}$$

Applying Equation 3.25 to each steam in $\mathcal{S}$, the service demand for each steam is computed. Because each stream could be the lowest priority in the worst case, only the maximum of them can be seen as the total service demand for stream set $\mathcal{S}$. Therefore, the timing and backlog constraints for $\mathcal{S}$ can be guaranteed by τ^* with which $\mathsf{bdf}(\gamma,\, \tau^*)$ bounds the maximum of individual streams:

$$\tau^* = \max\left\{\tau : \mathsf{bdf}(\gamma,\tau) \geq \max_{i\in N}\{\beta^A_{i,total}(\gamma)\},\, \forall\gamma\geq 0\right\} \tag{3.26}$$

3.1.4.8 EDF Scheduling with Global Backlog

The approach to get the total service demand of $\mathcal{S}$ for global backlog is different to the approach for distributed backlog. Without loss of generality, we assume that a backlog with size Q is shared by all event streams of $\mathcal{S}$.

From (169), the total service demand for all n streams with respect to EDF scheduling can be bounded by the sum of their arrival curves:

$$\beta^A \geq \sum_{i=1}^{N} \alpha_i^u(\gamma - D_i) \tag{3.27}$$

Based on this result, we refine our algorithms.

For the HAD algorithm, because there is no backlog for each evaluation, the related deadline for each event $e_{i,j}$ in every stream S_i remains D_i. Therefore, the service demand to guarantee deadline requirement of all streams is

$$\beta^\flat(\gamma) = \sum_{i=1}^{n} \alpha_i^u(\gamma - D_i, t^\top) \tag{3.28}$$

In the case of Equation 3.14 of the WCG algorithm, the backlogs of different streams needs to be considered. We apply the backlog demands fro all streams thereof:

$$\beta^\flat(\gamma) = \sum_{i=1}^{n} \big(\alpha_i(\gamma - D_i,\, t^\perp) + w_i \cdot B_i(\gamma,\, t^\perp)\big)$$

The same applies to Equation 3.17 of the EDG algorithm at time $t_w^\perp$.

Now we consider the backlog-size constraint. Besides the sum of all arrival curves, the constraint in Equation 3.13 additionally needs to consider events with the longest execution time, i.e., $\max_{i\in n}\{w_i\}$. Therefore, it is revised as

$$\beta^\dagger(\gamma) = \sum_{i=1}^{n} \alpha_i^u(\gamma) - Q \cdot \max_{i\in n}\{w_i\}$$

The backlog constraint in (3.15) is more complex, because the backlog is not empty and contains events from different streams. The remaining capacity of the backlog is

$$\max_{i\in n}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^\perp)|} \sum_{i=1}^{N} x_{i,j} \cdot w_i$$

where $x_{i,j} = 1$, $\forall j$ for Stream S_i, otherwise 0. Therefore, it is revised as

$$\beta^\dagger(\gamma) = \sum_{i=1}^{N} \alpha_i^u(\gamma, t^\perp) - \Big(\max_{i\in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t^\perp)|} \sum_{i=1}^{N} x_{i,j} \cdot w_i\Big)$$

The last revision is (3.18) of the EDG algorithm, where the estimated future events of all streams need to be counted. Therefore it is revised as

$$\beta^\dagger(\gamma) = \sum_{i=1}^{n} \alpha_i^u(\gamma, t_w^\perp) - \Big(\max_{i\in N}\{w_i\} \cdot Q - \sum_{j=1}^{|E(t_w^\perp)|} \sum_{i=1}^{n} x_{i,j} \cdot w_i - \sum_{i=1}^{n} \alpha_i^l(\epsilon)\Big)$$

Table 3.1: Event stream setting according to (72; 78).

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
period (msec)	198	102	283	354	239	194	148	114	313	119
jitter (msec)	387	70	269	387	222	260	91	13	302	187
deday (msec)	48	45	58	17	65	32	78	-	86	89
wcet (msec)	12	7	7	11	8	5	13	14	5	6

Table 3.2: Power profiles for devices according to (48).

Device Name	P_a (W)	P_s (W)	P_σ (W)	t_{sw} (S)	E_{sw} (mJ)
Realtec Ethenet	0.19	0.125	0.085	0.01	0.8
Maxstream	0.75	0.1	0.05	0.04	7.6
IBM Microdrive	1.3	0.5	0.1	0.012	9.6
SST Flash	0.125	0.05	0.001	0.001	0.098

3.1.4.9 Performance Evaluations

This section provides simulation results for the proposed adaptive dynamic power management schemes. All the results are obtained from an AMD Opteron 2.6 *GHz* processor with 8 *GB* RAM. The simulator is implemented in MATLAB by applying MPA and RTS tools from (172).

We take the event streams studied in (72; 78) for our case studies. The specifications of these streams are depicted in Table 3.1. Parameters `period`, `jitter`, and `delay` are used for generating arrival curves previously defined and `wcet` represents the worst-case execution time of an event. The relative deadline D_i of a stream S_i is defined by a *deadline factor*, denoted as χ, of its period, i.e., $D_i = \chi * p_i$. To compare the impact of different algorithms, we simulate traces with a 10 sec time span. The traces are generated by the RTS tools (172) and conformed to the arrival curve specifications. The history window γ^h is 200 msec. In our simulations, we adopt the power profiles for four different devices in (48), depicted in Table 3.2.

In this work, we evaluate our two PPM schemes, i.e., WCG-HAD and EDG-HAD. To show the effects of our scheme, we report the average idle power that is computed as the total idle energy consumption divided by the time span of the simulation:

$$\frac{E_{sw} \cdot number\_switches + \sum on\_time \cdot P_\sigma}{total\_time\_span}$$

For comparison, two other power management schemes described in (78) are measured as well, i.e., a periodic scheme (OPT) and a naive event-driven scheme (ED). The OPT scheme is a periodic power management (PPM) scheme which controls the device with a fixed on-off period. The lengths of the on and off periods are optimally computed with respect to the average idle power by an offline algorithm. The ED scheme turns on the device whenever an event arrives and turns off when the device becomes idle.

We simulate different cases of single and multiple streams. For multiple streams, we only report results for random subsets of the 10-stream set due to space limitation. $\mathcal{S}(3,4)$, for instance, represents a case considering only streams S_3 and S_4 in Table 3.1. For FP scheduling policy of a multiple-stream set, the stream index defines the priority of a stream. Considering again stream set $\mathcal{S}(3,4)$, for instance, S_3 has higher priority than S_4.

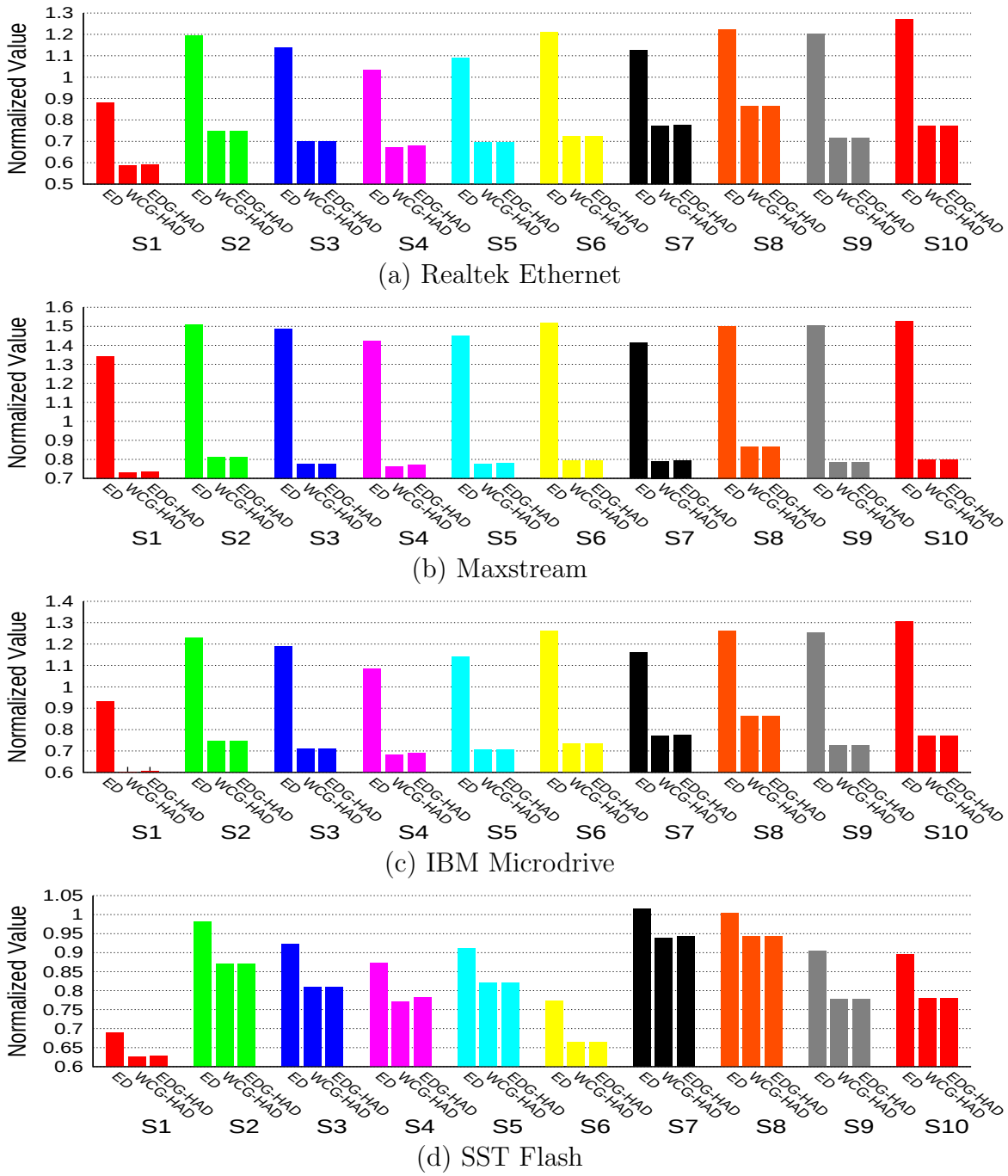
The simulation results for single-stream cases, multiple-stream cases, and computational expenses are reported. Although the periodic power management has provided interesting results, in the thesis we illustrate only the results concerning the adaptive solution. The adaptive case is more interesting for the topic of the thesis itself.

3.1.4.10 Single Stream

Firstly, we show the effectiveness of the proposed WCG-HAD and EDG-HAD schemes comparing to the OPT and ED schemes for single-stream cases. Because OPT does not consider the size of the system backlog, for fair comparison, we smooth out the backlog-size effect by setting backlog size to a relative large number, i.e., 60 events for this experiment. Figure 3.12 shows the *normalized* values of average idle power with respect to OPT for streams in Table 3.1 individually running on all four devices in Table 3.2. As depicted in the figure, both our proposals outperform the pure event-driven scheme as well as the optimal PPM scheme for all cases. On average, 25% of the average idle power is saved with respect to OPT for the deadline factor $\chi = 1.6$.

In general, larger ratio of `jitter` and smaller ratio of `wcet` with respect to a given period result in better energy saving, for instance, the cases for streams S_1 and S_6. On the contrary, the chance to reduce static power is less with larger `wcet` ratio, as in the case of streams S_2, S_7, and S_8. S_8 has the biggest `wcet-period` ratio, thereby conducting the least energy saving for all four devices. Another observation is that the overhead caused by the break-even time does not really affect the optimality of the average idle power. As shown in the figure, the ratio for a given stream does not change significantly for different devices, although the break-even time is considerably different for the four devices, e.g., 18.2 *ms* for the SST Flash and 152 *ms* for the Maxstream.

We also outline how the average idle power changes when the relative deadline of a stream varies. Figure 3.13 compares the four schemes by varying the deadline factor χ for streams S_8 and S_4. As the figure shown, our online schemes again outperform the other two. Another observation is that OPT can achieve good results only when the relative deadline is large. For the cases of small relative deadlines, it is even worse than ED. Our online schemes, on the contrary, can tackle different deadlines smoothly. The reason is that our online schemes consider the actual arrivals of event, resulting in a more precise analysis of the scheduling decision. Note that ideally our two online schemes, i.e., WCG-HAD and EDG-HAD, should produce identical results, because the WCG and EDG algorithms should theoretically converge to the same mode-switch moment, given a same trace. The slight deviation depicted in these two figures is due to the pessimistic activation decision in Line 4 of the EDG algorithm. This deviation is expected to become larger for multiple stream cases.



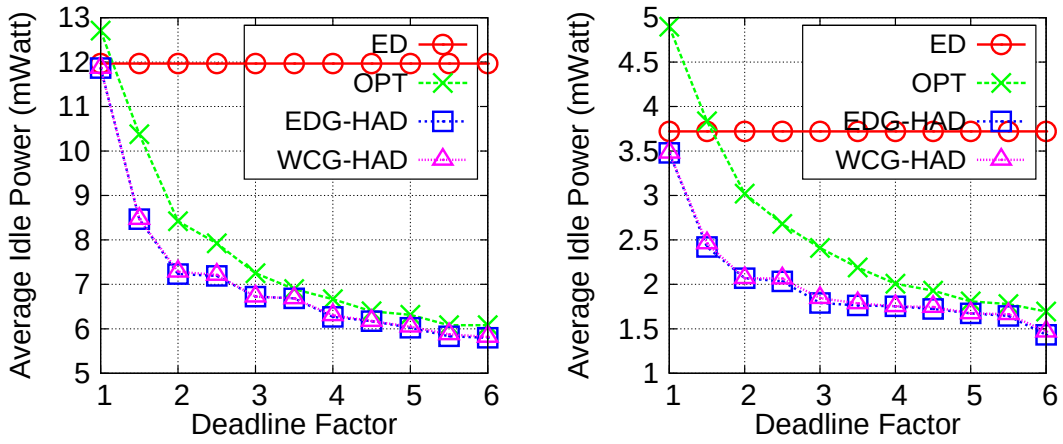
(a) Realtek Ethernet

(b) Maxstream

(c) IBM Microdrive

(d) SST Flash

Figure 3.12: Normalized average idle power consumption with respect to the OPT PPM scheme for single stream cases individually running on four different devices with deadline factor $\chi = 1.6$.



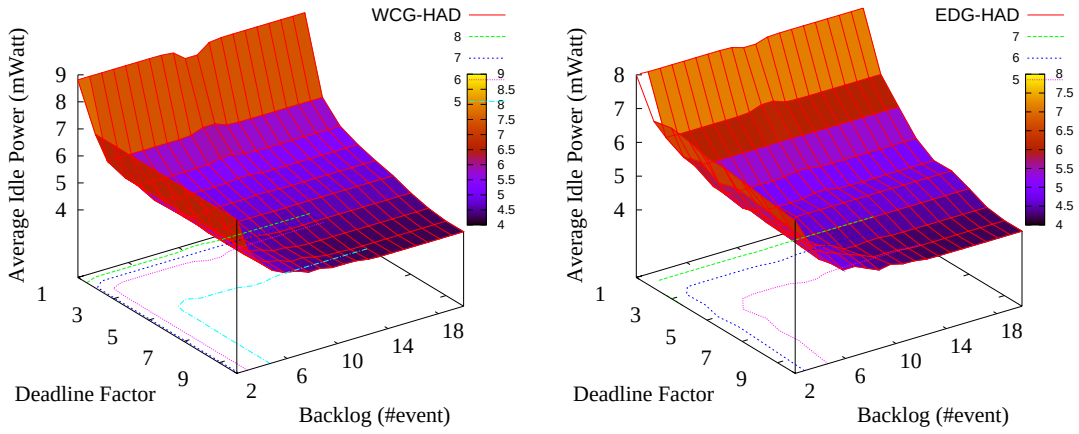
(a) Stream S_8 (b) Stream S_4

Figure 3.13: Average idle power consumption of different deadline settings on Realtek Ethernet.

3.1.4.11 Multiple Streams

We present simulation results for multiple-stream cases in this section. Figure 3.14 and Figure 3.15 depict simulation results for stream set $\mathcal{S}(6, 9, 10)$ running on Realtek Ethernet with distributed and global backlog allocation schemes, respectively. The results from these two figures demonstrate the effectiveness of our solutions for multiple streams. These results confirm as well as following statements: a) when relative deadline and backlog size are small, the average idle power is large, where the chances to turn off the device are slim. b) Increasing the relative deadline or backlog size individually helps reducing the idle power for only a certain degree. More increments do not conduct further improvements. c) Increasing both relative deadline and backlog size can effectively reduce the idle power, where more arrived events can be procrastinated and accumulated for each activation of the device. Another observation is that the global backlog scheme results in higher idle power compared to distributed backlog scheme in the cases of small backlog sizes. The reason is that the backlog demand curve has to consider the maximal value of the worst-case execution time of all streams in the set, incurring pessimistic predictions of activation moments.

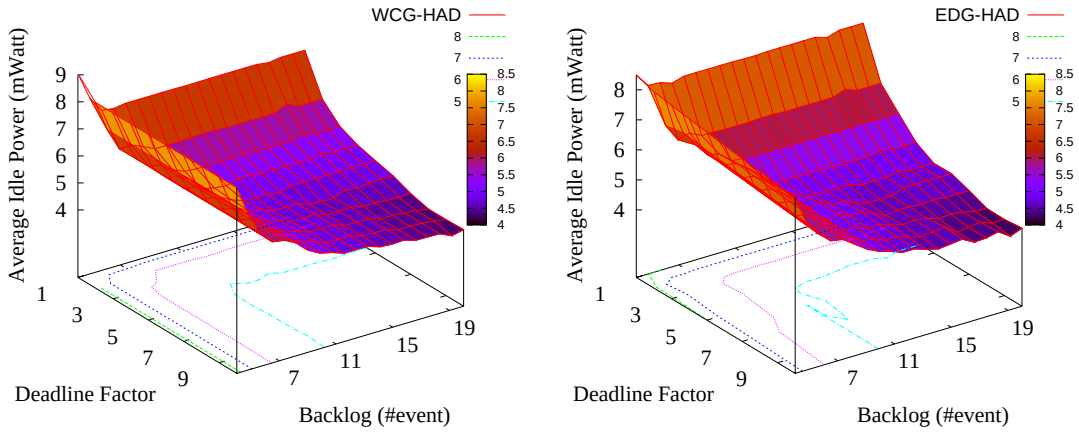
Computation Expense We also demonstrate the efficiency of our schemes by reporting the computational expenses. Figure 3.16 shows the numbers of activations of our algorithms and Figure 3.17 depicts the worst, best, and average case computational expenses for an activation. From Figure 3.16, we can find out that, given a same stream set, the numbers of activations for the EDG algorithm do not vary often as the relative deadline changes, which confirms to the principle of the algorithm. The fluctuations are caused by event arrivals when the device is at active mode. Such events do not activate the algorithm. The second observation is that the numbers of activations for EDG are depended on the numbers of streams running on the device while the numbers of activations for WCG quickly converge even more streams are involved. The reason



(a) EDF Scheduling (b) FP Scheduling

Figure 3.14: Average idle power consumption with respect to different deadline and backlog settings on Realtek Ethernet for stream set $\mathcal{S}(6, 9, 10)$ under distributed backlog allocation.

is that the activations of WCG are determined by the predicted turn-on moments that are depended on the backlog size and relative deadline. When the backlog size and relative deadline are large enough, the number of activations is one, no matter how many streams are added to the system. From above facts, one might conclude that WCG is better. EDG is, however, meaningful in the case when event arrivals are sparse. In such cases, the number of activations of EDG will be less than that of WCG. The results shown in the figure are caused by the dense-event trace generated by the RTS tools. Figure 3.17 presents the worst, best, and average case computational expenses of an activation of the proposed algorithms with respect to different deadline factors for stream sets $\mathcal{S}(1\text{–}4)$, $\mathcal{S}(3\text{–}6)$, and $\mathcal{S}(2, 4, 6, 8)$ individually running on Realtek Ethernet. Results for FP scheduling coupled with distributed backlog scheme and EDF scheduling coupled with global backlog scheme are shown in Figure 3.17(a) and Figure 3.17(b), respectively. We neglect the results for EDF scheduling with distributed backlog scheme due to the similarity to the FP case. From the figure, we can conclude that both our algorithms are efficient. The worst, best, and average case computation expenses of each activation are within the range of millisecond and are acceptable to the stream set in Table 3.1. By virtue of the new construction of the bounded delay function, the computation time are retained almost constant even with large relative deadlines. In general, EDG is more expensive than WCG and HAD, which are confirmed with the definition in Section 3.1.4.4. The last observation is that the computation expenses are not neglectable. There are also means to tackle this problem, for instance, setting the computation overhead as a safe margin for the computed sleep period or putting the activation itself as the highest priority events of the system. We do not elaborate them here, since they are not the focus of this set of papers. As shown in the figure, both schemes require a small computation time and the increment for the case of a longer relative deadline is considerably small, which makes our algorithms applicable online.



(a) WCG-HAD Scheme

(b) EDG-HAD Scheme

Figure 3.15: Average idle power consumption with respect to different deadline and backlog settings for stream set $\mathcal{S}(6, 9, 10)$ running on Realtek Ethernet under global backlog allocation and EDF scheduling policy.

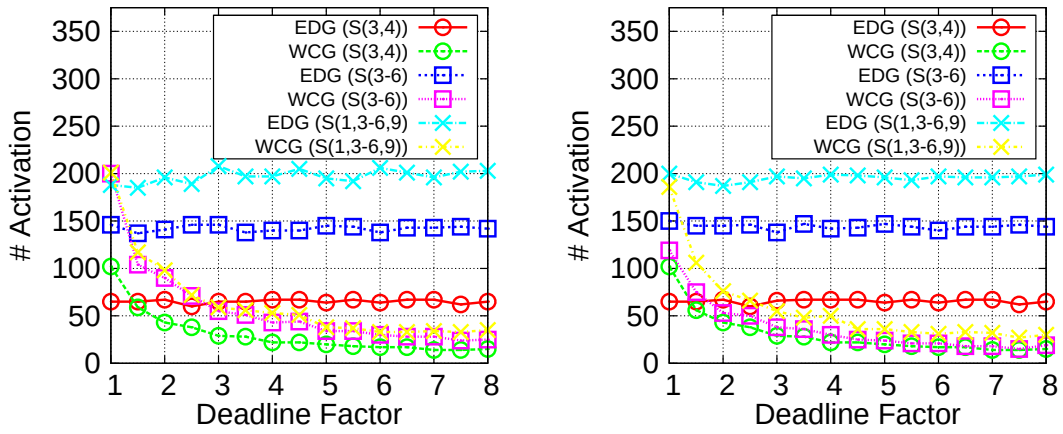
We can conclude that it is possible to develop efficient adaptive mechanisms (in this example related to the resource demand) not computationally expensive.

The work described in this part of the dissertation has been presented with the papers (77; 78; 79).

3.2 Energy Aware Scheduling with Constrained Resource

Embedded systems cover a very wide spectrum of application domains, from consumer electronics to biomedical systems, surveillance, industrial automation, automotive, and avionics systems. In particular, the technology evolution of sensor and networking devices paved the way for plenty of new applications involving distributed computing systems, many of them deployed in wireless environments, exploiting the mobility and the ubiquity of components. Moreover, in most cases, devices are battery operated, making energy-aware algorithms of paramount importance to prolong the system lifetime.

In each node of the system, at the processor level, two main mechanisms can be exploited to save energy: the DVS and the DPM. The former is used to reduce the dynamic energy consumption by trading the performance for energy savings. For DVS processors, a higher supply voltage, generally, leads to both a higher execution speed/frequency and also to a higher power consumption. On the other hand, DPM techniques are applied to control the change of the system mode leading to consume less leakage power, e.g., by postponing the tasks execution as long as possible still guaranteeing the schedulability of those tasks. At the network level, the energy consumption due to communication is usually managed by DPM techniques, although



(a) EDF scheduling

(b) FP scheduling

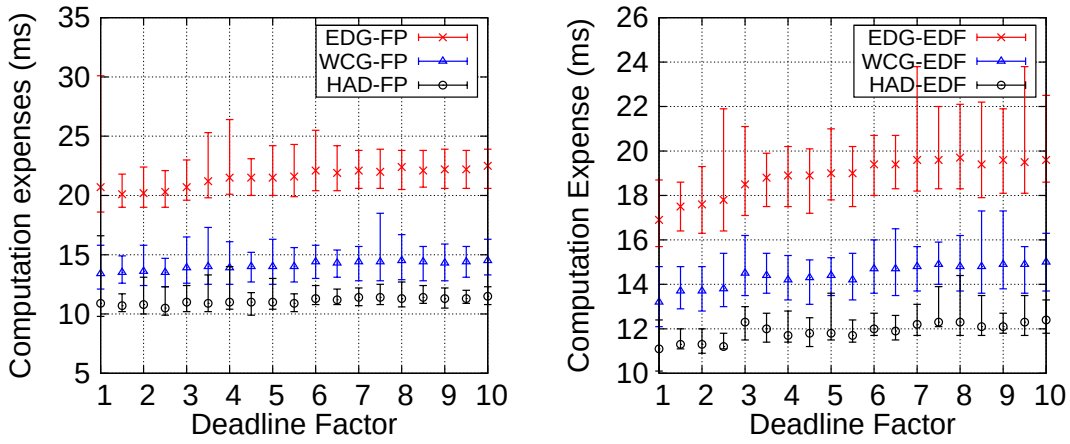
Figure 3.16: Numbers of Activations for different deadline settings of stream sets $\mathcal{S}(3,4)$, $\mathcal{S}(3\text{–}6)$, and $\mathcal{S}(1,3\text{–}6,9)$ running on Realtek Ethernet under distributed backlog allocation with individual backlog size of 10 events.

other mechanisms have been proposed in the literature, as the Dynamic Modulation Scaling (DMS) (138).

When the dynamic power is dominant, DVS techniques are used to execute an application at the minimum processor speed that guarantees meeting real-time constraints. Conversely, when static power is dominant, there exists a critical processor speed below which the energy wasted is greater than that consumed at the critical speed (47). For this reason, some authors recently proposed energy-aware algorithms that combine DVS and DPM techniques to improve energy saving (62; 177).

In wireless distributed embedded systems the quality of service offered is important as well as the power consumption. Messages have to be transmitted in order to guarantee the desired quality, (50; 89). The transmission bandwidth provided to each node allows to send messages and the sending itself represents an energy cost to take into account. Although a lot of research has been done to reduce power consumption while guaranteeing real-time requirements, see next section, most papers focus either on task scheduling or network communication. Instead, a co-scheduling of task and messages allows to explore more degree of freedom and might allow to save more energy to each node composing the system. The natural constrained resource is the energy but mostly the bandwidth which has to be given through the classical resource reservation mechanisms.

In (137), the combination of the two has to be explored in order to find optimal solutions to the energy consumption problem. Moreover the resource constrained condition (apart the energy consumption) has to be considered. We have then have extended the analysis, started with (77; 78; 79), by applying DVS and DPM together. Most of all is the introduction of resource constraints, such as the communication bandwidth, that defines the novelty of the proposed solution. It lead the way to the possibility of modeling complex systems, including distributed systems.



(a) Distributed backlog allocation

(b) Global backlog allocation

Figure 3.17: Worst, best, and average case computation expenses of an activation of the proposed algorithms with respect to different deadline factors for three different 4-stream sets $\mathcal{S}(1\text{–}4)$, $\mathcal{S}(3\text{–}6)$, and $\mathcal{S}(2,4,6,8)$ individually running on Realtek Ethernet.

DPM and DVS represent two different ways of applying the computational resource for task scheduling. These methodologies act on two different perspective of the resource reservation problem. In particular, DVS techniques modify the available resource and are able to adapt the resource provisioning rate to the actual resource demand from the applications. It can be seen as a greedy shaper techniques (93) applied to the computational resource. See Section 4.2.9 for more details. DPM techniques instead, by postponing the resource demand as much as possible, delay the resource demand. In the former section we have seen an example of adaptive DPM scheduling. Such example shows the capability of adaptation to dynamic conditions (arrival of the tasks) by varying the resource demand. This is the dual aspect of adapting the resource provided: *the demand is equivalent to the minimal resource requirement.* In the remaining part of the chapter both DVS and DPM are applied in order to find optimal (or at least sub-optimal) solutions to adapt to changing conditions of the scheduling system.

3.2.1 System Models

We consider a distributed real-time embedded system composed by wireless nodes. Each node executes a set of independent tasks that need to exchange information with tasks running in other nodes and accomplishing the system goal. A node is modeled as component $c = (\Gamma, S, M, B)$ which takes as input a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$, a scheduling algorithm S, a message set
$M = \{s_1, \ldots s_m\}$ and a transmission bandwidth B.

Task are scheduled by the node processor according to the given scheduling policy S, while messages are transmitted during the intervals in which the bandwidth B is made available by the adopted protocol. Note that a node is not required to work during the remaining intervals.

The analysis we are proposing is focused on a bandwidth allocation protocol that provides a slotted bandwidth according to a Time Division Multiple Access (TDMA) scheme. To decouple task execution from the communication activity, all tasks in a node build packets and move them to a shared communication buffer in the processor memory. When the channel is available, packets are transferred from the communication buffer to the transceiver for the actual transmission.

As outputs, each component could provide a set of performance indexes, such as message delays, task response times, and the energy consumption. At the moment we provide only energy consumption performances; other indexes can be added with a minimal effort, i.e. the response time R as a function of the task assignments. The component interface is schematically illustrated in Figure 3.18

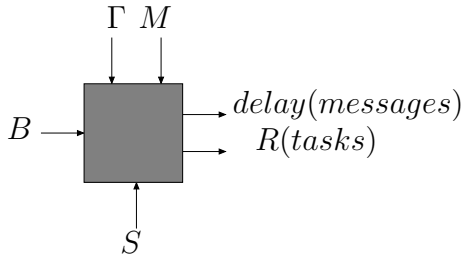


Figure 3.18: Node interface.

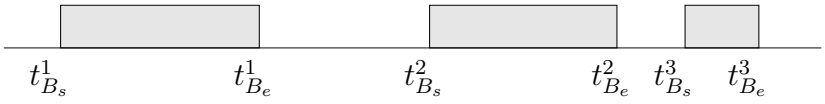


Figure 3.19: Bandwidth assignment.

Workload and Resource Models An application Γ consists of a set of periodic tasks, where each task $\tau_i = (C_i, T_i, D_i)$ is characterized by a worst-case execution time C_i, a period T_i, and a relative deadline D_i. Each task τ_i produces a message stream $s_i = (m_i, D_{m_i})$ characterized by a payload m_i and a deadline (relative to the task activation) for the message transmission or reception.

In order to decouple the message production from the job execution we suppose that the message is generated at the end of the job which executes without slack, so the message activation time is equal to the job deadline. The produced messages are enqueued in a buffer and then transmitted as soon as the bandwidth becomes available. Assuming that packets ready to be transmitted are stored as soon as they are created and that the time for moving them is negligible, then message transmission does not affect task scheduling.

In each node, the computational resource (i.e., the processor) is assumed to be always available at any time t, hence it is modeled as a straight line $f(t) = t$. On the other hand, the communication bandwidth B is assigned by a bandwidth manager (running in a master node) in a slotted fashion. In general, the transmission bandwidth

is modeled as set of disjointed slots $B = \{b^1, \dots b^r\}$, where each slot is described by a start time $t^i_{B_s}$ and an end time $t^i_{B_e}$. An example of slotted bandwidth assigned to a node is shown in Figure 3.19.

Power Model Each node consists of a CPU (processing element) and a Transceiver (transmitting and receiving element). Each device can be in one of the following states:

- *sleep*. In this state, the device is completely turned off and consumes the least amount of power P_σ; however, it takes more time to switch to the active state.
- *standby*. In this state, the device does not provide any service, but consumes a small amount of power P_s to be ready to become active within a short period of time.
- *active*. In this state, a device performs its job, executing tasks or handling messages. The power consumed in this state is denoted as P_a.

For a processor that supports DVS management, the power consumed in active mode depends on the frequency at which the processor can execute. Such a frequency is assumed to vary in a range $[f_{min}, f_{max}]$, while the processor execution speed s is defined as the normalized frequency $s = f/f_{max}$. In particular, for the processor in active mode we use the power consumption model derived by Martin et al. (111), which can be expressed as

$$P_a(f) = a_3 f^3 + a_2 f^2 + a_1 f + a_0 \tag{3.29}$$

where

- a_3 is the third order coefficient related to the consumption of the core sub-elements that vary both voltage and frequency;
- the second order term a_2, describes the non linearities of DC-DC regulators in the range of the output voltage;
- a_1 is the coefficient related to the hardware components that can only vary the clock frequency;
- a_0 represents the power consumed by the components that are not affected by the processor speed (like the leakage).

Switching from two operating modes takes a different amount of time and consumes a different amount of energy which depends on the specific modes, as shown in Figure 3.20. In particular, the following notation is used throughout the chapter: $t_{a-\sigma}$ and $E_{a-\sigma}$ are the time and the energy required for active-sleep transition, while the active-standby transition is described by t_{a-s} and E_{a-s}. For all devices we have that $P_\sigma < P_s < P_a$ and $t_{s-a} < t_{\sigma-a}$. In (137), we assume also that switching between the standby mode and the active mode has negligible overhead, compared to the other switches, which is the same assumption made by other authors (174; 178).

A simplified power consumption model is adopted for the transceiver to concentrate on the interplay between DVS and DPM for the processor. The communication bandwidth is then considered as a constraint for serving the schedule that minimizes power consumption while guaranteeing a desired level of performance. In particular, a transceiver is assumed to be either in *on* (equivalent to the active state) or *off* (equivalent to the sleep state) mode only (not in *standby*). Whenever the transmission bandwidth is available the transceiver is considered in on mode; the power used to transmit and receive messages is assumed to be equal to P_{on}, that is: $P_{tx} = P_{rx} = P_{on}$. Whenever the transmission bandwidth is not available, the transceiver is assumed to be in off mode with a power consumption equal to P_{off}.

Table 3.3 summarizes all the allowed modes with their characteristics, while Figure 3.20 illustrates the mode transition diagram and the transition costs in terms of time and energy. It is depicted a more complex case than 3.1 because two devices are consider at one time.

	Radio On	Radio Off
CPU Sleep	/	$P_\sigma + P_{off}$
CPU Standby	/	$P_s + P_{off}$
CPU On	$P_a(f) + P_{on}$	$P_a(f) + P_{off}$

Table 3.3: Power model: allowed power modes and power contributions.

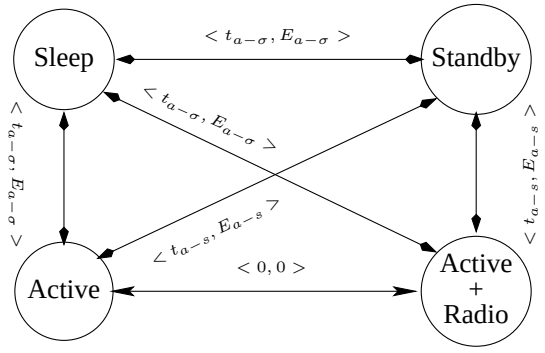


Figure 3.20: Power model: transition time and energy costs.

3.2.2 Schedulability Analysis

In real-time systems, the demand bound function (dbf) and the supply bound function (sbf) are typically applied to verify the schedulability of real-time applications under certain scheduling algorithms and resource provisioning (21; 144). In particular, the $\mathsf{dbf}(t)$ of an application Γ describes the resource requirement that the application demands to the scheduling element in any interval $[0, t)$. On the other hand, the $\mathsf{sbf}(t)$ describes the resource amount the scheduler supplies in any interval $[0, t)$. Intuitively,

the real-time constraints of a scheduling component are met if and only if, in any interval of time, the resource demand by the component is always below the resource supply curve, as shown in Figure 3.21 with various sbf depicted. The following of the section instantiates such an analysis under earliest deadline first and fixed priority scheduling paradigms.

In the case of an EDF scheduling paradigm, Baruah (19; 21) showed that the dbf of the task set Γ is

$$\mathsf{dbf}(t_1, t_2) = \sum_{i \in \Gamma} \left(\left\lfloor \frac{t_2 + T_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1}{T_i} \right\rceil \right) C_i,$$

and the schedulability of a task set is then guaranteed if and only if

$$\forall t_1, t_2 \quad \mathsf{dbf}(t_1, t_2) \leq \mathsf{sbf}(t_1, t_2). \tag{3.30}$$

The dbf represents both the minimal resource demand from the application Γ and the minimal feasible service requirement sbf^* that guarantees the tasks execution within their timing constraints, $\mathsf{sbf}^* =$
dbf.

Under fixed priority scheduling, the analysis can be carried out using a similar approach (78), but is not reported here for space limits.

Once the $\mathsf{dbf}(t)$ has been computed for an application Γ, the minimum supply bound function that guarantees the feasibility of Γ is $\mathsf{sbf}^*(t) = \mathsf{dbf}(t)$. In the processing model considered in (137) the processor supply function is a straight line $\mathsf{sbf}^l(t)$ that increases with a constant speed s whenever the processor is in active mode while it is steady if the processor is in standby or sleep mode. In particular, the minimum *straight line* supply bound function sbf^{l*} above sbf^* (which keep Γ feasible) is

$$\mathsf{sbf}^{l*} = min\{\mathsf{sbf}^l | \mathsf{sbf}^l \geq \mathsf{sbf}^*\}.$$

Every $\mathsf{sbf}^l \geq \mathsf{sbf}^{l*} \geq \mathsf{sbf}^*$ keeps the system feasible, because it anticipates the processor execution with respect to sbf^* and sbf^{l*}, which are feasible.

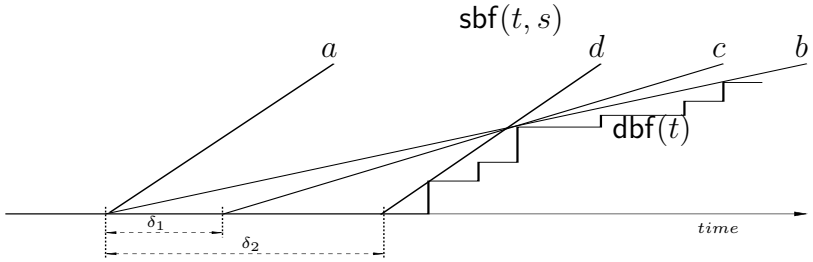


Figure 3.21: Supply Bound Function: different possible supply bound functions varying the resource provisioning rate s and the execution starting delay δ.

When applying DPM scheduling techniques it is possible to delay the task execution inserting a sleep/standby interval δ. In this case, the energy saving consist into finding the largest δ that still guarantee the feasibility of the application (see (78) for more

details) by postponing the task executions. That δ is computed by executing the processor at its maximum possible frequency f_{max} as is shown in Figure 3.21 with the sbf curve (d).

If no pending jobs are present at the actual instant t, selecting δ that finishes before the first activation time t_{act} produces only an energy waste because it reactivates the processor while the ready queue is still empty, thus, the δ is bounded by t_{act}.

The DVS technique instead, changes the processor execution frequency thus varying the sbf slope. By the analysis with
$\mathsf{sbf}^l(t, s_{max})$ (sbf as function of t and speed s, in this case the maximum speed s_{max}), the t_{feas} is derived as the last time instant that a schedulable activation of the task set Γ can start. The DPM, through the sbf-dbf analysis, derives such a t_{feas} as the maximum waiting δ from the actual analysis instant t before the task execution. Applying the DVS after the DPM considerations means anticipating such activation time to t_x (with $t_x \leq t_{feas}$) by executing at a smaller frequency as shown in Figure 3.21 with sbf (c) and (b), consequently saving processing energy. The construction of any sbf as we are doing, follows the constraint that sbf $\geq$ dbf, thus any sbf assures the schedulability of the task set if it is assumed schedulable by any other scheduling algorithm.

3.2.3 Energy Aware Scheduling

In Figure 3.22 it can be seen a task execution example with 3 tasks that activate at time 3, 8 and 10 sec (*Activations* on the figure). There is also a defined transmission bandwidth where the system is forced to transmit. A normal task scheduling would have started as soon as the tasks are active and with the maximum allowed execution speed (dashed line). Instead, a task scheduling that takes into account energy matters postpones as much as possible the task execution up to the task deadlines (the continuous line starting at $t = 25$sec with maximum allowed execution frequency) and adapts the execution frequency in order to cope better with the transmission bandwidth (continuous line starting at $t = 15$sec).

Such an example motivates the need of an appropriate task execution to save energy. This section defines a scheduling algorithm that integrates DPM and DVS techniques to reduce energy consumption, while coping with the available transmission bandwidth and guaranteeing the application timing constraints. The algorithm is named the Energy Aware Scheduling (EAS).

The EAS algorithm is applied at a generic time instant t; it computes the domain $[t_{min}, t_{max}]$ for the candidate next activation point t_a that satisfies DPM requirements and timing constraints at the first step. The DPM is applied to compute the maximum t_{feas} at which the processor can start executing at its maximum speed s_{max} keeping the task set schedulable. Second, task executions are anticipated with s_{max} (still keeping the maximum possible execution speed), to approach the starting point t_{B_s} of the transmission bandwidth. The final step selects the minimum processor speed s needed to keep the task set schedulable. In order to take message communication into account, task schedule is arranged to overlap with the bandwidth schedule. In this way the message sending correspond to the task execution, allowing saving more energy.

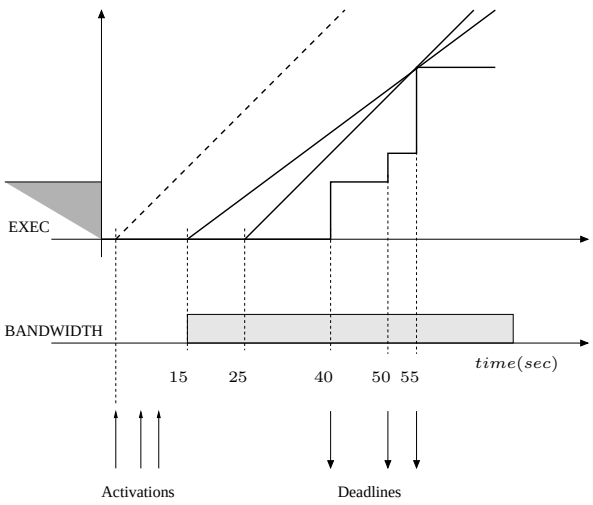


Figure 3.22: An example of Energy Aware Scheduling that applies DVS and DPM and copes with the transmission bandwidth.

The objective of the EAS algorithm is to compute that time t_a that minimizes the energy consumption of the next task scheduling and message transmission. Any valid activation point t_a must take into account the feasibility bound t_{feas}, the starting of the transmission bandwidth t_{B_s},

and the next activation time t_{act}[1]. Those dependencies define the interval $[t_{min}, t_{max}]$ where t_a belongs to, $t_a \in [t_{min}, t_{max}]$. The value t_{max} is the minimum among t_{feas} and t_{B_s}. If the processor has no pending jobs, the value of t_{min} is set to the next activation time t_{act}, otherwise $t_{min} = t$. By the definition of the interval $[t_{min}, t_{max}]$, the actual selection of t_a is done by computing the time that minimizes the energy consumption from now, time t, to the end of the evaluation period t_F (discussed in Section 3.2.2. Algorithm 6 resumes the sequence of steps of the EAS algorithm, while Figure 3.23 depicts the EAS application sequence and the result. The feasibility of any task set scheduled according the EAS is guaranteed by construction because each step keeps the feasibility.

The system energy consumption $E(t_a)$ is computed using the following formula

$$E(t_a) = (t_a - t)P_{\sigma/s} + (t_F - t_a)P_a(s(t_a)f_{max}) + \\ +2E_{a-\sigma} + E_{radio}(t, t_F), \tag{3.31}$$

which is the collection of all the consumption models detailed in the former section. In particular $E_{radio}(t, t_F)$ is the energy the transceiver consumes in $[t, t_F]$ as a function of the available bandwidth and $P_{\sigma/s}$ the not-working power consumption which is equal to P_σ if $t_a - t \geq 2t_{a-\sigma}$ and P_s otherwise.

As we said, the problem to be solved is to find t_a in the research interval $[t_{min}, t_{max}]$ that minimize the energy consumption as

$$t_a \mid min_{t_a \in [t_{min}, t_{max}]}\{E(t_a)\}. \tag{3.32}$$

[1] t_{act} is the closest next activation time of the task set after actual time t. By the task set activation time it is intended the first task activation time after t among all the tasks composing Γ.

Algorithm 6 Energy Aware Scheduling - EAS

procedure $t \mid t \notin B$
  Compute the $\mathsf{dbf}(t)$;
  Compute $\mathsf{sbf}^{l*}(t) = \mathsf{sbf}^{l}(t, f_{max})$ and obtain t_{feas};
  Calculate $t_{max} = min\{t_{feas}, t_{B_s}\}$;
  if No pending jobs at t **then**
    $t_{min} = t_{act}$;
  else
    $t_{min} = t$;
  end if
  Find $t_a \in [t_{min}, t_{max}] \mid min_{t_a} E(t_a)$;
  if $t_a \geq t + 2t_{a-\sigma}$ **then**
    Put the processor in sleep state in $[t, t_a]$;
  else
    Put the processor in standby in $[t, t_a]$;
  end if
  Compute the min frequency f_a or slope s_a guaranteeing feasibility.

In the next section such a relationship will be deeply exploited by comparing the energy saving contributions from the DVS and DPM.

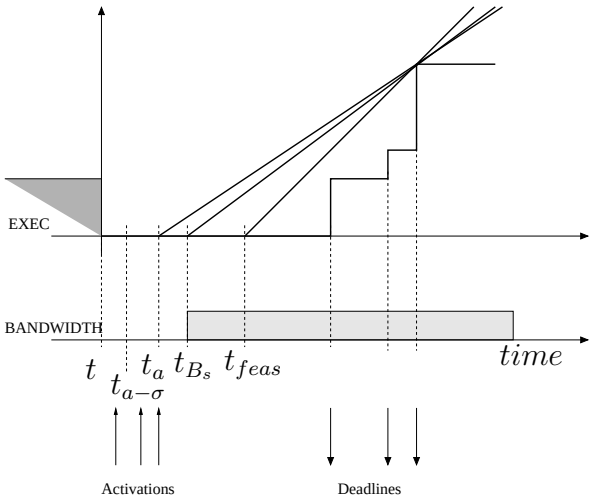


Figure 3.23: Feasibility and bandwidth guarantee: EAS algorithm with three possible processor executions together with the time instants when are applied.

3.2.4 Energy Aware Scheduling: implementation Details

In the previous section we have presented our scheduling algorithm that applies DVS and DPM techniques with the common idea of saving energy. The constrained transmission bandwidth is assumed provided by the network coordinator. In this way we

can concentrate on the node behavior and its energy saving strategies.

In this section we focus on some characteristics of the algorithm to better understand its behavior. First is analyzed how to select the instants in which execute the algorithm, then a more detailed analysis of the energy minimization problem is carried out, and finally the computation cost of the method is evaluated.

3.2.4.1 EAS Applicability

The EAS algorithm has been developed in order to be applied at each scheduling point, such as job activation and termination, preemption points, etc. The EAS overhead suggests to use it only when it could produce the maximum results. A natural choice is the instants when the ready queue empties and the processor is about to enter the idle state; using the algorithm there allows to maximize the idle time and better exploit the advantages of DPM. By the *idle time* it is intended the scheduling idle time, where the processor ready queue is empty. Another interesting point where to apply the EAS algorithm is the bandwidth finishing instant t_{B_e}. Indeed, by forcing the processor to be active during the transmission bandwidth interval, the possibility of slack time for the running tasks increases. So t_{B_e} is another application point of the algorithm and where it is possible to recover the eventual slack time. Contrarily, any processor idle time inside the bandwidth B is not an interesting point where to apply the EAS algorithm because of one of the reasonable assumption done. Indeed, it has been assumed to choose the frequency at the beginning of the processor active period hence, inside the bandwidth, it is only possible to apply the same frequency as it was before entering the bandwidth slot.

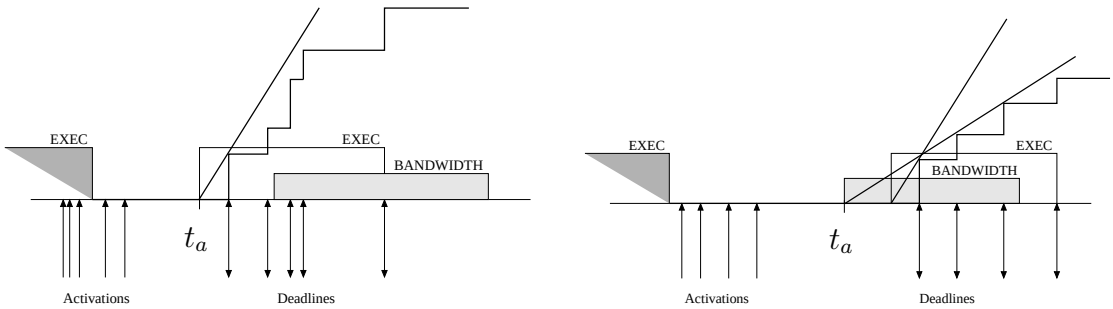
In this work we consider the bandwidth purely as a constraint to the processor scheduling and the energy saving problem we are tackling with. Our system node receives the bandwidth and cannot control it in any mode allowing to assume that it is enough to let all the messages produced by the task set being transmitted[1].

Scenarios: We have defined cases that recall the possible scenarios that can happen in a generic time interval of the system execution. We consider the actual time t, where either the processor has just emptied the task execution queue and is about to go in sleep, or the transmission bandwidth is just ended. According to the next bandwidth chunk and the processor demand of the next task execution (next after t), there are different possible starting time for the tasks execution which result in different energy consumption conditions.

Case 1: Task scheduling starting before the transmission bandwidth The application execution is required to start before the beginning of the bandwidth $t_a \leq t_{B_s}$, due to tasks timing constraints as depicted by Figure 3.24(a). In this case the resulting demand bound function requires the task scheduling starts before the beginning of the transmission bandwidth. The energy consumption is

$$P_\sigma(t_a - t) + P_a(s(t_a))(t_{B_s} - t_a) + [P_a(s(t_a)) + \\ + P_{on}](t_{B_e} - t_{B_s})$$

[1]In the future analysis, we plan to consider the bandwidth as resource to be optimized as well as the system energy and not just a constraint.



(a) Case 1: the feasibility analysis forces the execution to start before the bandwidth

(b) Case 2: execution can start at the beginning of the bandwidth

Figure 3.24: Scenarios: the execution starts before or at the beginning of the transmission bandwidth

which is given by the cost of the processor for executing before the bandwidth $(P_a(s(t_a))(t_{B_s} - t_a))$, the cost of the mandatory execution and transmission inside the bandwidth and the transceiver energy cost due to the transmission bandwidth $([P_a(s(t_a)) + P_{on}](t_{B_e} - t_{B_s}))$.

Case 2: Scheduling starting at the transmission bandwidth In Figure 3.24(b) is represented the second case where t_{feas} happens after the beginning of the transmission bandwidth. This way the processor activation can be advanced in order to meet the beginning of the bandwidth t_{B_s}. The EAS algorithm tends to overlap the processor activation time t_a to the beginning of the bandwidth in order to cope with the bandwidth constraint and save more energy. Although, t_a can still be different than t_{B_s} because it is the result of the energy minimization problem. The energy consumption is case of $t_a = t_{B_s}$ is

$$P_\sigma(t_a - t) + [P_a(s(t_a)) + P_{on}](t_{B_e} - t_{B_s}),$$

which differs from the previous case by the $P_a(s(t_a))(t_{B_s} - t_a)$: there is no energy consumption before the bandwidth. Whenever $t_a = t_{B_s}$ it is also possible to anticipate the task execution with respect to t_{B_s} to let tasks produce messages ready to be transmitted at the beginning of the bandwidth. In this way the bandwidth would be better applied by the node even if the energy consumption slightly deteriorate. Since we are focusing our analysis on the energy consumption, we privilege $t_a = t_{B_s}$ even if our approach is easily scalable to solve a wider range of problems.

3.2.5 Energy Minimization

The EAS algorithm every time is applied to compute the next execution starting point t_a for the processor. That t_a depends on the processor speed s and on its domain $[t_{min}, t_{max}]$. The relationship $s(t_a)$ among the speed and the activation point is quite intuitive and depends on the demand bound function of the task set. An heavy loaded processor results in a high dbf and consequently a high speed is required for the feasible execution of task set itself. Moreover, a slight difference among two dbf results in different speeds that keep the dbfs feasible. By that strict dependency, it is impossible

to derive a closed-form formula for $s(t_a)$. Instead we can conclude its convexity because is defined as the maximum of straight lines with slope $\frac{y}{x-t_a}$ starting at t_a,

$$s(t_a) \stackrel{def}{=} \max\{\frac{y}{x-t_a}\} \quad \forall x, y \ \in \mathsf{dbf}.$$

Each (x, y) is a points of the dbf where to control the slope/speed.

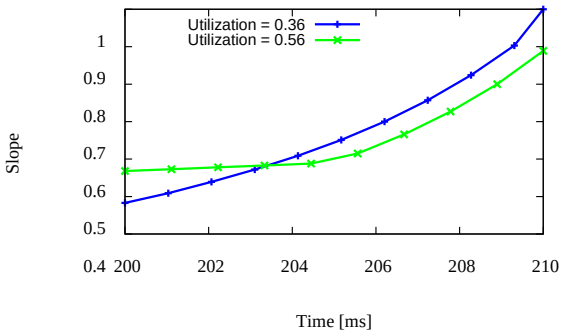


Figure 3.25: Slope/activation time relationship $s(t_a)$ depending on the demand bound function. Two task set with utilization $U = 0.36$ and $U = 0.56$ respectively.

Figure 3.25 depicts the dependency of the execution speed s_a to the task set and the dbf associated in a simulated case. By the picture it is possible to guess the convexity of $s(t_a)$. Figure 3.27 shows a general dependency among the energy consumption and the model of the processor applied. According to the model, the DVS or the DPM could be more or less applied.

The energy consumption $E(t_a)$, computed by Equation 3.32, has to be minimized evaluating the energy and workload requirements in $[t, t_F]$. The end instant t_F depends on the next time in which the EAS algorithm will be executed that could be either the next scheduling idle time or the end of the bandwidth slot t_{B_e}. Such a dependency is a consequence of the policy applied, because in the same interval it could happen to have different services provided and then different workloads executed. Pure DVS starting to execute at the beginning of the interval could provide a different computational resource than a combination of DVS and DPM where executing at t_a with $t_a \geq t$. Figure 3.26 details the different services provided in the same interval by varying the execution speed and the processor activation point t_a.

To resolve the previous dependency we decouple the ending instant t_F from t_a by choosing t_F as the maximum among the scheduling idle times. The DVS applied at the beginning of the interval (time t) with the minimum feasible speed results in maximizing the time distance between t and the beginning of the next idle time, keeping the time constraints guarantee. The value of t_F is the minimum between the DVS idle time and the t_{b_E}.

Looking for the activation time t_a that minimizes the energy consumption $E(t_a)$ must take into account the service supplied by the processor during the interval $[t_a, t_F]$. The research of the minimum does not have to find the value of t_a that imposes the minimum energy consumption, but the one that requires the minimum energy in order to provide the amount of service needed by the workload under the the constraints.

The function that has to be minimized is not the plain energy but the ratio between the energy required in the interval $[t, t_F]$ with the chosen t_a and the service supplied in that interval $S(t_a) = (t_F - t_a)s_a$. To compute t_a the function that has to be minimized is then

$$t_a | \min_{t_a} \frac{E(t_a)}{S(t_a)}. \tag{3.33}$$

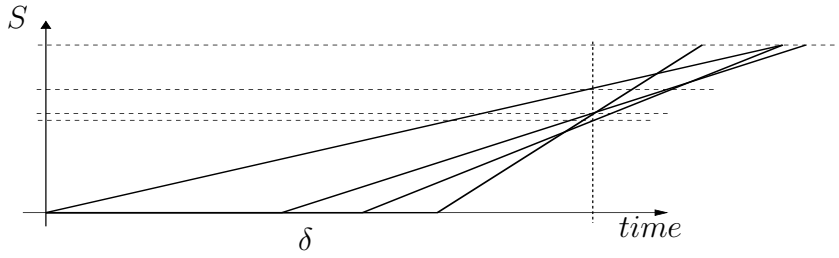


Figure 3.26: Different services S provided in an interval δ by different execution policy applied.

The EAS algorithm looks for the minimum ratio $\frac{E(t_a)}{S(t_a)}$ by spanning $s(t_a)$ and varying t_a. By the analysis of Equation 3.31 and 3.33 we derive that $E(t_a)$ is convex in the interval $[t_{min}, t_{max}]$. The convexity is guaranteed because Equation 3.31 is the composition of linear functions $(P_\sigma(t_a - t))$, convex functions $(P_a(s(t_a)))$ and convex functions multiplied by linear and positive functions in such interval $(P_a(s(t_a))(t - t_F)$. The ratio $\frac{E(t_a)}{S(t_a)}$ is convex as well as the ration among a convex function and a linear function. With a convex $\frac{E(t_a)}{S(t_a)}$ the minimum exists and can be found with well known and efficient methods, in particular we have applied the bisection method. There are two singular cases for that minimum. $t_a = t_{min}$ that shows the condition where the DVS effect is prominent with respect to the DPM one. Vice versa, $t_a = t_{max}$ shows a major effect of the DPM over the DVS. Since EAS has to be applied on-line, the research algorithm has to be quick.

Computational Cost The complexity of the EAS algorithm comes form the components of the algorithm itself. By the order of their application in EAS, there is

1. the computation of the dbf which has a polynomial complexity $O(n)$ where n is the number of tasks activation in the analysis interval $[t, t_F]$.
2. To compute t_{feas}, once known the dbf, requires to compute the intersection among the sbf^l* and x-axes which means $O(n)$.
3. The computation of the two bounds t_{min} and t_{max} has a complexity of $O(n)$ due to the min/max operations and mostly t_{act} which is searched.
4. Finally, finding t_a means finding $t_a \mid min\ E(t_a)$. To solve this we have applied the bisection method which has a polynomial complexity.

Taking into account all the contributions, the EAS has a polynomial complexity that makes it applicable on-line.

3.2.6 Simulations

This section presents some simulation results achieved on the proposed EAS method. An event-driven scheduler and an energy controller simulator has been implemented in C language and interfaced to Gnuplot.

Simulation Setup The simulator receives a task set and a bandwidth assignment as inputs. The task set is executed with a chosen scheduling policy. The energy consumed to schedule tasks and to transmit messages is computed at each simulation run. A simulation run consists of scheduling one task set with the assigned bandwidth until the task set hyper-period *hyp*. The power consumption $\frac{E}{hyp}$ in the hyper-period is then considered.

The scheduling policies applied are:

- *EDF* with no energy considerations, meaning that the processor is assumed always active at the maximum frequency, even if tasks are not ready to execute.
- *pureDVS* on top of an EDF scheduling algorithm. Only speed scaling is applied off-line to guarantee feasibility and the processor speed is set to that value. Online changes are not allowed.
- *pureDPM* where the task execution is postponed as much as possible and then scheduled by EDF. The execution is at the maximum processor speed.
- DVS and DPM are combined with EDF through the *EAS* algorithm.

A simulator infrastructure automatically generates a stream of tuples (U, n_t, B, n_B), where U denotes the utilization of the generic task set Γ, n_t the number of tasks, B the communication bandwidth (expressed as a percentage of the hyper-period), and n_B the number of slots in which the bandwidth has been split. Both the task set utilization and the number of tasks are controlled by the task set assignment (U, n_t). The bandwidth assignment (B, n_B) allows to control both the total bandwidth and its distribution within the hyper-period.

Given the total utilization factor U, individual task utilizations are generated according to a uniform distribution (24). Each bandwidth slot is set in the hyper-period with a randomly generated offset. To reduce the bias effect of both random generation procedures, 1000 different experiments are performed for each tuples (U, n_t, B, n_B) and the average is computed among the results obtained at each run.

Two different CPUs have been considered: the Microchip DsPic (DSPIC)[1] and the Texas Instruments (TI)[2], both using the CC2420 transceiver as communication device. Table 3.4 and Table 3.5 report the parameters that characterize the power model of the processors and the transceiver used in these tests, according to the models described in Section 6.1.1. Minimum and maximum frequencies of the CPUs are taken from the device data-sheets, whereas the coefficients $[a_0, a_1, a_2, a_3]$ comes form Equation 3.29.

[1] DSPIC33FJ256MC710 microprocessor

[2] TMS320VC5509 Fixed-Point Digital Signal Processor

CPU	P_s [mWatt]	f $[f_{min}, f_{max}]$ [Mhz]	$P_a(s)$ $[a_0, a_1, a_2, a_3]$ [mWatt]	P_σ [mWatt]	t_{sw} [sec]
TI	–	$[25, 200]$	$[7.7489, 17.5, 168.0, 0.0]$	0.12	0.00125
DSPIC	9.9	$[10, 40]$	$[25.93, 246.12, 5.6, 0.0]$	1.49	0.020

Table 3.4: Power profiles for processing devices.

Transceiver	P_s [mWatt]	P_a [mWatt]
CC2420	0.066	62.04

Table 3.5: CC2420 Transceiver power profile.

Simulation Results In a first simulation, we tested the power consumption of the CPUs as a function of the activation time. Figure 3.27 shows a general dependency of the power consumption from the model adopted for the processor. The figure shows also that both CPUs are DVS sensitive, in the sense that both privilege DVS solutions than the pure DPM ones. Indeed, the DSPIC and the TI exhibit a lower energy at t_{min} than at t_{max} (respectively 160 and 195 in this case as one of the interval of analysis along the whole execution interval). This means that a pure DVS solution costs less than a pure DPM one. Moreover, the DSPIC shows a global minimum inside the interval, meaning that a combined policy is able to reduce energy consumption. The time value corresponding to the minimum is the t_a that has to be found.

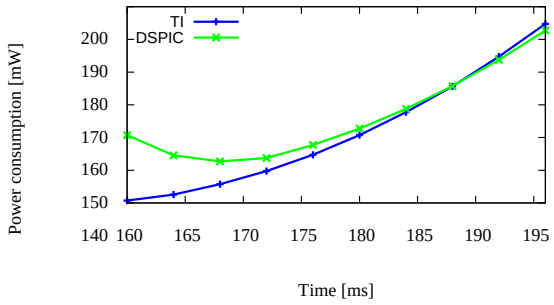


Figure 3.27: Energy consumption in one interval with the two CPUs and the same task set. The energy is obtained by varying the activation time t_a within $[t_{min}, t_{max}]$.

Figure 3.28 compares the two architectures, showing a higher energy consumption for the DSPIC. The power consumption has been averaged to the hyper-period of each task set. Note that both the CPUs have a dependency on the utilization.

We also investigated the effects of the transmission bandwidth to the energy consumption of the system. The results are reported in Figure 3.29, which illustrates the power consumption as a function of the bandwidth assignment. Note how the dependency is stronger with respect to the bandwidth amount, because the transmission cost

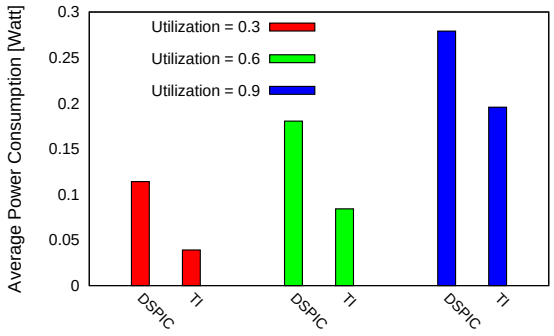


Figure 3.28: CPU comparison by varying the utilization; $n_t = 4$, $B = 0.3$ and $n_B = 3$.

increases when there is more bandwidth available; in fact, we assumed the CPU remains active while the bandwidth is available. Moreover we assumed to have messages available to be transmitted, so that the bandwidth is fully used for transmission with an increasing cost when the assigned bandwidth increases. On the other hand, the dependency with respect to the bandwidth allocation slots (how much B is split) is quite weak. This is because message deadlines were assumed to be large enough not to create a scheduling constraint.

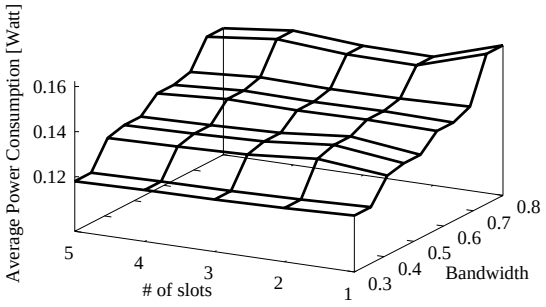


Figure 3.29: Average power consumption varying the bandwidth assignment; $U = 0.3$, $n_t = 4$.

Figure 3.30 shows how the EAS policy is affected by the task set, in terms of U and n. Notice that the power consumption is significantly affected by the utilization but not much by the number of tasks.

Figure 3.31 compares the EAS policy with respect to the *pureDVS*. The results are quite similar, since the considered CPUs are both sensitive to DVS. Nevertheless, the EAS is able to exploit the DPM capabilities and the available bandwidth to reduce the power consumption in all the task set assignments, mainly when the processor is not heavily loaded (low utilization cases).

Finally, Figure 3.32 and Figure 3.33 compare the four scheduling policies (for TI and DSPIC, respectively), under the same B, n_B, and n_t conditions, but for different task set utilizations. Notice how the EAS policy outperforms the other policies, especially for low utilizations. For high utilization, EAS and *pureDVS* exhibit the same

performance (but lower power consumption with respect to the *pureDPM* and EDF). This happens because, for high utilization there is no room for DPM improvements and only DVS is effective. Also note that, for very low utilizations, *pureDPM* provides better results than *pureDVS*. This is due to the fact that $U = 0.1$ would require a speed lower than the TI minimal speed, hence *pureDVS* forces the CPU to have a speed greater than the utilization, providing more service than required.

To conclude, the EAS algorithm is proved to be effective with respect to the other policies because it looks for the minimum energy consumption in $[t_{min}, t_{max}]$ and in any possible condition. If the minimum is found in t_{min} or t_{max}, the combined method is equivalent to the pure DVS or the pure DPM, respectively. Most of the time, however, the minimum is found inside the $[t_{min}, t_{max}]$ interval, so that the EAS is able to further reduce energy consumption with respect to the pure versions.

The paper (137) deeply exploits the concepts presented in this section.

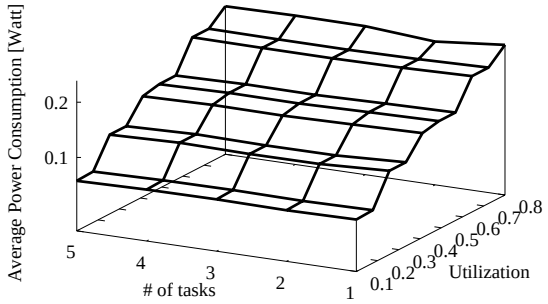


Figure 3.30: Average power consumption by varying U and n_t. EAS policy applied with $B = 0.5$ and $n_B = 3$ and the TI processor.

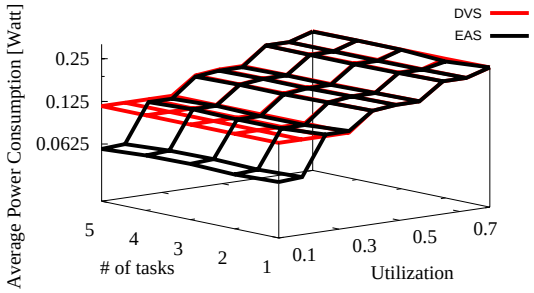


Figure 3.31: Average power consumption by varying U and n_t. The two policies applied with $B = 0.5$ and $n_B = 3$ and the TI processor.

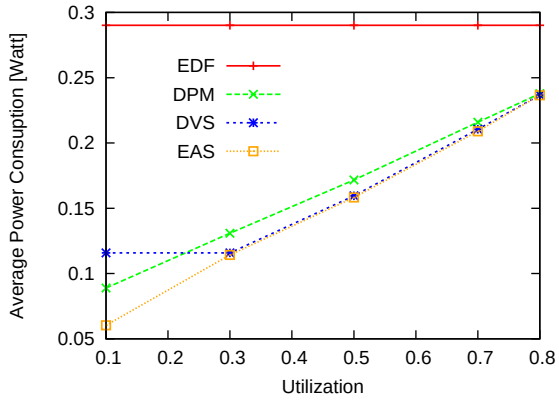


Figure 3.32: Average power consumption by varying U. All the policies are applied with $B = 0.5$, $n_B = 3$ and $n_t = 4$ and the TI processor is considered.

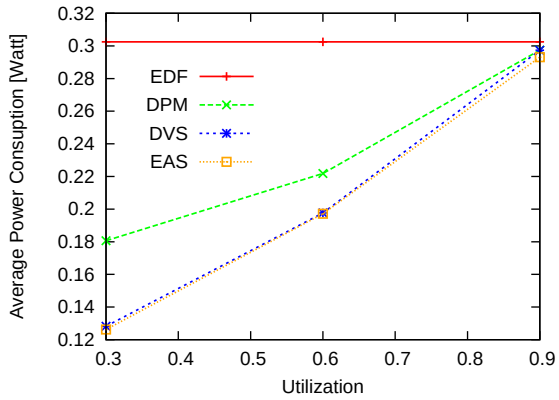


Figure 3.33: Average power consumption by varying U. All the policies are applied with $B = 0.5$, $n_B = 3$ and $n_t = 4$ and the DSPIC processor is considered.

Chapter 4

Reservation Mechanisms

A number of methods have been developed that allow scheduling and resource allocation decisions to be made for a single real-time application or set of cooperating real-time applications. Many such approaches rely on the full knowledge of the available resources. Besides, often resources are provided statically so that the application schedulability can be carried out once for all.

4.1 Survey

In this chapter, we review the major existing solutions for the bandwidth reservation for shared processing systems. The survey focuses on the analysis of the reservation techniques for single processor platforms, with relation to real-time servers, hierarchical schedulers, reclaiming techniques, etc.

4.1.1 Fixed Priority Servers

The most used algorithms for the bandwidth reservation in fixed priority systems are the deferrable server (156) and the sporadic server (147; 148). While DS has a simpler implementation, SS is able to achieve a larger schedulable utilization. Both algorithms are able to solve the poor performance of the polling server presented in (54; 140), and the processor reservation approach presented by Mercer et al. in (116), which is similar to a polling server. The Priority Exchange Server (PES) has been introduced in (97; 149). Moreover, it has no advantage over a sporadic server, but requires a more complex implementation (147; 148). Fixed priority algorithms that implement stealing mechanisms have been proposed in (95), and (132). Two complementary schemes for the reclaiming of unused bandwidth are the Capacity Sharing Server presented in (23) and the HisReWri algorithm proposed in (14). The Dual Priority mechanism has been first introduced in (56) and later extended in (22).

Polling Server The polling server is based on a periodic task with specific priority p_S, period P_S and capacity Q_S. The replenishment rule executes upon task activation with a periodic replenishment at the multiple of the period. The aperiodic requests are

served when available, till the capacity is consumed. Otherwise, the capacity is lost, which means that if a request is available at the beginning of the server period, then it is served with the server capacity Q_S, otherwise, if a request comes after that instant, it is not served until next server period.

Deferrable Server Deferrable server considers periodic task, usually with the highest priority to serve promptly aperiodic tasks, but any specific priority could be. The replenishment is made upon task activation with a periodic rate at the multiple of P_S. The consumption rule is that the aperiodic requests are served when the server still has capacity, while the capacity is lost at the end of the period.

Sporadic Server Sporadic servers assumes sporadic task with a specified priority, a period P_S and capacity Q_S in terms of computation time. The server is *active* when the executing priority is no lower than the priority of the server while it is *idle* when the executing priority is lower than the priority of the server. Initially, the server is idle and its budget is Q_S . When the server becomes active at time t_1, the replenishment time is set to $t_1 + P_S$. When the server becomes idle or Q_S becomes 0 at time t_2, the (next) replenishment amount is set to the amount of capacity consumed in time interval between the last replenishment time and t_2. Aperiodic requests are served when the server is granted for execution.

There is the corresponding dynamic version of those servers whenever they are scheduled (their capacity) with dynamic priority schedulers.

4.1.2 Dynamic Priority Servers

Apart the dynamic priority version of the former servers (servers with deadlines scheduled by dynamic priority scheduling mechanisms to assigne the resource they require) PS, DS and SS, there are two more interesting server mechanisms to be detailed.

Total Bandwidth Server and related variants Among real-time servers based on EDF scheduling, the Total Bandwidth Server (TBS) presented by Spuri et al. in (150) can be adopted when execution times are known upon job arrivals. Whenever a new job with execution requirement C_i needs to be scheduled, the server budget and deadline are set, respectively, to C_i and $max\{t, D_{old} + C_i/U_S\}$, where U_S is the server reserved bandwidth. U_S is the only parameter describing the TBS server. The Constant Utilization Server (CUS) presented in (60; 107) has similar characteristics, except for the fact that instead of immediately replenishing the budget when a new job needs to be executed, it waits until the server deadline. The Total Bandwidth (TB) (38) is an improvement over TBS that obtains optimal responsiveness by properly decreasing the server deadline. Since it is necessary to know in advance the execution requirements of the scheduled entities, neither of the above servers (TBS, CUS, TB) is suitable for open environments.

The TBS provides guaranteed bandwidth in a long run with an utilization U_S. The deadline assignment rule is the basis of such a server. The server deadline D_S is initialized as 0. When an event arrives at time t, the deadline of this event (as well as

D_S) is set to $max\{t, D_S\} + \frac{C_i}{U_S}$, where C_i is the required (worst) computation time of the event. The disadvantage is that the required (worst) computation time C_i of the event has to be known a priori.

Constant Bandwidth Server A server that does not need any information on the execution times of the scheduled entities is the Constant Bandwidth Server (CBS) presented by Abeni and Buttazzo in (2). It has a self-reclaiming mechanism that allows using in advance the capacity reserved to future jobs of the executing task. This mechanism is easily obtained by postponing the server deadline as soon as the budget is exhausted, without needing to wait until the next replenishment instant Z_k. Therefore, no suspended state is needed, and the reactivation time Z_k becomes useless. The drawback of this approach is that it is prone to the deadline-aging problem, so that neither CBS can be efficiently used to serve systems in which tasks are to be executed with more complex requirements than the simple First-Come First-Served execution.

The Soft Constant Bandwidth Server (S-CBS) provides a constant bandwidth Q_S each period P_S. The actual budget q_S is initialized as Q_S and the server deadline D_S is set 0. Each served event is assigned to the current server deadline D_S and the budget q_S is decreased by the served amount of computation. When q_S reaches 0, the new server deadline becomes $D_S + P_S$ and q_S is replenished to Q_S immediately. The server is active at time t if there are pending events; otherwise the server is idle, and the events are served in a FIFO manner. When the server is idle at time t and an event arrives, if $\frac{q_S}{D_S t} < \frac{Q_S}{P_S}$, the server becomes active; otherwise, the server deadline is set to $t + P_S$ and q_S is set to Q_S.

The Hard Constant Bandwidth Server (H-CBS) is similar to the S-CBS, but the budget replenishment is done when the old server deadline expires.

The budgeting mechanisms on both the CBS server implementations guarantees that at each server period P_S no more than Q_S computation time is provided to the application managed.

4.1.3 Resource Reclaiming

To improve the distribution of the unused bandwidth left by applications managed by the servers and that execute for less than what declared, it is possible to adopt particular bandwidth reclaiming mechanisms. The reclaiming mechanism is referred to reclaim the unused resource in rela-time systems.

In the last decade, many different techniques have been presented for the reclaiming of unused capacity in a server-based real-time system. The considerable attention that has recently been dedicated to this problem can be motivated with the typical issues that can arise while designing a reservation-based environment in an effective way. The need not to over-reserve the capacity dedicated to a particular task or application suggests one to assign, when possible, server parameters according to some average execution value, using worst-case parameters only for very critical instances. In order to ensure good system performances, soft real-time and best effort processes can then reclaim over-allocated capacity from servers that did not need it. Previously proposed works dealing with this problem presented a large number of different techniques to

distribute the spare capacity in an effective way, allowing overrun handling and fast system responsiveness. However, it is not clear how these approaches are related, and to what extent they contribute to solve the addressed problems. To better understand the mechanisms under the various reclaiming algorithms, we will first distinguish the unused (reclaimable) bandwidth into the following typologies.

- Not-admitted bandwidth: it is the share of the CPU that has not been accounted for in the admission control test. It corresponds to the capacity left when the sum of the bandwidths of the admitted servers is lower than the capacity of the computing platform.
- Inactive capacity: it is the capacity associated to servers that are not backlogged and that since their last activation executed for less than their fair share. In other words, this is the capacity that is left by admitted servers that temporarily dont have tasks or jobs in their ready queue. This is the bandwidth safely reclaimed by the GRUB algorithm.
- Cache capacity: it is the remaining budget of servers that have an earlier completion (an underrun) and that are known to activate themselves again at least after the next server deadline. This is the kind of capacity reclaimed by CASH and BASH algorithms.

There are also other kinds of capacities that can be reclaimed in a less safe way. For the moment, we will focus only on the reclaiming techniques that do not violate the temporal isolation property of the admitted servers. We hereafter detail such techniques, but first some definitions.

Definition 4.1.1 (Virtual Time)**.** *The value of virtual time V_j at any time is a measure of how much of the j-th component* reserved *service has been consumed by that time.*

Definition 4.1.2 (Work Conserving)**.** *A scheduling algorithm or server mechanisms is work-conserving if and only if it never idles processors when there exists at least one active task awaiting the execution in the system.*

Definition 4.1.3 (Deadline Aging)**.** *CBS server mechanisms can suffer the deadline aging problem when their deadline is continuously post-poned and high priority taks/server arrives to preempt them, so they cannot execute.*

- A sort of implicit reclaiming mechanism is used by any work-conserving server. Systems holding this property will never be idle when an application is waiting to be executed. Therefore, the bandwidth left unused by an application will always be used by some other application.
- A simple rule that can be easily added to the server formerly presented in to render it work-conserving is resetting to Inactive the state of all servers when the processor is idle. In this way, backlogged servers that were in suspended state will immediately switch to the Contending state, allowing some application to be scheduled.

- Another option to add the work-conserving property to the presented server is implementing the time-warping mechanism used by IRIS and BEBS. According to this mechanism, whenever the system is idle because all servers are either non-backlogged or in suspended state, the reactivation time Z_k of each suspended application A_k is decreased by $Z_{min} - t$, where Z_{min} is the first reactivation time among all the suspended servers:

$$\forall A_k \ : \ Z_k = Z_k - (Z_{min} - t).$$

  This is sufficient to avoid an idle condition when there are backlogged servers waiting to be executed. Somewhat counterintuitively, this solution shows a fairer distribution of the available bandwidth than with the previously described approach.

- For non work-conserving systems, a simple way to reclaim the not-admitted bandwidth can be provided by assigning such bandwidth to a new server that will work as a capacity tank. This server will supply additional execution to other servers that might need a further share of bandwidth to satisfy a temporary overrun.

- The inactive capacity may be reclaimed implementing a smart mechanism adopted by GRUB. The virtual time V_k of an executing application A_k is updated at a rate $slope_{active}/slope_k$, instead than at a rate $1/slope_k$, where $slope_{active}$ represents the sum of the $slope_k$ of each admitted application A_k that is either in *contending*, *non-contending* or *suspended* state, i.e. excluding all inactive applications.

- The capacity may be safely reclaimed only when there are valid reasons to believe that a non-backlogged server will not become active before a particular time-instant. The capacity that the server would have used in the considered time interval may then be safely assigned to other servers, as with CASH (40) and BASH (41) algorithms. The typical case is when a soft real-time task encapsulated into a server has an early completion. Since that task will not be activated until its next period, the unused bandwidth can be assigned to a different application, for instance to the next scheduled server. Nevertheless, the cache capacity will still be associated to the original server deadline (priority), and accordingly scheduled. Particular care must be taken when the system is idle. In this case, the capacity must be properly decremented (see (40; 41) to avoid capacity overallocation). This kind of reclaiming is not particularly suitable for hierarchical systems or, in general, for servers that handle more than one task at a time. In these cases, in fact, it is very difficult to guarantee that a server will not reactivate before a certain time-instant.

In complex systems where more than one server are applied, resource reclaiming assumes also a larger perspective. The resource can be reclaimed form subset of servers, see Section 6.1 for a detailed example about that.

Not admitted capacity reclaiming: IRIS and BEBS To solve this problem, Marzario et al. proposed IRIS (112), a server that does not suffer from the deadline-aging problem, and that can be efficiently used for the implementation of open environments. It is equivalent to our basic server, with the additional work-conserving property obtained through a time-warping mechanism: whenever the system is idle, the reactivation time of each Suspended server is uniformly decreased, so that the earlier reactivation time coincides with the current time. A server almost identical to IRIS has been presented by Brandt et al. in (13): the Best-Effort Bandwidth Server (BEBS). It is an improvement over the Rate-Based Earliest Deadline scheduler (RBED), an earlier server presented by the same group in (30). The only difference between IRIS and BEBS is in the actions associated to the time-warping operation. IRIS decreases the reactivation time of every server and decreases as well the deadline of the server(s) that first reactivates. BEBS instead does not decrease any deadline, obtaining a fairer reclaiming.

Inactive capacity reclaiming: GRUB and SHRUB Lipari and Baruah presented in (100) a CBS-based approach to reclaim the reserved capacity left free by inactive servers: the Greedy Reclamation of Unused Bandwidth (GRUB). The virtual time V_k of an executing application A_k is updated at a rate $slope_{active}/slope_k$, where a_{active} is the sum of the a_k of each admitted application A_k that is not in Inactive state. In this way, in each time interval dT, an executing server reclaims the share of bandwidth that has not been reserved to servers that have backlogged work to do, i.e. the share $(1 - U_{active})dT$. Note that this includes both the inactive and the not-admitted capacity. GRUBs reclaiming is greedy because this excess capacity is entirely given to the executing server. Fairer reclaiming strategies may be derived by distributing the inactive capacity according to some particular policy. The Shared Reclamation of Unused Bandwidth (SHRUB) presented in (18; 127) distributes the reclaimable bandwidth according to weights assigned to each application.

Cash capacity reclaiming: CASH and BASH Two algorithms that are able to reclaim cache capacities have been presented by Caccamo et al.: CASH (40) and BASH (41). When one knows that a server will not be activated until a certain time instant, the unused bandwidth is assigned to another application. This capacity is associated to the original server deadline, and accordingly scheduled. When the system is idle, the cache capacity must be properly decremented. CASH decreases the earliest deadline CASH capacity by the amount of idle time. BASH recomputes each BASH capacity as $slope(D_k - T_{idle})$, where T_{idle} is the last idle time. Simplified CASH-based servers are presented in (35).

Aggressive reclaiming: SLASH, BACKSLASH and CSS Four resource reservation algorithms have been presented in (99), each one improving over the previous one. From the simplest one to the most aggressively reclaiming one, they are: SRAND, SLAD, SLASH and BACKSLASH. While the first two servers has been designed just to show that it is better to assign the unused bandwidth to the highest priority task (as in CASH), the other two servers are more interesting. SLASH adds a self-reclaiming

mechanism that recharges the capacity postponing the deadline, as with CBS, and reclaims unused capacity from other servers using its original (unextended and, therefore, earlier) deadline. This allows than with CASH, since the executing server can use a higher priority. BACKSLASH further increases the reclaiming capabilities by retroactively allocating unused capacity to servers that used in advance their own future capacity (with self-reclaiming). The Capacity Sharing and Stealing (CSS) server presented in (125) integrates the reclaiming mechanism used by SLASH with the possibility to steal bandwidth reserved to inactive non-isolated servers.

Other dynamic servers Ghazalie and Baker introduced in (69) the Deadline Deferrable Server (DDS), Deadline Sporadic Server (DSS) and Deadline Exchange Server (DXS), adapting to the EDF case the corresponding algorithms previously defined for fixed priority systems. Spuri and Buttazzo introduced five different servers under dynamic priorities in (109): Dynamic Priority Exchange (DPE), Dynamic Sporadic Server (similar to the DSS in (69)), Earliest Deadline Last (EDL) and Improved Priority Exchange (IPE). Other bandwidth isolation algorithms with rather complex implementations are the Bandwidth Sharing Server (BSS) (104), its improved version BSS-I (101), and the Processor Sharing with Earliest Deadline First (PShED) (105). In (60; 61), Deng et al. introduced the analysis of open environment for realtime systems. Their two-level hierarchical implementation is based on TBS and CBS servers. Two attempts for providing the bounded-delay property to CBS and GRUB have been presented, respectively, in (102) (H-CBS) and (6) (HGRUB).

4.1.4 Other Server Mechanisms

Hierarchical Loadable Schedulers (HLS) (134) is a framework for the composition of existing scheduling algorithms using hierarchical scheduling, providing a guaranteed scheduling behavior to the applications. Another framework that supports a hierarchy of arbitrary schedulers, without providing compositional guarantees, is the CPU inheritance scheduling proposed by Ford and Susarla in (67). Offline strategies to deal with aperiodic workloads have been presented by Fohler et al. in (66; 82).

Overrun handling mechanisms for different task models When different task models are adopted, other mechanisms have been designed for the bandwidth reclaiming in overrun conditions. For instance, Buttazzo and Stankovic proposed in (39) the Robust Earliest Deadline (RED) scheduling algorithm for applications composed of firm real-time tasks. In (37), a related algorithm is described: Robust High Density (RHD). Koren and Shasha presented in (88) an on-line scheduling algorithm, called Dover, that has an optimal competitive factor.

Baruah and Haritsa (20) proposed an on-line scheduling algorithm (ROBUST) that maximizes processor utilization during overload conditions, given a minimum slack factor for all tasks. Thomas et al. proposed Spare CASH, an algorithm that adapts the reclaiming mechanism of CASH for a different model of firm real-time tasks (162). For adaptive tasks that may change their rate, Buttazzo et al. formulated an algorithm in which rate changes are modeled using spring coefficients (34). The Variable Rate Execution Model (VRE) in (70) is a similar (broader) model that implements and

provides schedulability conditions for systems in which task execution rates change dynamically.

4.2 Resource Guarantee

According to the real-time server theory, a server component is developed in order to guarantee a certain behavior to the application it directly manages. We are interested in modeling server components and deriving worst case and best case bounds on their properties to guarantee the behavior expected. In particular we focus our attention on the resource service interface of the server components provides.

As briefly introduced in the former sections a server can be seen as a process which provides resource to its applications. Thus, through its parameters, a server asks for a resource to an higher level resource scheduler. That resource is then passed to the applications beneath the server. The resource scheduler can schedule the resource according to either dynamic or static priority policies, which means that the server process has to have a priority assigned in order to be scheduled. The principle is exactly the same as the task scheduling, where most of the time and most of the server models assume the deadline equal to their period.

In case of dynamic priority server scheduling the specific server is not aware of the exact amount of resource it could have available because at each scheduling instance that amount depends on the other servers and their actual scheduling priority and cannot be known a priori, see Figure 4.3 for details. It is possible to do pessimistic assumptions in order to derive bounds to the resource received by the server (resource that has to be partitioned by the server according to its parameters) in case of dynamic priority server scheduling. Those assumptions consider each server as executing as the lowest priority server among all the servers at the same scheduling level. In the worst case, a dynamic priority server component S receives resource $\bar{\beta}_S = \beta \oslash \sum_{i \neq S} \beta_i(0)$[1] resource as the one left by the rest of the scheduled components. The resource are considered as the lower bound for schedulability reasons. By β_{S_i} it is represented the resource demand of server i, or better the resource amount the server is going to partition according to its parameters (Q_{S_i}, P_{S_i}). While in the best case it could receive $\dot{\beta}_S \equiv \beta$. The worst case is the only guarantee that can be given.

In case of fixed priority scheduling, because of the implicit hierarchy among the servers due to the statically defined priority, each server knows the exact amount of resource β_S is going to receive. The i-th server receives the residual resource from the $i-1$ high priority servers, so the resource left unused by those. See Figure 4.2 for some details.

A server S, as depicted in Figure 4.1, within the RTC framework can be represented by a tuple $(\beta_S, \hat{\beta}_S, \beta'_S)$, where β_S is the resource S receives as a portion of the service the main computational resource component or the higher level computational resource provider. That resource is a portion of the total resource available β. Instead, $\hat{\beta}_S$ is

[1] $\beta(\gamma) \oslash \alpha(\gamma) \stackrel{def}{=} (max\{inf_{\lambda \geq \gamma}\{\beta^u(\lambda) - \alpha^l(\lambda)\}, 0\}$ which is the subtraction in the min-plus algebra, (93; 159).

the resource that S provides to the application it manages (a portion of the received amount β_S). Finally, β'_S is the amount of β_S which has not been partitioned by S.

$$\beta'_S \stackrel{def}{=} (min\{inf_{\lambda \geq \gamma}\{\beta^u_S(\lambda) - \hat{\beta} l_S(\lambda)\}, 0\}, sup_{0 \leq \lambda \leq \gamma}\{\beta'^l_S(\lambda) - \hat{\beta}^u_S(\lambda)\})$$

β_S, $\hat{\beta}_S$ and β'_S, are server component interfaces and by them it is possible to define server guarantees. Figure 4.1 shows the server component interface, the resource partitioning and the relationship among the server and its application in details. We recall that real-time servers manages real-time applications forwarding resources to their applications allowing them to execute and then being scheduled by a real-time scheduler.

In this section we analyze server guarantee analysis by taking into account the differences among dynamic scheduling servers and static scheduling versions.

In the rest of the dissertation, where not explicitly pointed out, the deadline a server assigns to its application event streams is assumed to be equal to the server period. Furthermore, in the following we denote by $\beta(\gamma)$ the total resource amount available by the set of servers in the interval domain, γ.

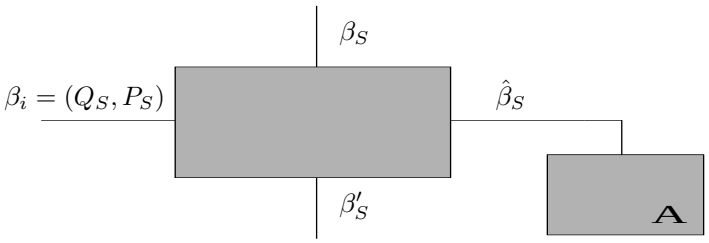


Figure 4.1: Periodic server resource model and interaction with applications A.

4.2.1 Polling servers

A *polling server* (PS) can be implemented by a periodic task $S = (P_S, Q_S)$. According to the scheduling policy applied we distinguish PS for dynamic scheduling (DPS) and PS for static scheduling (SPS). To the event streams it directly manages, a PS is a component that provides a resource supply of Q_S during every interval of length P_S, but if no requests are pending, PS suspends itself until the beginning of its next period. The capacity is not preserved if there is no workload to be served. The analysis of the PS in terms of service curve has been carried out by Wandeler et al. in (170). We improve the guarantees that have been derived by proposing tighter bounds.

DPS Under a dynamic priority scheduling assumption, it is impossible to know when the server application receives the resource within a period. A DPS server can only guarantee to supply that Q_S resource at the end of its period. Furthermore it can

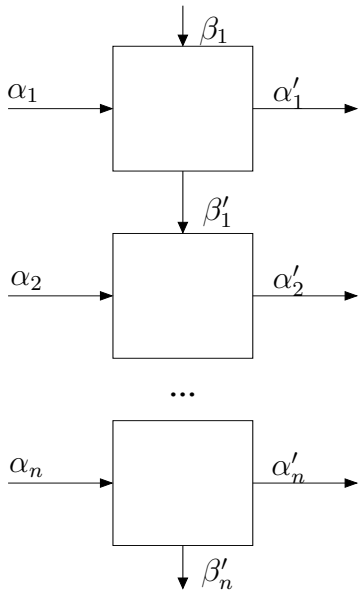


Figure 4.2: Fixed priority server scheduling.

happen to have an entire period with no resource applied at all, because at its beginning no requests were pending. A guarantee then is given by

$$\begin{aligned} \hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{DPS}(\gamma), \hat{\beta}^l_{DPS}(\gamma)\right); \\ \hat{\beta}^u_{DPS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\ \hat{\beta}^l_{DPS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l(\gamma), \lceil\tfrac{\gamma-2(P_S-Q_S)-P_S}{P_S}\rceil Q_S\right\}. \end{aligned} \tag{4.1}$$

Both the lower and upper bound are obtained by considering the total amount of resource received by the server scheduler β. This gives pessimistic bounds, but the only that can be derived. Furthermore, the lower bound tells that at most the server has to wait $2(P_S - Q_S) - P_S$ before providing its budget.

Lemma 4.2.1 (DPS Guarantees)**.** $\hat{\beta}_S = \left(\hat{\beta}^u_{DPS}(\gamma), \hat{\beta}^l_{DPS}(\gamma)\right)$ *are the server DPS guarantees.*

Proof. The best case service provisioning (which is the upper bound) happens at the beginning of each server period, whenever the resource is available, which means the minimum among β^u and $\lceil\frac{\gamma}{P_S}\rceil Q_S$ in the interval domain. The worst-case situation happens when in one instance the resource has been provided at the beginning of the server period, the next period the application was not ready (so the resource has not been applied) and at the subsequent period the resource is assigned to the server at the and of the period. This means that the server could wait $2(P_S - Q_S) + P_S$ before it can provide its resource to the applications. Furthermore, the available resource has to be taken into account, which means the minimum among β^l and $\lceil\frac{\gamma-2(P_S-Q_S)-P_S}{P_S}\rceil Q_S$ has

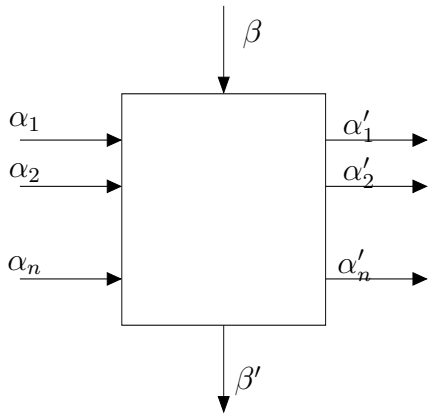


Figure 4.3: Dynamic priority server scheduling.

to be considered. It demonstrates the lemma; see Figure 4.5 for the visual explanation of the scenario. □

SPS Analyzing the hierarchy of SPS servers in a system architecture it is possible to know exactly when each server S receives the Q_S resource, β_S is the resource the server receives and where it is coded the static priority hierarchy of the server component. Still there is the possibility of having one period without any resource applied, hence the resource is guaranteed only from the second period in an interval domain analysis. It is possible to guarantee the service passed to the server application as

$$\begin{aligned} \hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{SPS}(\gamma), \hat{\beta}^l_{SPS}(\gamma)\right); \\ \hat{\beta}^u_{SPS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u_S(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\ \hat{\beta}^l_{SPS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l_S(\gamma), \lfloor\tfrac{\gamma-P_S}{P_S}\rfloor Q_S\right\}. \end{aligned} \tag{4.2}$$

Lemma 4.2.2 (SPS Guarantees)**.** $\hat{\beta}_S = \left(\hat{\beta}^u_{SPS}(\gamma), \hat{\beta}^l_{SPS}(\gamma)\right)$ *are the server SPS guarantees.*

Proof. In case of FP scheduling, the server knows the amount of resource β_S it is going to receive in any interval γ, and most of all when it is receiving such resource. The reasoning is the same of the dynamic priority case apart the fact that it can be used the available resource β_S and not larger bounds. In the worst-case, that server has to wait one period before receiving the resource to provide to its application, see Figure 4.6. This is the lower bound of the server resource provisioning and it demonstrates the lemma. □

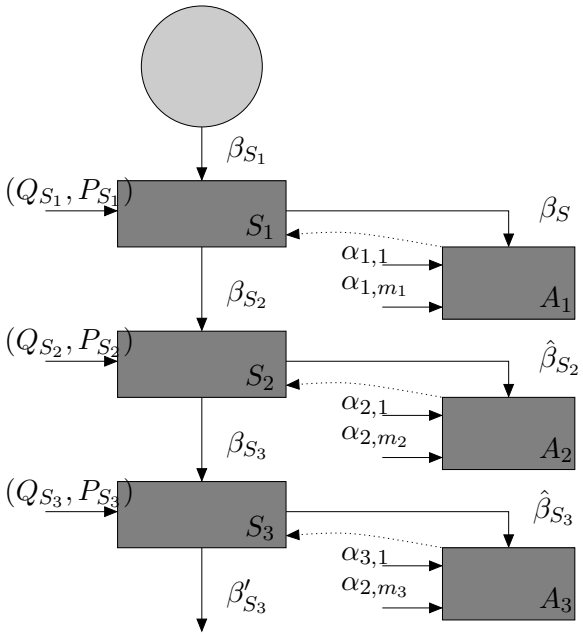


Figure 4.4: Resource scheduling among servers.

Figure 4.5: Worst-case resource supply of a DPS server (Q_S, P_S).

4.2.2 Deferrable servers

Deferrable servers (DSs) can be implemented as a periodic tasks $S = (P_S, Q_S)$, but it differs from the polling server by the way the resource Q_S is supplied along the period. The capacity is kept during the period even if not used by the application. This allows to serve more promptly arriving events. By that rule, it does not happen to have an entire period where the resource is not provided.

A deferrable server as a polling server, has two different versions according the scheduling algorithm applied, and to derive their service guarantee we can reason in the same way as already done for the PS.

With a dynamic policy it is not known where, in each period, the resource will be assigned. The *dynamic deferrable server* (DDS) can only guarantee to have Q_S resource at the end of the period, so as in case of DPS we apply β to get the resource supply rate.

Figure 4.6: Worst-case resource supply of a SPS server (Q_S, P_S).

$$\begin{aligned}
\hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{DDS}(\gamma), \hat{\beta}^l_{DDS}(\gamma)\right); \\
\hat{\beta}^u_{DDS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\
\hat{\beta}^l_{DDS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l(\gamma), \lceil\tfrac{\gamma-2(P_S-Q_S)}{P_S}\rceil Q_S\right\}.
\end{aligned} \tag{4.3}$$

Lemma 4.2.3 (DDS Guarantees)**.** $\hat{\beta}_S = \left(\hat{\beta}^u_{DDS}(\gamma), \hat{\beta}^l_{DDS}(\gamma)\right)$ *are the server DDS guarantees.*

Proof. The demonstration is based on the same one applied for Lemma 4.2.1. The difference come from the fact that, since the DS maintain the budget in case of no task are ready to be served, the waiting time in the worst-case is shorter than in case of the PS. Indeed only $2(P_S - Q_S)$ is the time interval with no resource provisioning. Figure 4.7 describes the worst-case interval where no resource is applied. □

Figure 4.7: Worst-case resource supply of a DDS server (Q_S, P_S).

While, with a static scheduling policy, the *static deferrable server* (SDS) knows when it receives the β_S resource within the period due to the scheduling priority and we derive the following service resource guarantee.

$$\begin{aligned}
\hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{SDS}(\gamma), \hat{\beta}^l_{SDS}(\gamma)\right); \\
\hat{\beta}^u_{SDS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u_S(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\
\hat{\beta}^l_{SDS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l_S(\gamma), \lfloor\tfrac{\gamma}{P_S}\rfloor Q_S\right\}.
\end{aligned} \tag{4.4}$$

Lemma 4.2.4 (SDS Guarantees)**.** $\hat{\beta}_S = \left(\hat{\beta}^u_{SDS}(\gamma), \hat{\beta}^l_{SDS}(\gamma)\right)$ *are the server SDS guarantees.*

Proof. The demonstration is based on the same demonstration applied for Lemma 4.2.2. In the worst-case a DS servers does not have to wait one period before providing its resource if the application is not available, see Figure 4.8. This is because the budget is preserved by the server. □

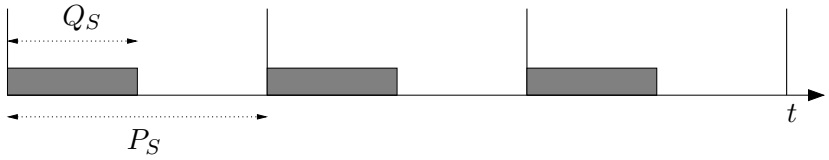


Figure 4.8: Worst-case resource supply of a SDS server (Q_S, P_S).

4.2.3 Sporadic servers

The *sporadic server* (SS) algorithm creates a periodic task to serve aperiodic or periodic stream requests, and like the DS, it preserves its capacity until a request occurs. However, SS differs from the deferrable server in the way it replenishes its capacity. Whereas DS replenishes periodically its capacity to its full value at the beginning of its period, SS does only after the resource has been consumed. Since Sprunt et al. in (96) have proven that a sporadic server behaves like a normal periodic task, the guaranteed resource supplied, in bounds, is given by the same relationship derived for the DS. We keep the distinction between the dynamic priority and static priority versions of the server, namely respectively *dynamic sporadic server* (DSS) and *static sporadic server* (SSS).

DSS

$$\begin{aligned}
\hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{DSS}(\gamma), \hat{\beta}^l_{DSS}(\gamma)\right); \\
\hat{\beta}^u_{DSS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u_S(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\
\hat{\beta}^l_{DSS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l_S(\gamma), \lceil\tfrac{\gamma-2(P_S-Q_S)}{P_S}\rceil Q_S\right\}.
\end{aligned} \tag{4.5}$$

Lemma 4.2.5 (DSS Guarantees)**.** $\hat{\beta}_S = \left(\hat{\beta}^u_{DSS}(\gamma), \hat{\beta}^l_{DSS}(\gamma)\right)$ *are the server DDS guarantees.*

Proof. The demonstration is based on the same applied for Lemma 4.2.3. □

SSS

$$\begin{aligned}
\hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{SSS}(\gamma), \hat{\beta}^l_{SSS}(\gamma)\right); \\
\hat{\beta}^u_{SSS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u_S(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\
\hat{\beta}^l_{SSS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l_S(\gamma), \lfloor\tfrac{\gamma}{P_S}\rfloor Q_S\right\}.
\end{aligned} \tag{4.6}$$

Lemma 4.2.6 (SSS Guarantees). $\hat{\beta}_S = \left(\hat{\beta}^u_{SSS}(\gamma), \hat{\beta}^l_{SSS}(\gamma)\right)$ *are the server SSS guarantees.*

Proof. The demonstration is based on the same demonstration applied for Lemma 4.2.4. □

4.2.4 Time Division Multiple Access Server

Among the time-driven servers models we investigate the TDMA one. A time-driven server is a model where the resource supply is driven by a predefined timing pattern that depends only on the server properties. In particular, a TDMA server assigns computational resource, so time, as slots that repeats each cycle of the server. A TDMA server is a pure periodic server with a main cycle and the slots always assigned in the same position within the main cycle. By S we represent the whole cycle and the TDMA server which is composed by parts S_i, each of them can manage an application. Among the properties of such server we denote the budget Q_{S_i} as the slot assigned to the application managed by S_i and P_S the main cycle of the TDMA server.

In the TDMA case there is no worst-case or best case but a strict resource assignment pattern. So assuming the slot ordered from the first assigned to the last one and the server S_i as the TDMA sub-server associated to the i-th server, the guarantees offered by the i-th slot are

$$\hat{\beta}_{S_i}(\gamma) = \hat{\beta}^u_{TDMA_i}(\gamma) \stackrel{def}{=} min\left\{\beta^u_S(\gamma), \left\lfloor \frac{\gamma}{P} \right\rfloor Q_{S_i}\right\}.$$

β_S is the resource amount which has been assigned to the whole TDMA server, while β_{TDMA_i} represents the bound to resource provided by the i-th slot of the TDMA server. The staircase function lower bounds the service a TDMA server can provide, as detailed by Figure 4.9.

Lemma 4.2.7 (TDMA Guarantees). $\hat{\beta}_{S_i} = \hat{\beta}^u_{TDMA_i}(\gamma)$ *is the server TDMA slot i guarantees.*

Proof. The demonstration is straight forward form the TDMA definition. Since the resource assignment is strict for each slot (including the slot order), the i-th slot every each P interval receives Q_{S_i} amount of resource. In the interval domain, this means that every $P - Q_{S_i}$ with no resource available, the resource starts to be available. The fully availability of the budget Q_S is after an interval of P. □

4.2.5 Total Bandwidth Server

The *total bandwidth server* (TBS) has been proposed by Spuri et al. in (109; 150) as an alternative to the "classical" server view. It allows to overcome the problem of the late scheduling of the applications managed by servers with large deadlines (we remind the usual assumption of server deadline equal to server period). A solution to such a problem is to assign a possible earlier deadline to each request. The assignment must be

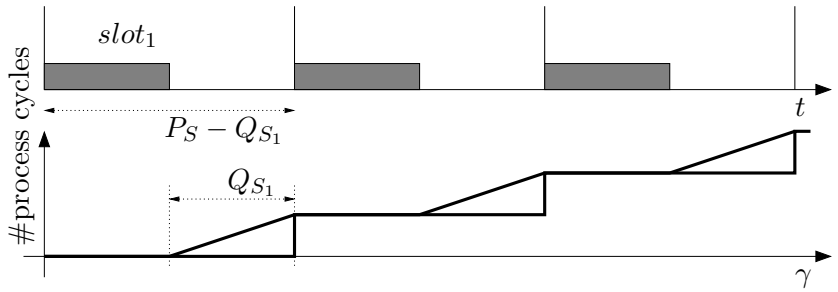


Figure 4.9: Resource supply of a TDMA server (Q_{S_i}, P_S). The staircase function lower bounds the service curve of the server.

done in such a way that the overall processor utilization of the server application load never exceeds the guaranteed maximum value U_S. A TBS server is then characterized by its bandwidth U_S and applies to dynamic scheduling policy only. Each event stream $\tau_i = (T_i, C_i)$, once processed by a TBS server, receives a deadline $d_i = max\{r_i, d_{i-1}\} + \frac{C_i}{U_S}$. The deadline is assigned so that the server demand does not exceed the given bandwidth.

In terms of service, the TBS algorithm denotes only that to each event stream i, it is provided C_i workload every $\frac{C_i}{U_S}$. But nothing can be guaranteed concerning the service curve β_{TBS} of the server. The server service provisioning depend also by the rest of the task set according to the EDF scheduling mechanism applied. The TBS server does not implements any budget mechanism and does not have any protections against overloads conditions.

4.2.6 Constant Bandwidth Server

Abeni et al. in (2) have defined the *constant bandwidth server* (CBS) as a particular periodic model that manages task for dynamic periodic scheduling policies only. A CBS processes each of its event stream received as input in order to assign them a deadline according to specific rules.

The constant bandwidth server has a periodic representation (P_S, Q_S) where the server actual budget $c_S(t)$ is recharged immediately at its maximum value Q_S after being exhausted. When this happens, the resulting output event stream from the server gets an increased deadline $d'_S = d_S + P_S$ (d_S is the former deadline of the server) by the period of the server P_S. The CBS guarantees that, if $u_S(t)$ is the fraction of processor time managed by the server time by time i.e., its actual bandwidth, its contribution to the total utilization factor will not be greater than $U_S = Q_S/P_S$ even in case of overloads.

This version of the CBS is the soft one, where soft stands for the ability of replenish the server capacity any time it is exhausted. Such a mechanism does not allow to derive significative guarantees in the service provided. Indeed, supposing an application that continuously arrives demanding resource to the server; the server quickly exhausts its budget and immediately replenishes it extending the deadline for the application execution. In the worst-case that server (and its deadline) can be preempted by high priority tasks/servers, which means the application server will be always postponed

without being executed. This is the worst-case condition which means no minimum service guaranteed. That is close to the TBS case.

The hard CBS version differ from the soft one by mean of the replenishing mechanism. Once exhausted the budget, the hard CBS has to wait until next period to replenish its budget. A hard CBS server guarantees is then

$$\begin{aligned} \hat{\beta}_S(\gamma) &= \left(\hat{\beta}^u_{CBS}(\gamma), \hat{\beta}^l_{CBS}(\gamma)\right); \\ \hat{\beta}^u_{CBS}(\gamma) &\stackrel{def}{=} min\left\{\beta^u_S(\gamma), \lceil\tfrac{\gamma}{P_S}\rceil Q_S\right\} \\ \hat{\beta}^l_{CBS}(\gamma) &\stackrel{def}{=} min\left\{\beta^l_S(\gamma), \lceil\tfrac{\gamma-2(P_S-Q_S)}{P_S}\rceil Q_S\right\}. \end{aligned} \tag{4.7}$$

In case of hard CBS, at most Q_S resource will be given at the beginning of each period P_S.

Lemma 4.2.8 (CBS Guarantees)**.** $\hat{\beta}_S = \left(\hat{\beta}^u_{CBS}(\gamma), \hat{\beta}^l_{CBS}(\gamma)\right)$ *are the server CBS guarantees.*

Proof. The demonstration is based on the same one applied for Lemma 4.2.3. □

The CBS implements a budget mechanism, contrary to the TBS one, which allows it not to overcome the promised budget in case of something not correct in the task modeling such as overrunning tasks with respect to the assumed worst case execution time.

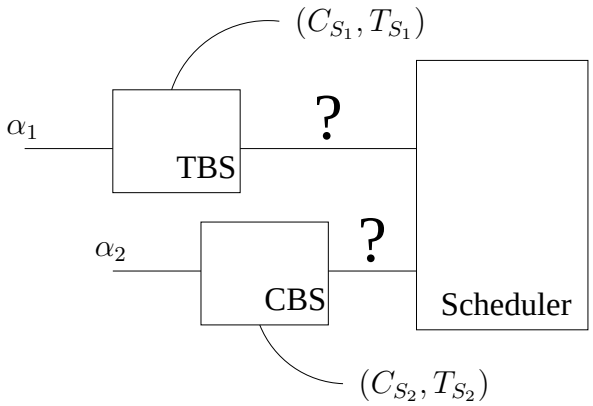


Figure 4.10: Servers view and their applications. A CBS together with a TBS to manage input applications. The application once handled by the servers are passed to the dynamic priority scheduler changed in their parameters according the server behavior (deadline and budgeting mechanisms).

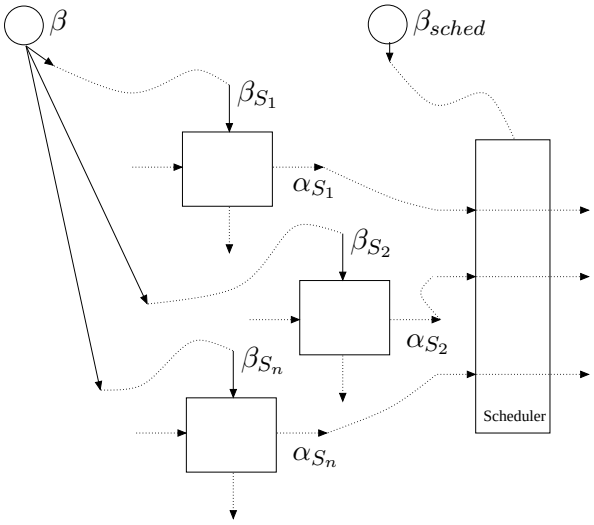


Figure 4.11: A resource reservation system architecture with n generic server models with guaranteed (continuous lines) and not guaranteed (dashed arrows) interfaces. The resource for the servers is partitioned with respect to the one for the scheduler.

4.2.7 Server guarantees

In the previous section we have defined the models for the set of servers considered. Some more comments are required to deeply understand the guarantee of those models. In the Table 4.1 we resume the bounds of the resource service curve $\hat{\beta}_S$ each server S guarantee to its application.

An accurate investigation of the server behavior denotes that they are represented, in their best cases and worst cases, as periodic resource model; but an on-line scheduling algorithm uses some rules for dynamically allocating the resource. Therefore, an on-line algorithm may produce different partitions every time it is executed, depending on the arrival times and execution times of the application tasks. This results in partitions that are not necessarily periodic. Comparing the guarantee obtained for the static and dynamic version of the servers we can notice that in case of static servers, the bounds are more tight because it i possible to know in advance the available resource, and then derive better bounds. The bounds for the dynamic servers are instead derived in the worst possible conditions, hence only large bounds are available.

Although DS and SS servers have different algorithms, it is remarkable how the DS and SS offer the same guarantees. The periodic nature of both servers results in the common worst case and best case analysis previously presented.

DS, SS and CBS servers, as well as TBS are *work conserving*, according their definition, because of at any time there is a resource request and the budget is not expired; then the request is provided from the server at last. The PS instead, it is not work conserving because only requests ready at the beginning of the server period are served, not those that arrived within the period. This results in large bounds for PS servers coming from worst-case conditions.

Server	$\hat{\beta}_S = (\hat{\beta}^u_S, \hat{\beta}^l_S)$
DPS	$\left(min\left\{\beta^u(\gamma), \lceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l(\gamma), \lceil\frac{\gamma-2(P_S-Q_S)-P_S}{P_S}\rceil Q_S\right\}\right)$
SPS	$\left(min\left\{\beta^u_S(\gamma), \rceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l_S(\gamma), \rfloor\frac{\gamma-P_S}{P_S}\lfloor Q\rfloor_S\right\}\right)$
DDS	$\left(min\left\{\beta^u(\gamma), \lceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l(\gamma), \lceil\frac{\gamma-2(P_S-Q_S)}{P_S}\rceil Q_S\right\}\right)$
SDS	$\left(min\left\{\beta^u_S(\gamma), \lceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l_S(\gamma), \lfloor\frac{\gamma}{P_S}\rfloor Q_S\right\}\right)$
DSS	$\left(min\left\{\beta^u(\gamma), \lceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l(\gamma), \lceil\frac{\gamma-2(P_S-Q_S)}{P_S}\rceil Q_S\right\}\right)$
SSS	$\left(min\left\{\beta^u_S(\gamma), \lceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l_S(\gamma), \lfloor\frac{\gamma}{P_S}\rfloor Q_S\right\}\right)$
TDMA	$\left(min\left\{\beta^u_S(\gamma), \left\lceil\frac{\gamma-(P_S-Q_{S_i})}{P_S}\right\rceil Q_S\right\}, min\left\{\beta^u_S(\gamma), \left\lceil\frac{\gamma}{P_S}\right\rceil Q_S\right\}\right)$
TBS	-
CBS	$\left(min\left\{\beta^u(\gamma), \lceil\frac{\gamma}{P_S}\rceil Q_S\right\}, min\left\{\beta^l(\gamma), \lceil\frac{\gamma-2(P_S-Q_S)}{P_S}\rceil Q_S\right\}\right)$

Table 4.1: Server interfaces: service guarantees.

The TBS server model, according its algorithm analysis, does not offer any guarantee in terms of resource applied, so the resuming table does not show bounds in the TBS row, see Table 4.1.

Table 4.2 exploits how the reclaiming mechanism can be implemented with servers. Basically, throughout the feedback between the server, which provides the resource, and the application, which demands for such a resource, it is possible to evaluate the resource unused by the server application. That resource is the portion β^r_S of the resource reserved by the server that its application α does not need. The feedback is implemented a the minimum among the resource provision and the resource demand, respectively applying the lower and the upper bounds. The unused resource can then be recovered and passed to other components of the system architecture.

The reclaimed resource is then given as the difference among the provided resource and the used one, $\beta^r = \beta \oslash \alpha^d(0)$ in case of EDF scheduling. In case of FP ones, the reclaimed resource is more complex to compute, but can be obtained as the residual resource at the end of the fixed priority scheduling hierarchy among the task composing the server application.

Reasoning with dynamic priority scheduling policies, in case of PS, the resource demand is bounded by the demands of the tasks available at the beginning of the server period. Besides, in the interval domain and with the non-deterministic analysis it is impossible to derive better bounds than the worst-case. With traces instead (not the curves but the real trace of the task arrival stream in the interval domain) it could be possible to define correctly the workload amount at the beginning of each server period. But in this way the powerfulness of the RTC analysis is lost. For the work conserving servers the application demand is given by

$$\alpha^d(\gamma) = \sum_{i=1}^{m} \alpha^d_i(\gamma)$$

as the cumulative resource demand of an application Γ. Intuitively, the resource demand for PSs is smaller than the one for the other server. Furthermore, the guaranteed bounds of PSs are larger with respect to the other server mechanisms. This allows to conclude that the reclaimed resource in case of a polling server is larger than what the rest of the server can do.

Server	$\hat{\beta}_S$
DPS	$(min\{\hat{\beta}^u_{DPS}, \check{\alpha}^{du}\}, min\{\hat{\beta}^l_{DPS}, \check{\alpha}^{dl}\})$
SPS	$(min\{\hat{\beta}^u_{SPS}, \check{\alpha}^{du}\}, min\{\hat{\beta}^l_{SPS}, \check{\alpha}^{dl}\})$
DDS	$(min\{\hat{\beta}^u_{DDS}, \alpha^{du}\}, min\{\hat{\beta}^l_{DDS}, \alpha^{dl}\})$
SDS	$(min\{\hat{\beta}^u_{DDS}, \alpha^{du}\}, min\{\hat{\beta}^l_{DDS}, \alpha^{dl}\})$
DSS	$(min\{\beta^{Au}_{DSS}, \alpha^{du}\}, min\{\beta^{Al}_{DSS}, \hat{\alpha}^{dl}\})$
SSS	$(min\{\hat{\beta}^u_{SSS}, \alpha^{du}\}, min\{\hat{\beta}^l_{SSS}, \alpha^{dl}\})$
TBS	(α^u, α^l)
CBS	$(min\{\hat{\beta}^u_{CBS}, \alpha^{du}\}, min\{\hat{\beta}^l_{CBS}, \alpha^{dl}\})$

Table 4.2: Server service guaranteed for the applications and the reclaiming mechanism. The EDF case is described.

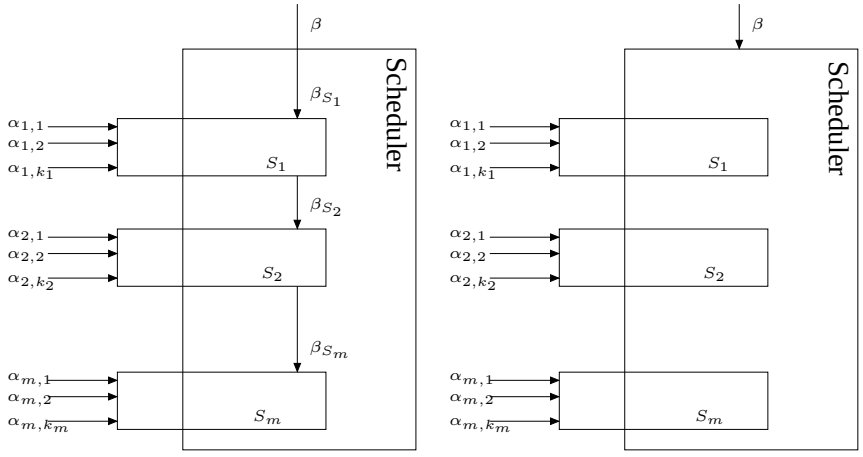


Figure 4.12: Server composition with the fixed priority and dynamic priority scheduling policy respectively.

Since β is equivalent to the supply bound function (both described in the interval domain), the guarantees obtained as service curves β are exactly equivalent to those in terms of sbf.

4.2.8 Service Guarantee Improvements

The bounds derive in Section 4.2 can be improved by reasoning under different perspectives. Those bounds are step-wise curves, driven by the floor operator applied.

Instead, applying linear curves it is possible to tighten the bounds of the server service provisioning as follows.

Considering the case of CBS, the amount of completed computation in the time interval $(t_1, t_1 + \gamma)$ is guaranteed to be no less than

$$\lfloor \frac{\gamma}{P_S} \rfloor Q_S + max\{\gamma - \lfloor \frac{\gamma}{P_S} \rfloor P_S - (P_S - Q_S), 0\},$$

otherwise, there exists some time t in time interval (t_1, t_2) such that the server is not active and $\frac{q_S}{D_S - t} \geq \frac{Q_S}{P_S}$. Than the corresponding lower bounded of the server guarantees is obtained as a periodic function shifted by $2(P_S - Q_S)$ from the origin. Such a function after $2(P_S - Q_S)$ has a linear increase of Q_S, then an holding period of $P_S - Q_S$ and the it repeats.

In case of polling server with the highest priority (SPS), the guarantees are

$$\begin{cases} 0 & \gamma \leq l \\ \lfloor \frac{\gamma - l}{P_S} \rfloor Q_S + max\{\gamma - l + \lfloor \frac{\gamma - l}{P_S} \rfloor P_S - (P_s - Q_S), 0\}. & \text{otherwise} \end{cases}$$

by defining the interval I as $l = P_S - Q_S$. If there is no available service in l then the linear periodic function starts with available service for Q_S and unavailable service for $P_S - Q_S$.

The static priority version of the deferrable server with highest priority (worst-case analysis) guarantees

$$\lfloor \frac{\gamma}{P_S} \rfloor Q_S + max\{\gamma - \lfloor \frac{\gamma}{P_S} \rfloor P_S - (P_s - Q_S), 0\}.$$

The sporadic server, in the same conditions, offers the same guarantees than the deferrable one

$$\lfloor \frac{\gamma}{P_S} \rfloor Q_S + max\{\gamma - \lfloor \frac{\gamma}{P_S} \rfloor P_S - (P_s - Q_S), 0\},$$

which is the same as the deferrable one.

To conclude, the TDMA server guarantees per slot is defined following the other servers. In the interval domain, in case of the first slot Q_{S_i} it is

$$\lfloor \frac{\gamma}{P_S} \rfloor Q_{S_i} + max\{\gamma - \lfloor \frac{\gamma}{P_S} \rfloor P_S - (P_s - Q_{S_i}), 0\},$$

The former bounding are shifted version of the TDMA curve, recalling the similarities among the service guarantee and the periodicity of them.

That linear functions improve the former guarantees by considering the linear increase of the resource provisioning, and not just stair-case function. In Figure 4.9 it has been shown the differences among the two bounding: the linear one (called TDMA curve) and the stair-case one. This way we have better bounds by which obtain better schedulability analyses.

4.2.9 Greedy Shapers

Another modeling for servers is possible with as the greedy shapers whenever applied to service resource curves. Indeed a greedy shaper with a shaping curve σ delays events of an input event stream, so that the output event stream has σ as an upper arrival curve, and it outputs all events as soon as possible (168). Generally it is considered a greedy shaper with shaping curve σ, which is sub-additive and with $\sigma(0) = 0$. Assume that the shaper buffer is empty at time 0, and that it is large enough so that there is no event loss. In (93), Le Boudec and Thiran proved that for an input event trace R to such a greedy shaper, the output event trace R is given by $R = R \otimes \sigma$. In practice, a greedy shaper with a shaping curve $\sigma(\gamma) = min_i b_i + r_i\gamma$ with $\sigma(0) = 0$ can be implemented using a cascade of leaky buckets (93). Every leaky bucket i has a bucket size b_i and a leaking rate r_i, and the leaky buckets are arranged with decreasing leaking rate within the cascade. Initially all buckets are empty. When an event arrives at a leaky bucket stage, a token is generated. If there is enough space in the bucket, the token is put into the bucket and the event is sent to the next stage immediately. Otherwise, the event is buffered until the bucket emptied enough to put the token in.

Wandeler et al. in (168) have derived the properties of an abstract greedy shaper.

Proposition 4.2.9 (Abstract Greedy Shapers)**.** *Assume an event stream that can be modeled as an abstract event stream with arrival curves (α^u, α^l) serves as input to a greedy shaper with a sub-additive shaping curve σ with $\sigma(0) = 0$. Then, the output of the greedy shaper is an event stream that can be modeled as an abstract event stream with arrival curves*

$$\alpha^u_G = \alpha^u \otimes \sigma$$
$$\alpha^l_G = \alpha^l \otimes (\sigma \overline{\oslash} \sigma)$$

Further, the maximum delay and the maximum backlog at the greedy shaper are bounded by

$$d_{max,G} = Del(\alpha^u,)$$
$$b_{max,G} = Buf(\alpha^u,)$$

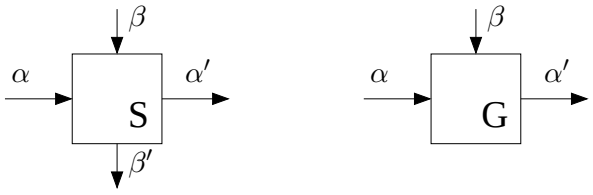


Figure 4.13: Server S and greedy shaper G implementation of resource reservation mechanisms applied to the application α.

In Figure 4.13 it is represented the duality among servers and greedy shapers. The server receives the resource and partition that for the applications, while the greedy shaper acts directly on the arrival curve shaping the them according to its function, that in this case is the rule of the server. The shaper consider the total service available as part of the shaping function.

Chapter 5

Dynamic Systems

In this chapter we first define a generic model for mode changes and then we apply such a model to analyze mode-changing systems as we details in the last part of the thesis.

Actual real-time systems are dynamic because they can change their behavior at runtime. Pedro et al. in (128) define an operating mode as the behavior of the system which is described by a series of functionalities and their temporal restrictions. Complex real-time systems can have different working modes that alternate during their execution. A mode change request is the event that triggers a mode change transition. This event can be produced by any task with the necessary knowledge of the internal and external state. A mode change request (MCR) can be initiated by the environment or by the system; it is asynchronous and can happen at any time during a mode execution. The request may occur during the system execution in a particular mode, in the ready state, or other situations. The system has to either avoid simultaneous requests or to provide the mechanism required to deal with this situation. In future developments this assumption will be released allowing to develop a more general analysis. We denote the instant as t_{MCR}.

The mode change request starts a mode change from an origin mode or old mode, *mode I*, to the destination mode or new mode which is *mode II*. In order to exclude interference of multiple mode changes, we assume that a new MCR cannot occur during a transition between modes. We will refer to the mode and the parameters at the mode in which a change is initiated with the index I. The target mode of a change, which is named mode II will be referred with II. Each mode comprises a set of tasks to execute.

The mode change become effective when the new mode can restart; t_{MCO} is the minimum instant when the new mode is operative.

Figure 5.1: Mode change: stable modes and transition stage.

Figure 5.1 depicts the mode transition stage. The transition among two steady states is intended as the condition where the system is affected by both the two steady states: the systems is changing its parameters. The transition can be considered concluded once the old mode does not affect anymore the system behavior. Which means that the new mode is effective as it has been since the first activation of the system with no interferences from other situations. Such a status could be extremely hard to verify, especially because it requires specific metrics to evaluate the end of the transient. Such metrics have not been derived so far due to the complexity of the actual real-time systems. Thus, it will be investigate in future works. For the moment, we assume the transition ending once the new mode can safely restart, so at time t_{MCO}. The delay δ is then defined as the time interval from the mode change request until the mode change become effective $\delta = t_{\mathsf{MCO}} - t_{\mathsf{MCR}}$. The task set or parameters of single tasks can change only at mode switches.

A component which changes its mode can have different behaviors from the mode change request. Broadly we can classify them into a) a transition where at the t_{MCR} the component is aborted. This means that all the activities of that component are aborted as soon as the mode change request appears. The second possible transition is b) a transition where the component continues to provide its service till the end of the transition stage. In that period the component continue to work with its old settings. The instant when the component is aborted is t_{ab}. In case a) $t_{\mathsf{ab}} = t_{\mathsf{MCR}}$ while in case b) it is $t_{\mathsf{ab}} = t_{\mathsf{MCO}}$, but in general the abortion can take place any time in between $[t_{\mathsf{MCR}}, t_{\mathsf{MCO}}]$ $t_{\mathsf{MCR}} \leq t_{\mathsf{ab}} \leq t_{\mathsf{MCO}}$. The component abortion can depend on many other situations included schedulability reasons. In this dissertation we consider only the two possible transitions as the most representative ones. The generic one (abortion any time in between) will be accurately detailed in future works, but its a special case of the ones tackled with this work.

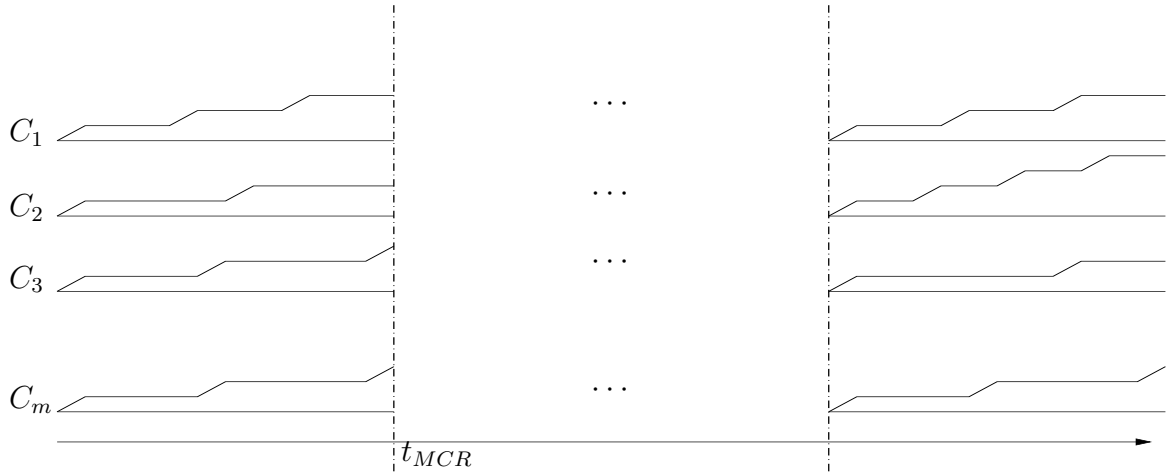


Figure 5.2: Mode change: transition stage. m components of a real-time system are represented with their curves in the steady states condition. During the mode transition the curves have to be derived.

In any mode change the following properties must be satisfied:

1. *Schedulability*: Given that each mode is schedulable in steady state conditions, all deadlines must be met during the transition; that is the case of hard real-time systems. In general we could say that a certain QoS has to be maintained.

2. *Promptness*: The mode change has to occur within a bounded interval of time, that is the new mode has to reach its steady state as early as possible.

3. *Periodicity*: All periodic tasks must be executed periodically, even in the presence of a mode change. The activation pattern of these tasks should remain unaltered during the transition.

4. *Consistency*: Shared resources must be used in a consistent way in order to avoid data corruption.

For the sake of simplicity, consistency will be neglected at this stage of the work, considering the hypothesis of independent tasks.

In the rest of the chapter we will analyze what happens during a mode change transition, Figure 5.15 illustrates our objective: deriving guarantees for resource demand and resource provisioning of the changing components. In this way we are able to *details what happens during all the stages of the system: old mode, new mode and mostly during the transition stage.*

5.1 Motivational Examples

The next motivational examples illustrates different situations in which the system is feasible within each of his modes, but during the mode transition either a reservation server cannot use all their available budget within its period or the served application cannot meet its timing constraints.

The idea of those examples is to show real cases where dynamic systems require accurate changing policies in order to keep up the determinism degree the real-time systems require.

The first example show the problems with multi-moded applications.

Example 5.1.1. *A multi-mode real-time system is composed by two servers, S_A and S_B, that are scheduled with a fixed priority policy, where S_A has higher priority than S_B. Each S_i is characterized by a budget Q_i and a period P_i every which the budget is provided; S_i is denoted as $S_i(Q_i, P_i)$. Mode I consists of $S_A(2, 4)$ and $S_B(3, 7)$, and mode II consists of $S_A(3, 7)$ and $S_B(3, 7)$.*

The servers can be seen as periodic tasks which budget has to be scheduled. The application made by servers is schedulable by Rate Monotonic in each mode. However, if a mode transition occurs at time $t = 4$ from mode I to mode II, S_B is not able to provide its budget on time due to the interference of the changing server S_A, as illustrated in Figure 5.3.

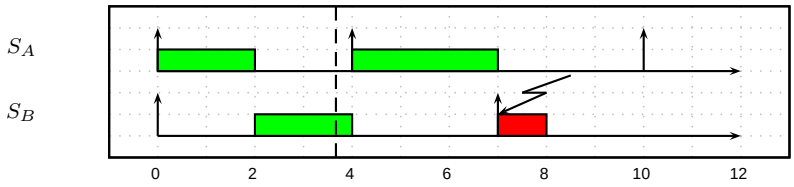


Figure 5.3: Fixed priority scheduling: unfeasible mode change at $t = 4$ from $(S_A(2,4), S_B(3,7)$, to $(S_A(3,7), S_B(3,7))$.

Example 5.1.2. *A multi-moded real-time system is composed by two servers which are scheduled by the Earliest Deadline First with relative deadline equal to the server period. In mode I* $\{S_A^I(5,10)$ *and* $S_B^I(3,6)\}$*, while in mode II* $\{S_A^{II}(4,8)$ *and* $S_B^{II}(3,6)\}$*. The first server* S_A *changes its mode at time* $t = 8$*.*

The two servers are equivalent to tasks that have to be scheduled and although the task set is feasible in both the modes (total utilization less than or equal to 1 *so that the servers can provide their budget on time), during the mode change the second server cannot provide entirely its budget during a transition period, as showed in Figure 5.4.*

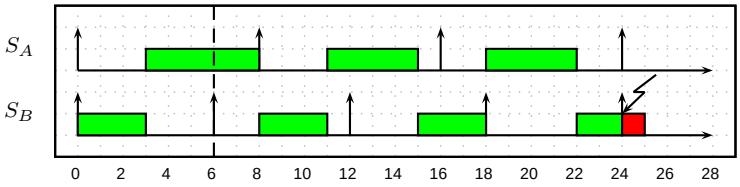


Figure 5.4: EDF scheduling: unfeasible mode change at $t = 6$ from $S_A = (5,10), S_B = (3,6)$ to $S_A = (4,8), S_B = (3,6)$.

These two examples illustrate that reconfiguring a server at runtime may cause capacity miss of other servers: the budget is not fully delivered to the applications. The required changes need an appropriate changing strategy.

Example 5.1.3. *In a partitioned architecture a set of servers manage applications. One polling servers* S_A *with the lower priority among the rest of the servers can operate in two modes. In the first mode it has a budget of* $3msec$ *and a period of* $10msec$*,* $S_A^I(3,10)$*. In the second mode* S_A *has parameters* $S_A^{II}(1,4)$*. The mode change happens at time* $t = 10msec$*. Since the server* S_A *is assumed the lowest-priority server, it is able to provide its budget just at the end of its period.*

Figures 5.5 and 5.6 show two peculiar cases where the server S_A *manages an application made by an aperiodic task* τ_A*. The aperiodic task is assumed to activate at time* $t = 5msec$ *and execute at the end of each server period for the amount of the budget available at that time instant. In the first case depicted by Figure 5.5, the application* τ_A *with computation time of* $5msec$ *and period* $15msec$ *(*$\tau_A(3,15)$*) is not feasible whenever scheduled by a mode changing* S_A *in the condition described formerly. Instead, In the second case on Figure 5.6, a slightly different application* $\tau_A(3,17)$ *is schedulable in case of mode changing server.*

This third example shows the application problems in case of changing servers: The example details how a proper design of the application to a mode changing server maintain the real-time requirements of the system.

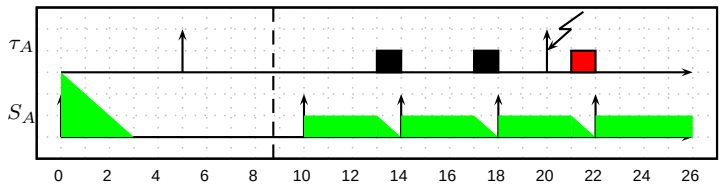


Figure 5.5: Unfeasible server and task scheduling with $\tau = (3, 15)$, $S^I = (3, 10)$ and $S^{II} = (1, 4)$; mode change at $t_{MCR} = 10$.

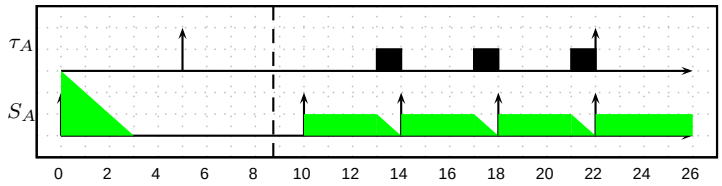


Figure 5.6: Feasible server and task scheduling with $\tau = (3, 17)$, $S^I = (3, 10)$ and $S^{II} = (1, 4)$; mode change at $t_{MCR} = 10$.

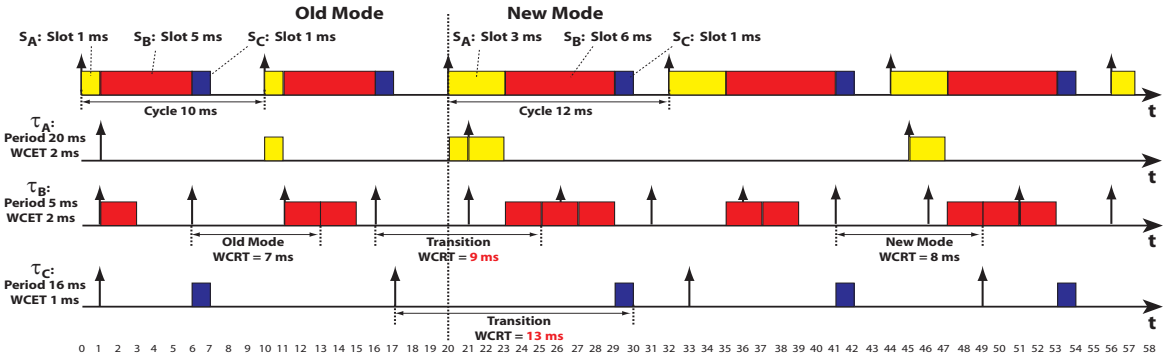


Figure 5.7: TDMA servers reconfigured at $t = 20$ms (dashed line) causes longer WCRTs for tasks τ_B and τ_C.

To illustrate the different problems that may occur during systems reconfigurations, we have chosen three more examples of systems with TDMA servers (167), static polling servers (140), and CBSs (5). Similar examples can be derived with other kinds of servers and show that a naive online change of parameters is not able to guarantee the system schedulability in hard real-time scenarios.

Example 5.1.4. *Consider Figure 5.7. Three TDMA servers, S_A, S_B, and S_C can operate in two modes, denoted as Old Mode and New Mode. We suppose that given an operating mode, all TDMA servers operate with the same period which equals the cycle of the TDMA. When there is a mode change, the allocated slots in the TDMA and the*

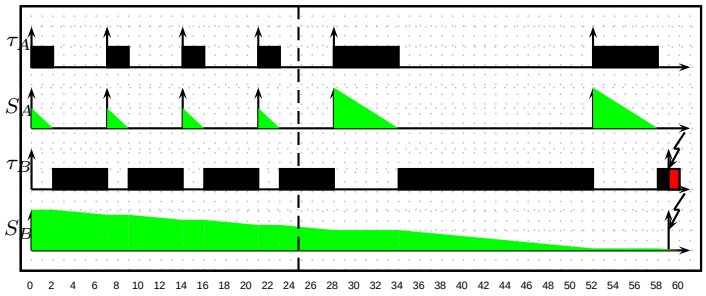


Figure 5.8: Polling server S_A reconfigured at $t = 28$ms (dashed line) causes a deadline miss for task τ_B and a capacity miss for server S_B.

cycle of the TDMA may change. When a server slot becomes available, it is available regardless of whether there is workload to use it.

Server S_A serves a single task τ_A with worst-case execution time (WCET) of $2ms$ and period of $20ms$ which we will denote as $(2, 20)$. In Old Mode, server S_A has a reserved slot of $1ms$ in a TDMA cycle of $10ms$ denoted as $(1, 10)$. In New Mode, server S_A has parameters $(3, 12)$. Server S_B serves a single task τ_B with parameters $(2, 5)$. The server in Old Mode has parameters $(5, 10)$ and in New Mode $(6, 12)$. Server S_C serves a single task τ_C with parameters $(1, 16)$. The server in Old Mode has parameters $(1, 10)$ and in New Mode $(1, 12)$.

Figure 5.7 shows a server reconfiguration performed at time $t = 20ms$. For task τ_B this means that it suffers longer worst-case response time (WCRT) of $9ms$ during the reconfiguration whereas its WCRT is $7ms$ in Old Mode and $8ms$ in New Mode. Similarly task τ_C has a longer WCRT during the reconfiguration equal to $13ms$, whereas in Old Mode it is $10ms$ and in New Mode $12ms$.

Hence, a reconfiguration of TDMA servers may cause several tasks to miss deadlines.

Example 5.1.5. *Consider Figure 5.8. Two polling servers, S_A and S_B, are scheduled with the fixed priority policy. Server S_A has higher priority. It can operate in two modes. In Old Mode it has a budget of $2ms$ and a period of $7ms$, denoted as $(2, 7)$. It serves a single task τ_A with WCET of $2ms$ and deadline equal to period of $7ms$, denoted as $(2, 7)$. In New Mode S_A and τ_A have parameters $(6, 24)$ and $(6, 24)$, respectively. Server S_B and its task τ_B operate in a single mode and their parameters are $(40, 59)$ and $(40, 59)$, respectively. The system is schedulable separately in both modes.*

Figure 5.8 shows a server reconfiguration without a proper transition algorithm. Server S_A and task τ_A simultaneously enter Mode II at time $t = 28ms$ which leads to a capacity miss for server S_B and a deadline miss for task τ_B at time $t = 59ms$ even though the mode change was performed at the end of the periods for server S_A and task τ_A.

The example illustrates that reconfiguration of a server may cause other servers to not be able to deliver their guaranteed budgets.

Example 5.1.6. *Consider Figure 5.9. CBS S_A can operate in two modes. In Old Mode it has a budget of $4ms$ with a period of $5ms$ denoted as $(4, 5)$. It serves a single task τ_A*

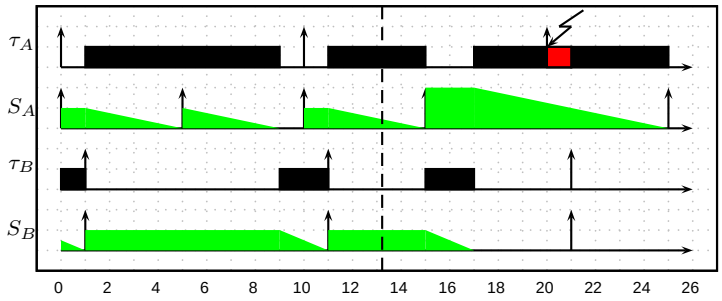


Figure 5.9: CBS S_A reconfigured at $t = 15$ms (dashed line) causes a missed deadline for task τ_A.

with WCET of $8ms$ *and deadline equal to period of* $10ms$ *denoted as* $(8, 10)$*. In New Mode, the parameters for* S_A *are* $(8, 10)$ *and* τ_A *is unchanged. CBS* S_B *serves a single task* τ_B *with parameters* $(2, 10)$ *and* $(2, 10)$*, respectively. The system is schedulable when server* S_A *is either in Old Mode or in New Mode as* $U = U_{S_A} + U_{S_B} = 1$*.*

Figure 5.9 shows a reconfiguration for server S_A *at the end of a server deadline at time* $t = 15ms$ *which leads to a missed deadline for task* τ_A *at time* $t = 20ms$*.*

The example illustrates that reconfiguration of a server may cause the application that it serves to miss deadlines.

In summary, the problems observed during online reconfiguration of servers and application fall in two classes:

1. **Isolation violation:** a reconfiguration of one server may cause other servers to not be able to deliver their guaranteed capacities.

2. **Deadline violation:** a reconfiguration of a server may affect the application that it serves by making it miss deadlines.

Safe reconfiguration algorithms will have to address both problems in order to be suitable for hard real-time systems.

5.2 Application Mode Change

To detail the scheduling in case of multi-moded applications we consider a generic application consisting of a set Γ of n periodic tasks. Each task τ_i is characterized by the model $\tau_i = (O_i, C_i, T_i, D_i)$. The parameters are not fixed, but can be modified by the application at any instant in time. Even the computation time is allowed to change for those tasks that can be executed in different versions. In particular, when $C_i = 0$ we assume that task τ_i is suspended, until some explicit activation is triggered by some other task.

Some tasks can be useless at a certain point due to no more use of a certain functionality; either new requirements can come out during the life-cycle of the system.

From a scheduling point of view, the application mode changing operations can be classified into two types:

1. Operations that increase the task set utilization:
   - Activating a new task;
   - Increasing the execution time of an active task;
   - Increasing the activation frequency of an active periodic task.
2. Operations that decrease the task set utilization:
   - Suspending an active task;
   - Decreasing the execution time of an active task;
   - Decreasing the activation frequency of an active periodic task.

Without loss of generality, we start considering the elementary case of a mode change involving two periodic tasks, one to be suspended and one to be activated. Note that more general cases involving more tasks can be seen in a compositional manner from the basic scenarios.

The critical time interval that need to be analyzed is the mode change window, where the schedulability of the whole task set has to be guaranteed.

More formally, we consider a real-time application that experiences a transition from a task set Γ^I to a task set Γ^{II}, where Γ^{II} is derived from Γ^I by deleting task τ^I and adding task τ^{II}. We assume that both task sets are schedulable in their steady state condition, under a given scheduling algorithm. The schedulability of the third condition, the transition one in between the two steady state, has to be analyzed.

Two are the extreme conditions for the task behavior in case of mode changes we are considering. Each one affecting the phase transition in different ways.

- *transitionA*: at t_{MCR} the the old mode *aborts* interrupting the its execution till the new mode is operative.
- *transitionB*: the old mode application keeps on executing and requesting service until the new mode is operative.

The two cases result in a different behavior of the application, and hence different requirements for the system. The differences will be exploited in the next sections. Mostly, for application mode changes, it is considered the case where the deleted task τ^I is immediately terminated at the time instant in which the mode change is requested. We investigate this case with the intent of finding the earliest time at which task τ^{II} can be activated without jeopardizing the schedulability of the system.

System Assumptions The operating mode is the behavior of the system, described by a series of functionalities, their temporal restrictions and a schedule, consisting of a set of tasks. This definition allows to deal with modes at different levels of abstraction. The lowest level, defined as a set of tasks and associated temporal restrictions.

In single-mode, a task τ_i behaves unchanged for the whole life of the system; this means that its parameters are never changed. In multi-mode systems, these parameters may vary between different modes, thus leading to a task being described by tuples of

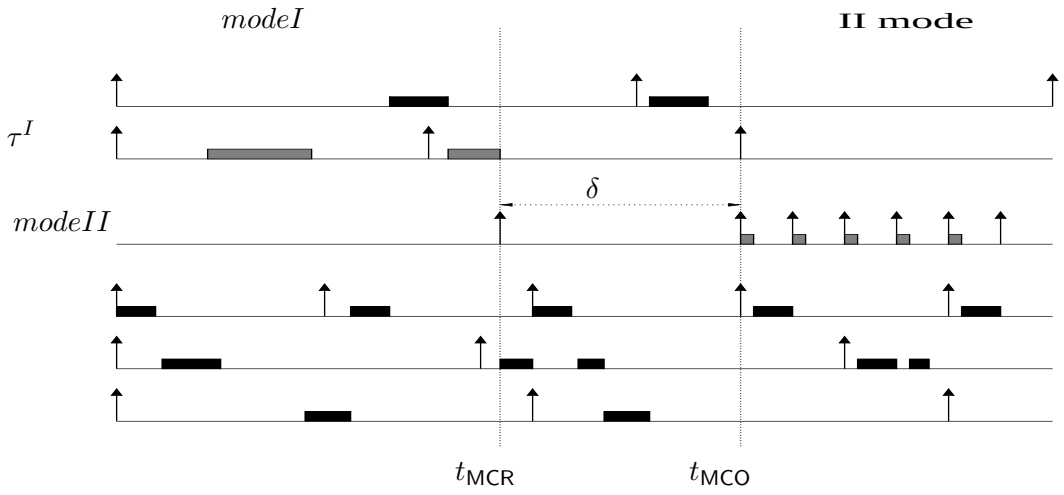


Figure 5.10: Three time intervals in a mode change; in black the tasks that remain unchanged, while in gray the ones that are going to change (the old and the new task).

(O_i, C_i, T_i, D_i), one for each mode in the system. In particular, in a system with m modes of operation, M_1 to M_m a task τ_i can be seen as a set of tuples like

$$\tau_i = \{\tau_i^{M_1}, \tau_i^{M_2}, \cdots, \tau_i^{M_m}\},$$

where

$$\begin{aligned} \tau_i^{M_1} &= (O_i^{M_1}, C_i^{M_1}, T_i^{M_1}, D_i^{M_1}, P_i^{M_1}) \\ \tau_i^{M_2} &= (O_i^{M_2}, C_i^{M_2}, T_i^{M_2}, D_i^{M_2}, P_i^{M_2}) \\ \tau_i^{M_m} &= (O_i^{M_m}, C_i^{M_m}, T_i^{M_m}, D_i^{M_m}, P_i^{M_m}). \end{aligned}$$

A task may even not exist in a particular mode; this condition can be modeled with $C_i^j = 0$. Other parameters can compose the task model. Important in our analysis framework are r_i, the arrival time of each job of task τ_i which is necessary to describe the task in case of mode changing. Moreover we require $s_{j,i}$ as the exact starting time of the j-th job instance.

The simplest and more significative case is the one where the old task set is $\Gamma^I = \{\tau_1, \tau_2, \cdots, \tau^I, \cdots, \tau_n\}$, and the new one, after the transition interval, is $\Gamma^{II} = \{\tau_1, \tau_2, \cdots, \tau^{II}, \cdots, \tau_n\}$, where τ^{II} substitutes τ^I that has to be aborted during one of its instances. Few more variables are needed by the model we are building up. The time at which the mode change is initiated t_{MCR}, the time at which the old task is aborted t_{ab} (that in general could be $t_{ab} \geq t_{MCR}$), see Figure 5.11.

5.2.1 Schedulability Analysis

We can give our definition of system stability with relation to the mode change, and that can be easily translated in system schedulability.

Definition 5.2.1. *The system is stable if the three task sets* $\Gamma^I, \Gamma_{mode-change},$ *and* Γ^{II} *in their respective activation time interval, are schedulable.*

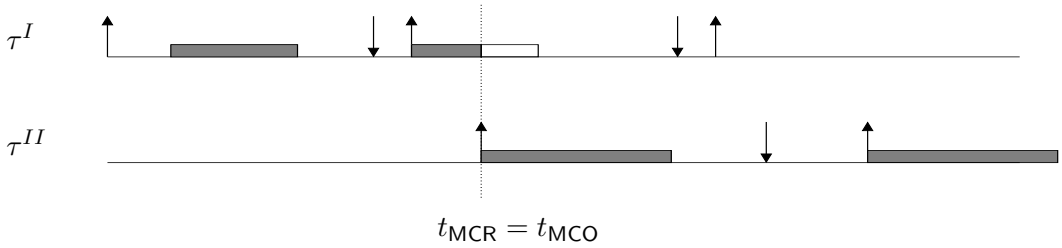


Figure 5.11: The abortion of a task and the starting time of a new task in case the new execution can start immediately after the old one. In white the unused execution time of the old task. The case depicted is a case with deadlines less than the periods.

By $\Gamma_{mode-change}$ we intend the task set during the transition phase, where one task is going to be aborted and a new one is activated at its place; the mode change task set $\Gamma_{mode-change}$ includes the two tasks that interfere during the mode transition, τ^I, τ^{II}.

Considering that Γ^I is well defined (each task has a pre-defined model, where computation time and all the composing parameters are known in advance), the only degree of freedom we have is about the new task. In particular we can modify only its offset, because its other parameters are fixed at design time.

The goal of our schedulability analysis, is to model the new task offset as a function of the transition parameters,

$$O^{II} = f(\Gamma^I, t_{MCR}, t_{ab}, C^{II}, T^{II}, D^{II}), \tag{5.1}$$

and then find the smallest possible value that keep the system feasible. For all offsets larger than O^{II} the application remain feasible. As a reminder, the offset of the new task is equivalent to the transition delay δ we are looking for as the parameter describing the transition. The main concern is to keep the system feasible but the transition has to be quick for QoS matters: the quicker it is the sooner the system will start providing the expected functionalities encoded in the new mode.

5.2.1.1 Utilization Approach

Before going into more complex results, we improve a proposal from by Buttazzo et al. in (34) where they developed a sufficient condition to compute a delay δ ($\delta \equiv O^{II}$) after which the new task can start execution leaving the system stable. The idea is that, aborting the task at t_{ab} with $e^I(t_{ab})$ computation time executed so far, Figure 5.12, it will last

$$\delta = d^I - e^I(t_{ab})/U^I, \tag{5.2}$$

before the old task releases all the utilization hired at its job arrival time. That would be the safe interval to let the new mode start. As previously said, $t_{\mathsf{ab}} + \delta$ expresses the starting time of τ^{II}, t_{MCO}, so is the offset of such new task $\delta \equiv O^{II}$. Instead, by d_i we intend the absolute deadline of the i-th task depending on the actual task instance. This solution can cope with any kind of transition since t_{ab} is generic.

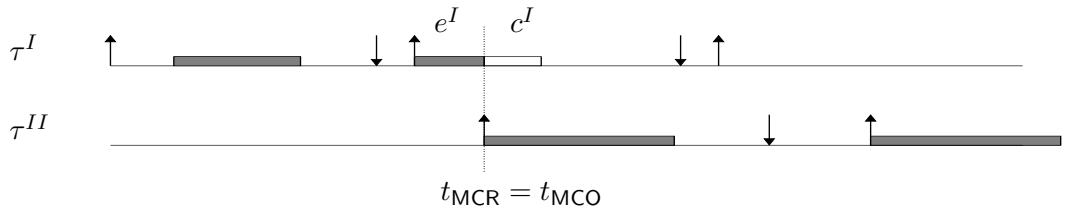


Figure 5.12: Remained and executed computation at abortion time t_{ab}

$$U^I = \frac{C^I}{T^I} = \frac{e^I}{T'^{II}}; \quad T'^{II} = \frac{e^I}{C^I}T^I = \frac{C^I - c^I(t_{ab})}{C_i}T_i = T^I - \frac{c^I(t_{ab})}{C^I}T^I = T^I - \frac{c^I(t_{ab})}{U^I}$$

Starting from the utilization that the task under execution is going to free, Buttazzo et al. in (34), postulate the possibility to enter and start a new task only after an amount of time in which the remained utilization (the one left unused or still hired but not "applied "by the specific task) is completely refilled. In other words, the new task can start only after it has freed the whole bandwidth requested to execute. An improvement to the approach briefly described, come from not only reasoning with just a couple of task (the couple (τ^I, τ^{II}) affected by the mode change), but considering the whole system with the entire utilization applied. The new task can start once it has enough band to hire and not when its substitute task τ^Ihas release enough for it. In the EDF case, the total utilization a feasible system can offer is 1 (106) whenever the deadlines are equal to the periods. It means that the new task can start since the system computational bandwidth, with the old task set at time t_{ab} U_{Γ^I}, plus the band required by τ^{II}, U^{II}, is less than or equal to 1. Is enough that the old task frees U_x given by

$$U_{\Gamma^I} - U_x + U^{II} \leq 1,$$

to have the system feasible during the mode change as well as during the modes. This means that it is not the whole U_i band, but less if there is spare utilization from the application,

$$U_x \geq U_{\Gamma^I} + U^{II} - 1.$$

When the old task has freed enough bandwidth U_x, the new task can safely start, the

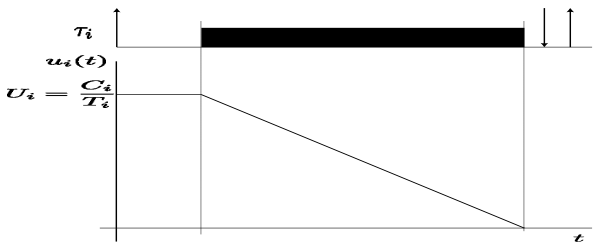


Figure 5.13: Utilization still "hired" by the task during its execution

least amount is then $U_x = U_{\Gamma^I} + U^{II} - 1$, which means

$$U_x = \begin{cases} U_{\Gamma^I} + U^{II} - 1 & \text{if } U_{\Gamma^I} + U^{II} \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

It is then possible to compute the new starting time δ' obtained when there is enough band left free considering the whole task set, and not after all the utilization of the old task τ^I has been freed. It become

$$\delta' = d^I - \frac{e^I(t_{ab})}{U_x}.$$

$\delta' = -\infty$ is possible according to the previous equation in case of $U_x = 0$. A more correct expression would be

$$\delta' = \begin{cases} d^I - \frac{e^I(t_{ab})}{U_x} & \text{if } U_x \neq 0 \vee d^I \geq \frac{e^I(t_{ab})}{U_x} \\ \delta & \text{otherwise,} \end{cases} \tag{5.4}$$

with δ the delay obtained from Equation 5.2. The new result improves the solution in (34) since $U^I \geq U_x$, is immediate to prove that $\delta \geq \delta'$. The method proposed so far offers a necessary and sufficient condition and is extremely easy to be applied on-line because has a very low computational overhead. Its main drawback is that it works well only in case of deadlines equal to the period. Otherwise, the necessary and sufficient utilization condition we can build for an EDF scheduling would be too much pessimistic. That can also be applied as sufficient condition with fixed priority scheduling by considering the hyperbolic bound, (26; 27).

5.2.1.2 Fixed-Priority Scheduling Scheme

The former utilization based approach has many advantages, but with more general conditions (deadlines different than the period) the results offered are not accurate anymore. The scheduling theory suggests other solutions to be applied to this class of problems and in order to find a better starting time for the new task. That will lead to a more prompt system, hence, able to reach the next steady state after the mode transition, in a reduced time interval. The rest of the methods proposed require to consider a specific scheduling algorithm. In this section we refer to any of the fixed-priority scheduling paradigms. The classic literature knows well that the schedulability test for FP scheduling policy applies the response time analysis. However its is well known the complexity of such a test based on RTA; the pseudo-polinomial complexity of the RTA does not allow to apply it on-line, as we are claiming to do in any transitions. Recently are came out works that reduces the complexity of the RTA deriving bounds: the objective is to apply them only as acceptance test. Since our target is to find the best new task starting point, it could be seen as a sort of acceptance test (delay the task till the system can afford to let it execute). We can apply the same techniques already discussed in the introduction with significative modifications. Indeed, in (55) the response time bound has been applied as a sufficient schedulability test that we are going to review.

Davis et al. in (55) and Bril et al. in (32) have derived the schedulability test in case of jitter by considering the upper bound of the response time R^{ub},

$$\forall i \quad R_i^{ub} \leq D_i - J_i \;\; R_i^{ub} = \frac{B_i + C_i - F_i + \sum_{\forall j \in hp(i)} [U_j J_j + C_j(1-U_j)]}{1 - \sum_{\forall j \in hp(i)} U_i} + F_i$$

This exploits the worst-case analysis of the response time which gives just a sufficient condition. So far, the offset of task has not been considered. We can add it integrating the bound with the task offset obtaining:

$$R_i^{ub} = \frac{B_i + C_i - F_i + \sum_{\forall j \in hp(i)} [U_j(J_j - O_i) + C_j(1-U_j)]}{1 - \sum_{\forall j \in hp(i)} U_i} + F_i,$$

as a rather incremental version of the one derived by Davis et al. in (55). According to our premises, only one task, the new one, has an offset, hence we can divide the analysis into parts, as follows. We consider all the tasks with an higher priority than the new task, all those tasks that can interfere with τ^{II}.

$$\forall i \in hp(II) \; R_i^{ub} = \frac{B_i + C_i - F_i + \sum_{\forall j \in hp(i) \wedge \neq new} [U_j J_j + C_j(1-U_j)]}{1 - \sum_{\forall j \in hp(i)} U_i} + \frac{U^{II}(J^{II} - O^{II})}{1 - \sum_{\forall j \in hp(i)} U_i} + F_i,$$

which is equivalent to

$$\forall i \in hp(II) \; R_i^{ub} = R_i' + \frac{U^{II} J^{II}}{1 - \sum_{\forall j \in hp(i)} U_i} - \frac{U^{II} O^{II}}{1 - \sum_{\forall j \in hp(i)} U_i},$$

with $R_i' = \frac{B_i + C_i - F_i + \sum_{\forall j \in hp(i) \wedge \neq new} [U_j J_j + C_j(1-U_j)]}{1 - \sum_{\forall j \in hp(i)} U_i} + F_i$. Furthermore

$$\forall i \in hp(II) \; R_i^{UB} = R_i'' - \frac{U^{II} O^{II}}{1 - \sum_{\forall j \in hp(i)} U_i},$$

has been obtained through $R_i'' = R_i' + + \frac{U^{II} J^{II}}{1 - \sum_{\forall j \in hp(i)} U_i}$. Applying the feasibility criteria, is then easy to derive a set of constraints to what we are looking for. Indeed, imposing $R_i \leq D_i - J_i$ as schedulability condition for all the tasks of our task set, including the old task and the new task, we obtain

$$\forall i \; R_i \geq (R_i'' - D_i + J_i) \frac{1 - \sum_{\forall j \in hp(i)} U_i}{U^{II}}.$$

The value of O^{II} that satisfies all the previous conditions is carried out as the minimum among all the ones obtained with high priority tasks than τ^{II}.

$$O^{II} = min\{\forall i | O^{II}, R^{II} + J^{II} - D^{II}\}_0,$$

and $\delta \equiv O^{II} \geq R^{II} + J^{II} - D^{II}$ derives from the schedulability condition of the new task τ^{II}. The $_0$ operator denotes the non negative function $f_+(x) = max\{0, f(x)\}$.

In case of fixed-priority scheduling policies even the concept of workload can be applied to verify the schedulability of task sets. For a task τ_i its level-i workload $w_i(t)$ is the total amount of time the processor is busy to serve τ_i and its high priority tasks in any interval length of t. By extension, $w_0(t) = 0$ for all t. It can be computed as

$$w_i(t) = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j, \tag{5.5}$$

where $hp(i)$ is the set of high priority tasks with respect to τ_i. The same reasoning for RTA can be applied to the workload case and obtain the offset or delay for the new mode.

Task Modeling In terms of workload demand we can find a bounding function able to model the new macro task. The idea is to model the couple of tasks affected by the mode change (τ^I, τ^{II}) with only one but having a different set of parameters to describe its behavior. In this way to afford the *single task mode-change* problem implies, for FP scheduling algorithms, to consider the couple of tasks adjacent in term of priority. No other tasks than τ^I and τ^{II} can fall in the priority interval given by (τ^I, τ^{II}). That reduce the generality of application of such method, but it still represents a good starting point to develop a general approach. Considering the upper bound of the *level-i* workload as a linear curve, which is the combination of linear upper bounds of each task, we have

$$w_i^{ub}(t) = \sum_{\forall j \in hp(i)} U_j t + O_j \equiv$$
$$\sum \forall j \in hp(i) U_j t + \sum_{\forall j \in hp(i)} O_j,$$

where $U_j = C_j/T_j$ is the slope of each composing linear curve, while $offset_j$ is its offset. $O = \sum_{\forall j \in hp(i)} O_j$. It still remains to check if exist a t that satisfies the scheduling condition. The solution

$$L_i = min_{0 \leq t \leq T_i} w_i^{ub}(t)/t$$
$$L = max_{1 \leq i \leq n} L_i,$$

in accordance to (31; 32; 55), is given as a pseudo-polinomial problem with two dimensions (n, t).

As a sufficient condition, the FP scheduling algorithm schedules τ_i if $L_i \leq 1$. The whole task set Γ is schedulable if $L \leq 1$. Even if is possible to apply the reduced set by Bini et al (25) to have the number of checks reduced, but still the complexity order remains.

5.2.1.3 Dynamic Scheduling Scheme

In case of dynamic priority scheduling policies we can apply the processor demand criteria to verify the schedulability of multi-moded applications. Again, we have to figure out which is the best starting time for the new task. That leads to a system more prompt system, hence able to reach the next steady state after the mode transition, in a smaller amount of time.

The hypothesis we are considering imply tasks no more synchronous, due to the new task activation time. Hence, busy period has to be found between any interval (t_1, t_2), with $t_1 \leq t_2$, along the system evolution. The classical processor demand formula (21) comes from two main parts. The number of instance of task τ_k with activation time bigger or equal than t_1 and deadline less or equal than t_2 is given by

$$\eta_k(t_1, t_2)_0 = (\lfloor \frac{t_2 - O_k - D_k + T_k}{T_k} \rfloor - \lceil \frac{t_1 - O_k}{T_k} \rceil)_0,$$

The processor demand criteria, according to (21), is

$$\forall t_1, t_2 \quad g(t_1, t_2) = \textstyle\sum_{i=1}^{n} \eta(t_1, t_2) \leq (t_2 - t_1).$$

The computational demand of a task set can be precisely described by the demand bound function, introduced by Baruah et al. (21) to express the total computation that must be executed by the processor in each interval of time when tasks are scheduled by EDF. For any given periodic task τ_i activated at time $t = 0$, its demand bound function $\mathsf{dbf}_{\tau_i}(t)$ in any interval $[0, t]$ is given by

$$\mathsf{dbf}_{\tau_i}(t) = \max \left\{ 0, \left(\left\lfloor \frac{t - D_i}{T_i} + 1 \right\rfloor C_i \right) \right\}.$$

Hence, the computational demand of a task set Γ of periodic tasks synchronously activated at time $t = 0$, can be computed as the sum the individual demand bound functions of each task, that is:

$$\mathsf{dbf}_{\Gamma}(t) = \sum_{\tau_i \in \Gamma} \mathsf{dbf}_{\tau_i}(t).$$

By definition, the demand bound function provides an upper bound of the resource requested by the task set in each interval of time coming from $\tau_1, \tau_2, \ldots \tau_n$ and mostly τ^I and τ^{II} both contributing to the resource demand during the transition. While $\tau_1, \ldots \tau_n$ and τ^I are synchronous, τ^{II} has an offset O^{II} which is also the delay of the transition of the mode change request $O^{II} = t_{ab} + \delta$. The whole set of contributions to the processor demand, during a mode change, is listed below. Considering the task model given, we have two caes, two different resource demand and two different schedulability guarantees.

$$\begin{cases} [d^{last}, \infty], & g_{\Gamma \setminus \{\tau^I\}}(t_1, t_2) + \lfloor \frac{t_2 - t_1 - D^I + T^I}{T^I} \rfloor C^I - c^I(t_{ab}) + \lfloor \frac{t_2 - D^{II} + T^{II}}{T^{II}} \rfloor \\ & - \lceil \frac{t_1 - O^{II}}{T^{II}} \rceil C^{II} \leq t_2 - t_1 \\ (0, d^{last}), & g_{\Gamma \setminus \{\tau^I\}}(t_1, t_2) + \lfloor \frac{t_2 - t_1 - D^I + T^I}{T^I} \rfloor C^I + \lfloor \frac{t_2 - D^{II} + T^{II}}{T^{II}} \rfloor - \lceil \frac{t_1 - O^{II}}{T^{II}} \rceil C^{II} \leq t_2 - t_1, \end{cases} \tag{5.6}$$

where the contribution depends on where the measuring interval is considered as we have detailed in the formulas. d^{last} is the last deadline of the old task τ^I after its abortion. The first part of demand $g_{\Gamma\setminus\{\tau^I\}}(t_1,t_2)$ is the contribution of all the tasks composing the old task set without τ^I. τ^I deserves a particular attention according the specific interval the processor demand is going to be computed; indeed it is going to be aborted to its resource demand it is affected by t_{ab}. The quantity $\lfloor\frac{t_2-t_1-D^I+T^I}{T^I}\rfloor C_i \equiv g^*_{\tau^I}(t_1,t_2)$ is the resource amount of task τ^I that after the abortion time does not offer contribution to the processor demand, so the actual resource demand is given as:

$$g^*_{\tau^I}(t_1,t_2) = \begin{cases} \lfloor\frac{t_2-t_1-D^I+T^I}{T^I}\rfloor C^I & \text{if } t_1 \le t_2 \le t_{ab}; \\ \lfloor\frac{t_{ab}-t_1-D^I+T^I}{T^I}\rfloor C^I & \text{if } t_1 \le t_{ab}, t_2 > t_{ab}; \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

This contribution has to be subtracted with the remained computation time of τ^I at its abortion time, demand unused by the task itself. The quantity of processor resource demanded by a generic task τ_k can be approximated, bounding the number of jobs that the task execution fits in the time window $[t_1,t_2]$. Increasing such a number, the processor demand increases, leading to an upper bound of the processor demand. A only sufficient condition, that can be used to guarantee the feasibility of a task set, can be obtained from the previously described approximation, and now described in detail. If we limit the offset of a task to its period, considering indeed, tasks where $O_k \le T_k$ we can have the following

$$\eta_k(t_1,t_2)_0 \le (\lfloor\frac{t_2-O_k-D_k}{T_k}\rfloor - \lceil\frac{t_k-T_k}{T_k}\rceil)_0 \le ((\frac{t_2-O_k-D_k}{T_k}+1) - \frac{t_k-T_k}{T_k})_0$$

by removing even the floor and ceiling functions. We might find many bounds to transform the open formula in a closed one, and to extract the values we need to set up a significative analysis. A good bound is defined according the following assumptions:

$$\lceil\frac{t_1-O^{II}}{T^{II}}\rceil \ge \lceil\frac{t_1-T^{II}}{T^{II}}\rceil;$$

in case of $O^{II} \le T^{II}$. Otherwise, with $T^{II} \le O^{II} \le 2T^{II}$ the bound is given by $\lceil\frac{t_1-2T^{II}}{T^{II}}\rceil$ and continuing with the trend, is always possible to define a lower bound for the number of jobs with arrival time less than or equal to t_1. The expression results in

$$O^{II} \ge \frac{g_{\Gamma\setminus\{\tau_i\}} + g_{\tau^I} - (t_2-t_1) - D^{II}U^{II} - \lceil\frac{t_1-T^{II}}{T^{II}}\rceil C^{II} + C^{II} + t_2U^{II}}{U^{II}}, \tag{5.8}$$

removing the ceiling and floor functions through out the specific bounds in case of $[0,d^{last}]$. Instead, in case of $[d^{last},\infty)$ it is

$$O^{II} \ge \frac{g_{\Gamma\setminus\{\tau_i\}} + g_{\tau^I} - (t_2-t_1) - D^{II}U^{II} - \lceil\frac{t_1-T^{II}}{T^{II}}\rceil C^{II} + C^{II} + t_2U^{II} - c^I(t_{ab})}{U^{II}}, \tag{5.9}$$

The main idea is to find the better bound in order to have a close result to the exact value.

A new task set Γ^{II} differs from the old one Γ^{I} by only one task, the old version that expires before the new one starts executing. During the mode transition the task set can be represented with $\Gamma_{mode-change}$, and includes the old task and the new one. We recall that the new task is the only one having an offset O^{II} different than 0. The other tasks are synchronous.

Theorem 5.2.2. *Given a task set of n tasks Γ^{I} feasible, a mode change requests at time $t_{MCR} \equiv t_{ab}$, and the new feasible task set Γ^{II} with $n-1$ simultaneous old tasks and a new one not synchronous (with an offset O^{II}), works safely (without missing any deadline even while the mode change happens), even during the transition, if the mode changes as an offset*

$$O^{II} = \begin{cases} max_{t_1,t_2} \frac{g_{\Gamma-\{\tau I\}}+g_{\tau I}-(t_2-t_1)-D^{II}\cdot U^{II}-\lceil\frac{t_1-2T^{II}}{T^{II}}\rceil C^{II}+C^{II}+t_2\cdot U^{II}}{U^{II}} \\ if\ t_1 \le t_2 < d^{last} \\ IS\ FEASIBLE \\ max_{t_1,t_2} \frac{g_{\Gamma-\{\tau I\}}+g_{\tau I}-(t_2-t_1)-D^{II}\cdot U^{II}-\lceil\frac{t_1-2T^{II}}{T^{II}}\rceil\cdot C^{II}+C^{II}+t_2\cdot U^{II}-c^{I}(t_{ab})}{U^{II}} \\ if\ t_1 \le t_2 \wedge t_2 \ge d^{last} \wedge t_2 \le D^{*} \\ IS\ FEASIBLE \end{cases} \tag{5.10}$$

Proof. By definition of processor demand and the theorem regulating the guarantee test, (21), the demonstration immediately comes. Considering the offset O^{II} as indefinite variable, the results derives from the exploitation of the unknown variable. For the moment we consider

$$D^{*} = max\{d_k | d_k \le t_{ab} + min(L^{*}, H)\}$$

where H is the task set $\Gamma_{mode-change}$ hyper-period and

$$L^{*} = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U},$$

like especially specified in (21). In order to verify the task set schedulability during the transition, the resource demand in such a phase, $g_{\Gamma_{mode-change}}$, has to be less than or equal to the computational resource available t

$$\forall t\ g_{\Gamma_{mode-change}} \le t.$$

From Equation 5.8 and Equation 5.9 the two offset conditions are obtained as a function of t_1 and t_2. The schedulability has to be guaranteed for any t_1 and t_2, hence the maximum of the offsets that demonstrates the theorem. □

O^{II} as in Equation 5.10 makes the mode change safe. The value of k and the number of values to be verified in Equation 5.10 depends on how big is the new task offset with respect the new task period. The resulting formula is very easy to be solved; it is sufficient to do few tries to tune the right offset and the right bound. That can be even done in parallel. It is also possible to carry out the error made

using such a bound, respect to the possible optimal value (the smaller instant possible in which the new task can restart). The error in the new task offset is due to the approximation applied to obtain Equation 5.10 in a closed form. The error is at max $2T^{II}/U^{II}$; indeed, removing the floor, that rounds to the lower integer, we can have at max $2T^{II}/U^{II}$ more to be considered into the processor demand. Furthermore, the second bound adds uncertainty, less than or equal to $2T^{II}/U^{II}$ to be considered for an exact analysis.

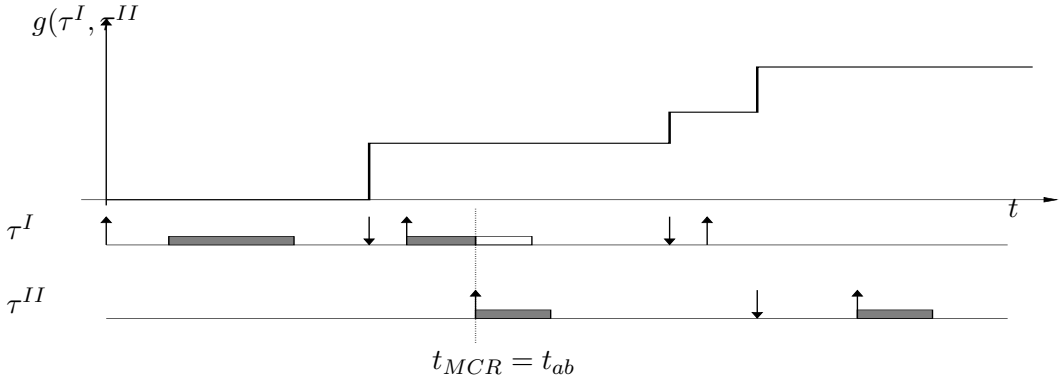


Figure 5.14: Processor demand: an example of the processor demand of two tasks where first one is aborted after one instance, and the second one taking place of the first.

Since the former theorem gives us two possible results according the case that could be considered, we have to define a unique result.

The procedure we have built-up comes from considering the two cases as

- $t_1, t_2 < d^{last}$ CaseA,
- $t_1, t_2 \geq d^{last}$ CaseB

and obtain

$$O^{II} = min\{O_A^{II} \text{ if feasible}, O_B^{II} \text{ if feasible}\}$$

including even the case where both of the two are unfeasible. In that scenario the mode change is unfeasible.

The error we commit can be the starting point for a refinement sequence step toward the optimal value; reducing the offset one unit at the time till the task set (in the mode change progress, the whole task set the system schedules) remains feasible.

In some applications is possible to leave the result as it comes out from Equation 5.10. There are systems than can effort such a time interval where the new task does not start, and complete its functionality only after such a delay. Such systems privilege the possibility to compute on-line the offset, even a non precise offset. On the other hand, there are systems that needs the the optimal value to be as much prompt as possible and let the new task start as soon as possible. Since the optimal value requires the refinement step, hence more computation time, is more likely usable off-line, without degrading much the system real-time performance. With a certified uncertainty, the offset can range in between, it could happen that it assumes values bigger than its period; a bound tuning could be required with the formula seen right before.

Optimality The possibility to reduce the time interval where to check schedulability allows to speed up the computation of the new task offset. Since the value obtained above is not optimal because derived through bounds we can improve such a result toward the optimality. First of all we have to define what we consider by an optimal result.

Definition 5.2.3. *The optimal offset of the new task is the smallest offset between the set of feasible offsets.*

$$O^{*II} \stackrel{def}{=} min\{O_k | O_k \in \Psi_{\Gamma_{mode-change}}\}$$

While by *feasible* offsets we mean the offsets that leave the task set feasible during the transition phase. With $\Psi_{\Gamma_{mode-change}}$ we intend region described by all the new task offsets that guarantee the feasibility of mode changing application during the mode transition. The value computed with the previous formula can be a good enough as a starting point for an algorithm exploring the solution space seeking for an optimal result. Starting from that, one unit of time at the time and testing the schedulability of the new task set with the offset analysis, we can reach the optimal offset via an exhaustive search, as outlined by the Algorithm 7.

Algorithm 7 Optimal algorithm

Input: offsetBound
Output: optimal
1: $offsetOptimal = offsetBound - 1$;
2: **while** $\Gamma^{II}(offsetOptimal)$ is feasible **do**
3: check feasibility of $\Gamma^{II}(offsetOptimal)$;
4: offsetOptimal–;
5: **end whilereturn** $offsetOptimal - 1$;

5.2.1.4 Real-Time Calculus and Application Mode Change

With real-time calculus it is possible to carry out the analysis of mode transition for multi-moded applications. Perathoner et al. (153) have developed an upper bound to the resource demand of an application during the transition stage. Even in that case it has been considered the old task aborted at time t_{MCR}, while the new one starts at t_{MCO}. The case in which the old task continues to execute after t_{MCR} has not been taken into account. They have proposed solutions both in case of fixed and dynamic priority scheduling by considering the transition arrival curve α^T as the curve that upper bounds the task workload. In particular it has been studied the mode changing task and its workload.

Theorem 5.2.4 (Transition Demand Bound Function)**.** *Given an application and its demand bound functions at the old mode* dbf^I*, and at the new mode* dbf^{II}*, its transition demand bound function* dbf^T *during a mode change with a delay* δ *is*

$$\mathsf{dbf}^T(t) = sup_{0 \leq \lambda \leq t}\{\mathsf{dbf}^I(t-\lambda) + \mathsf{dbf}^{II}(\lambda - \delta)\}. \tag{5.11}$$

Proof. The proof of the transition demand bound function comes straight from (129), where the resource demand is upper bounded by dbf^T in the interval domain analysis. The demand bound function is exactly the demand curve α^d. □

The resulting dbf^T at the transition stage obtained with Equation 5.11 is safe and offers a necessary and sufficient condition to study the mode transitions. The upper bounding linear approximation of the transition resource demand is given by $\Delta^T = max\{\Delta^I, \Delta^{II}\} + \delta$ and $slope^T = min\{slope^I, slope^{II}\}$. The scheduling conditions that applies the transition bounded-delay approximations of the transition demand bound functions is just sufficient.

The transition demand bound function has been derived in order to have all the possibility covered. Once known the demand bound function in all the possible conditions of the application (steady states and mode transitions) it is possible to set up a complete schedulability analysis for multi-moded applications in order to derive δ with the RTC framework too. As a reminder δ is equivalent to the offset of the new task O^{II}, and with the real-time calculus such a delay represents a shift left for the arrival curve, in particular for the arrival curve of the new task.

5.3 Server Mode Change

In the previous section it has been studied the case with mode changing application. Consistent results have been proposed with a minimum δ that has to be waited for a feasible transition. In this section is the server component to be analyzed. We analyze multi-moded servers which could change their parameters because of changing applications or due to external reasons. We analyze the capacity of changing operational mode at run-time affecting the resource reservation of the servers and consequently the application schedulability.

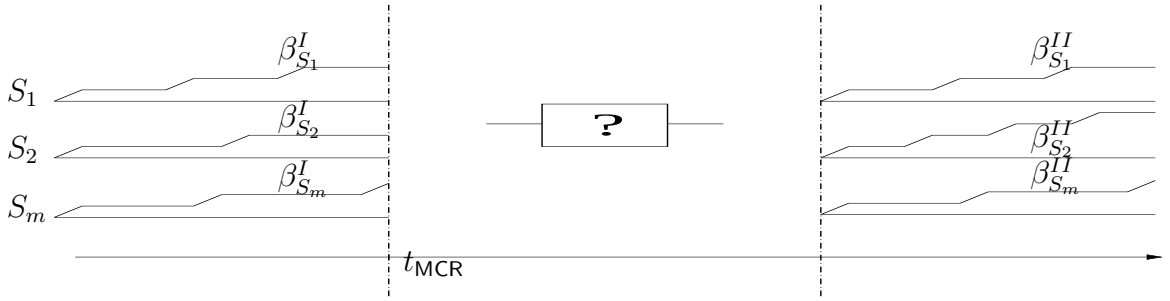


Figure 5.15: Server mode change: transition stage with undefined guarantees.

5.3.1 System Model and Backgrounds

From the platform side, computational resources are provided by reservation servers. As already stated, we consider the class of server algorithms that can be described by a periodic server abstraction, $S = (Q_S, P_S)$.

A Servers can change its parameters at run-time to cope with changing conditions. Whenever a server S switches from an *old mode* $S^I = (Q^I, P^I)$ to a *new mode* $S^{II} = (Q^{II}, P^{II})$, the corresponding supply bound function changes from $\mathsf{sbf}_S^I(t)$ to $\mathsf{sbf}_S^{II}(t)$. The supply bound function that describes the computational resource provided by the server across the mode transition is denoted by $\mathsf{sbf}_S^T(t)$. The bounded-delay approximations are denoted by $\mathsf{bdf}^I = (slope^I, \Delta^I)$, $\mathsf{bdf}^{II} = (slope^{II}, \Delta^{II})$, and $\mathsf{bdf}^T = (slope^T, \Delta^T)$, respectively. The bounded-delay function lower bounds the service provisioning.

From t_{MCR} on, all the required changes in the system are initiated, while the new mode begins at t_{MCO} after a delay $\delta \geq 0$ from the mode change initiation, $t_{\mathsf{MCO}} = t_{\mathsf{MCR}} + \delta$. The mode transition is the stage between the two steady modes, starting at time t_{MCR} and ending when the new mode become effective. Since t_{MCR} and t_{MCO} are not always the same, the mode change analysis has to work $\forall\ t_{\mathsf{MCR}}$ and t_{MCO} such that $t_{\mathsf{MCR}} \leq t_{\mathsf{MCO}}$. In case of multi-moded servers the application schedulability has to be guaranteed during the mode change transition and in order to set up a schedulability criterion that works in any possible condition for the server and the applications, it is important to identify the minimum resource amount provided in any possible interval. This includes the transition between different modes where the service provisioning is affected from the old and the new mode.

Two can be the server behavior in case of mode change, which means that two can be the different transition phases.

- *transitionA*: at t_{MCR} the the old mode *aborts* interrupting the service provisioning until the new mode is operative. This leave a hole in the service provisioning equal to the delay δ that has to be waited.

- *transitionB*: the old mode server keeps on providing its service until the new mode is operative. The service provisioning is kept with the same parameters until t_{MCO}.

The two cases results in a different behavior of the server and then of the whole system. The differences will be explained in the next sections. All the intermediate cases, with abortion time t_{ab} such that $t_{\mathsf{MCR}} < t_{\mathsf{ab}} < t_{\mathsf{MCO}}$ can be easily inferred from the transitionB case.

We assume that the system is feasible in each mode, that is in both steady states conditions, and wants to derive the condition in which feasibility can also be preserved during mode transitions.

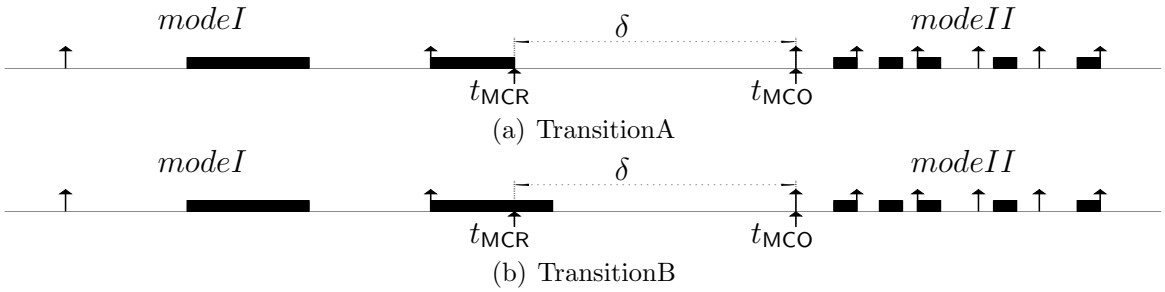
The computational resource the server provides is used by the application by the server.

Definition 5.3.1 (Server Schedulability)**.** *A server is said to be schedulable if its budget is provided on time within each period.*

Definition 5.3.2 (Application Schedulability)**.** *An application is said to be schedulable by a server if all its tasks are able to meet their deadlines.*

Definition 5.3.3 (System Schedulability)**.** *A system consisting of multiple applications managed by a set of reservations said to be schedulable if both servers and applications are schedulable.*

5.3.2 Server Transitions

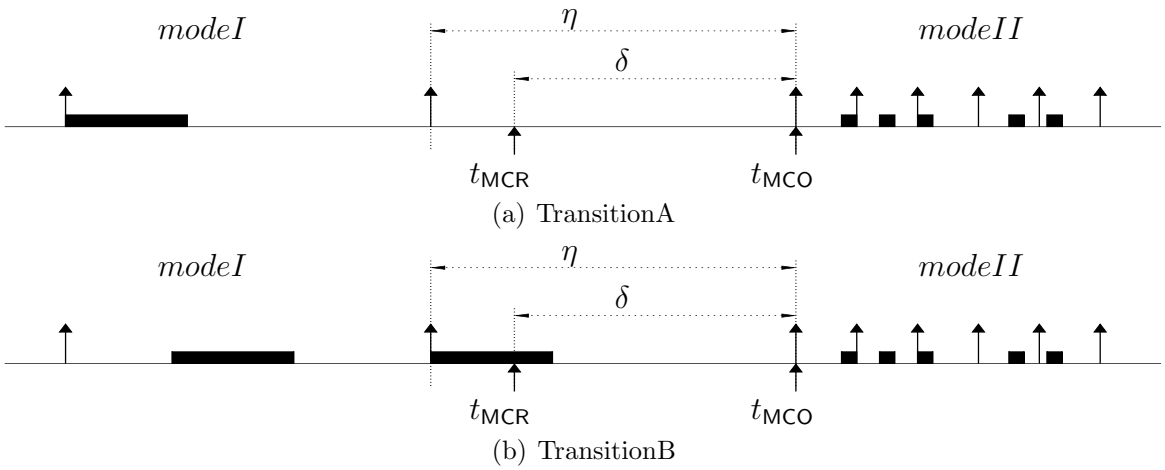


(a) TransitionA

(b) TransitionB

Figure 5.16: TransitionA and transitionB. The service provision aborts or continues after the mode change t_{MCR} respectively.

Since the server behaves differently either in case of transitionA or transitionB the service provisioning will be different accordingly. Given a delay of δ for the new mode, it is possible to derive guarantees to the transition supply bound functions as follows.



(a) TransitionA

(b) TransitionB

Figure 5.17: Server mode change and mode transition cases.

With multi-moded servers it is important to identify the minimum amount of resource provided in any stage of the server. This includes the transition between different modes, where the service provisioning is affected from both the old and the new mode. The transition supply bound function sbf^T is the resource provisioning during the transition stage, while the transition bounded-delay function bdf^T lower bounds the transient service provisioning.

5.3.3 Transition Guarantees

The following theorems provide the sbf^T functions for the two types of transitions introduced in Section 6.2.1 (transitionA and transitionB).

Theorem 5.3.4 (Mode Change sbf - transitionA)**.** *Let S be a periodic server with steady states parameters (Q^I, P^I) and (Q^{II}, P^{II}) and with corresponding supply bound functions sbf^I and sbf^{II}. Let t_{MCR} be the time at which the mode change is requested and let t_{MCO} be the time at which the new mode is started, after a delay δ, such that $t_{\mathsf{MCO}} = t_{\mathsf{MCR}} + \delta$. If the server aborts the old mode at time t_{MCR} (transitionA case), then the resource provisioning during the transition stage is lower bounded by*

$$\mathsf{sbf}^T(t) \;=\; \inf_{0\le\lambda\le t}\{\mathsf{sbf}^I(t \,-\, \lambda \,-\, \eta \,+\, P^I \,-\, Q^I) \,+\, \mathsf{sbf}^{II}(\lambda \,+\, P^{II} \,-\, Q^{II})\}, \quad (5.12)$$

being $\eta = t_{\mathsf{MCR}} - t^{\mathsf{last}} + \delta$ and t^{last} the initial of the last period of the old mode.

Proof. Since the mode change can occur any time during the period interval of the first mode, the worst-case service provisioning occurs when the t_{MCR} is at the beginning of P^I. The worst-case resource supply is when the old mode provides its budget at the beginning of the last period before the change, while the new mode provides resources at the end of the subsequent periods. This leaves a hole of $\eta + P^I - Q^I + P^{II} - Q^{II}$ in the service provisioning. If $R(t)$ is the amount of computation that has been delivered up to time t, we are looking upon a bound of such a resource in a generic interval $[r, s)$ with $s = r + t$ centered in $[t_{\mathsf{MCR}}, t_{\mathsf{MCO}}]$.

$$\begin{aligned} R[r,s) &= R^I[r, t_{\mathsf{MCO}}) + R^{II}[t_{\mathsf{MCO}}, s) \\ &\quad\; \lambda := s - t_{\mathsf{MCO}} \\ &= R^I[r, s-\lambda) + R^{II}[s-\lambda, s) \\ &= R^I[s-t, s-\lambda) + R^{II}[s-\lambda, s) \end{aligned}$$

and each R is lower bounded by its sbf. In this case,
$R^I[s-t, s-\lambda) \ge \mathsf{sbf}^I\left(t - \lambda - (\eta + P^I - Q^I - 2(P^I - Q^I))\right)$ where the service provisioning has a delay of $\eta + P^I - Q^I$. Instead,
$R^{II}[s-\lambda, s) \ge \mathsf{sbf}^{II}\left(\lambda - (P^{II} - Q^{II} - 2(P^{II} - Q^{II}))\right)$ with a delay of $P^{II} - Q^{II}$. Both cases are represented in Figure 5.17(a). Thus

$$\begin{aligned} R[r,s) &\ge \mathsf{sbf}^I(t - \lambda - \eta + P^I - Q^I) \\ &\quad\; +\mathsf{sbf}^{II}(\lambda + P^{II} - Q^{II})\ \forall\lambda \\ &\le \inf_{0\le\lambda\le t} \mathsf{sbf}^I(t - \lambda - \eta + P^I - Q^I) \\ &\quad\; +\mathsf{sbf}^{II}(\lambda + P^{II} - Q^{II}), \end{aligned}$$

that proves the theorem. □

Theorem 5.3.5 (Mode Change sbf - TransitionB)**.** *Let S be a periodic server with steady states parameters (Q^I, P^I) and (Q^{II}, P^{II}) and with corresponding supply bound functions sbf^I and sbf^{II}. Let t_{MCR} be the time at which the mode change is requested and let t_{MCO} be the time at which the new mode is started, after a delay δ, such that*

$t_{\mathsf{MCO}} = t_{\mathsf{MCR}} + \delta$. *If the server continues to provide its old mode service until time* t_{MCO} *(transitionB case), then the resource provisioning during the transition stage is lower bounded by*

$$\mathsf{sbf}^T(t) = \inf_{0 \le \lambda \le t} \{\mathsf{sbf}^I(t - \lambda - \eta + 2P^I - Q^I) + \mathsf{sbf}^{II}(\lambda + P^{II} - Q^{II})\}, \quad (5.13)$$

being $\eta = t_{\mathsf{MCR}} - t^{\mathsf{last}} + \delta$ *and* t^{last} *the initial of the last period of the old mode.*

Proof. The worst-case service provisioning occurs when the old mode provisions its resource at the beginning of its last period, while the new mode provisions at the end of its period. This leaves a hole in the service provisioning of $\delta + P^{II} - Q^{II}$. For any interval $[r, s)$ such that $r \le t_{\mathsf{MCR}} \le t_{\mathsf{MCO}} \le s$ the amount of computational resource provided $R(t)$ is

$$\begin{aligned} R[r,s) &= R^I[r, t_{\mathsf{MCO}}) + R^{II}[t_{\mathsf{MCO}}, s) \\ &\quad \lambda := s - t_{\mathsf{MCO}} \\ &= R^I[r, s-\lambda) + R^{II}[s-\lambda, s) \\ &= R^I[s-t, s-\lambda) + R^{II}[s-\lambda, s) \end{aligned}$$

and each R is lower bounded by its sbf. In this case
$R^I[s-t, s-\lambda) \ge \mathsf{sbf}^I\left(t - \lambda - (\eta - Q^I - 2(P^I - Q^I))\right)$ where the service provisioning has a delay of $\eta - Q^I$. Instead,
$R^{II}[s-\lambda, s) \ge \mathsf{sbf}^{II}\left(\lambda - (P^{II} - Q^{II} - 2(P^{II} - Q^{II}))\right)$ with a delay of $P^{II} - Q^{II}$. Both cases are represented in Figure 5.17(b). Thus

$$\begin{aligned} R[r,s) &\ge \mathsf{sbf}^I(t - \lambda - \eta + 2P^I - Q^I) \\ &\quad + \mathsf{sbf}^{II}(\lambda + P^{II} - Q^{II})\ \forall\lambda \\ &\le \inf_{0 \le \lambda \le t} \mathsf{sbf}^I(t - \lambda - \eta + 2P^I - Q^I) \\ &\quad + \mathsf{sbf}^{II}(\lambda + P^{II} - Q^{II}), \end{aligned}$$

that proves the theorem. □

The obtained sbf^Ts can be used to guarantee the service provisioning during the mode change transitions.

With Equation 5.12 and Equation 5.13 it has been derived the transition supply bound function as a function of η and consequently of the delay δ, $\mathsf{sbf}^T = f(\delta)$.

In case on no delays among the ending of one mode and the beginning of the new one, $\delta = 0$ and $t_{MCO} = t_{MCR}$, it is

$$\mathsf{sbf}^T(t) = inf_{0 \le \lambda \le t}\{\mathsf{sbf}^I((t - \lambda) + \mathsf{sbf}^{II}(\lambda + P^{II} - Q^{II})\},$$

in both the transition cases.

Transition Bounded-Delay Function With the bounded-delay modeling it is possible to derive the transition bounded-delay functions for servers for the transition supply bound functions. First, the same idea of Equation 5.12 and Equation 5.13 can be applied with bounded-delay functions obtaining

$$\mathsf{bdf}_A^T(t) = inf_{0\leq\lambda\leq t}\{\mathsf{bdf}^I((t-\lambda-\eta+P^I-Q^I)+\mathsf{bdf}^{II}(\lambda+P^{II}-Q^{II})\}, \quad (5.14)$$

in case of transitionA, while

$$\mathsf{bdf}_B^T(t) = inf_{0\leq\lambda\leq t}\{\mathsf{bdf}^I((t-\lambda-\eta+2P^I-Q^I)+\mathsf{bdf}^{II}(\lambda+P^{II}-Q^{II})\}, \quad (5.15)$$

in case of transitionB.

In the bounded-delay model the transition resource supply results in a representation with $(slope^T, \Delta^T)$, where Δ is the largest interval of time with no resource provisioning during the transition. It happens, during the transition, that the delay δ contributes to the worst-case conditions for the service provisioning from the servers.

For transitionA cases it is

$$\begin{aligned} slope_A^T &= min\{slope^I, slope^{II}\} \\ \Delta_A^T &= \eta + P^{II} - Q^{II}. \end{aligned} \quad (5.16)$$

TransitionB cases instead, results in

$$\begin{aligned} slope_B^T &= min\{slope^I, slope^{II}\} \\ \Delta_B^T &= \eta - Q^I + P^{II} - Q^{II}. \end{aligned} \quad (5.17)$$

We can observe how $\Delta_A^T \geq \Delta_B^T$, which is reasonable because of the nature of the transitionA: the service provisioning is interrupted before the transitionB case, so less service is provided during the transition. Furthermore, depending on η, hence δ, the transition bounded-delay function can be larger than the mode bounded-delay functions. Case by case $\mathsf{bdf}^I, \mathsf{bdf}^{II}$ and bdf^T can have different relationship. In particular, given $\eta = P^I$, which is one of the possible conditions, we have many possibilities to order the 3 bounded-delay functions depending on the parameters before and after the mode change of the server.

Both the transition supply bound function and the transition bounded-delay functions depends on the transition delay δ, $\mathsf{sbf}^T(t,\delta)$ and $\mathsf{bdf}^T(t,\delta)$.

Equation 5.16 and 5.17 define the relationship between the transition delay δ and the transition service provisioning delay Δ^T.

5.4 Server Schedulability

We have verified that both the applications and servers can change their parameters at run-time. Mostly we know how to model their transitions.

Once defined the demand and supply bound function that bound the resource demand and supply in all the server-application conditions (either old mode or new mode

or the transition stage), it is possible to generalize the schedulability criteria to multi-moded systems.

WIth the dbf and sbf representation developed so far we can model each component, server and application, in any of its state (mode I, mode II and transition). This means that we can setup schedulability conditions in any component state applying the model that better abstract the component status. Allowing both the applications and the servers to change many can be the combinations that has to be verified in order to guarantee the system schedulability in any possible scenario.

- stable conditions: application in its mode I or mode II and the server in its mode I or mode II. The schedulability guarantee applies the demand and supply bound functions in steady conditions.
  - Both the server and applications in their mode I status: $\forall\ t\ \mathsf{dbf}^I \leq \mathsf{sbf}^I$.
  - Both the server and applications in their mode II status: $\forall\ t\ \mathsf{dbf}^{II} \leq \mathsf{sbf}^{II}$.
  - The server is in mode I and the application in mode II: $\forall\ t\ \mathsf{dbf}^{II} \leq \mathsf{sbf}^I$.
  - The server is in mode II and the application in mode I': $\forall\ t\ \mathsf{dbf}^I \leq \mathsf{sbf}^I$.
- During the transitions: one of the two component is in transition, which means that either the demand or the supply bound functions are represented with the transition model, or both in case of simultaneous transitions of the servers and their applications.

5.4.1 Transition Schedulability

In case of transitions (which is the most interesting case) the schedulability has to be guaranteed. Supposing that both the application and the server are changing it follows following.

Lemma 5.4.1 (Mode Change EDF Schedulability)**.** *Given a server S handling an application Γ that is feasible in each in a steady state condition, if both S and Γ change their at the same time, then the application is feasible during the transition phase if*

$$\forall\ t\quad \mathsf{dbf}_\Gamma^T(t) \leq \mathsf{sbf}_S^T(t). \tag{5.18}$$

Using the the bounded-delay linear approximation, the feasibility condition becomes:

$$\forall\ t\quad \mathsf{dbf}_\Gamma^T(t) \leq \mathsf{bdf}_S^T(t), \tag{5.19}$$

Note that when the application increases its resource request, a short transition delay in the server adaptation is good for the application. However, a too short delay could result in a service over-provisioning that would steel bandwidth from the other servers, so jeopardizing the schedulability of the other applications during the transient. On the other hand, a large delay in the server adaptation would affect the schedulability

of the application itself. In general, there is a trade-off between schedulability of the application and the schedulability of the servers.

In case of fixed priority scheduling such as Rate Monotonic (RM), the schedulability condition applies the workload and becomes as follows.

Theorem 5.4.2 (Mode Change RM Schedulability)**.** *An application* Γ *and a server* S *are schedulable in mode I, mode II and during mode transition if*

$$\forall\ \tau_i\ \in \Gamma\ \exists\ t\ \ w_i^T(t) \leq \mathsf{sbf}_S^T(t). \tag{5.20}$$

where w^T is the transition level-i workload. The transition level-i workload can be obtained in the same way as the transition demand bound function. Considering the bounded delay model it is

$$\forall\ \tau_i\ \in \Gamma\ \exists\ t\ \ wbf_{\tau_i}^T(t) \leq \mathsf{bdf}_S^T(t), \tag{5.21}$$

with the workload bound function, wbf to bound the level-i workload.

The supply bound function that can be guaranteed in all the stages a server pass trough can be more pessimistic than the single mode guarantees. Though, it depends on the interval δ that has to be awaited before letting the new mode start. The bdf^T lower bounds all the possible transition service supply, and δ is given as a parameter for that change. From the application point of view, the earliest the mode can change the better it is. The problem is that such a resulting bound function has to cope, together with the rest of the servers, with the resource provided by the upper layers. Then, the delay δ comes from both the schedulability analysis of the application-server sub-system and the schedulability analysis of the set of servers composing the whole system.

Inter-Server Schedulability Analyzing the schedulability of the application with respect to the service provided by its server we can derive a maximum delay $\delta^\flat$ in the service provisioning that can be afforded by the application and its timing requirements. The delay is obtained by comparing the sbf (or its relate bdf) and the dbf. In case of EDF scheduling policy we obtain $\delta^{EDF,\flat}$ as the maximum δ that guarantee $\mathsf{dbf}^T(t) \leq \mathsf{sbf}^T(t,\delta)$. In case of bdf it is $\delta'^{EDF,\flat} = max\{\delta \mid \mathsf{dbf}^T(t) \leq \mathsf{bdf}^T(t,\delta)\}$.

In case of FP scheduling $\delta^{FP,\flat}$ is the maximum among the δ delays that satisfies $w_i^T(t) \leq \mathsf{sbf}^T(t,\delta)$ for all task i. By w_i we intend the workload bound function as defined in Chapter 2. With bounded-delay functions it is $\delta'^{FP,\flat} = max_i\{max\{\delta \mid w_i^T(t) \leq \mathsf{bdf}^T(t,\delta)\}\}$.

By intra-server schedulability we refer to the schedulability of the application based on the resources provided by the server. From the analysis performed by applying either Condition (5.18), Condition (5.19), Condition (5.20), or Condition (5.21) , we can derive a maximum delay $\delta^\flat$ after which the new mode can safely start. For all $\delta \leq \delta^\flat$ the server mode change is feasible for the application because it will result in a larger resource supply. With the bounded-delay functions it is obtained $\delta'^\flat$ which is larger than the one from the sbfs because of the bounding of the bounded-delay functions to the supply bound ones.

Inter-Server Schedulability By inter-server schedulability we refer to the schedulability of the servers when they adapt their parameters, independently of the behavior of the served applications. Such an analysis can be performed using the results achieved for multi-mode task sets, which is well known in the real-time literature. The analysis can easily be applied to periodic servers by considering them as periodic tasks that must receive Q units of computational resource every period P. Therefore, the effect of a mode change in a server has to be investigated with respect to the entire set of servers composing the system.

Crespo et al. (133) have surveyed solutions to the mode-change problem together with proposing new solutions in case of fixed priority servers. Guanming (71) instead, proposed a solution for EDF scheduling policies. Perathoner et al. (153) have investigated the task mode change problem applying both the scheduling paradigm and came out with sustainable solutions. By applying Equation 5.11 to the mode changing server resource request and considering Equations 5.18 and 5.20 with respect to the specific scheduling paradigm applied, the task-wise analysis results in a minimal delay $\delta^\sharp$ that does not affect the schedulability of other servers. By schedulability of a server it is intended the possibility of provide its periodic budget on time together with the schedulability of the applications each server manages. $\forall \delta \geq \delta^\sharp$ the set of server remains schedulable because a larger δ in this case results in a lower resource request for the resource scheduler.

Any change after a delay $\delta \geq \delta^\sharp$ keeps the system feasible, but reduces the amount of resource provided to the application managed by the changing server.

With the help of one examples it is possible to show how the mode change analysis of applications applies to servers to derive the minimum delay $\delta^\sharp$ for the changing server by considering a set of server composing the system.

Example 5.4.3. *Considering two servers such that $S_1^I = (5, 10)$ and $S_2^I = (2, 4)$ in their first mode, and $S_1^{II} = S_1^I = (5, 10)$ and $S_2^{II} = (4, 8)$. The first server does not change its mode. Requested a mode change at $t_{MCR} = 2$ the set of server results unfeasible during the transition with a $\delta = 0$, so S_2^{II} starting at t_{MCR}. Through the utilization-based approach the minimum that has to be waited is $\delta = 2$ from Equation 5.2 which has been applied instead of Equation 5.4 because the total utilization of the servers is 1. $\delta = 2$ is the delay $\delta^\sharp$ the second server has to wait. For all the delay larger than $\delta^\sharp$ the server set results feasible, while $\forall \delta < \delta^\sharp$ the server set suffer a deadline miss.*

A real-time system with servers and applications is feasible during mode transitions if the delay is less than or equal to a threshold derived from the intra-server analysis and greater than or equal to a threshold $\delta^\flat$ derived from the inter-server analysis; that is, $\delta \leq \delta^\flat \wedge \delta \geq \delta^\sharp$. The resulting feasibility region for δ is then

$$\Phi = \begin{cases} 0 & \text{if } \delta^\sharp > \delta^\flat \\ \forall \delta \mid \delta^\sharp \leq \delta \leq \delta^\flat & \text{if } \delta^\sharp \leq \delta^\flat \end{cases} \tag{5.22}$$

This result is stated in the following theorem.

Theorem 5.4.4. *Consider a multi-mode system with m servers S_i, each managing an application Γ_i, such that the system is feasible in any of its working mode. If server S_i performs a mode transition with a delay δ, the system remains feasible if $\delta \in \Phi_i$, where Φ_i is the feasibility region of server S_i from Equation (5.22). If Φ_i is empty, the transition is unfeasible under any condition.*

Proof. The theorem directly comes form the construction of the region Φ_i, since the δs in Φ_i have been obtained by guaranteeing the feasibility of the set of servers in the system. □

Algorithm 8 describes how to compute a transition delay that keeps the system feasible.

Algorithm 8 Algorithm to compute the transition delay for the mode changing server S_i. The policy is the strategy adopted by the system to handle mode transitions.

Input: Γ, $\Gamma_S = \{S_1, \ldots, S_m\}$ and the mode changing server S_i, $S_i \in \Gamma_S$;
Output: transition delay $\delta \in \Psi$;
$\delta^\flat$ = intra-serverSchedulability(Γ_i, S_i);
$\delta^\sharp$ = inter-serverSchedulability(Γ_S, S_i);
δ = policy($\delta^\flat, \delta^\sharp$);

5.5 Resource Reservation Analysis

The server and the application requirements during a mode transition are encoded into the feasibility region Φ in terms of transition delays that the set of servers and the applications can tolerate.

5.5.1 $(slope, \Delta)$-Space

Formerly we have derived a $(slope, \Delta)$ representation of servers where each tuple $(slope_i, \Delta_i)$ represents a resource supply. Even applications can be mapped into the $(slope, \Delta)$-space. Indeed, depending on the scheduling policy, it is possible to derive the application representation in the $(slope, \Delta)$-space as the feasibility region of the application itself according to the scheduling paradigm.

The feasibility region of an application is intended as all the service supply $(slope, \Delta)$ spots that enforce the timing constraints of the application itself. Such a region is obtained by referring to the whole computational resource available for the application. Besides, a single $(slope, \Delta)$ spot bounds many applications and their demand bound function, and an application resource demand can be bounded by different $(slope, \Delta)$ tuples.

The FP schedulability criteria with workloads (24) is

$$\forall i \; \exists t \; \in \; schedP_i \; : \; w_i(t) \leq slope(t - \Delta),$$

in terms of $(slope, \Delta)$ means that

$$\begin{aligned} \forall i \; \exists t \; \in \; schedP_i \; : \; \Delta \leq t - \frac{w_i(t)}{slope} \\ \forall i \; \Delta \leq max_{t \in schedP_i}\{t - \frac{w_i(t)}{slope}\} \\ \Delta \leq min_i max_{t \in schedP_i}\{t - \frac{w_i(t)}{slope}\} \end{aligned} \tag{5.23}$$

Equation 5.23 defines the application feasible region under FP. Each application Γ has its own depending on the tasks composing the application, both in terms of their parameters and their timing requirement.

For an EDF scheduling policy the application feasible region is more easy to find. The application feasibility region is defined as

$$\forall t \; \in \; \mathrm{D} \; : \; \mathsf{dbf}(t) \leq slope(t - \Delta),$$

with

$$dbf(t) = \sum_i max\{0, \lfloor \frac{t - T_i - D_i}{T_i} \} C_i.$$

It means that

$$\begin{aligned} \forall t \; \in \; \mathrm{D} \; : \; \delta \leq t - \frac{\mathsf{dbf}(t)}{slope} \\ \Delta \leq min_{t \in \mathrm{D}}\{t - \frac{\mathsf{dbf}(t)}{slope}\} \end{aligned} \tag{5.24}$$

Equation 5.24 defines the application feasible region as the minimum among curves in the $(slope, \Delta)$ domain. In the $(slope, \Delta)$-space such a region represents the application Γ and its scheduling algorithm *sched*.

As we already stated, the scheduling criteria compares the supply bound function of servers and the demand bound function of applications. It allows also to translate the usual schedulability criteria in the interval domain into the $(slope, \Delta)$-space, as we have showed.

Figure 5.18 depicts the feasible region of an application when is scheduled by EDF or by FP. The two feasibility regions are represented by the bold curves while the intermediate curves which are used to derive the regions are the normal curves. It is remarkable that the EDF region superpose the FP one.

A servers applied to the applications limit the service provisioning with respect to the whole resource available. So server bounded-delay functions are spots in the $(slope, \Delta)$-space affecting the application feasible region by cutting some part as not allowed anymore because of the new service provisioning of the server applied. It is important to note how a larger bdf increases the feasibility region of the applications. By larger bdf it is intended a provisioning with reduced delay Δ and/or and increased slope *slope*, see Figure 5.20. Figure 5.20 depicts the feasibility region of an application

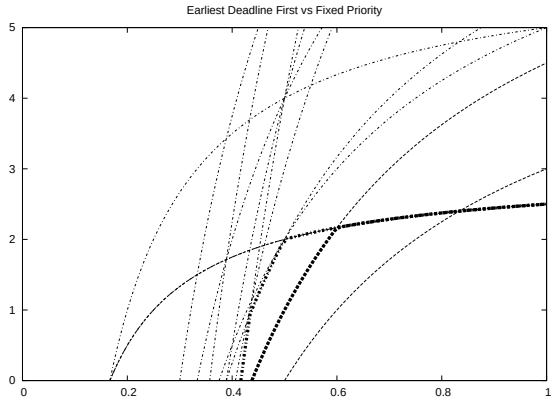


Figure 5.18: Feasibility region in the $(slope, \Delta)$-space: EDF vs FP feasibility regions in bolt. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

scheduled with fixed-priority, and a server with $slope = 0.8$ and $\Delta = 1$. The colored rectangle describes all the service provisioning lower than the amount provided by the server. In that case it is the resource smaller than $\mathsf{bdf} = (0.8, 1)$. The intersection among the two regions defines the new feasibility region for the application.

In case of multi-moded servers, a mode change is equivalent of having three different servers, one at mode I and the second at mode II with two different constraints on the feasibility region of the application. As demonstrated in the previous section, the transition stage affects the application region by changing the resource provisioning. It is then such as having another server sbf^T affecting the application feasibility region. This is the third server.

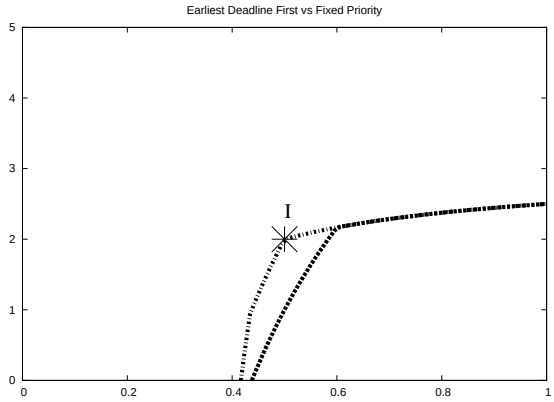


Figure 5.19: Feasibility region in the $(slope, \Delta)$-space: EDF vs FP with different feasibility regions. A feasible application under EDF not feasible under FP in case of server applied. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

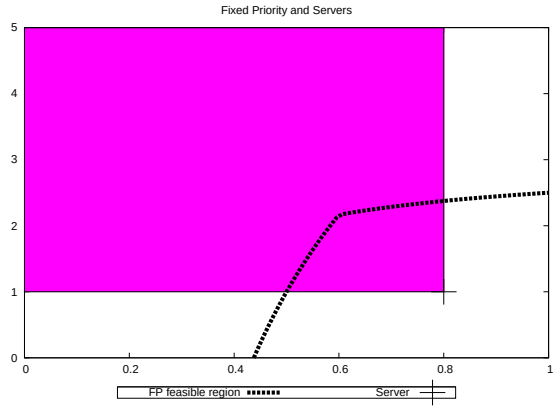


Figure 5.20: Feasibility region with FP in the $(slope, \Delta)$-space: feasibility region bounds and a server at $(0.8, 1)$. The colored region describe a lower resource provisioning by servers with respect to $0.8, 1)$. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

5.5.2 Space Solution Analysis

From the analysis of the application feasible region it is possible to infer both the server/service requirements of a feasible multi-moded application and the application requirements to cope with multi-moded servers.

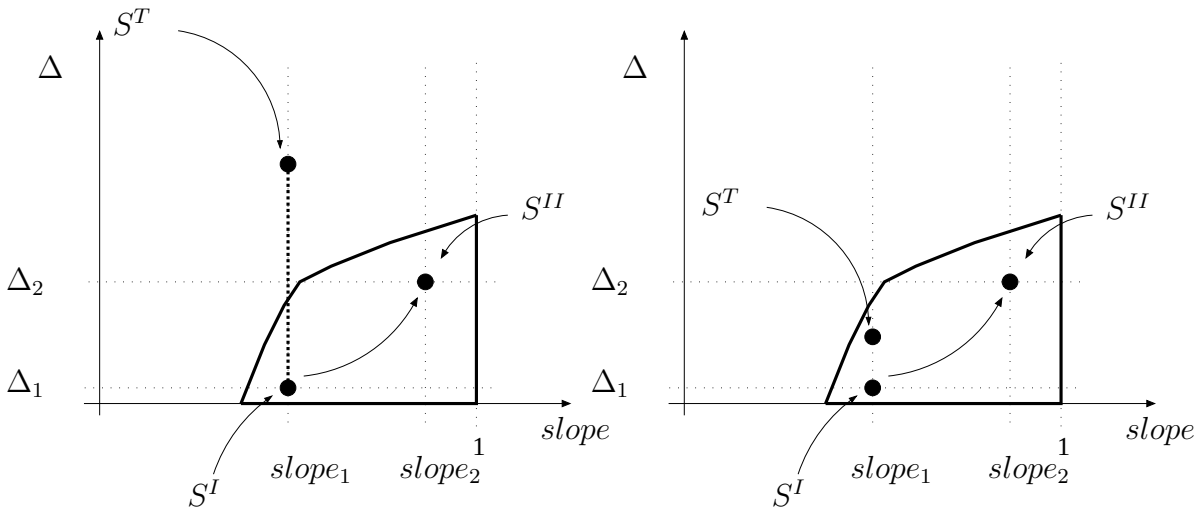


Figure 5.21: Mode transition in the $(slope, \Delta)$-space: an example of unfeasible transition together with an example of a feasible transition. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

Difference among scheduling policies, in the $(slope, \Delta)$-space are mapped into different feasibility region for the application as previously detailed. Equation 5.23 and 5.24 describe the feasibility regions for the applications in the $(slope, \Delta)$-space Ω, as a function of the application and the scheduling algorithm, $\Omega = f(\Gamma, sched)$. In particular, EDF has a bigger region than fixed priority; which means a well known concept to

the real-time community: *applications feasible under EDF could not be feasible under FP.*

The server partitions the resource that has to be passed to the application, hence it reduces the whole resource according to the server parameters and the scheduling applied to the servers. It is then true that a server reduces the feasibility region of an application which is obtained in the case of unlimited resources available. An application which is feasible with the whole computational resource available could not be feasible anymore once managed by a server. Moreover, the effect of a server applied to an application are different with respect to the scheduling policies applied. A server applied to a EDF scheduling could affect less the schedulability of an application, by still keeping the application feasible. In case of FP feasibility regions, a server has more chances to turn a feasible application into an unfeasible one; see Figure 5.19, where the server applied keeps the application feasible in the EDF case, but not in the FP one.

A server that makes an application unfeasible has to be studied in order to see what could be the countermeasure that would make the application feasible again i.e., the use of other possible servers or changing its parameters, in order to keep the application feasible, which is the main objective of our analysis.

Concerning multi-moded servers, the schedulability in the two steady states conditions (old and new modes) does not guarantee the schedulability of the transition. This means that the server mode change transition could result in a tuple $(slope, \Delta)$ outside the feasibility region of the application even though the service guarantee of the two steady states are inside that region. It is then an analysis of the feasibility region and the transition guarantees to conclude about the feasibility of the application and a multi-moded server. Figure 5.21 shows examples of feasible transition and an unfeasible one with respect to a real-time application; because of the changing resource reservation applications cannot be feasible anymore. We remind that the usual assumption about mode change transition is that the component is feasible in its steady states conditions. Which, in the server case, means that the server service guarantees in stable conditions are inside the feasibility region of the server application.

The problem addressed in this dissertation considers the case in which the two points corresponding to the server steady state modes fall in the feasibility region, and the proposed analysis allows verifying the feasibility of the application also during the transition phase, assuming it is performed with a given delay δ. In the case the transition point falls outside the feasibility region, the delay delta can be used as a design parameter to reach feasibility during the transition, if that is possible.

The servers provides service in a stair-case manner and not with straight lines as with the bdf models. This means that the sbf extends the space of feasible solutions making more cases feasible. The analysis with bounded-delay functions is pessimistic because the bdf approximates the sbf with a linear lower bound. Thus, it results in only sufficient conditions. A more accurate analysis would require a more complex space solution. The feasibility region would be larger than the $(slope, \Delta)$ one but more difficult to be managed. With a tighter bound the space of feasible solution (feasible applications) increases, reducing the unfeasible region. Moreover, an application derived within that space is assured to be feasible since $\mathsf{sbf} \geq \mathsf{dbf}$. Instead, an application

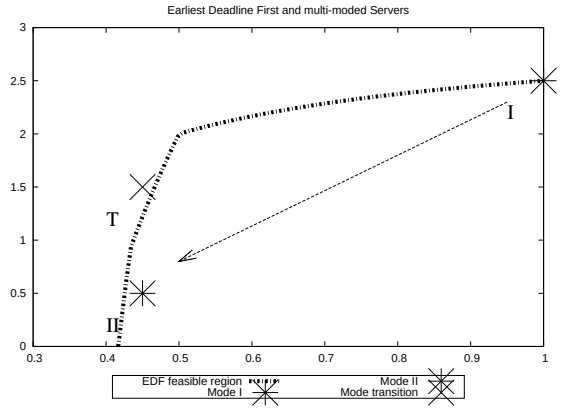


Figure 5.22: Feasibility in the $(slope, \Delta)$-space: application scheduled with EDF and server changing from $S^I = (1, 2.5)$ to $S^{II} = (0.45, 0.5)$. The mode transition is unfeasible with a maximum possible delay in the resource supply $\Delta = 1.5$. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

outside that region is not feasible during the mode change, while it has to be chosen to be feasible in the two mode as the main assumption.

Despite the more accuracy by analyzing the supply bound function, the $slope, \Delta)$-space resulting from the application of the bounded-delay approximation, is more easy to investigate.

5.5.3 $(slope, \Delta)$-space Sensitivity Analysis

Sensitivity analysis projects the distance from the feasibility region border to the component parameters and describes metrics to evaluate the sensitivity of the feasibility concept with respect to the parameter changes.

By applying the concepts form (28) the sensitivity analysis can be considered for the $(slope, \Delta)$-space. Once defined an application it is possible to take decisions about

- how much the feasibility is affected by the parameters?
- The parameters of the server mode by checking the feasibility requirements in terms of *slope* and Δ. Which are feasible parameters for the server?
- Given a server, how far is an unfeasible server from the application feasibility region?
- Exploited the differences among the server mechanism in terms of service provisioning, which is the best mechanism for the assigned application and the assumed scheduling policy?

Those are the main questions that the sensitivity analysis aim to answer. In the next chapter we will give a particular example related to adaptive conditions.

Since the $(slope, \Delta)$ representation is an approximation of the supply bound function, the sensitivity analysis will offer only sufficient conditions. An accurate analysis

can be done on the $(slope, \Delta)$-space to see what are the differences between a precise schedulability analysis with supply-bound function and an approximated one with the help of bounded-delay functions.

With a partitioned system it is also possible to reason in the other-way-around: choose an application or improve the one available on order to be feasible with respect to the developed server. Once defined the space (representation space and feasible region inside), a multiple approach is then possible: a) define the applications with a given server through its feasible region; b) how far is an unfeasible application form the given server?

In this dissertation we are interested in the server transitions among two working modes. Once defined the server properties in terms of supply bound function and even bounded-delay function at the steady states and along the mode transition; our interest is to evaluate the feasibility or not of the server transition. Another interesting aspect of the analysis is to define a feasible real-time application that copes with a changing server. We leave this aspects to future works.

In the previous section we have derived the bounded-delay model of a server as a function of the interval δ that the new mode has to wait before it can start. The transition guarantees models a third server (the transition server) which bounded-delay function (and consequently the supply-bound function) depends on such δs. We consider the delay as the parameter where to apply sensitivity analysis; thus we investigate the effects of mode transitions with another abstraction: the $(slope, \Delta)$-space. This way we can also derive feasible solutions for mode changing server.

The $(slope, \Delta)$-space allows to model the application in terms of its feasibility region. With the dbf modeling it is possible to map each state of the application (mode I, mode II and transition) in the space. The mode transition application is a fake application, but since its resource demande is bounded by the transition demand bound function dbf^T, it can be represented and studied in the $(slope, \Delta)$-space. There will be mode I, mode II and mode transition feasibility regions for an applications allowing to verify in that space all the application conditions.

Concerning the servers, it exist the server in mode I and mode II which are two spots in the $(slope, \Delta)$-space, but also the mode transition which is represented by the supply bound function, and consequently the bounded-delay function during the mode transition.

5.5.3.1 Mode Changing Delay

Intra-server and inter-server analysis provides two bounds $(\delta^\flat, \delta^\sharp)$ for the delay δ. Both those bounds have to be satisfied. A partitioned system with servers and applications is feasible during mode transitions if $\delta \leq \delta^\flat \wedge \delta \geq \delta^\sharp$; the resulting in the region Φ formerly defined. That region applies to the $(slope, \Delta)$-space, and with that it is possible to consider sensitivity analysis to the parameter δ in such space.

A server representing a point inside the feasibility region of an application means that the application is feasible with such a server providing resource to it. This means that the service provisioning slope $slope$ is enough for the application and the delay Δ does not affect the timing guarantees of the application. In case of multi-moded

servers, the assumption that the application is feasible in both the modes (mode I and mode II) implies that $(slope^I, \Delta^I)$ and $(slope^{II}, \Delta^{II})$ are inside the feasibility region of the application. Instead, the $\mathsf{bdf}^T = (slope^T, \Delta^T)$ can reside outside that region which means that too less resource is provided by the server during the transition to guarantee the application timing requirements. This would make the application unfeasible during the mode change. We have already seen how Δ^T results from the given delay δ and the server configurations (the servers involved by the mode change as detailed by Equations 5.16 and 5.17).

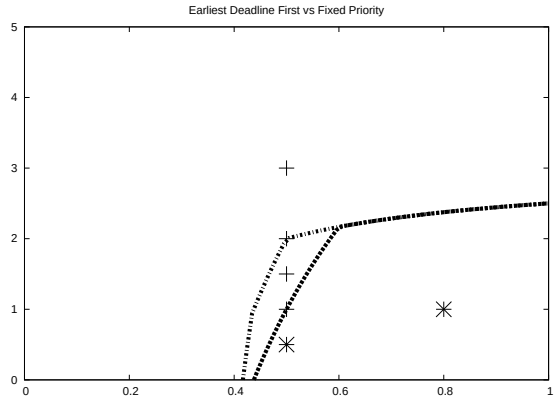


Figure 5.23: Feasibility in the $(slope, \Delta)$-space: EDF vs FP and server with mode change. 4 mode changes are represented with delay $\Delta^T = 1, 1.5, 2, 3$. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

Figure 5.19 shows an interesting case where the mode change is feasible in case of EDF scheduling, while not for FP. In particular is the bdf^T that does not guarantee the application, while the modes do as expected. Figure 5.23 instead, has represented 4 possible transitions. Given an application, the first with $\Delta^T = 1$ keeps the application feasible for both FP and EDF scheduling policies applied. Both the second and the third cases, $\Delta^T = 1.5$ and $\Delta = 2$ leave the application feasible only for EDF. The fourth, $\Delta = 3$ is an unfeasible situation for both the scheduling cases.

Through the framework proposed we can also investigate the difference among the possible transitions. Figures 5.24 and 5.25 show the differences among the two transitions (in terms of Δ^T) respectively with an EDF and a FP scheduling policy applied. We notice how the delay Δ^T is less in case of transitionB, as expected. It is noticeably how Δ^T does not depend on the scheduling policy applied.

Given an application, from the graphical analysis of Figure 5.24 and Figure 5.25 it is possible to conclude less strict bounds to the server parameters in case of EDF with respect to the FP case. The conclusion is general and does not depend on the kind of the application provided.

We comment also the fact that the transition guarantee depends on the transition order and not only on the two states (modes) involved. In case of transition from S^I to S^{II} the effect is different than the transition form S^{II} to S^I thus resulting into two different bdf^T. The order of the mode change (which mode is first) also affects the delay it has to be waited by an application in order to receive the service.

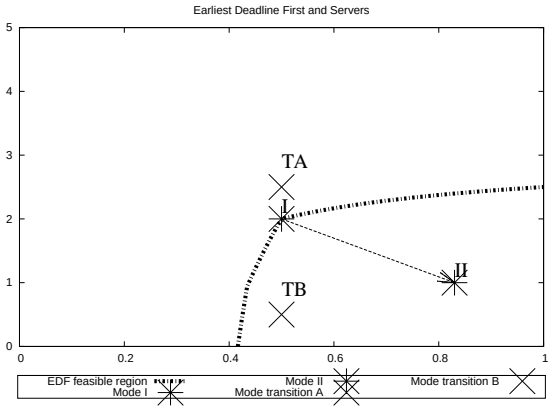


Figure 5.24: Feasibility with EDF in the $(slope, \Delta)$-space: bounds to the server parameters in the two modes $S^I = (1, 2)$ and $S^{II} = (2.5, 3)$ and during the with $\eta = 2.5$. TransitionA and transitionB are. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

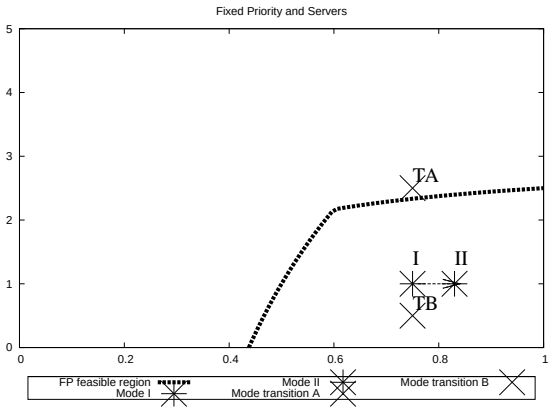


Figure 5.25: Feasibility with FP in the $(slope, \Delta)$-space: bounds to the server parameters in the two modes $S^I = (1.5, 2)$ and $S^{II} = (2.5, 3)$ and during the transition with $\eta = 2.5$. TransitionA and transitionB are represented. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

Example 5.5.1. *Given a server S with possible periods P_S $(2.8, 3)$ and possible budgets Q_S $(2, 2.5)$. Assuming the server changing with a transition of kind transitionA and a delay such that $\eta = 1.3$ time unit assigned by the intra-server and inter-server analysis. The application is given and scheduled by EDF scheduling algorithm. The possible transitions are*

1. *$S^I = (2, 2.8)$ and $S^{II} = (2.5, 3)$. In that case the transition is ruled by a bdf^T with $slope^T = min\{0.71, 0.83\} = 0.71$, and $\Delta^T = \eta + 3 - 2.5 = 1.8$.*
2. *$S^I = (2, 3)$ and $S^{II} = (2.5, 2.8)$. In that case the transition is ruled by a bdf^T with $slope^T = min\{0.6, 0.89\} = 0.6$, and $\Delta^T = \eta + 2.8 - 2.5 = 1.6$.*
3. *$S^I = (2.5, 3)$ and $S^{II} = (2, 2.8)$. In that case the transition is ruled by a bdf^T with $slope^T = min\{0.83, 0.71\} = 0.71$, and $\Delta^T = \eta + 2.8 - 2 = 2.1$.*

4. $S^I = (2.5, 2.8)$ *and* $S^{II} = (2, 3)$. *In that case the transition is ruled by a* bdf^T *with* $slope^T = min\{0.89, 0.6\} = 0.6$, *and* $\Delta^T = \eta + 3 - 2 = 2.3$.

Figure 5.26 show the example with the four possible combination of the two server parameters in case of tranitionA.

The same example can be obtained in case of transitionB, of course with a lower Δ^T for each case due to the transition itself.

As we already mentioned, there are two main problems related to a server mode change which concerns the application the server manages and the rest of the partitioned system. A quick mode change could not affect much the application real-time constrains, while it can violate the resource isolation created by the partitioned system and jeopardize the scheduling of other servers. The solutions of the server mode change problem cames from a trade-off among the two concurrent conditions, where the server, the application and the whole system schedulability has to be guaranteed. As a reminder, by feasibility of the system in this case it is intended both the schedulability of the servers in the sense we gave previously, and the schedulability of the applications the server manages.

From the intra-server and inter-server feasibility analysis it has been derived how the delay affects the system schedulability. The relation 5.22 defines the bounds for the delay. The resulting interval represents the feasibility regionΦ for δ as all the posible delays that keep the system feasible during the mode change. The region depends on the set of servers composing the system, the applications and the servers directly involved in the mode change.

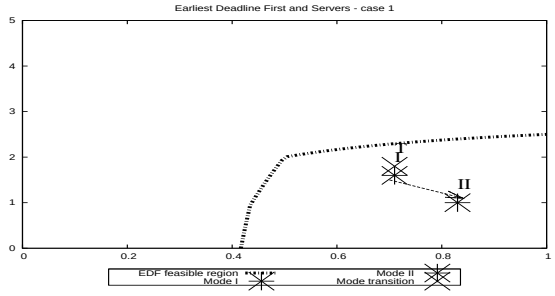
The solutions to the mode changing problems require to find Φ and select there the best δ according to certain requirements of the system. If is required a quick mode change, it is chosen the smallest value within Φ, or in case it is required to apply less resource possible during the change, than δ^{max} in Φ is the solution to that problem.

The differences among the transitions determines different delays. It is then possible to apply different kind of transitions in order to accomplish the delay requirements of a system. In particular, if the allowed delays are small enough it is possible to apply transitionB in case of server mode changes. If that is not possible (intra-server analysis does not allow such a small delay) then the solution is a kind of transition like transitionA.

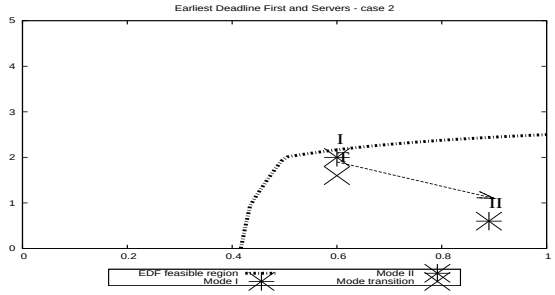
The server mechanism is another degree of flexibility in the mode changing problem. The mechanisms have different resource provisioning resulting in different Φ regions for the delay. The appropriate server can be chosen in order to keep the system feasible during the transition and satisfy the constraints of the problem, hence obtaining δ.

At last also the applications can be modified in order to find better Φ regions (better in the sense of the problem which has to be solved), and fit better those regions.

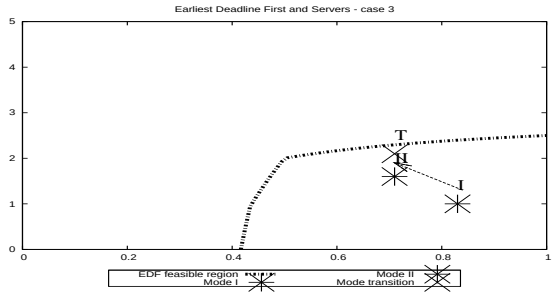
All of these solution have to be applied in a mode change problem framework.



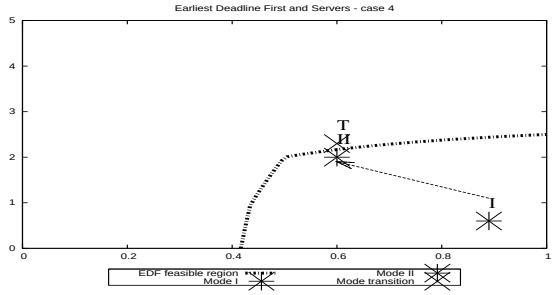
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 5.26: Four possible cases in the $(slope, \Delta)$-space depending on the combination of the old mode and new mode of a mode changing server. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

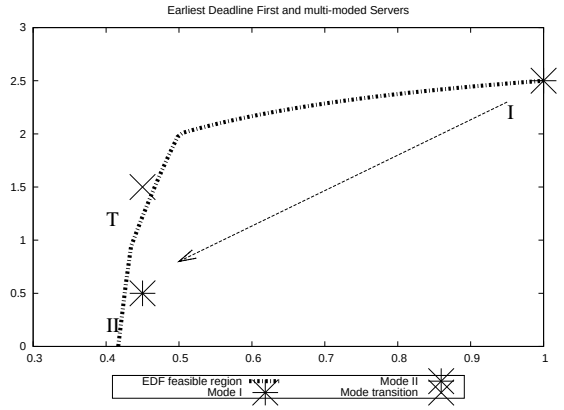


Figure 5.27: Feasibility region in the $(slope, \Delta)$-space: maximum affordable delay for a feasible transition in case of a server changing from $S^I = (1, 2.5)$ to $S^{II} = (0.45, 0.5)$ and scheduled with EDF. The application $\Gamma = \{\tau_1 = (0.5, 3), \tau_2 = (2, 8)\}$.

5.5.4 Mode Change Resource Reservation

An example of mode change is illustrated in Figure 5.28 and Figure 5.29 for an application with two periodic tasks, $\tau_1 = (0.5, 3)$ and $\tau_2 = (1, 8)$, handled by a server changing from $S^I = (slope^I, \Delta^I) = (0.9, 2)$ to $S^{II} = (slope^{II}, \Delta^{II}) = (0.45, 0.5)$. The EDF scheduling policy is applied. Supposing that the inter-server schedulability analysis proves a minimum transition delay $\delta^\sharp = 0$, which means $\Delta^\sharp_A = 1.25$ and $\Delta^\sharp_B = 0$ respectively for transitionA and transitionB. From the feasibility analysis of the server and its application we obtain $\Delta^\flat = 1.88$ that in case of transitionA correspond to $\delta^\flat_A = 0.63$, while for transitionB is $\delta^\flat_B = 10.63$. The transitionA case results in a region $\Phi_A = [0, 0.63]$ while, the transitionB has $\Phi_B = [0, 10.63]$. Figure 5.28 shows the case of transitionA and the feasible region for the delays δ in terms of bounded-delay function delays Δ. Figure 5.29 shows the case of transitionB together with the fact that transitionB is the best possible transition because it provides the widest interval of feasible δ inside the application feasibility region. As a matter of fact the whole Φ_B in terms of Δs stays inside the application feasibility region.

The figure also reports the two values $\Delta^\flat$ and $\Delta^\sharp$ (derived from $\delta^\flat$ and $\delta^\sharp$ through Equation 5.16 or 5.17) which identify the range of transition delays that make the application feasible. For this application, the maximum Δ that can guarantee a feasible transition is $\Delta = 1.88$, corresponding to a maximum transition delay of either $\delta = 0.63$ or $\delta = 10.63$, respectively for transitionA or transitionB . The delays have been computed considering a mode change starting at the beginning of old mode period, $t_{\mathsf{MCR}} = t_{\mathsf{last}}$.

The proposed analysis can be used to find the best δ in accordance to a desired policy that reflects the requirements of the system. The policy included in Algorithm 8 affects the selection of δ in Ψ. In particular, if the goal is to minimize the application response time, then the minimum delay can be selected as $\delta = \min\{\delta \mid \delta \in \Psi\}$. If the goal is to minimize the resource required during the transition (which could be relevant for saving energy), then the maximum delay can be selected as $\delta = \max\{\delta \mid \delta \in \Psi\}$.

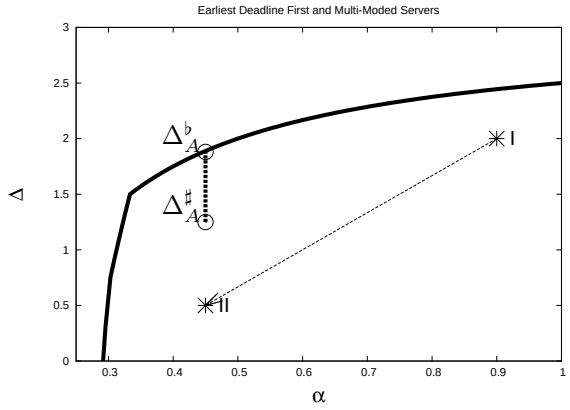


Figure 5.28: Feasibility region: application scheduled with EDF and server changing from $S^I = (slope^I, \Delta^I) = (1, 2.5)$ to $S^{II} = (slope^{II}, \Delta^{II}) = (0.45, 0.5)$. The application has two tasks $\tau_1 = (0.5, 3)$ and $\tau_2 = (1, 8)$. The transitionA is represented, and the segment describes Φ_A for the Δs.

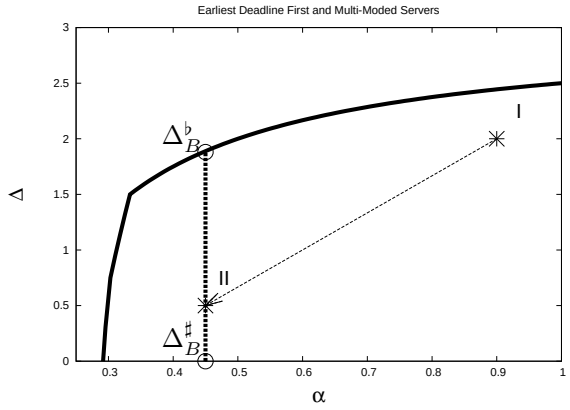


Figure 5.29: Feasibility region in the $(slope, \Delta)$-space: application scheduled with EDF and server changing from $S^I = (slope^I, \Delta^I) = (1, 2.5)$ to $S^{II} = (slope^{II}, \Delta^{II}) = (0.45, 0.5)$. The application has two tasks $\tau_1 = (0.5, 3)$ and $\tau_2 = (1, 8)$. The transitionB is represented, and the segment describes Φ_B for the Δs

Summarizing, a mode change framework for resource reservations has different degrees of freedom for making feasible an unfeasible transition.

- *Resource reservation parameters*: the resource reservation parameters before and after the transition affect the transition itself and can be set to make an unfeasible transition feasible.
- *Kind of transition*: the different transition types determine different transition delays because of their resource supply. In particular, transitionB provides a larger service than transitionA, which means that a larger delay can be afforded by the application when transitionB is applied. By selecting the abortion time during the transition it is possible to modulate the solution in order to satisfy the delay requirements of a system: if the allowed delays are small enough, then transitionA has to be preferred; otherwise (if intra-server analysis does not allow such small delays) transitionB has to be applied.
- *The server mechanism*: the particular server mechanism adopted for implementing a reservation affects the resource provisioning, thus affecting the Φ region for the delay. Hence an appropriate server can be chosen to keep the system feasible during the transition and satisfy the constraints of the problem.

5.5.5 Case Study

We now present a case study of a multi-moded resource reservation with the EDF scheduling policy applied. Let us consider two periodic servers $S = (Q, P)$ characterized by the following initial modes: $S_1^I = (2, 4)$ and $S_2^I = (5, 10)$. The first server manages an application Γ_1 composed by two tasks $\tau_{1,1} = (2, 20)$, $\tau_{1,2} = (5, 30)$ with relative deadlines equal to periods. The second server manages a single task $\tau_{2,1} = (6, 12)$. At time $t = t_{\mathsf{MCR}} = 2$, S_1 is required to change its parameters from $S_1^I = (2, 4)$ to $S_1^{II} = (4, 8)$, while S_2 remains unchanged $S_2^{II} = S_2^I$. In this case, the system is asking server S_1 to provide more resource with a larger period, but with the same bandwidth. The second server guarantees the same resource provisioning to its application during the system changes.

The transition bounded-delay functions for the server S_1 result to be $\mathsf{bdf}_B^T = (0.5, \gamma + 2)$ with $\Delta_B^T = \gamma + 2$ for transitionB, and $\mathsf{bdf}_A^T = (0.5, \gamma + 2)$ with $\Delta_A^T = \gamma + 6$ in case of transitionA applied.

The inter-server analysis provides a minimum delay $\delta^\sharp = 2$ for guaranteeing the feasibility of the other server S_2, see Example 5.4.3 for more details about the computation of $\delta^\sharp$. In other words, adapting S_1 no earlier than 2 time units from the mode change request, the second server is not affected by the mode change and its resource provisioning is guaranteed also during transitions. This translates into bounded-delay functions with a delay $\Delta_B^\sharp = 6$ and $\Delta_A^\sharp = 10$, respectively for transitionB and transitionA.

The intra-server analysis provides the maximum transition delay $\delta^\flat$ that keeps the served application feasible during the transition. That is, $\delta^\flat = \max\{\gamma \mid \mathsf{bdf}^T(t, \gamma) \geq \mathsf{dbf}^T(t)\} - t_{\mathsf{MCR}} + t^{\mathsf{last}}$. For a transitionB it is $\delta_B^\flat = 12$, while for transitionA it is $\delta_B^\flat = 8$.

Hence, the feasibility regions is $\Phi_B = [2, 12]$ for transitionB and $\Phi_B = [6, 8]$ for transitions of type transitionA. The optimal transition delay δ can then be chosen within these intervals, based on the adopted system policy. In terms of Δ the interval is $[6, 16]$ for transitions like transitionB, as represented by Figure 5.31. Figure 5.32 shows the case of transitionA with the region Φ translated into the segment for the Δs, $[10, 16]$. Figure 5.30 shows that Γ_1 is feasible in both modes of server S_1 and the bounded-delay functions requirements during the transitions.

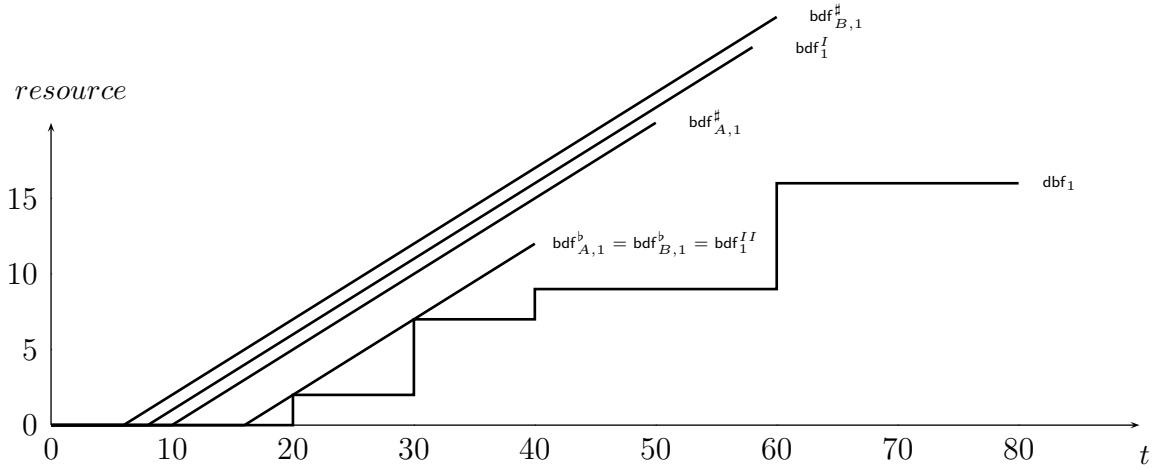


Figure 5.30: Demand curves (dbf) of the application Γ_1 and resource curves (bdf) for both the transitions (transitionA and transitionB) and the steady states of server S_1.

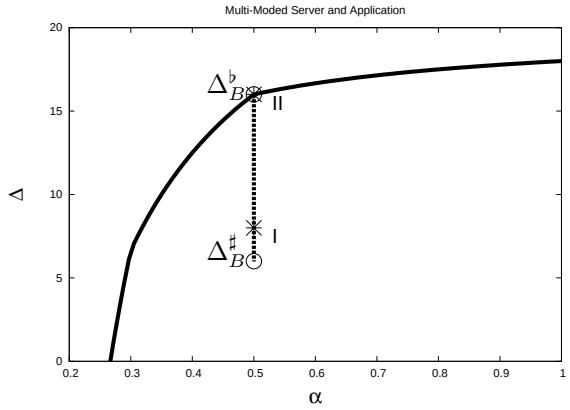


Figure 5.31: Feasibility region in the *slope*, Δ)-space with EDF: bounds to server S_1 parameters in case of transition from mode I, $S^I = (2, 4)$ to mode II, $S^{II} = (4, 8)$. The feasibility region for the delay δ is derived from the maximum Δ^T allowed, represented by the segment. TransitionB is represented.

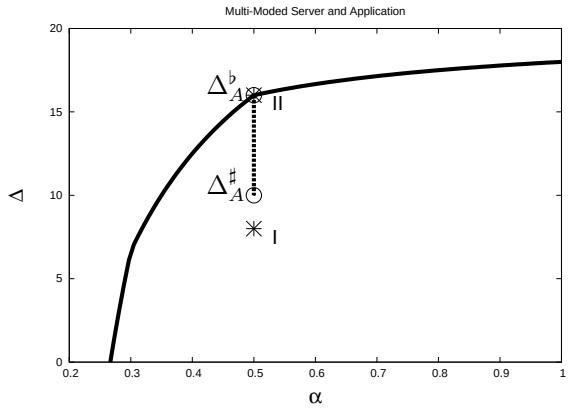


Figure 5.32: Feasibility region in the *slope*, Δ)-space with EDF: bounds to server S_1 parameters in case of transition from mode I, $S^I = (2, 4)$ to mode II, $S^{II} = (4, 8)$. The feasibility region for the delay δ is derived from the maximum Δ^T allowed, represented by the segment. TransitionA is represented.

Chapter 6

Resource Adaptation

Static, expert-intensive approaches to real-time resource allocation have a proven track record for programming embedded real-time control systems. They have a number of drawbacks that make them inappropriate for programming applications in more dynamic real-time environments such as multimedia workstations or home multimedia systems. In particular, the underlying assumptions that the available resources are fixed and known and that only a single application or set of cooperating applications are being run do not hold.

Nowadays, dynamic real-time applications ask for real-time systems that can adapt their behavior at run-time by changing their operating mode. In particular, our scenario assumes that servers, who manages applications by supplying the resource they requires, need to be reconfigured dynamically to adapt the resource reservations and reflect the changes in the system or its environment. Such reconfigurations need to be performed online without jeopardizing schedulability. It is therefore essential to develop appropriate resource reconfiguration criteria and algorithms to manage the criticality of the transition phase.

The limitation in the resource makes it impossible to concurrently execute large task sets in the system. It is therefore desirable to apply strategies that can optimally allocate the computational resource to the most demanding applications. In addition, a static allocation of resources may not be the right solution in many dynamic scenarios. In certain cases it is desirable to manage computational resource in order to dynamically adapt the system behavior to the application which are being executed. For instance, static TDMA-based resource allocation allocates each application by the same resource amount with no privileges for those applications which could require more resources because executing in an high quality (or high consuming) mode. If the resource allocation is adaptively reconfigured at the occurrence of application mode changes, instead, those application will be privileged only when really needed.

As we will show in the rest of the thesis the idea is to develop resource reservation mechanisms that allow system adaptation to changing conditions. What has been investigated so far are *distributed* (see Section 6.2) and *global* (Section 6.1) solutions.

In case of distributed (or local) solutions each server decides when to activate its new version evaluating its proper conditions. The rest of the system and its conditions are not taken into account apart for schedulability conditions. This is equivalent of

having a server monitor (SM) that once a server requires a change, it provides the server with the new set of parameters obtained guaranteeing the schedulability of the whole system. The other server/component of the system remain unchanged. With distributed solutions there are only *active* component, those that request to change. Figure 6.1 depicts a server monitor. In this case the server monitor can apply an acceptance test in order to see if the mode or transition is safe for the whole system and the application the server manages. If it is not, the transition will not not allowed.

Global solutions instead, account for the whole system status and look for a global optimal (or sub-optimal) solution to adapt the resource distribution to the new conditions. The system components can change (event those not directly interested by the change) to let the main change be possible. The system feasibility, even in this case, is the major guarantee with the global solutions. We distinguish among

- *active* components, as those that require the change because their changing conditions;
- *passive* components, as those that are required to change to let the active once change;
- and the *inactive* component, as the components that remains unchanged.

In the future, we will investigate deeper *cooperative* solutions, where more complex SMs will drive the transition by changing passive components in order to let the active once change more promptly. The improvements in the mode transition stage come form the resource that can be reclaimed form the passive components. That is the way we intend cooperative solutions. There will be comparative study among the two possible solutions in order to verify the performances of global and distributed methods. Finally trade off solutions can be required depending on the specific scenarios. In this case the SM will explore more complex transitions (with intermediate steps) to let any king of transition and new mode be reachable.

In this last part of the thesis we outline adaptive resource reservation mechanisms presented as an example of adaptive TDMA server and a global resource distribution mechanism for wireless sensor networks. The solutions proposes two adaptive resource reservation mechanisms which applies to specific scenarios.

6.1 Adaptive Bandwidth Allocation in Wireless Sensor Networks

Wireless Sensor Networks are natural candidates for pervasive monitoring applications. A common scenario consists of tens or hundreds of nodes that monitor environmental phenomena of interest on large and scarcely accessible areas. Generally, WSN nodes are powered by batteries and communicate through limited-bandwidth wireless protocols such as IEEE 802.15.4 (92). Therefore, energy and bandwidth are scarce resources that must be controlled to keep the system operational for as long as it is possible.

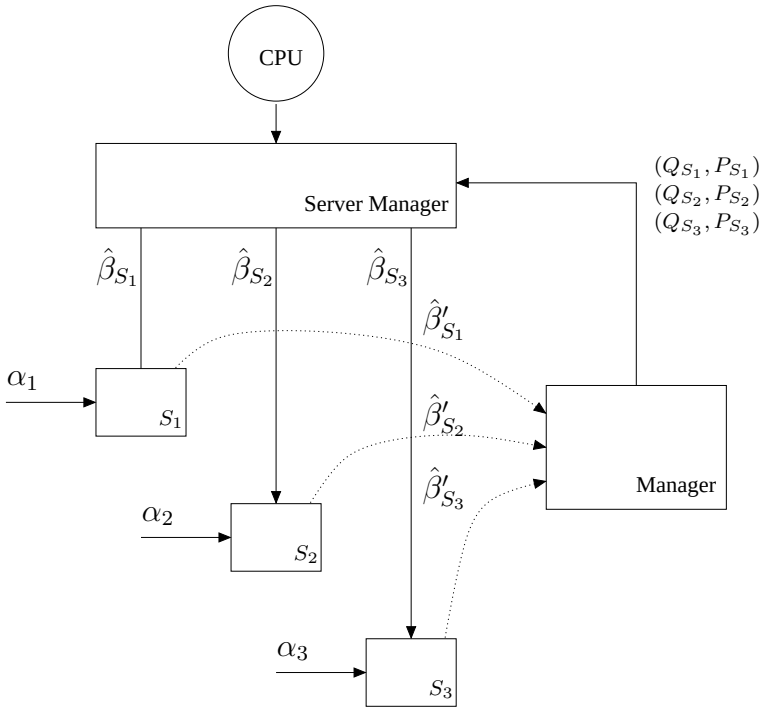


Figure 6.1: Server monitor that manages server mode changes. The feedback from the servers describes the new condition for the servers, in this case the new resource requirements.

Recent advances in technology have made it possible to use WSN for more complex resource demanding applications such as multimedia streaming. In such context, low quality audio and video data are captured by the nodes, elaborated locally and even transmitted through the network. For instance, Kulkarni et al. (91) proposed a multi-tier heterogeneous camera based surveillance network called SensEye; Sensor Andrew (1) is a campus wide infrastructure deployment of heterogeneous WSN motes for sensing and control at Carnegie Mellon University. They make use of CMUcam3 (42) vision sensors for computer vision applications like human tracking.

However, the limitation in bandwidth and energy makes it impossible to concurrently transmit large amount of data in the network. It is therefore desirable to apply strategies that can optimally allocate bandwidth to the most demanding nodes. In addition, a static allocation of bandwidth may not be the right solution in many dynamic scenarios. In certain cases it is desirable to manage energy and bandwidth to dynamically adapt the system behavior to the phenomena which are being monitored. For instance, static TDMA-based transmission allocates each potential transmitter by the same bandwidth amount with no privileges for those nodes close to the occurring phenomena. If the bandwidth allocation is adaptively reconfigured at the occurrence of interesting events, instead, those nodes will be privileged only when really needed. For energy matters the same considerations apply: it is useless to let a device node transmit high data volumes without any event notification content. A node very far from the event can be kept in sleep mode without any disruption of the global appli-

cation whereas the closer nodes must be kept on and transmitting at the maximum resolution. Dynamic nodes exhibit different operative modes where the mode change is forced by the occurrence of events.

With the paper (136) we propose a *component-based* architecture for WSN. The network consists of a set of *components* organized in a hierarchical structure that mimics the underlying cluster-tree network topology. A component can either be a node or a cluster of nodes. Each component is able to automatically reconfigure itself and the underlying nodes upon the occurrence of events, such as detection of an environmental situation of interest, the low-battery level of a node, the insertion of a new component, or the removal of an existing component. We develop reconfiguration algorithms which seek to optimize the overall *quality* level provided by the WSN.

The IEEE 802.15.4 protocol is the most popular standard for WSNs and specifies the Medium Access Control (MAC) sub-layer and the Physical Layer of Low-Rate Wireless Personal Area Networks (LR-WPANs) (92). According to the protocol, a WSN can be operated in the so-called beacon-enabled mode, with the nodes connected in a star topology. In this mode the star makes use of a superframe that enables the real-time capabilities of the standard by using a dynamic TDMA.

Although a lot of work has been carried out in real-time communications over the IEEE 802.15.4, the proposed approaches mainly tackle the problem of point-to-point communication within the star. Relevant examples are: iGAME (89), where the authors propose a network calculus methodology to study the bandwidth allocation problem and provide analysis in case of Round-Robin policy; optimal GTS scheduling (GSA) (122) that try to minimize the total number of unallocated time-slots by applying an EDF-based strategy.

An important issue for WSNs is the capability to cope with reactive paradigms, i.e. change the bandwidth allocation according to the event appearing in the environment. An Adaptive GTS Allocation Scheme has been proposed in (80) to dynamically adapt the bandwidth assigned by the coordinator to other nodes in order to match their actual requirement. This algorithm is focused on fairness and minimum average latency and does not consider any real-time communication constraints. Another example of adaptive systems is BACCARAT (50), where the authors propose an adaptive solution designed to provide real-time constraints and maximize event detection efficiency under variable network load conditions.

Starting from the star topology it is possible to create more complex scenarios. This is the case of the so called cluster-tree topology, where different coordinators interconnect each other in a hierarchical way.

The IEEE 802.15.4 protocol supports the cluster-tree topology, nevertheless it is not clear how to accomplish this in case of beacon-enabled Personal Area Networks (PANs). Among the works addressing this problem we mention (90) where Koubaa et al. present a time division mechanism and propose a methodology to achieve fair distribution of bandwidth. An adaptive beacon scheduling scheme has been proposed by Cho et al. (49), making use of power control and cluster grouping.

Other works moved their focus on the real-time routing problem for the cluster-tree topology. Trdlička et al. (164), gave an optimal solution for the off-line problem.

6.1.1 System Model

In (136) we consider a classical WSN monitoring application with real-time requirements. The system consists of a set of sensor nodes that collects data about events appearing in a large area. The collected data are therefore packed in messages and sent towards a single collection point called sink, with bounded transmission delay.

Among the several possible network configurations adopted in WSNs, we consider hierarchical structure for its flexibility and scalability. At the lowest level we consider star topologies where a *Control Coordinator* (CC) manages either *End-Devices* (EDs) to form a *leaf cluster* or other coordinators at the lower levels; in the latter they are named PAN coordinators. In order to obtain larger scale networks, the star topology can be extended by interconnecting clusters in a hierarchical way, thus creating what is called a *cluster-tree topology*.

In Figure 6.2 the leaves of the tree represent the EDs (nd_i) with sensing capabilities that are monitoring the environment. All the other nodes are CCs C_k, and have the main functions of maintaining the topology and allocating the bandwidth that was assigned by its upper layers to its children in a hierarchical structure. The bandwidth allocated to the leaf nodes will be used to transmit data messages. To simplify the presentation, and without loss of generality, we assume that the CCs are not sensing, thus their assigned bandwidth can be entirely redistributed to the children. This hierarchical topology results in a hierarchical bandwidth allocation problem.

To properly react to external events, the ED may operate in one of M different *operating modes*. Each mode involves a different requirement in terms of energy and bandwidth. The operating mode of an ED is controlled by its CC. In (136) we assume that all nodes execute the same application code. Therefore, all of them can operate in one of the M possible modes.

We denote by Γ_k the k-th leaf cluster; $\Gamma = \{\Gamma_1, \cdots, \Gamma_{LC}\}$ is the set of all leaf clusters, with LC the total number of leaf clusters. Finally, Π is the set of all control coordinators. The notation is illustrated in Table 6.1.

We are addressing the problem of dynamically allocating and reconfiguring the bandwidth assigned to each cluster in the hierarchy. The corresponding software module is part of the MAC layer. We also focus our analysis at the cluster level not addressing the node details. How the bandwidth is distributed among the single nodes and the policies applied within the clusters is left to the next works.

In this case study we consider adaptive resource reservation mechanisms (the communication resource) and the analysis take care of the retrievement and the adaptation of the resource whenever the system conditions change. The transitions are not specifically considered so that the real-time guarantee cannot be explicitly assured, but we have developed methods in order to keep the transition interval at the minimum possible. This reduces the effects of the transition on the predictability of the system. Such a solution can still offer real-time guarantees in case of systems with large latency such as the wireless sensor networks with message communication.

To mention that, in (50) we have also addressed the transition problems with bandwidth allocations in case of WSNs. In that situation (at a lower abstraction level than (136)) the real-time constraints are guaranteed during any phase of the system.

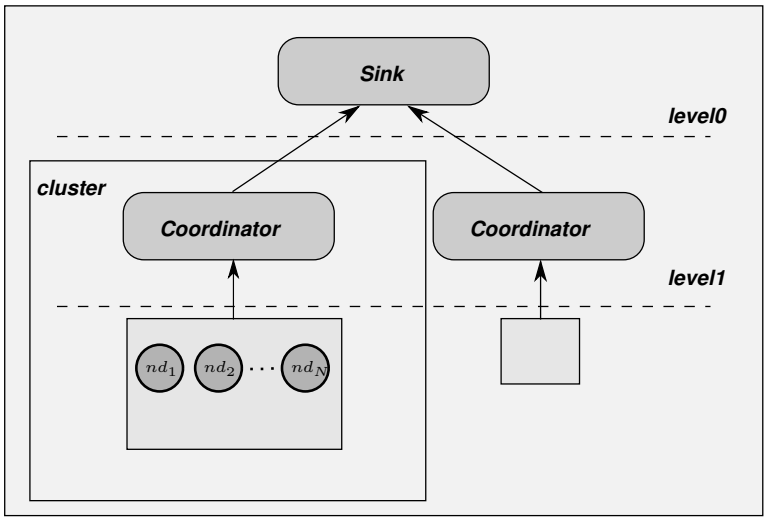


Figure 6.2: WSN hierarchical architecture with end devices, control coordinators and clusters elements.

6.1.2 Real-Time Components

The network nodes can be seen as basic computational units that implement services with temporal requirements. In the same way, clusters are aggregation of units that can be seen as a unit. Therefore, it is natural to apply a component-based approach to the network model.

In our scenario, the network components implement functional services with temporal requirements, expressed by a *Real-Time Interface* (RTI) (57; 74). Figure 6.3 depicts the component and the component interface abstraction. To model RTI we make use of an approach similar to the Real-Time Calculus (RTC) (159): a Real-Time Interface of a generic network component has input and output variables related to event streams (arrivals) and resource availability (services).

Arrival Curves. The RTC describes the trace of an event stream by means of a cumulative function $R(t)$, defined as the workload amount of the event stream in the time interval $[0, t)$. While R describes a concrete trace of an event stream, a tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ provides an abstract event stream model of bounds on admissible traces of an event stream. The upper arrival curve $\alpha^u(\Delta)$ is the upper bound on the workload of events monitored in any time interval of length Δ, and the lower arrival curve $\alpha^l(\Delta)$ is the lower bound. We refer to (159) for a detailed discussion.

The concept of arrival curve $\alpha(\Delta)$, unifies many common timing models of event streams. In our case α models the transmission event stream of each ED node, where the workload is intended as the payload of each transmission event. The timing requirements D of such a transmission arrival curve is the maximum delay affordable by a transmission.

Service Curves. the concrete resource availability can be described by a cumulative function $S(t)$, that is defined as the number of available resources, e.g., processor cycles or bus capacity, in the time interval $[0, t)$. A tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves provides an abstract resource model. β^u and β^l are

C_i	Control Coordinator node
nd_i	End Device node
Γ_k	k-th leaf cluster
Γ	set of all the leaf clusters
Π	set of all control coordinators
α	component transmission arrival curve
α^d	component resource demand curve
β	component service curve
μ_j	j-th operating mode of an ED
E_k	cluster k energy level
$E_{k,j}$	cluster k mode j energy level
q_{nd_i}	ED nd_i quality level
q_j	mode j quality level
$x_{k,j}$	number of nodes of cluster k in mode μ_j
N_k	total number of nodes of cluster k
M	number of modes per node
LC	number of leaf clusters

Table 6.1: System notation

respectively the upper and lower bound to the available resources in any time interval of length Δ.

The service curve abstraction $\beta(\Delta)$ models any possible resource supply in the interval domain, including the bandwidth provisioning by the control coordinators in WSNs.

Real-Time Calculus throughout transmission arrival and service curves is able to model the classical WSN non-determinism by providing bounds to the transmission event stream and the service provisioning. By guaranteeing the bound the analysis is feasible for all the possible value within the bounds themselves.

ParagraphReal-Time Interfaces The RTIs, as applied in (136), are special instances of assume/guarantee interfaces tailored towards guarantees on the delays of events (57; 161; 169). In our framework, an ED nd_i of the k-th cluster is described by an input/output interface which specifies in input (α_{n_i}, D_{nd_i}) as the node transmission rate and its timing requirement, and $\beta^A_{nd_i}$ to define the node bandwidth request as the service output interface to other components. Leaf-cluster CC C_k, together with upper layers coordinators, have $\beta^A_{C_k}$ and $(\beta_{C_k}, \beta^G_{C_k})$, with β_{C_k} the actual bandwidth provided and $\beta^G_{C_k}$ the guaranteed bandwidth level, as the input and output interfaces respectively. A leaf-cluster coordinator is connected with many nodes; $\beta^A_{C_k}$ and $(\beta_{C_k,nd_i}, \beta^G_{C_k,nd_i})$ are the interfaces of C_k respectively to upper layer coordinators and nd_i. Figure 6.3 shows the abstraction of a generic component and its real-time interface where time requirements are translated into bandwidth requirements.

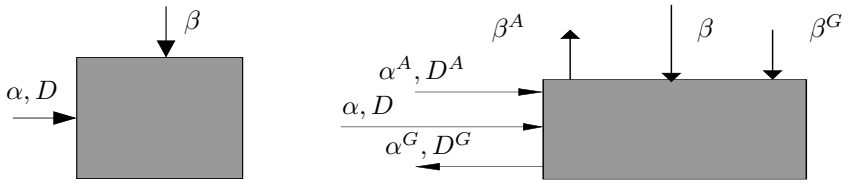


Figure 6.3: Generic (ED or CC) real-time component and its interface abstraction.

6.1.2.1 Component Model

In the component-based WSN system we are modeling, the components are the ED nodes and the CC nodes. The parameters describing the i-th ED nd_i are

- the energy E_{nd_i}, that measures the actual energy level of the node, e.g. transmission energy or other models;
- the quality q_{nd_i}, describing a quality index of the node services, e.g., transmission quality or other metrics; this is a scalar number, for example in interval $[0; 100]$, with 0 denoting the minimum and 100 the maximum satisfaction for the end-user;
- α_{nd_i} and D_{nd_i} which is the transmission arrival curve and its maximum delay;
- β_{nd_i}, as the allocated bandwidth for transmission.

The component model is affected by the actual ED component operative mode. Thus, the node nd_i at its j-th mode μ_j has

$$\begin{aligned}
\alpha_{nd_i}(\mu_j) = \alpha_{nd_{i,j}} &= (\alpha^u_{nd_i,j}, \alpha^l_{nd_i,j}) \\
\beta_{nd_i}(\mu_j) = \beta_{nd_i,j} &= (\beta^u_{nd_i,j}, \beta^l_{nd_i,j}) \\
E_{nd_i}(\mu_j) &= E_{nd_i,j} \\
q_{nd_i}(\mu_j) &= q_{nd_i,j},
\end{aligned}$$

with the resulting node model $(nd_i, j) = (E_{nd_i,j}, q_{nd_i,j}, \alpha_{nd_i,j}, \beta_{nd_i,j})$ as a function of the mode μ_j.

A reasonable WSN multi-mode problem could be the one in which the ED can operate in 3 different modes: μ_3 corresponds to a low quality and energy consumption operating mode; μ_2 as medium-quality working mode which means medium rate sensing, transmission and energy consumption; μ_1 for high-quality node functionality which means high transmission rate and consequently high energy consumption.

$$\begin{aligned}
q(\mu_1) \geq q(\mu_2) \geq q(\mu_3) \\
E(\mu_1) \geq E(\mu_2) \geq E(\mu_3) \\
\alpha(\mu_1) \geq \alpha(\mu_2) \geq \alpha(\mu_3) \\
\beta(\mu_1) \geq \beta(\mu_2) \geq \beta(\mu_3).
\end{aligned}$$

The CC model is simpler than the ED one and does not depend on the component mode. The k-th control coordinator component C_k is described by the bandwidth it receives from the upper layer β_k, and the one it passes to the lower level components $\{\beta_{k,v} : \forall v \in$ all the CC nodes composing the k-th cluster$\}$.

6.1.2.2 Composability Criteria

In real-time systems, the composability of temporal interfaces is equivalent to the classical feasibility concept: two real-time components are composable if the time requirements of the two components are guaranteed after their composition.

Node composability A sensing node nd_i with an transmission event stream requires a certain amount of bandwidth $\beta^A_{nd_i}$ to transmit the output messages on time. The minimum resource amount required to satisfy the node transmission request is $\alpha^d_{nd_i} \stackrel{def}{=} \alpha_{nd_i}(\Delta - D_{nd_i}) = \beta^A_{nd_i}$, which is also the minimum feasible stream transmission rate which represents also the minimum resource amount required to satisfy the node transmission request. $\beta^A_{nd_i}$ is the resource demand of node nd_i. The node nd_i receives $\beta_{nd_i}(\Delta)$ bandwidth amount from its coordinator C_k: $\beta_{nd_i}(\Delta) = \beta_{C_k,nd_i}$.

Lemma 6.1.1 (Node Composability Criteria)**.** *A generic node nd_i is composable to a generic coordinator C_k if*

$$\forall\ \Delta \quad \beta^A_{nd_i}(\Delta) \leq \beta^G_{C_k,nd_i}(\Delta)$$

Proof. The proof is a straightforward application of the composability definition by Wandeler et al. (160). □

The composability condition of Lemma 6.1.1 guarantees the components service compliance together with the transmission event deadlines.

In the same way, it is possible to state a composability criteria for the coordinators.

Lemma 6.1.2 (Coordinator Composability Criteria)**.** *A control coordinator C_g is composable to an upper level control coordinator C_k if*

$$\forall\ \Delta \quad \beta^A_{C_g}(\Delta) \leq \beta^G_{C_k}(\Delta)$$

Proof. The demonstration comes from the former Lemma 6.1.1 and the component composability definition (160) for RTIs. □

The bandwidth composability guarantees the real-time properties of the components after their composition.

Figure 6.4 shows an example of interface composition of components with hierarchical architecture.

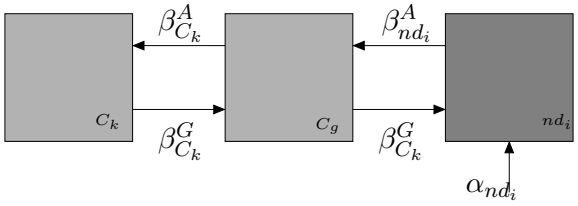


Figure 6.4: Interface composability between a node nd_i, its coordinator C_g and the coordinator of the coordinator C_k.

6.1.3 Optimization Problem

The problem of allocating bandwidth to the clusters and assigning a mode to each ED can be modeled as an optimization problem. The goal is to maximize a global *quality index* that depends on the operating modes of the ED. Constraints are on energy and on the bandwidth requirements of the components, as described in Section 6.1.2.2.

An optimization problem maximizing a generic quality index in a WSN has to consider both topological and functional aspects of the application to be executed. These aspects are coded in the optimization problem as a set of constraints. In our case, we consider *dynamic constraints*: whenever the system has to react to an external event, the set of constraints needs to be updated due to parameter changes (energy and quality); a new problem must be solved to find a solution consistent with the new conditions.

In terms of notation, we define $x_{k,j}$ as the number of nodes of cluster k operating in the j-th mode, $M = 3$ as the number of possible modes; $\overline{x}_k = [x_{k,1}, x_{k,2}, x_{k,3}]$ describes the distribution of the nodes of the cluster among the modes. N_k is the number of nodes of the cluster k and β_k as the bandwidth received by the k-th cluster. We assume also that each cluster has the same quality index $\overline{q} = [q_1, q_2, q_3]$, with q_j the quality associated to the j-th mode.

Our proposal formulates the bandwidth allocation and mode assignment as two separate single-objective optimization problems, instead of a single problem with two objectives.

6.1.3.1 Bandwidth allocation

First, we start with the bandwidth allocation problem that considers the part of the system topology without the EDs.

We introduce the concept of *cluster weight* to calculate the proportion of bandwidth to allocate, thus accommodating design preferences based on the geographic location of the clusters. This results in the cost function

$$\overline{c} \cdot \overline{\beta}^T, \tag{6.1}$$

with each component c_k of $\overline{c} = [c_1, \ldots, c_N]$ being the weight of cluster k and $\overline{\beta} = [\beta_1, \ldots, \beta_N]$. By $(\ )^T$ it is intended the transposition operation; in particular it applies to arrays and transposes a row array into a column array.

Constraints. At the coordinator level enforcing the network topology requires the incoming bandwidth for each cluster to be the sum of the incoming bandwidths of the child clusters/nodes. Given a cluster k, the *topology constraint* is

$$\forall \Delta \ \sum_{g=1}^{N_k} \beta_{k,g}(\Delta) \leq \beta_k(\Delta). \tag{6.2}$$

In order to operate all the nodes of the k^{th}-leaf cluster, it demands a minimum bandwidth requirement of $N_k \alpha^d_{k,3}$ when all of its nodes are in lowest operating mode μ_3. On the other hand, the maximum bandwidth demand of the k^{th}-leaf cluster is bounded by $N_k \alpha_{k,1}$ where all the nodes operate in the highest operating mode μ_1. As a remainder, $\alpha^d_{k,3}$ and $\alpha^d_{k,1}$ denote respectively the low-mode and high-mode bandwidth resource demand of any ED node of the k-th leaf cluster, assuming the node bandwidth requests equal for any node within the cluster. This second topology constraint has to bound the leaf cluster bandwidth to the minimum and maximum bandwidth amount requested by the ED nodes composing the cluster

$$\forall k \ \in \ \Gamma \quad N_k \alpha^d_{k,3} \leq b_k \leq N_k \alpha^d_{k,1}. \tag{6.3}$$

6.1.3.2 Mode Assignment

In order to seek for the optimal mode assignment at any leaf cluster, the bandwidth assigned to the leaf cluster has to be known first.

At the leaf cluster level we associate to each mode a quality, and we want to maximize the number of nodes in the higher quality modes, so our value function is:

$$\overline{q} \cdot \overline{x}^T. \tag{6.4}$$

Constraints. For any leaf cluster, the *topology constraints* define the number of EDs per cluster and enforce a feasible solution for the hierarchical topology of the cluster itself. For the k-th leaf cluster, the number of nodes is given by the topology of the network, so that the sum of the components of $\overline{x}$ is constrained to be equal to the number of nodes in the cluster:

$$\sum_{j=1}^{M} x_{k,j} = N_k. \tag{6.5}$$

Since an ED component transmits its information upwards in the logical network topology, it requires bandwidth from its CC. The *bandwidth constraints* describe within the leaf cluster, the dependency among the bandwidth request and availability as:

$$\forall \Delta \ \overline{\alpha}^d_k(\Delta) \cdot \overline{x}^T_k \leq \beta_k(\Delta), \tag{6.6}$$

where for a generic leaf cluster k each mode j has its proper transmission demand curve $\alpha^d_{k,j}$, and β_k is the bandwidth provided by the cluster coordinator; $\overline{\alpha^d}_k = [\alpha^d_{k,1}, \alpha^d_{k,2}, \alpha^d_{k,3}]$.

To take into account the energy consumption of the EDs and to control the energy assigned to each cluster we introduce an additional constraint:

$$\overline{E}_k \cdot \overline{x}_k^T \leq E_k, \tag{6.7}$$

where $\overline{E}_k = [E_{k,1}, E_{k,2}, E_{k,3}]$ models the energy consumption per mode of all the nodes and E_k is the maximum amount of energy that the k-th cluster is allowed to consume.

6.1.3.3 Linearization

Using α^d and β curves as constraints, requires a substantial amount of computation to solve a non-linear optimization problem. In order to obtain a linear problem we approximate Constraints (6.2), (6.3), and (6.6) using linear bounds to the arrival and service curves.

In case of arrival curves, the bounds are $a^*\Delta \leq \alpha^d(\Delta) \leq a^\sharp\Delta$; and $b^*\Delta \leq \beta(\Delta) \leq b^\sharp\Delta$ for the service curve. In a conservative approximation, we use $a\Delta$ with $a = a^\sharp$ to bound the transmission bandwidth requirement α^d, and $b\Delta$ with $b = b^*$ or $b^\sharp$ to bound service curves. In particular, b^* is applied to approximate the β at the right hand side of Equation 6.2, while $b^\sharp$ is applied to the left hand side βs.

The linearized bandwidth constraints are $\overline{a} \cdot \overline{x}^T \leq b$, and $\sum_g^{N_k} b_{k,g} \leq b_k$.

The solution obtained with the simplifications above becomes an approximated one, but makes the problem solvable with standard linear optimization techniques. See (68) for a discussion on the loss introduced by this simplification.

Referring to the topology notation in Section 6.1.1 the bandwidth allocation optimization problem, with the bandwidth constraint linearization becomes:

$$\begin{aligned} &\max\ \overline{c} \cdot \overline{b}^T \\ &\text{subject to} \\ \forall h\ \in\ \Pi\quad &\sum_{g=1}^{N_h} b_{h,g} \leq b_h. \\ \forall k\ \in\ \Gamma\quad &N_k a_{k,3} \leq b_k \leq N_k a_{k,1} \end{aligned} \tag{6.8}$$

The linearized mode assignment optimization problem $\forall\ \Gamma_k \in \Gamma$ is

$$\begin{aligned} &\max\ \overline{q}_k \cdot \overline{x}_k^T \\ &\text{subject to} \\ &\sum_{j=1}^{M} x_{k,j} = N_k \\ &\overline{a}_k \cdot \overline{x}_k^T \leq b_k \\ &\overline{E}_k \cdot \overline{x}_k^T \leq E_k. \end{aligned} \tag{6.9}$$

We first solve the global bandwidth allocation problem (6.8), then we solve the mode assignment problem (6.9) for every leaf cluster.

The flexibility obtained by decoupling the problem into two sequential optimization problems allows to model an increased set of WSN applications. Nevertheless, our approach can be easily extended by applying other objective functions taking into account different WSN conditions.

6.1.4 Optimization Algorithms

We broadly classify the algorithms to find a solution to the problem into design (off-line) and execution (on-line) algorithms.

1. The *off-line* or *design stage* algorithm computes an optimal initial condition for the WSN system; the initial mode per each node is obtained by using the simplex algorithm on the optimization problem. The initial problem at the design stage is not constrained by energy or quality requirements, since we assume that at the beginning all the nodes have the same battery level and quality index.

2. The *on-line* or execution stage algorithm is employed upon occurrence of any dynamic event: whenever there is an occurrence of an event at a location monitored by a leaf cluster, it could trigger a change in the quality requirement of the nodes attached to that cluster, thus asking for a change in the demanded bandwidth. The on-line problem has to quickly provide a sub-optimal feasible solution allowing the WSN to promptly change mode within a short delay.

6.1.4.1 Off-line Optimization

The off-line solution for mode assignment is solved using the two single objective functions, one for the bandwidth allocation and the other for mode assignment as explained in Section 6.1.3. The Global Mode Assignment (GMA) algorithm is based on the linearized equations of the optimization model and it is described below.

Algorithm 9 Global Mode Assignment (GMA) Algorithm: off-line optimal feasible configuration assignment of the WSN

Input: System as nodes architectures, node modes, node constraints.
Output: Run-time initial feasible configuration of the WSN: $(\overline{x}_k^0, b_k^0)\ \forall \Gamma_k \in \Gamma$.

```
1: Solve bandwidth allocation optimization problem (6.8).
2: b̄^0 = [b_1^0, ··· , b_LC^0].
3: for Γ_k ∈ Γ do
4:     Solve mode assignment optimization problem (6.9) for Γ_k.
5:     x̄_i^0.
6: end for
```

The solution is obtained with the help of the simplex algorithm. The result of the GMA algorithm is used as an initial configuration to set up the WSN. At this stage, all the leaf clusters have equal importance and the bandwidth assignment is based on the number of nodes within each cluster which determines the lower and upper bound for the incoming bandwidth.

6.1.4.2 On-line Problem

Since it is not feasible to run the optimal bandwidth and mode assignment problem on-line, we have developed algorithms to quickly achieve feasible sub-optimal solutions to the optimization problem.

We consider two types of transitions that can trigger the request for on-line optimization. The first transition is the occurrence of a physical event that increases the importance of that specific leaf cluster. The second transition is due to a decrease/increase in the available energy, which forces the nodes to work in a low operating mode, thus consuming less energy. In both cases, the event can request a change to a higher quality mode (which requires more bandwidth), or a change to a lower quality mode (which frees some bandwidth).

We propose two on-line algorithms for bandwidth reallocation and two on-line algorithms for mode reassignment. We start by describing the bandwidth reallocation algorithms.

6.1.4.3 Bandwidth Re-Allocation Algorithms

The on-line algorithm is invoked by occurrence of an external event. In this case, the system has to decide the importance of the event and accordingly increase the weight (which can also be considered as priority) assigned to that leaf cluster. Depending on the coverage of the search for bandwidth reclamation we propose two algorithms.

The **Local Bandwidth Re-Allocation Algorithm (LBRA)** tries to reallocate the bandwidth among the sibling clusters according to the new weight assigned to each of its child cluster. The weight assigned to the child cluster determines the fraction of the parent's bandwidth assigned to that particular cluster. Since the bandwidth is reclaimed locally within the sibling clusters, this is a local reclaiming algorithm. The complexity of the search is reduced at the expense of a potential loss of overall system quality.

The **Global Greedy Bandwidth Reclaiming Algorithm (GGBRA)** is an on-line global bandwidth search algorithm. Any occurrence of an external event triggers the algorithm, which in turn updates the weight assigned to each child cluster. The algorithm considers weight as a cluster importance. Instead of reallocating the bandwidth among the sibling clusters, this algorithm first tries to update the mode for a certain number of nodes.

The event can trigger a request for a higher quality mode or for a lower quality mode. The algorithm has to find a way to obtain more bandwidth from the sibling clusters. If it is not possible to do so, the algorithm goes up in the hierarchy triggering a request for more bandwidth to the upper layers.

6.1.4.4 Mode Re-assignment Algorithms (MRA)

The mode reassignment algorithms try to update the modes of the nodes attached to those specific leaf clusters which were affected due to on-line bandwidth reclamation algorithms. In case the cluster gains more bandwidth due to higher importance, the nodes must be moved to higher quality mode and vice-versa. The mode assignment

Algorithm 10 Local Bandwidth Re-Allocation Algorithm (LBRA)

Input: System: parent cluster, leaf cluster.
Output: Bandwidth Re-allocation β_k $\forall k \in \Pi$, Mode Re-assignment.

```
if Energy drops below certain threshold then
    Change the nodes to lower mode
    Calculate available bandwidth at the parent cluster
    Propagate the available Bandwidth and buffer it at the PAN Coordinator
else if Occurrence of an important event and buffered bandwidth available at the PAN Coordinator then
    Increase the weight and priority of the leaf cluster
    Calculate the required energy to move the nodes to higher mode
    Reclaim the buffered energy from the PAN Coordinator
else if Occurrence of an important event and no buffered bandwidth at the PAN Coordinator then
    Increase the weight of the leaf cluster
    Divide b_k among the sibling clusters such that it satisfies constraints (6.2), (6.3).
    Assign new b_k.
end if
Call Mode Re-assignment Algorithm - MRA
```

during the off-line configuration was managed by solving the simplex algorithm at every leaf cluster. Due to the high computation complexity of simplex algorithm we propose two alternative on-line algorithms for this purpose.

This problem is similar to a bin packing problem. In this case, we need to distribute b_k bandwidth among N_k nodes of a leaf cluster Γ_k in order to achieve the maximum quality. Since the total system quality depends on the local quality, we solve this problem as a local problem. However, an optimal solution to this local mode assignment problem does not mean that we have a global optimal mode assignment. We show this fact in Section 6.1.6 by comparing the total quality achieved by this algorithm with the GMA algorithm.

The **GReedy Mode Assignment Algorithm (GRMA)** is based on the greedy solution similar to the bin packing problem. We first try to allocate as many nodes as possible in the higher quality mode, such that the required bandwidth is less than the reallocated bandwidth. Any remaining bandwidth is then assigned to lower quality modes in a greedy manner.

The **Optimal Hyper-plane based Mode Assignment Algorithm (OHMA)** provides an optimal solution to the local mode assignment problem. Instead of solving the simplex algorithm, this algorithm considers the intersection of the total number of nodes constraint (6.10), with the available bandwidth, the energy constraints (6.11) and (6.12) respectively. The intersection is a hyper-plane, which in case of three-dimensional (three modes) problem is a line segment, and the solution is one of the two end points of the intersecting line segment as shown in Figure 6.5. The algorithm

Algorithm 11 Global Greedy Bandwidth Reclaiming Algorithm (GGBRA)

Input: System: parent cluster, leaf cluster.
Output: Bandwidth Re-allocation. $b_k \ \forall \Gamma_k \in \Gamma$.
Output: Mode Re-assignment.

```
1: if Energy drops below certain threshold then
2:     Change the nodes to lower mode
3:     Calculate available bandwidth at the parent cluster
4:     Propagate the available Bandwidth and buffer it at the PAN Coordinator
5: else if Occurrence of an important event and buffered bandwidth available at the
   PAN Coordinator then
6:     Increase the weight and priority of the leaf cluster
7:     Calculate the required energy to move the nodes to higher mode
8:     Reclaim the buffered energy from the PAN Coordinator
9: else if Occurrence of an important event and no buffered bandwidth at the PAN
   Coordinator then
10:    Increase the priority of the leaf cluster
11:    Calculate requiredb required to move n nodes to higher quality mode.
12:    while parentNode ≠ PAN do
13:        for Γ_k ∈ Γ & reclaimed b_k ≤ required b_k do
14:            if then i nodes in high quality modes available
15:                move i nodes to lower quality mode.
16:                Calculate reclaimed b_k
17:            end if
18:        end for
19:    end while
20: end if
21: if reclaimed b_k ≥ required b_k then
22:    Move n nodes of leaf cluster to higher mode.
23: end if
24: Call MRA
```

selects one of the two vertices depending on the quality vector, which would point towards higher total quality vertex.

The constraints of each three-dimensional problem (local or composed one) are

$$x_1 + x_2 + x_3 = N \tag{6.10}$$

$$E_1 x_1 + E_2 x_2 + E_3 x_3 \leq E \tag{6.11}$$

$$a_1 x_1 + a_2 x_2 + a_3 x_3 \leq b. \tag{6.12}$$

In particular, equation $x_1 + x_2 + x_3 = N$ states that the solution has to be a point on that plane, while $E_1 x_1 + E_2 x_2 + E_3 x_3 \leq E$ represents an upper bound to the bandwidth constraint $a_1 x_1 + a_2 x_2 + a_3 x_3 \leq b$ as detailed by Figure 6.5.

The intersections among the constraints is a set of vertexes v where the optimal solution can be found. In v the solution vertex has to be chosen by comparing the cost

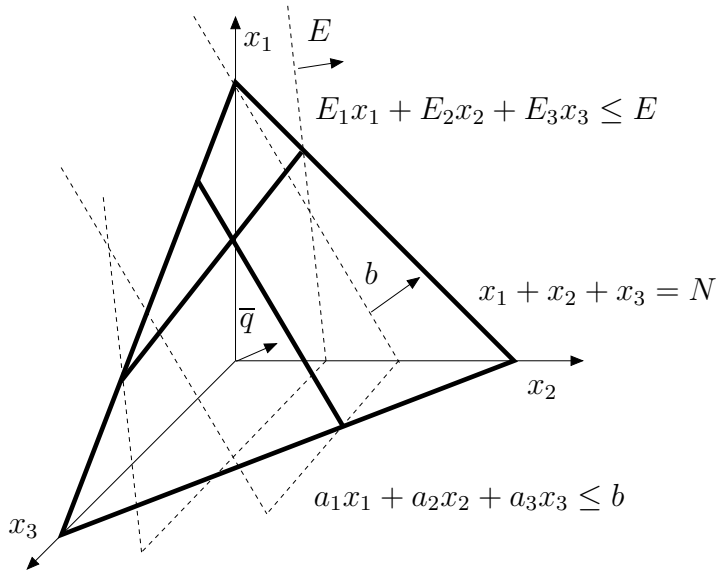


Figure 6.5: Cluster optimization problem: hyper-planes representation.

function $\overline{q} \cdot \overline{x}^T$ computed for each of the vertexes. The vertex with the maximum cost function is the optimal solution and (x_1, x_2, x_3) are the optimal mode assignments for the cluster.

As an example Figure 6.6 shows the case with the $x_1 + x_2 + x_3 = N$ and $a_1x_1 + a_2x_2 + a_3x_3 \leq b$ constraints. The intersection among them and the planes $x_1 = 0$, $x_2 = 0$ and $x_3 = 0$ ($x_1,\ x_2, x_3\ \geq 0$ are implicit constraints of the problem) results in 3 vertices

$$P_1(x_1 = 0) \begin{cases} x_1 = 0 \\ x_2 = N - \frac{b - Na_2}{a_3 - a_2} \\ x_3 = \frac{b - Na_2}{a_3 - a_2} \end{cases}$$

$$P_2(x_2 = 0) \begin{cases} x_1 = N - \frac{b - Na_1}{a_3 - a_1} \\ x_2 = 0 \\ x_3 = \frac{b - Na_1}{a_3 - a_1} \end{cases}$$

$$P_3(x_3 = 0) \begin{cases} x_1 = N - \frac{b - Na_1}{a_2 - a_1} \\ x_2 = \frac{b - Na_1}{a_2 - a_1} \\ x_3 = 0 \end{cases}$$

The set of vertexes v is

$$v = (v_1, v_2, v_3) = (P_1, P_2, P_3).$$

In v there are the solution candidates from which extract the optimal one.

We have shown that by the analysis of the solution space together with the vector directions it is possible to express the admissible solutions of the leaf cluster optimization problem in a closed formula and apply it on-line.

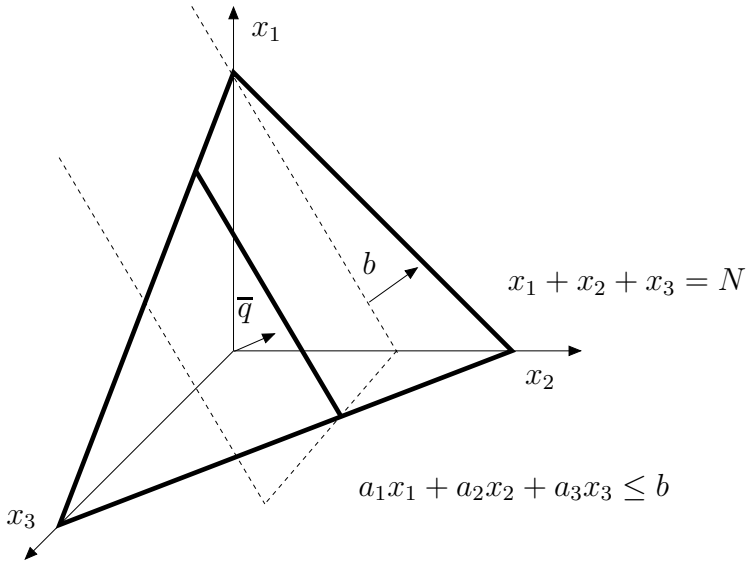


Figure 6.6: Optimal hyper-plane-based mode assignment.

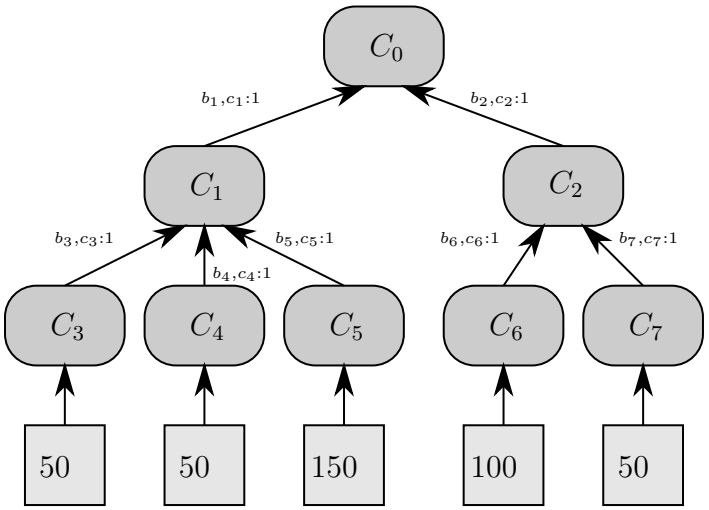
6.1.5 Example

In order to explain the bandwidth allocation and mode assignment with respect to different algorithms, we exemplify a simple dynamic topology reactive to physical events. This configuration is set up for the WSN topology as shown in Table 6.2. The leaf node C_3 to C_7 are the coordinators of WSN which communicate with the EDs responsible for monitoring the physical events. In order to assign the initial bandwidth allocation and modes, we run the off-line GMA algorithm. At this stage we consider that all the clusters C_k have equal importance and set the weights c_k equal to one. In this case study we have μ_1 as a high quality mode and μ_2 , μ_3 as medium and low quality operation mode respectively. The $\overline{\alpha}$ [4 2 1], $\overline{E}$ [4 2 1] and $\overline{q}$ [8 4 1] coefficients are set according to the mode quality.

Step 1: off-line optimization. We allocate an initial root bandwidth of 800 units for the entire system and run the off-line GMA algorithm. Every leaf cluster is assigned a maximum energy of 8000 units so that we can ignore the bandwidth and mode constraints arising due to limited energy.

The results of GMA are shown in Table 6.2. The total quality for this configuration, i.e $\sum_{\Gamma_k \in \Gamma} c_k \overline{q} \cdot \overline{x}_k^T$, is 1429.

Step 2 (GGBRA Algorithm). The configuration obtained by GMA algorithm is deployed in a physical environment with mode settings and network bandwidth values set according to the results obtained in Step 1. At this stage, there is an important event at leaf cluster 3, which triggers a change in the system state. According to the importance of the event, we increment the weight assigned to cluster 3 by one: $c_3 = c_3 + 1 = 2$. The system either initiates the GGBRA algorithm or LBRA algorithm. Table 6.3 shows the change in configuration, i.e bandwidth reallocation and mode reassignment, according to GGBRA.



$Mode$	C_3	C_4	C_5	C_6	C_7
μ_1	16	16	41	0	0
μ_2	33	33	108	0	0
μ_3	0	0	0	100	49

b_1	b_2	b_3	b_4	b_5	b_6	b_7
649	150	133	133	383	100	50

Table 6.2: Offline GMA Algorithm; initial weights $c_k = 1$

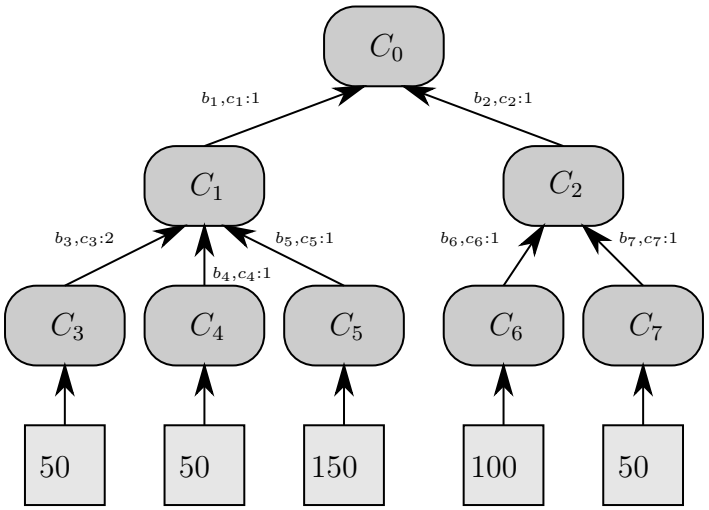
The GGBRA moves 10 nodes in C_3 from μ_2 to μ_1. This is because C_3 has gained more importance due to the occurrence of an external physical event. The GGBRA reclaims this bandwidth from cluster C_4 which is a sibling cluster with lower weight. Since it was possible to reclaim the required bandwidth from a sibling cluster, the GGBRA exited without moving up in the topology and searching for more complex. The total quality for this configuration is 1718. This increase in quality with respect to the initial total quality reflects the fact that a cluster with more importance has received more bandwidth.

Step 3 (LBRA Algorithm).

Similar to GGBRA, LBRA modifies the weight assigned to C_3, $c_3 = 2$. Now LBRA reallocates the bandwidth among the sibling clusters according to its weight. Then, it executes the mode reassignment algorithm to reassign the modes according to the new bandwidth. Since, this algorithm is local, the reconfiguration only affects the sibling clusters, whereas the other cluster components within the topology remains unaffected by this change. The result of LBRA algorithm is shown in Table 6.5. The bandwidth reallocation moves more bandwidth to cluster C_3 by reclaiming it from C_4 and C_5. The total quality as a result of LBRA algorithm is 1752 which is greater than the total quality achieved by GGBRA algorithm.

Step 4 (GMA Algorithm)

Only for comparison, we show the result of GMA algorithm by assigning the same weight assigned to C_3 as in case of LBRA and GGBRA algorithm. Table 6.5 shows



$Mode$	C_3	C_4	C_5	C_6	C_7
μ_1	26	16	41	0	0
μ_2	23	16	108	0	0
μ_3	0	17	0	100	49

b_1	b_2	b_3	b_4	b_5	b_6	b_7
649	150	150	116	383	100	50

Table 6.3: Online GGBRA Algorithm as a result of an event at C_3

$Mode$	C_3	C_4	C_5	C_6	C_7
μ_1	49	16	41	0	0
μ_2	0	33	41	0	0
μ_3	0	0	67	100	49

b_1	b_2	b_3	b_4	b_5	b_6	b_7
649	150	200	133	316	100	50

Table 6.4: Online LBRA Algorithm as a result of an event at C_3

this bandwidth reallocation and mode reassignment with off-line GMA algorithm at the instance of physical event at cluster C_3.

$Mode$	C_3	C_4	C_5	C_6	C_7
μ_1	49	12	11	0	0
μ_2	0	36	138	0	0
μ_3	0	0	0	100	49

b_1	b_2	b_3	b_4	b_5	b_6	b_7
649	150	200	133	316	100	50

Table 6.5: Offline GMA Algorithm as a result of an event at C_3

The GMA algorithm runs an off-line simplex algorithm which globally re-distributes the bandwidth in order to get an optimal result. In this case the bandwidth is re-

distributed among the sibling clusters of C_3 without effecting other clusters, due to the effect that C_6 and C_7, which have lower importance, are already having all its nodes in lowest operating mode. The total quality obtained by GMA is 1813 which is higher than the quality obtained by the on-line algorithms.

6.1.6 Simulations

In this section we describe the performance of different algorithms with respect to the total quality of the WSN. By the total quality we mean the aggregate of number of nodes in each mode for every leaf cluster Γ_k. $\sum_{\Gamma_k \in \Gamma} c_k \overline{q} \cdot \overline{x}_k^T$ where c_k is the importance of the k-th cluster. The total quality function is quite general and with its parameters allows to model most of the quality function that real applications want to pursue.

The simulation study was performed in MATLAB. A topology data structure was modeled in a matrix form to hold the WSN topology. The topology structure included the coordinator to end-device relationships and the number of end devices assigned to each subcluster. The structure maintained the energy levels, weights and quality parameters for the leaf clusters. The global optimization algorithm for bandwidth and mode assignment was developed to dimension the WSN topology for different available bandwidth, energy and total number of end devices. These initial results were then used by online algorithms described in Section 6.1.4 to run the simulation and study its effect on the total system quality by redistributing the assigned bandwidth and modes according to an occurrence of a simulated event.

We show the analysis for a fixed topology as described in Section 6.1.5. We compare the online algorithms with the global optimal algorithm as described in Section 6.1.4. The graph in Figure 6.7 compares all three algorithms against the total number of sensing nodes in the WSN. We keep the energy assigned to each cluster and total bandwidth of the network fixed. The energy value is selected such that it does not affect the bandwidth constraints, i.e, all the nodes can operate in maximum mode with sufficient available energy. In each of three algorithms the total quality increases with the number of nodes up to a certain point. This is because the total available bandwidth is more than sufficient for all the nodes to operate in the highest mode and we have maximum quality. There after the mode selection depends on the applied algorithms up to a certain point. In this range, it can be seen from the figure that the global optimal algorithm always gives better total quality compared to the on-line algorithms. After a saturation point, increasing the total number of nodes does not increase the total quality. This is due to the fact that our analysis helps us to determine the maximum number of nodes with which we can guarantee a minimum bandwidth to every node within the network.

Figure 6.8 shows the comparison among the total network quality values obtained with different algorithms. The total bandwidth and the number of nodes in the system is fixed; we can see how increasing energy allows nodes to operate in higher modes and hence use more bandwidth. Energy directly affects the maximum allowed bandwidth for every cluster, so as the energy increases the bandwidth increases and more nodes can operate in higher modes. This can be seen in the figure where increasing energy results in increasing total quality. This trend can be observed up to a certain point

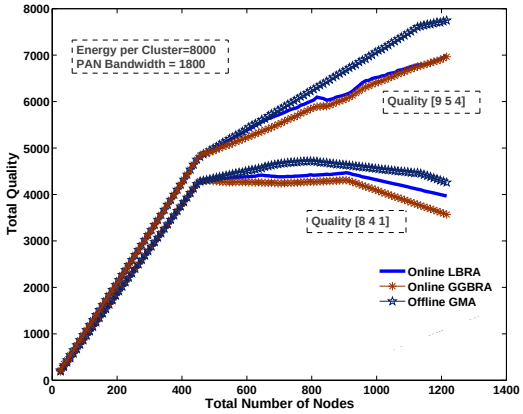


Figure 6.7: Number of Nodes vs Total Quality.

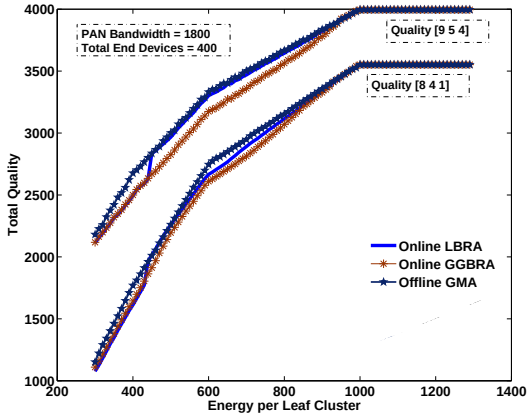


Figure 6.8: Energy per Cluster vs Total Quality.

where any further increase in cluster energy cannot increase the assigned bandwidth due to the constraint of Equation 6.6 constraint. After this saturation point the total quality remains almost unchanged.

The results illustrated clearly show the complexity of adaptive resource management in case of large systems. Those systems can be even the case of nowadays hierarchical embedded systems. Besides, with the work in (136) it has been proposed efficient solutions to complex problems and mostly it has been shown the direction to be taken in order to get even more efficient solutions.

The work here presented has been published in (136).

6.2 Resource Adaptation with TDMA Servers

The server architecture paradigm has been seriously considered in the past years for its ability to separate the scheduling concerns between the system and the application

levels.

A server mechanism is strictly connected with the resource partition idea, where a shared resource, e.g. CPU computation time, is used by several applications. Servers are used to isolate the temporal behavior of real-time tasks through resource reservations (116). Abeni and Buttazzo (5) introduced a bandwidth reservation mechanism (the Constant Bandwidth Server - CBS) that allows real-time tasks to execute in dynamic environments under a temporal protection mechanism, so that the server never exceeds a predefined bandwidth, independently on the actual requests of the server tasks.

Server models can be classified into *event-driven servers*: the servers are driven by the application requirements. The CBS and sporadic server (147) are typical examples. And *time-triggered servers*: the server resource supply is driven by a predefined timing pattern that depends only on the server properties. An example is the Time Division Multiple Access (TDMA) server where the resource is periodically partitioned (167). In particular, a TDMA server assigns time slots to its applications that repeat each cycle.

Nowadays, dynamic real-time applications ask for real-time systems that can adapt their behavior at run-time by changing their operating mode: the computing environment and the available resource of a system may change over time. For example, adding a new task into the system at runtime may result in a reduction of the computing resources being allocated to the existing tasks. Moreover a change in the operating mode of an application, e.g., from start-up to normal, or from normal to shut-down, may also demand re-allocation of the computing resources among the tasks. That and many other scenarios require flexible workload management and resource allocation.

Whereas a server manages an application by supplying the resource it requires, adaptive applications must rely on adaptive servers to meet their changing resource requirements. Servers need to be reconfigured dynamically to adapt the resource reservations and reflect the changes in the system or its environment. Such reconfigurations need to be performed online without jeopardizing schedulability. It is therefore essential to develop appropriate resource reconfiguration criteria and algorithms to manage the criticality of the transition phase.

6.2.1 Models

In this section, we introduce the basic modeling techniques that will be used for the analysis of the proposed scheduling server. Our techniques are based on the framework for Modular Performance Analysis with Real-Time Calculus (MPA-RTC) (43; 159). This is a compositional framework for system-level performance analysis of distributed real-time systems based on Network Calculus (51; 93). It analyzes the flow of event streams through a network of processing and communication resources in order to compute worst-case backlogs, end-to-end delays, and throughput.

A General Event Stream Model Application tasks are activated by the arrivals of events. The timing characteristics of event arrivals from the input streams are described abstractly with *arrival curves*. The tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of upper and lower arrival curves provides an upper and a lower bound on the number of events that arrive in *any* time interval of length Δ. If $R[s,t)$ is the cumulative function that denotes the number of events that arrive in the time interval $[s,t)$, then the following inequality holds

$$\alpha^l(t-s) \leq R[s,t) \leq \alpha^u(t-s) \quad \forall s < t,$$

where $\alpha^u(\Delta) = \alpha^l(\Delta) = 0$ for $\Delta \leq 0$. Arrival curves substantially generalize traditional event stream models such as periodic, periodic with jitter, and sporadic. Often the domain of arrival curves are workload units. Event-based arrival curves can be converted to workload-based arrival curves by scaling with the best-case/worst-case execution demand of events. In (136), we make use of the workload-based interpretation and assume that each event has a fixed execution demand. More general concepts for characterization of these units are discussed in (115).

A General Resource Model Processing and communication resources are also represented abstractly. Their availability is described by a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves which provide an upper and a lower bound on the available service in *any* time interval of length Δ. The service is expressed in a suitable workload unit that match the one of the arrival curve, such as number of cycles for computing resources or bits for communication resources. If $C[s,t)$ denotes the amount of workload units available from a resource in the time interval $[s,t)$, then the following inequality holds:

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s) \quad \forall s < t,$$

where $\beta^u(\Delta) = \beta^l(\Delta) = 0$ for $\Delta \leq 0$. In the next sections service curves will be used to express the resources supplied by a server to an application, and the resources demanded by an application from a server.

Processing Model and Analysis In real-time systems, event streams are typically processed by a sequence of HW/SW components. In the framework of MPA-RTC such processing or communication components are modeled by abstract performance components that act as curve transformers in the domain of arrival and service curves, where the transfer function depends on the modeled processing semantics. By connecting the inputs and outputs of components and constructing a network of abstract performance components, MPA-RTC can model complex applications and analyze them for worst-case backlogs, end-to-end delays, and throughput, for details see (43). The framework supports analysis for various scheduling policies such as fixed priority, EDF, first-in first-out, generalized processor share, TDMA, scheduling servers, and others.

Schedulability of an Application An application is schedulable if its real-time requirements are satisfied by the system. Using MPA-RTC it is possible to compute the minimum resource demand of an application as a single service curve $\underline{\beta^l}$, for details see (172; 173). Then, the system needs to provide a resource supply $\overline{\beta^l}$ that is greater or equal to the demanded one, in order for the application to be schedulable. This can be expressed as:

$$\overline{\beta^l}(\Delta) \geq \underline{\beta^l}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0}. \tag{6.13}$$

This is the schedulability condition of an application with a single operating mode. During runtime however, adaptive applications change between different operating modes which have different resource demands. Therefore, for each operating mode i of an application, the system needs to satisfy a different schedulability condition:

$$\overline{\beta^{l,i}}(\Delta) \geq \underline{\beta^{l,i}}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i. \tag{6.14}$$

6.2.2 Framework for Adaptive Servers with Guarantees

In this section, we give an overview of a framework with adaptive resource reservations. There are many scenarios for the use of such a framework and many different ways to realize it. We focus on the scheduling servers and their properties. In our framework, applications share a common processor using servers and we refer to them as Adaptive Servers with Guarantees (ASG) as they guarantee resource reservations and can be reconfigured dynamically while still providing a guarantee even during the reconfiguration.

We consider a uniprocessor system that runs a set of applications. Each application is scheduled on an individual ASG server. The servers provide resource reservations and guarantee isolation between applications. Applications can be of arbitrary complexity and they may even have their own schedulers, as in hierarchically scheduled systems (170). An ASG server is only concerned with guaranteeing a minimum service supply to its application. The system has a single Server Manager that can control the parameters of all servers (such as their budgets and period) and is able to communicate with the applications in order to accommodate their changing resource requirements.

The overall system framework is illustrated in Figure 6.9.

The Adaptive Server with Guarantees Servers are scheduled statically by a TDMA scheme. For each server a slot of fixed size Q called budget is reserved in the TDMA time-wheel. A server is activated, i.e., its budget becomes available, when the slot of the server arrives in the TDMA time-wheel. All servers in the system are activated periodically with the same period P which equals to the cycle of the TDMA. Servers can have different budgets but always a common period. An ASG server is denoted with the tuple (Q, P). A schedule of four ASG servers is illustrated in Figure 6.10.

Budgets are always given to applications regardless of whether they use them or not, like in a traditional TDMA schedule. In the following discussion, we assume

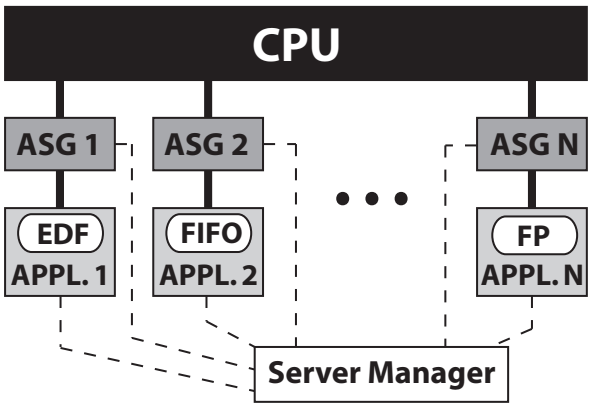


Figure 6.9: Overview of a system where the CPU is shared by applications through multiple ASG servers.

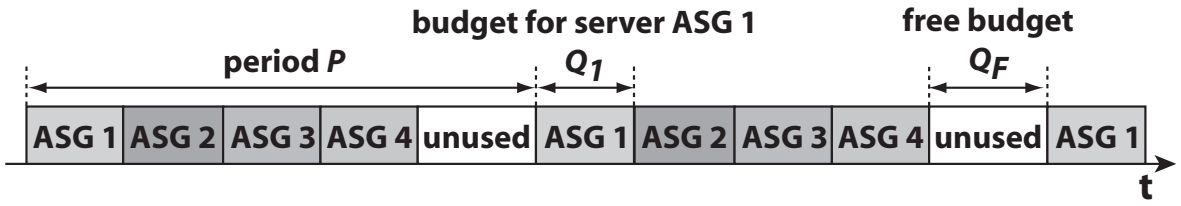


Figure 6.10: Schedule for four ASG servers.

that context switch overheads take negligible time but they can be trivially added to our analysis. The description of an ASG server can be summarized in the following definition.

Definition 6.2.1. *An ASG server (Q, P) guarantees to an application access to a shared resource for $Q > 0$ time units every $P > 0$ time units, where $Q \leq P$.*

The total utilization for a system with N ASG servers is defined as:

$$U = \frac{\sum_{i=1}^{N} Q_i}{P}, \tag{6.15}$$

as the sum of the single server utilizations Q_i/P. Such a system is schedulable when the total utilization is smaller or equal to 1:

$$U \leq 1. \tag{6.16}$$

When the total utilization is less than 1, there is some unused budget in the system, Q_F, called the free budget. We suppose that all ASG servers are scheduled from the beginning of every period one after the other, and the free budget is always at the end, as illustrated in Figure 6.10. The free budget may be given to non real-time applications on the basis that it can always be reclaimed by the system. The free budget is essential in our framework during reconfigurations as it will be shown in Section 6.2.3.

Resource Supply of an ASG An ASG server (Q, P) may not have access to the CPU for a time interval Δ that is upper bounded by $P - Q$. After this interval, the server will have guaranteed access to the resource for Q time units. Therefore, an ASG server cannot guarantee resource access for any interval of size $0 \leq \Delta \leq P - Q$. However, it guarantees service of $S(\Delta - (P - Q))$, in any interval $(P - Q) \leq \Delta \leq P$, where S is CPU speed, e.g. cycles per time unit. Without loss of generality, we assume that $S = 1$, as, all parameters in the system can be normalized according to this speed. Then the minimum resource supply of an ASG server (Q, P) in any time interval Δ can be lower bounded by the following function:

$$\beta_{Q,P}(\Delta) = \max\left(\left\lfloor \frac{\Delta}{P} \right\rfloor Q, \Delta - \left\lceil \frac{\Delta}{P} \right\rceil (P - Q)\right), \tag{6.17}$$

or more compactly as:

$$\beta_{Q,P}(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \left\{ \lambda - \left\lceil \frac{\lambda}{P} \right\rceil (P - Q) \right\}. \tag{6.18}$$

The minimum resource supply for an ASG server (Q, P) is illustrated in Figure 6.11.

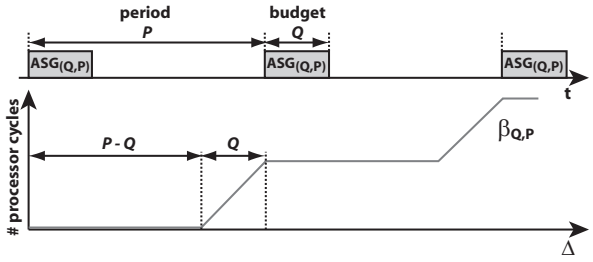


Figure 6.11: Resource supply of an ASG server (Q, P).

The minimum resource supply function in (6.18) is actually a lower *service curve* as known from Network and Real-Time Calculus (51; 93; 159). Service curves are abstract representations for the availability of processing and communication resources. A service curve $\beta(\Delta)$ gives a lower bound on the available service in *any* time interval of length $\Delta > 0$ where for $\Delta \leq 0$, $\beta(\Delta) = 0$. The service is usually expressed in a suitable workload unit such as number of cycles for computing resources or bits for communication resources.

Performance Analysis Application tasks are activated by the arrivals of events. The timing characteristics of event arrivals are described abstractly with *arrival curves* as known from Network and Real-Time Calculus. The arrival curve $\alpha(\Delta)$ denotes an upper bound on the number of events that arrive in *any* time interval of length $\Delta > 0$ where for $\Delta \leq 0$, $\alpha(\Delta) = 0$. Arrival curves substantially generalize traditional event stream models such as periodic, periodic with jitter, and sporadic. Often the domain of arrival curves are workload units. Event-based arrival curves can be converted to workload-based arrival curves by scaling with the best-case/worst-case execution

demands of events. The units of the arrival and service curves used in an analysis need to be the same. In (155), we use the workload-based interpretation and assume that each event has a fixed execution demand. More general concepts for characterization of these units are discussed in (115).

Now given the minimum resource supply of an ASG server and a characterization of the activation stream of the task, we can compute the worst-case response time (WCRT) for the task. To this end, we use results from Network and Real-Time Calculus where for a resource supply characterized with a service curve β and an input stream characterized with an arrival curve α, the WCRT of an event from the stream is the maximum horizontal distance between the arrival and the service curves computed as follows:

$$\sup_{\lambda \geq 0} \{\inf\{\tau \geq 0 : \alpha(\lambda) \leq \beta(\lambda + \tau)\}\} \triangleq \mathrm{Del}(\alpha, \beta). \tag{6.19}$$

Example 6.2.2. *To illustrate this let us consider Example 5.1.4 from Section 5.1. Consider server S_B in Old Mode which is an ASG server with budget $Q = 5$ msec and a period $P = 10$ msec. The respective service curve can be computed with equation (6.18). It serves a single periodic task τ_B with a period of 5 msec and WCET of 2 msec. The WCRT of the task computed with equation (6.19) is shown in Figure 6.12a. The computed WCRT is equal to the one observed on the trace in Figure 5.7.*

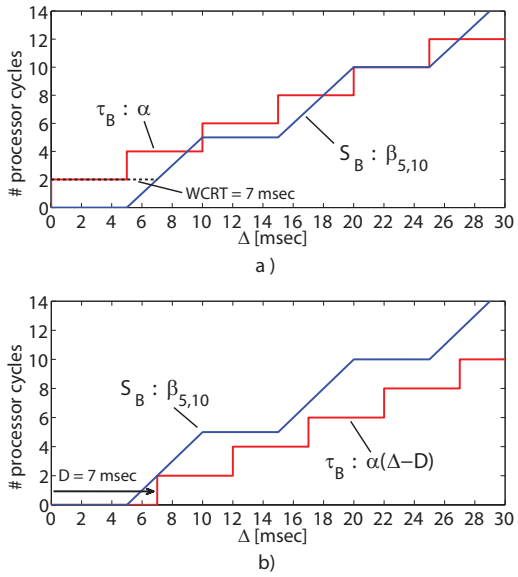


Figure 6.12: Server S_B and task τ_B WCRT analysis (a) and schedulability condition (b).

Schedulability An application is schedulable if its real-time requirements are satisfied by the system. If we consider the case of a single task, we may have the requirement that all activations are processed within a relative deadline D. Given (6.19), this is expressed as $\mathrm{Del}(\alpha, \beta) \leq D$. Inverting it w.r.t. β, we can compute a lower bound

on the minimum resource demand required to meet the deadline requirement. This is expressed as follows:

$$\underline{\beta}(\Delta) \geq \alpha(\Delta - D) \quad \forall \Delta \in \mathbb{R}^{\geq 0}. \tag{6.20}$$

In other words, the minimum resource demand has a lower service curve that equals to $\underline{\beta}(\Delta) = \alpha(\Delta - D)$.

By using previous results on demand bound functions by Baruah et al.(17) and interface-based design by Wandeler et al.(170) such a task is schedulable if a resource can supply service that is larger or equal to the demanded one. For an ASG server (Q, P), schedulability would mean that:

$$\beta_{Q,P}(\Delta) \geq \underline{\beta}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \tag{6.21}$$

where $\beta_{Q,P}$ is computed with (6.18).

In the case of task τ_B from Example 5.1.4, it is schedulable with a relative deadline $D = 7$ msec by server S_B with Old Mode parameters $(5, 10)$. This can be seen in Figure 6.12b where the service curve of server S_B is above the shifted arrival curve of task τ_B which expresses the resource demand of the task.

The same schedulability condition applies not only for single tasks, but even for complex applications as we can compute the minimum resource demand of an application as a single service curve $\underline{\beta}$, for details see (170).

Schedulability during a Reconfiguration A reconfiguration may change the server parameters such as their budgets and period from one mode to another. We consider a single reconfiguration. For a system with N ASG servers before a reconfiguration they operate with parameters (Q_i^O, P^O), $1 \leq i \leq N$, (for Old Mode), and after the reconfiguration with parameters (Q_i^N, P^N), $1 \leq i \leq N$, (for New Mode). We assume that the system is schedulable in Old Mode and New Mode separately, i.e., condition (6.21) is satisfied by assumption for all servers in Old Mode:

$$\beta_{Q_i^O,P^O}(\Delta) \geq \underline{\beta_i}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i,$$

and for all servers in New Mode:

$$\beta_{Q_i^N,P^N}(\Delta) \geq \underline{\beta_i}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i.$$

During a reconfiguration or the changing from one set of server parameters to another, the system should not suffer a degraded performance. Let us consider the two problems described in Section 5.1. To prevent isolation violations each server should be able to guarantee a service curve during a reconfiguration. To prevent deadline violations each server should be able to guarantee a service curve that is sufficiently large during a reconfiguration.

Let us denote as $\tilde{\beta}_i(\Delta)$ the service provided by an ASG server during time intervals Δ that span Old Mode, the Reconfiguration, and New Mode. In order to prevent a degraded performance during a reconfiguration we need to have for all servers that:

$$\tilde{\beta}_i(\Delta) \geq \min\{\beta_{Q_i^O,P^O}(\Delta), \beta_{Q_i^N,P^N}(\Delta)\} \; \forall \Delta \in \mathbb{R}^{\geq 0} \; \forall i. \tag{6.22}$$

The above condition ensures that each server guarantees during a reconfiguration at least the minimum of the services guaranteed in Old and New Modes. This implies that each application served by an ASG server during a reconfiguration is guaranteed that it will not violate the larger of the deadlines from Old and New Modes.

To illustrate this, consider server S_B from Example 5.1.4. During the transition from Old Mode to New Mode if it were able to meet condition (6.22), then the WCRT of task τ_B would have been at most the maximum of the WCRTs from the two modes which is 8 msec, and it would not have experienced the WCRT of 9 msec. This is illustrated in Figure 6.13.

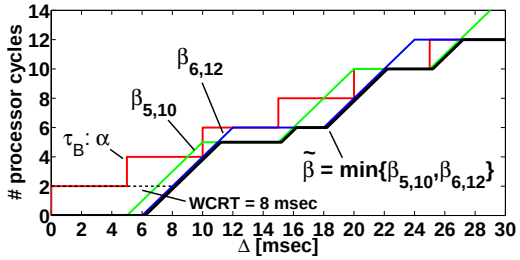


Figure 6.13: Condition (6.22) can guarantee a WCRT of 8 msec for task τ_B during the reconfiguration in Example 5.1.4.

6.2.3 Algorithms and Analysis

In this section, we classify the scenarios for feasible resource reconfigurations and provide schedulability analysis for each of them to show that they meet condition (6.22) with proofs available in (154). The proposed algorithms are implemented in the server manager and executed by it. Initiation of a reconfiguration can be done by an application in order to request a different resource reservation, or by the system manager in order to achieve better resource allocation. The proposed algorithms work regardless of what the reason for reconfiguration is.

Reconfigurations that do not require change of period have simple feasibility conditions and they do not require any pre-computed information except budgets and period as the decision for performing them can be made online. For the case of changing periods, conditions are much more involved as we will see, and some parameters need to be pre-computed and stored in the Server Manager to be used online.

We differentiate between reconfigurations that do not change the period of the servers, i.e., $P^O = P^N$, and those that do, i.e., $P^O \neq P^N$. The possible reconfiguration scenarios are summarized in Table 6.6.

Notation. The time of the k-th activation of server (Q_i, P) is denoted as $s_{i,k}$. The time when the free budget starts is $s_{F,k}$. An activation frame k contains the k-th activations of all servers and the free budget. The time when activation frame k starts is the activation time of the first scheduled server (Q_1, P) denoted as $s_{1,k}$ and it ends when the same server is scheduled again $s_{1,k+1}$. When we would like to differentiate between any of the parameters and indicate that they belong to the Old Mode, or the

Table 6.6: Reconfiguration Scenarios

$P^O = P^N$	Remove a server
	Decrease of a budget
	Add a server
	Increase of a budget
$P^O \neq P^N$	Increase of period $P^O < P^N$
	Decrease of period $P^O > P^N$

New Mode, we will add the superscripts O or N, respectively. In the Old Mode, all activation frames have the same length which equals to the period, $P^O = s^O_{1,k+1} - s^O_{1,k}$ for frames k in the Old Mode, unless otherwise stated. Similarly for the New Mode.

Algorithms that change the period of servers will require an intermediate phase called Reconfiguration where budgets and period will be different than the ones in Old and New Modes. Parameters belonging to the Reconfiguration will carry the superscript R when necessary. The notation is illustrated in Figure 6.14.

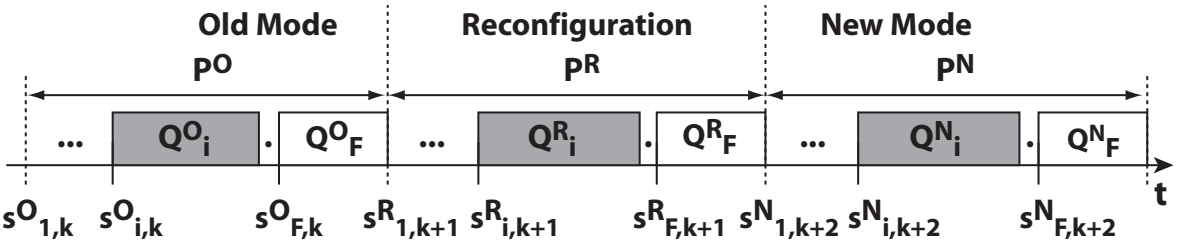


Figure 6.14: Notation. Three activation frames where activation frame k belongs to the Old Mode, frame $k+1$ to the Reconfiguration, and frame $k+2$ to the New Mode.

No Change of Period Here for brevity we do not differentiate between P^O and P^N but refer to the period as P. In these scenarios the last activation frame of the Old Mode which we denote as k is followed immediately by the first activation frame of the New Mode denoted as $k+1$.

Removing an Existing ASG Removing a server from the schedule means that in the Old Mode, it has budget $Q^O > 0$, and in the New Mode, its budget is $Q^N = 0$. The budgets of all other servers are unchanged. This is an operation that can always be performed since it decreases the utilization of the system by Q^O/P, and increases the free budget, $Q^N_F = Q^O_F + Q^O$.

Algorithm 12 describes removing server (Q^O_i, P) from a schedule with N servers. When the server is removed, activations of all preceding servers are unchanged while activations of succeeding servers are shifted earlier by the removed budget. This is illustrated in Figure 6.15 where the activation times of servers (Q^N_3, P) and (Q^N_4, P) have been shifted to the left by Q^O_2 in New Mode, and Q^O_2 has been used to increase

the free budget. The dashed boxes show where servers (Q_3^O, P) and (Q_4^O, P) would have been scheduled if there were no reconfiguration.

Algorithm 12 `Removing an ASG`

Input: $s_{j,k}^O$, $1 \leq j \leq N$ ▷ Schedule in last frame (k) of Old Mode
Input: P ▷ Current period
Input: (Q_i^O, P) ▷ Server to be removed
Output: $s_{j,k+1}^N$, $1 \leq j \leq N-1$ ▷ Schedule in first frame $(k+1)$ of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j < i$ **then**
3: $s_{j,k+1}^N \leftarrow s_{j,k}^O + P$
4: **else if** $j > i$ **then**
5: $s_{j,k+1}^N \leftarrow s_{j,k}^O + P - Q_i^O$
6: **end if**
7: **end for**

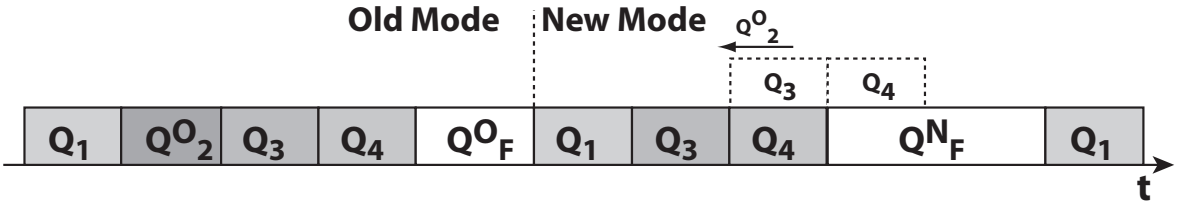


Figure 6.15: Removing server (Q_2^O, P) from a schedule of four ASG servers.

Theorem 6.2.3. *Removing server (Q_i^O, P) from a schedule of N servers using Algorithm 12 satisfies condition (6.22) for all other servers in the system as each of them gets at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j \geq \beta_{Q_j,P}$, $1 \leq j \leq N$, $j \neq i$.*

Proof. All proofs are omitted and can be found online in a technical report (154). □

Decreasing the Budget of an Existing ASG Decreasing the budget of a server means that in Old Mode, the server has budget $Q^O > 0$, and in New Mode, its budget is $0 < Q^N < Q^O$. The budgets of all other servers are unchanged. This is an operation that can always be performed since it decreases the utilization of the system by $(Q^O - Q^N)/P$, and increases the free budget, $Q_F^N = Q_F^O + (Q^O - Q^N)$.

Algorithm 13 describes decreasing the budget of server (Q_i, P) from Q_i^O in Old Mode to Q_i^N in New Mode in a schedule of N servers. In the first frame when the budget is decreased, activations of all preceding servers are unchanged while activations of succeeding servers are shifted earlier by the amount of decrease of budget. This is illustrated in Figure 6.16 where the activation times of servers (Q_3, P) and (Q_4, P) have been shifted earlier in New Mode by $(Q_2^O - Q_2^N)$, and $(Q_2^O - Q_2^N)$ has been used to increase the free budget. The dashed boxes show where servers (Q_3, P) and (Q_4, P) would have been scheduled if there were no reconfiguration.

Algorithm 13 `Decreasing the budget of an ASG`

Input: $s^O_{j,k}$, $1 \leq j \leq N$ ▷ Schedule in last frame (k) of Old Mode
Input: P ▷ Current period
Input: (Q^O_i, P) ▷ Server to be modified with Old Mode parameters
Input: (Q^N_i, P) ▷ Server to be modified with New Mode parameters
Output: $s^N_{j,k+1}$, $1 \leq j \leq N$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j \leq i$ **then**
3: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
4: **else if** $j > i$ **then**
5: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P - (Q^O_i - Q^N_i)$
6: **end if**
7: **end for**

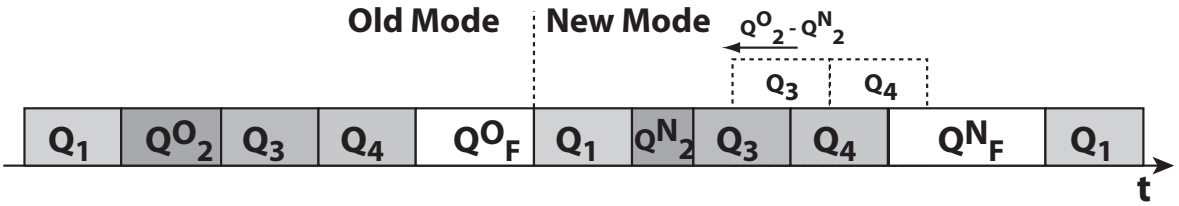


Figure 6.16: Decreasing the budget from Q^O_2 to Q^N_2 in a schedule of four ASG servers. The activation times of servers (Q_3, P) and (Q_4, P) have been shifted earlier in New Mode by $(Q^O_2 - Q^N_2)$, and $(Q^O_2 - Q^N_2)$ has been used to increase the free budget. The dashed boxes show where servers (Q_3, P) and (Q_4, P) would have been scheduled if there were no reconfiguration.

Theorem 6.2.4. *Decreasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers using Algorithm 13 satisfies condition (6.22) for all servers in the system. Unchanged servers get at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j \geq \beta_{Q_j,P}$, $1 \leq j \leq N$, $j \neq i$. For the decreased server, this is $\tilde{\beta}_i \geq \beta_{Q^N_i,P}$.*

Adding a New ASG Adding a server to the schedule means that in Old Mode, it has budget $Q^O = 0$, while in New Mode, its budget is $Q^N > 0$. Budgets of all other servers are unchanged. This is an operation that is feasible if there is sufficient free budget in the system:

$$Q^N \leq Q^O_F.$$

The reconfiguration decreases the free budget in the system, $Q^N_F = Q^O_F - Q^N$, and increases the utilization by Q^N/P.

Algorithm 14 describes adding server (Q^N_{N+1}, P) to a schedule of N servers. In the first frame where the server is added, it is scheduled at the beginning of the free budget slot. This is illustrated in Figure 6.17 with the activation times of existing servers not changing as the added server is scheduled after all other servers in New Mode. Free budget has been decreased by the budget of the server, $Q^N_F = Q^O_F - Q^N_5$. The dashed box shows where the free budget Q^O_F would have been if there were no reconfiguration.

Algorithm 14 `Adding an ASG`

Input: $s^O_{j,k}$, $1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: $s^O_{F,k}$ ▷ Start of free budget in frame (k) in Old Mode
Input: P ▷ Current period
Input: Q^O_F ▷ Free budget in Old Mode
Input: (Q^N_{N+1}, P) ▷ Server to be added in New Mode
Require: $Q^N_{N+1} \le Q^O_F$
Output: $s^N_{j,k+1}$, $1 \le j \le N+1$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: $\quad s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
3: **end for**
4: $s^N_{N+1,k+1} \leftarrow s^O_{F,k} + P$

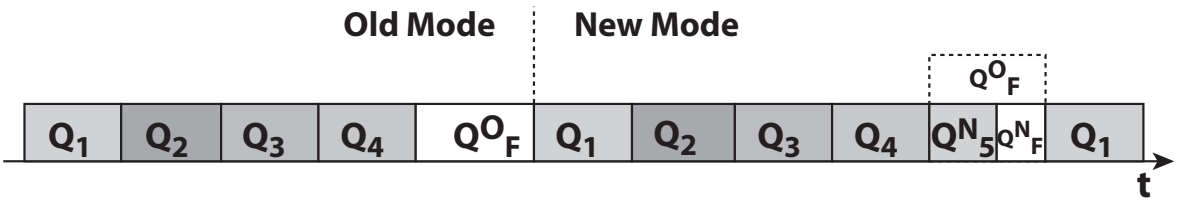


Figure 6.17: Addition of server (Q_5, P) to a schedule of four ASG servers.

Theorem 6.2.5. *Adding server (Q^N_{N+1}, P) to a schedule of N servers using Algorithm 14 satisfies condition (6.22) for all other servers in the system as each of them gets at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j = \beta_{Q_j,P}$, $1 \le j \le N$.*

Increasing the Budget of an Existing ASG Increasing the budget of a server means that in Old Mode it has budget $Q^O > 0$, and in New Mode it has budget $Q^N > Q^O$. Budgets of all other servers are unchanged. This is an operation that is feasible if there is sufficient free budget in the system:

$$Q^N - Q^O \le Q^O_F.$$

The reconfiguration decreases the free budget in the system, $Q^N_F = Q^O_F - (Q^N - Q^O)$, and increases the utilization of the system by $(Q^N - Q^O)/P$.

Algorithm 15 shows increasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers. In the first frame where the budget is increased, all preceding servers are activated earlier in the free budget of the previous frame by the amount of the increase of budget, and all succeeding servers are activated without change. This is illustrated in Figure 6.18; last frame of Old Mode has a decreased length, $P - Q^N_2 + Q^O_2$. This causes the activation times of server (Q_1, P) to be shifted earlier. Activation times of server (Q_3, P) do not change as the shorter activation frame cancels with the increased budget for all New Mode activations. Free budget has been decreased by the increase of server budget, $Q^N_F = Q^O_F - Q^N_2 + Q^O_2$. The dashed boxes show where the activations of servers (Q_1, P), (Q_2, P), (Q_3, P) and the free budget Q_F would have been if there were no reconfiguration.

Algorithm 15 `Increasing the budget of an ASG`

Input: $s^O_{j,k}$, $1 \leq j \leq N$ ▷ Schedule in last frame (k) of Old Mode
Input: P ▷ Current period
Input: Q^O_F ▷ Free budget in Old Mode
Input: (Q^O_i, P) ▷ Server to be modified with Old Mode parameters
Input: (Q^N_i, P) ▷ Server to be modified with New Mode parameters
Require: $Q^N_i - Q^O_i \leq Q^O_F$
Output: $s^N_{j,k+1}$, $1 \leq j \leq N$ ▷ Schedule in first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j \leq i$ **then**
3: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P - (Q^N_i - Q^O_i)$
4: **else if** $j > i$ **then**
5: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
6: **end if**
7: **end for**

%vspace*-0.2cm

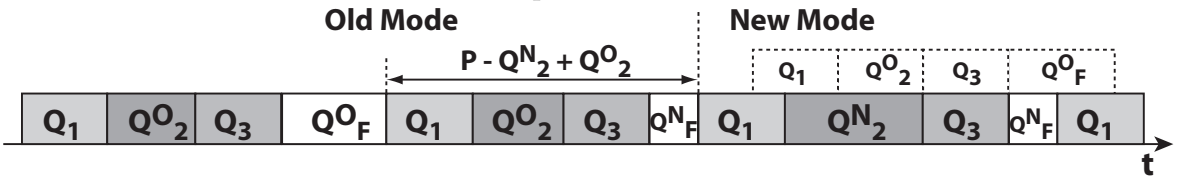


Figure 6.18: Increasing the budget of server (Q^O_2, P) to Q^N_2 in a schedule of three ASG servers.

Theorem 6.2.6. *Increasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers using Algorithm 15 satisfies condition (6.22) for all servers in the system. Unchanged servers get at least a guaranteed service during the reconfiguration of $\tilde{\beta}_j \geq \beta_{Q_j,P}$, $1 \leq j \leq N$, $j \neq i$. For the increased server this is $\tilde{\beta}_i \geq \beta_{Q^O_i,P}$.*

Change of Period We perform analysis given the configuration of the system (such as budgets and period) in Old and New Modes. The results of the analysis are whether a transition is feasible with the given configurations, and in the case of feasibility with what parameters it can be executed online.

Increase of Period We suppose that there are N servers in the system. In the Old Mode they operate with parameters (Q^O_i, P^O), $1 \leq i \leq N$, and in the New Mode with (Q^N_i, P^N), $1 \leq i \leq N$, where $P^O < P^N$. Assume that for every server we have that

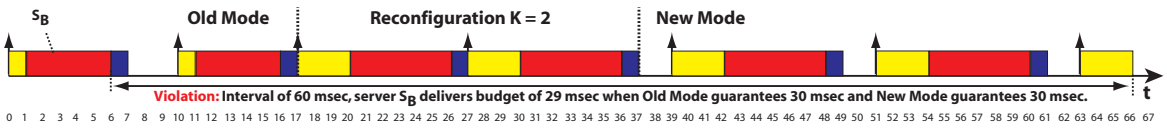


Figure 6.19: Violation of condition (6.22) when increasing period with $K = 2$ for server S_B from Example 5.1.4.

$Q_i^O \leq Q_i^N$. If this is not the case, namely there is a server that requires a smaller budget in the bigger period, $Q_i^O > Q_i^N$, we can reduce its budget first by using the algorithms proposed in Section 6.2.3 as we can be sure that schedulability is satisfied with the new budget in the smaller period, and then perform the reconfiguration involving increase of period.

The proposed reconfiguration algorithm is subject to the feasibility condition that the sum of all New Mode server budgets is smaller than the Old Mode period which is expressed as follows:

$$\sum_{i=1}^{N} Q_i^N \leq P^O. \tag{6.23}$$

The condition ensures that the increase of budgets does not lead to service guarantee violations in intervals of time beginning P^O time units before the reconfiguration and ending P^O time units after the reconfiguration. It can be related to the feasibility condition for Algorithm 15, $\sum_{i=1}^{N}(Q_i^N - Q_i^O) \leq Q_F^O$.

If condition (6.23) is not satisfied for a set of Old Mode and New Mode parameters, the reconfiguration algorithm would need to go through intermediate modes (budgets and periods) where for each successive pair of them condition (6.23) holds. We will not discuss this further and assume that the feasibility condition is met.

The algorithm for performing safely the increase of period can be summarized in three steps: (1) Increase to New Mode budgets following Algorithm 15. (2) Schedule the ASG servers for $K \geq 1$ activation frames using the New Mode budgets and Old Mode period. (3) Increase to New Mode period by increasing free budget. The second step of the algorithm we denote as the Reconfiguration phase which is K activation frames long. We suppose that it has Old Mode period but it can actually have a shorter period which would require a small modification of our analysis. At the moment we assume that K is given as input to the algorithm, later we will show how to compute it. Algorithm 16 describes the details for performing the increase of period. It is illustrated in Figure 6.20.

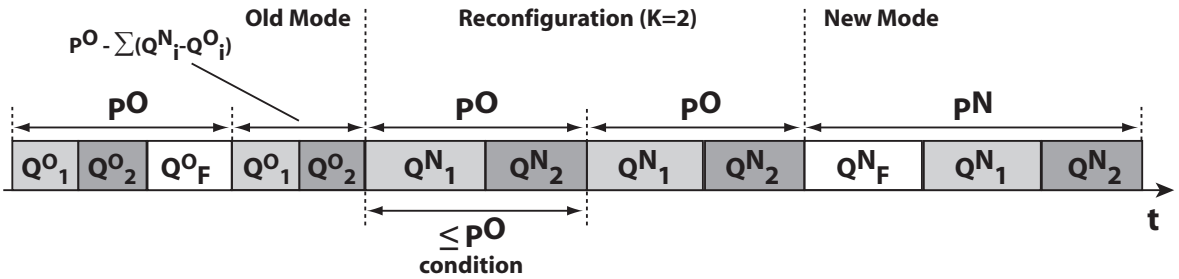


Figure 6.20: Increase of period with $K = 2$.

The following theorem gives a lower bound for the guaranteed resource supply of an ASG server during an increase of period reconfiguration.

Algorithm 16 `Increase of Period`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: P^O ▷ Old Mode period
Input: P^N ▷ New Mode period
Input: $(Q^O_i, P^O),\ 1 \le i \le N$ ▷ Servers in Old Mode
Input: $(Q^N_i, P^N),\ 1 \le i \le N$ ▷ Servers in New Mode
Input: K ▷ Number of activation frames during the Reconfiguration
Require: $\sum_{i=1}^{N} Q^N_i \le P^O$
Output: $s^N_{j,k+p},\ 1 \le j \le N,\ 1 \le p \le K$ ▷ Schedule in all frames during the Reconfiguration
Output: $s^N_{j,k+K+1},\ 1 \le j \le N$ ▷ Schedule in first frame ($k+K+1$) of New Mode

(* First frame of Reconfiguration - increase budgets *)
$s^R_{1,k+1} \leftarrow s^O_{1,k} + P^O - \sum_{i=1}^{N}(Q^N_i - Q^O_i)$
for $j \leftarrow 2$ to N **do**
  $s^R_{j,k+1} \leftarrow s^R_{j-1,k+1} + Q^N_{j-1}$
end for

(* All subsequent frames of Reconfiguration *)
for $p \leftarrow 2$ to K **do**
  for $j \leftarrow 1$ to N **do**
    $s^R_{j,k+p} \leftarrow s^R_{j,k+p-1} + P^O$
  end for
end for

(* First frame of New Mode - increase period *)
for $j \leftarrow 1$ to N **do**
  $s^N_{j,k+K+1} \leftarrow s^R_{j,k+K} + P^N$
end for

Theorem 6.2.7. *Reconfiguring a server from (Q^O_i, P^O) to (Q^N_i, P^N) in a schedule of N servers using Algorithm 16 provides at least a guaranteed service of:*[1]

$$\tilde{\beta}_i(\Delta) = \min\{\beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta),$$
$$(\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - K\cdot P^O + P^O - Q^O_i) \quad (6.24)$$
$$+ \beta_{Q^N_i,P^O}(K\cdot P^O)\} \quad (6.25)$$

which satisfies condition (6.22) when $K \ge 1$ is found as:

$$K = \max_{1\le i\le N}\Big\{\min\Big\{\kappa \mid \forall\Delta \in \mathbb{R}^{\ge 0},\ \kappa \in \mathbb{Z}^+,$$
$$(\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - \kappa\cdot P^O + P^O - Q^O_i) + \beta_{Q^N_i,P^O}(\kappa\cdot P^O)$$
$$\ge \min\{\beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta)\}\Big\}\Big\}$$

The guaranteed service in the above theorem can be explained informally as follows. It is computed as the minimum of the services from Old Mode, New Mode, and an expression which describes the service in time intervals that span Old Mode, Reconfiguration, and New Mode. The last one consists of two subexpressions. Expression (6.24) lower bounds the service guaranteed in the time window part that is *outside* of the Reconfiguration time window and hence the service curve depends only on the Old and the New Modes parameters, and it is 'shifted to the right' by the size of the

[1]The $\otimes$ is the min-plus convolution operator which is defined as: $(a \otimes b)(s) = \inf_{0\le\lambda\le s}\{a(s-\lambda) + b(\lambda)\}$

Reconfiguration time window which is at most $K \cdot P^O$ time units. Expression (6.25) lower bounds the service guaranteed only in the Reconfiguration time window which uses New Mode budgets with Old Mode period, and the service is defined for a fixed length interval of size $K \cdot P^O$.

In expressions (6.24) and (6.25), we can increase the size of the Reconfiguration phase by increasing the number of activation frames in it K. In order to meet condition (6.22) for each server, we have to find the *minimum* K that will make the guaranteed service $\tilde{\beta}_i$ greater or equal to the minimum of the Old and New Modes services. After doing this for all servers, we have to take the *maximum* K which will make the reconfiguration feasible for the whole system.

We can find the minimum K for a server efficiently by starting with an initial value of $K = 1$. If this is not feasible, we choose successive values of K by using binary search until the smallest one is found that is feasible. With bigger K we are increasing the service guaranteed in the Reconfiguration which is service greater than Old Mode and New Mode services (it has the larger New Mode budget and the smaller Old Mode period), therefore we are guaranteed to find a finite value which will make condition (6.22) satisfied.

We can illustrate this by considering server S_B from Example 5.1.4. It will need $K = 3$ to perform a safe reconfiguration from $(5, 10)$ to $(6, 12)$. This is illustrated in Figure 6.21 as well as the violations of condition (6.22) for $K = \{1, 2\}$. The trace showing the violation for $K = 2$ for server S_B is in Figure 6.19.

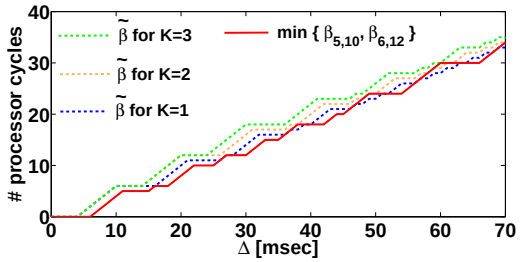


Figure 6.21: Effect of $K = \{1, 2, 3\}$ for server S_B from Example 5.1.4. Only $K = 3$ is feasible.

Decrease of Period This scenario is very similar to the one for increasing the period. Because of the lack of space, we only give the main points. In the Old Mode servers operate with parameters (Q_i^O, P^O), $1 \leq i \leq N$, and in the New Mode with (Q_i^N, P^N), $1 \leq i \leq N$, where $P^O > P^N$. We assume that for every server we have that $Q_i^O \geq Q_i^N$.

It is subject to the feasibility condition that the sum of Old Mode budgets is smaller than the New Mode period which is expressed as $\sum_{i=1}^{N} Q_i^O \leq P^N$.

The algorithm can be summarized in three steps: (1) Decrease to New Mode period by decreasing free budget. (2) Schedule the ASG servers for $K \geq 1$ activation frames using Old Mode budgets and New Mode period. (3) Decrease budgets by using Algorithm 13. Algorithm 17 describes the details for performing the decrease of period. It is illustrated in Figure 6.22.

Algorithm 17 `Decrease of Period`

Input: $s^O_{j,k},\ 1 \le j \le N$ ▷ Schedule in last frame (k) of Old Mode
Input: P^O ▷ Old Mode period
Input: P^N ▷ New Mode period
Input: $(Q^O_i, P^O),\ 1 \le i \le N$ ▷ Servers in Old Mode
Input: $(Q^N_i, P^N),\ 1 \le i \le N$ ▷ Servers in New Mode
Input: K ▷ Number of activation frames during the Reconfiguration
Require: $\sum_{i=1}^{N} Q^O_i \le P^N$
Output: $s^N_{j,k+p},\ 1 \le j \le N,\ 1 \le p \le K$ ▷ Schedule in all frames during the Reconfiguration
Output: $s^N_{j,k+K+1},\ 1 \le j \le N$ ▷ Schedule in first frame ($k+K+1$) of New Mode

(* First frame of Reconfiguration - decrease period *)
1: **for** $j \leftarrow 1$ to N **do**
2: $\quad s^R_{j,k+1} \leftarrow s^O_{j,k} + P^O$
3: **end for**

(* All subsequent frames of Reconfiguration *)
4: **for** $p \leftarrow 2$ to K **do**
5: $\quad$ **for** $j \leftarrow 1$ to N **do**
6: $\quad\quad s^R_{j,k+p} \leftarrow s^R_{j,k+p-1} + P^N$
7: $\quad$ **end for**
8: **end for**

(* First frame of New Mode - decrease budgets *)
9: $s^N_{1,k+K+1} \leftarrow s^R_{1,k+K} + P^N$
10: **for** $j \leftarrow 2$ to N **do**
11: $\quad s^N_{j,k+K+1} \leftarrow s^N_{j-1,k+K+1} + Q^N_{j-1}$
12: **end for**

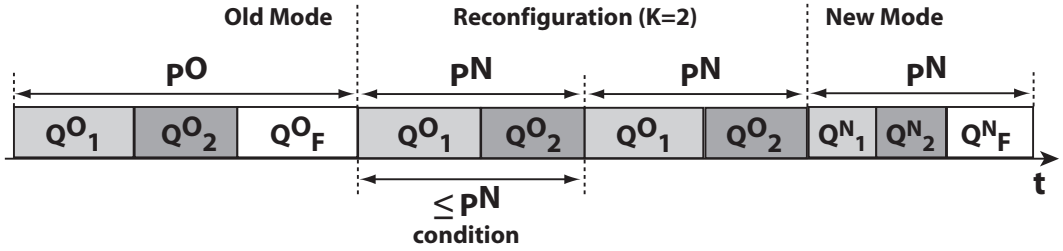


Figure 6.22: Decrease of period with $K = 2$.

Theorem 6.2.8. *Reconfiguring a server from (Q^O_i, P^O) to (Q^N_i, P^N) in a schedule of N servers using Algorithm 17 provides at least a guaranteed service of:*

$$\tilde{\beta}_i(\Delta) = \min\big\{(\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - K \cdot P^N + P^N - Q^N_i) + \beta_{Q^O_i,P^N}(K \cdot P^N),\ \beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta)\big\}$$

which satisfies condition (6.22) when $K \ge 1$ is found as:

$$K = \max_{1 \le i \le N} \Big\{ \min \Big\{ \kappa \mid \forall \Delta \in \mathbb{R}^{\ge 0},\ \kappa \in \mathbb{Z}^+, (\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - \kappa \cdot P^N + P^N - Q^N_i) + \beta_{Q^O_i,P^N}(\kappa \cdot P^N) \ge \min\{\beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta)\} \Big\} \Big\}$$

6.2.4 Case Study

Here, we consider a multi-mode real-time system that executes two applications. Application 1 can run in two modes denoted as Mode 1 and Mode 2. In mode 1, there is a single task which processes a single event stream described by a period $p = 5$ msec, jitter $j = 10$ msec, and minimum inter-arrival time between two events $d = 1$ msec. Each event has a worst-case execution time of $c = 2$ msec, and it needs to be processed within a relative deadline of $D = 9$ msec. Similarly, in mode 2 there is a single task but it processes an event stream with parameters $p = 40$ msec, $j = 20$ msec, $d = 20$ msec, $c = 7$ msec, and $D = 25$ msec. Application 2 is a single mode application, it has a single task that processes one event stream with parameters $p = 20$ msec, $j = 15$ msec, $d = 5$ msec, $c = 1$ msec, and $D = 30$ msec. The system schedules the two applications using two servers (Q_1, P) and (Q_2, P). We suppose that each context switch takes 0.3 msec. The utilization of the system, U, can be computed as $U = (Q_1 + 0.3 + Q_2 + 0.3)/P$.

The designer of this system needs to select the configuration parameters of the ASG schedule such as the *minimum* required budgets that make the two applications schedulable, and the size of the servers period. The design objective is to minimize utilization because other soft real-time applications use the unused resources while guaranteeing the real-time requirements. Then the solution depends on the mode that application 1 is currently in. Figure 6.23 shows the total utilization of the system as a function of the period of the servers considering the two modes of application 1, where the period varies from 1 msec to 50 msec. When application 1 is in mode 1, the system has the minimum utilization ($U = 0.768$) with servers period $P = 12.5$ msec, and allocated budgets for application 1 and application 2, $Q_1 = 8$ msec and $Q_2 = 1$ msec, respectively. When application 1 is in mode 2, however, the system has the minimum utilization ($U = 0.427$) achieved for period $P = 22.5$ msec, and budgets $Q_1 = 7$ msec and $Q_2 = 2$ msec.

Since the mode of application 1 changes dynamically during runtime, it is not possible to fix the parameters of the scheduler at design time. If the parameters are set to the optimal ones for mode 1, when operating in mode 2 the system would have 15% utilization overhead. Similarly fixing the parameters optimally for mode 2, the utilization overhead would be 14% when the system is in mode 1.

We can solve the above problem by using the algorithms proposed in (155). Let us consider two scenarios.

Scenario 1: When application 1 is in mode 1, we run the two ASG servers corresponding to the two applications with parameters $(8, 12.5)$ and $(1, 12.5)$ which give us the lowest system utilization. When application 1 switches to mode 2, it notifies the Server Manager (SM) and it requests a switch to the minimum budget for mode 2 of $(4.7, 12.5)$. The SM can grant this budget using Algorithm 13. Afterwards the SM manager can reconfigure the two ASG servers and increase their period to the one which makes the system utilization the smallest. The SM can use Algorithm 16 with $K = 1$ to reconfigure the system from $(4.7, 12.5)$ and $(1, 12.5)$ to $(7, 22.5)$ and $(2, 22.5)$.

Scenario 2: When application 1 has to switch back to mode 1, it first notifies the SM which by using Algorithm 17 with $K = 1$ reconfigures the two servers from $(7, 22.5)$ and $(2, 22.5)$ back to $(4.7, 12.5)$ and $(1, 12.5)$. Then the SM increases the budget for

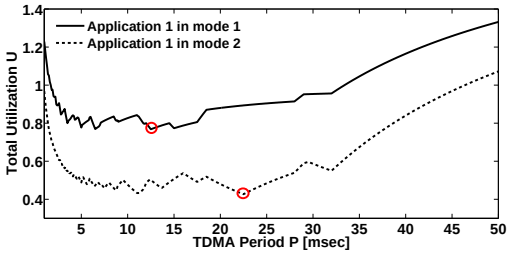


Figure 6.23: Total utilization for period varying from 1 msec up-to 50 msec considering the two different modes of application 1. The circles on the graphs denote the points of minimum utilization.

application 1 using Algorithm 15 from 4.7 to 8. Afterwards, application 1 is notified and can safely switch to mode 1.

Note that the SM takes advantage of the fact that mode 1 is more heavily loaded than mode 2 for application 1. Therefore, the SM optimizes the server period when the application is in the lightly loaded mode. This means that in Scenario 1, the application mode change is done before the resource optimization. And in Scenario 2, it is done after the resource optimization. This is feasible with our algorithms as they are completely deterministic and the time needed for a reconfiguration can be safely and accurately upper bounded in advance. It is also possible to perform the resource optimization when the system is more heavily loaded however, the reconfiguration process will take longer.

In summary, we can guarantee an optimal resource allocation in environments where applications are added or removed dynamically, or perform mode-changes. With the proposed algorithms the schedulability of the applications is never compromised during the reconfiguration process.

Setup: The servers and applications have been modeled with the Matlab Real-Time Calculus Toolbox (172). The exploration of the minimum required budgets for different periods in Figure 6.23 has been done with the Real-Time Interfaces methodology as described in (167). The exploration took less than 15 sec to perform on a commodity laptop considering discretization of the period with steps of 0.1 msec. The feasibility check for the value of K took less than 1 sec.

The work in this last section has been presented at the EMSOFT 2010 conference, (155).

Chapter 7

Conclusions

The main focus of this dissertation are the requirements of complex embedded systems. Nowadays systems are formed by dynamic components, which means that they require adaptive components able to cope with changing conditions of the system itself and/or the environment.

At the beginning, we have presented the adaptive resource problem with examples of complex and dynamic real-time systems. In order to provide real-time guarantees to those systems adaptivity techniques have to be implemented. Adaptivity in terms of adaptive service provisioning but also in terms of adaptive resource demand. Since most the literature considers dynamic applications, we have considered the case of changing resource reservations, hence mostly addressing the adaptive resource reservation problem.

We have investigated some server mechanisms deriving the guarantees in the resource provisioning to their real-time applications. Those guarantees has been first obtained in steady state conditions as the description of the minimum amount of resource the application can rely on in order to execute whatever is the scheduling policy applied.

The next step has been toward the analysis of the behavior of those servers along changing conditions. Whenever a server changes one of its parameter, the service provisioning changes. The application may then suffer those changes, so that the real-time schedulability is no more guaranteed. Thus, we have provided an analysis framework by which detail what happens to the server guarantees along any of it possible operational mode: steady states and changing conditions. Most important is the mode transition phase, where old mode effects and new mode effects are mixed up resulting in complex conditions. *To guarantee the predictability of the systems even the transition have to be studied.*

Through abstraction and approximated models we have investigated the transition phase in order to provide the transition guarantees necessary. Transition guarantees that are in terms of service provided by the servers to their applications. The analysis of the transition guarantees allows also to verify at design time the schedulability of component-based real-time systems in any condition and transition.

Finally, it has been considered two representative examples where to efficiently apply adaptive resource reservation mechanisms. Both are extreme cases which outline

the complexity but also the powerfulness of adaptive solutions. Powerfulness which has been measured in terms of efficient resource allocation and mostly, the responsiveness hence, the predictability degree, of the adaptive system.

That framework exploited by the dissertation represents a first solution toward efficient adaptive resource reservation mechanisms. A lot of interesting work has to be done in different directions.

- Extending the set of modeled servers. This way the set of modeled resource reservation mechanisms increases consequently extending the cases where to apply adaptive solutions. Exploring such a set would allow to find optimal solutions scenario by scenario and transition by transition.

- The investigation of multi-moded servers and applications concurrently acting on a real-time system. A complete dynamic system which resource requirements change during time, and which analysis combines the two presented in Chapter 5.

- The definition of a server monitor which applies complex solutions to rule the transition phase of servers. A central monitor that case by case

- The consequent final step would be the introduction of collaborative solutions where the adaptivity involves the whole systems and not just those components that require the modifications. This would open the possibilities for adaptive mechanisms leading toward complex but even more efficient solution to the adaptive resource reservation problem.

References

[1] **Sensor Andrew**. `http://www.ices.cmu.edu/censcir/sensor-andrew/`. 152

[2] Luca Abeni and Giorgio Buttazzo. **Integrating Multimedia Applications in Hard Real-Time Systems**. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*, pages 4–13, Madrid, Spain, dec 1998. 5, 16, 86, 99

[3] Luca Abeni and Giorgio Buttazzo. **Adaptive Bandwidth Reservation for Multimedia Computing**. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, page 70, Washington, DC, USA, 1999. IEEE Computer Society. 10

[4] Luca Abeni and Giorgio Buttazzo. **Hierarchical QoS Management for Time Sensitive Applications**. In *RTAS '01: Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01)*, page 63, Washington, DC, USA, 2001. IEEE Computer Society. 10

[5] Luca Abeni and Giorgio Buttazzo. **Resource Reservation in Dynamic Real-Time Systems**. *Real-Time Syst.*, **27**(2):123–167, 2004. 6, 7, 9, 10, 110, 172

[6] Luca Abeni, Claudio Scordino, and Giuseppe Lipari. **Serving non real-time tasks in a reservation environment**. In *Real-Time Linux Workshop*, 2008. 90

[7] APC (American Power Conversion). **Determining Total Cost of Ownership for Data Center and Network Room Infrastructure, http://www.apcmedia.com/salestools/CMRP-5T9PQG_R2_EN.pdf**, 2003. 37

[8] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. **Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling**. *Software Engineering Journal*, **8**:284–292, 1993. 14

[9] N.C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. **Hard Real-Time Scheduling: The Deadline-Monotonic Approach**. In *in Proc. IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, 1991. 14

[10] John Augustine, Sandy Irani, and Chaitanya Swamy. **Optimal Power-down Strategies**. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 530–539, October 2004. 37, 38

[11] Todd Austin, David Blaauw, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Wayne Wolf. **Mobile Supercomputers**. *IEEE Computer*, **37**:81–83, 2004. 36

[12] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. **Dynamic and Aggressive Scheduling Techniques for Power-aware Real-time Systems**. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 95–105, 2001. 37

[13] Scott Banachowski, Timothy Bisson, and Scott A. Br. **Integrating best-effort scheduling into a real-time system**. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004*, 2004. 89

[14] Scott A. Banachowski, Timothy Bisson, and Scott A. Brandt. **Integrating Best-Effort Scheduling into a Real-Time System**. In *RTSS*, pages 139–150, 2004. 84

[15] Philippe Baptiste. **Scheduling Unit Tasks to Minimize the Number of Idle Periods: A Polynomial Time Algorithm for Offline Dynamic Power Management.** In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 364–367, 2006. 37, 38

[16] S. Baruah. **Feasibility Analysis of Recurring Branching Tasks**. *Real-Time Systems, Euromicro Conference on*, **0**:138, 1998. 6

[17] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. **Generalized Multiframe Tasks**. *Real-Time Systems*, **17**(1):5–22, 1999. 178

[18] S. Baruah, G. Lipari, and L. Abeni. **Shrub: Shared Reclamation of Unused Bandwidth**. Technical report, RetisLab Scuola Superiore Sant'Anna, Pisa, 2008. 89

[19] Sanjoy K. Baruah. **Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks**. *Real-Time Syst.*, **24**(1):93–128, 2003. 4, 71

[20] Sanjoy K. Baruah and Jayant R. Haritsa. **Scheduling for Overload in Real-Time Systems**. *IEEE Trans. Computers*, **46**(9):1034–1039, 1997. 90

[21] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. **Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor**. In *IEEE Real-Time Systems Symposium*, pages 182–190, 1990. 15, 16, 34, 70, 71, 120, 122

[22] Guillem Bernat and Alan Burns. **Combining (/sub m//sup n/)-hard deadlines and dual priority scheduling**. In *IEEE Real-Time Systems Symposium*, pages 46–57, 1997. 84

[23] Guillem Bernat and Alan Burns. **Multiple Servers and Capacity Sharing for Implementing Flexible Scheduling**. *Real-Time Systems*, **22**(1-2):49–75, 2002. 84

[24] Enrico Bini and Giorgio C. Buttazzo. **Biasing Effects in Schedulability Measures**. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, June 2004. 32, 79, 134

[25] Enrico Bini and Giorgio C. Buttazzo. **Schedulability Analysis of Periodic Fixed Priority Systems**. *IEEE Transactions on Computers 53 (11), pp. 1462-1473, November 2004*, pages 1462–1473, 2004. 15, 22, 119

[26] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe Buttazzo. **A Hyperbolic Bound for the Rate Monotonic Algorithm**. *Proc. of the IEEE Euromicro Conference on Real-Time Systems*, pages 59–66, 2001. 117

[27] Enrico Bini, Giorgio C. Buttazzo, and Giuseppe Buttazzo. **Rate Monotonic Scheduling: the Hyperbolic Bound**. *IEEE Transactions on Computers 52 (7), pp. 933-942, July 2003*, pages 59–66, 2003. 117

[28] Enrico Bini, Marco Di Natale, and Giorgio C. Buttazzo. **Sensitivity analysis for fixed-priority real-time systems**. *Real-Time Systems*, **39**(1-3):5–30, 2008. 139

[29] B. Bouyssounouse and J. Sifakis. **Embedded Systems Design - The ARTIST Roadmap for Research and Developemnt**. *Lecture Notes in Computer Science, Springer Verlag*, 2005. 2

[30] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. **Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time and Non-Real-Time Processes**. *Real-Time Systems Symposium, IEEE International*, **0**:396, 2003. 89

[31] 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**. In *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 269–279, Washington, DC, USA, 2007. IEEE Computer Society. 119

[32] Reinder J. Bril, Elisabeth F. M. Steffens, and Wim F. J. Verhaegh. **Best-Case Response Times and Jitter Analysis of Real-Time Tasks**. *J. of Scheduling*, **7**(2):133–147, 2004. 118, 119

[33] G. Buttazzo and L. Abeni. **Adaptive rate control through elastic scheduling**. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, **5**, pages 4883–4888 vol.5, 2000. 10

[34] G C. Buttazzo, G Lipari, M. Caccamo, and Abeni L. **Elastic Scheduling for Flexible Workload Management**. *IEEE Transactions on Computers*, **51**:289–302, 2002. 90, 115, 116, 117

[35] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Plenum Publishing Co. (Series in Computer Science), 2005. 89

[36] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 1998. 3, 4, 6, 7, 16

[37] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2005. 90

[38] Giorgio C. Buttazzo and Fabrizio Sensini. **Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments**. *IEEE Trans. Comput.*, **48**(10):1035–1052, 1999. 85

[39] Giorgio C. Buttazzo, John A. Stankovic, Scuola Superiore, and S. Anna. **RED: Robust Earliest Deadline Scheduling**. In *Proc. of 3rd International Workshop on Responsive Computing Systems*, pages 100–111, 1993. 4, 90

[40] Marco Caccamo, Giorgio Buttazzo, and Lui Sha. **Capacity Sharing for Overrun Control**. In *IEEE Real-Time Systems Symposium*, pages 295–304, 2000. 88, 89

[41] Marco Caccamo, Giorgio C. Buttazzo, and Deepu C. Thomas. **Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times**. *IEEE Transactions on Computers*, **54**:198–213, 2005. 88, 89

[42] Carnegie Mellon Univ., Pittsburgh, PA. *CMUcam3 datasheet version 1.02*, Sep. 2007. 152

[43] S. Chakraborty, S. Künzli, and L. Thiele. **A General Framework for Analysing System Properties in Platform-Based Embedded System Designs**. In *DATE*, pages 190–195, 2003. 22, 26, 172, 173

[44] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler. **Interface-Based Rate Analysis of Embedded Systems**. In *RTSS*, pages 25–34, 2006. 30

[45] Jian-Jia Chen and Tei-Wei Kuo. **Procrastination for Leakage-Aware Rate-Monotonic Scheduling on A Dynamic Voltage Scaling Processor**. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 153–162, 2006. 38

[46] Jian-Jia Chen and Tei-Wei Kuo. **Procrastination Determination for Periodic Real-time Tasks in Leakage-aware Dynamic Voltage Scaling Systems**. In *International Conference on Computer-Aided Design (ICCAD)*, pages 289–294, 2007. 37, 38

[47] Jian-Jia Chen and Tei-Wei Kuo. **Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems**. In *ICCAD '07: Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 289–294, New York, NY, USA, 2007. ACM. 66

[48] Hui Cheng and Steve Goddard. **Online energy-aware I/O device scheduling for hard real-time systems**. In *Proceedings of the 9th Design, Automation and Test in Europe (DATE)*, pages 1055–1060, 2006. 60

[49] Jaejoon Cho and Sunshin An. **An Adaptive Beacon Scheduling Mechanism Using Power Control in Cluster-Tree WPANs**. *Wirel. Pers. Commun.*, **50**(2):143–160, 2009. 153

[50] Nastasi Christian, Marinoni Mauro, Santinelli Luca, Pagano Paolo, Lipari Giuseppe, and Franchino Gianluca. **BACCARAT: a Dynamic Real-Time Bandwidth AllocationPolicy for IEEE 802.15.4**. In *Proceedings of IEEE Percom 2010, International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS 2010)*, Mannheim, Germany, 2010. 66, 153, 154

[51] R. Cruz. **A Calculus for Network Delay, Parts 1 & 2**. *IEEE Transactions on Information Theory*, **37**(1), 1991. 172, 176

[52] Tommaso Cucinotta, Luigi Palopoli, and Giuseppe Lipari. **FRESCOR Delivarable D-AQ2v2: Control Algorithms for Coordinated Resource-Level and Application-Level Adaptation v2**, 2008. 10

[53] R. I. Davis and A. Burns. **Hierarchical Fixed Priority Pre-emptive Scheduling**. In *Proc. of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 2005. 7

[54] R.I. Davis, K.W. Tindell, and A. Burns. **Scheduling Slack Time in Fixed Priority Pre-emptive Systems**. In *Proceedings of the 14th IEEE Real-Time Systems Symposium (RTSS 1993*, 2004. 84

[55] Robert I. Davis and Alan Burns. **Response Time Upper Bounds for Fixed Priority Real-Time Systems**. In *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS 2008, Barcelona, Spain, 30 November - 3 December 2008*, pages 407–418. IEEE Computer Society, 2008. 117, 118, 119

[56] Robert I. Davis and Andy J. Wellings. **Dual Priority Scheduling**. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1995. 84

[57] L. de Alfaro and T. Henzinger. **Interface Theories for Component-base Design**. In *In EMSOFT'01: Embedded Software, Lecture notes in Computer Science 2211*, pages 148–165. Springer Verilag, 2001. 28, 155, 156

[58] L. de Alfaro and T. Henzinger. **Interface-Based Design**. In *To appear in the proceedings of the 2004 Marktoberdorf Summer School*, 2005. 28

[59] Augusto Born de Oliveira, Eduardo Camponogara, and George Lima. **Dynamic Reconfiguration in Reservation-Based Scheduling: An Optimization Approach**. In *15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 173–182, 2009. 10

[60] Z. Deng and J. W. s. Liu. **Scheduling Real-Time Applications in an Open Environment**. In *in Proceedings of the 18th IEEE Real-Time Systems Symposium, IEEE Computer*, pages 308–319. Society Press, 1997. 6, 85, 90

[61] Zhong Deng, Jane W.-S. Liu, Lynn Y. Zhang, Mouna Seri, and Alban Frei. **An Open Environment for Real-Time Applications**. *Real-Time Systems*, **16**(2-3):155–185, 1999. 17, 90

[62] Vinay Devadas and Hakan Aydin. **On the Interpaly of Dynamic Volatage Scaling and Dynamic Power Management in Real-Time Embedded Applications**. In *EMSOFT'08: Proceedings of the 8th ACM Conference on Embedded Systems Software*, pages 99–108, New York, NY, USA, 2008. ACM. 66

[63] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. **Incremental Schedulability Analysis of Hierarchical Real-Time Components**. In *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software (EMSOFT 2006)*, pages 272–281, October 2006. 7

[64] X. Feng and Al. Mok. **A Model of Hierarchical Real-Time Virtual Resources**. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS 2002)*, pages 26–35, December 2002. 5, 6

[65] Gerhard Fohler. **Changing Operational Modes in the Context of Pre Run-Time Scheduling (Special Issue on Responsive Computer Systems)**. *IEICE transactions on information and systems*, **76**(11):1333–1340, 1993. 9

[66] Gerhard Fohler, Tomas Lennvall, and Giorgio Buttazzo. **Improved Handling of Soft Aperiodic Tasks in Offline Scheduled Real-Time Systems using Total Bandwidth Server**. In *In Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation*, 2001. 90

[67] Bryan Ford and Sai Susarla. **CPU inheritance scheduling**. *SIGOPS Oper. Syst. Rev.*, **30**(SI):91–105, 1996. 90

[68] Lipari G. and E. Bini. **Resource Partitioning among Real-Time Applications**. In *ECRTS'03, IEEE Computer Society*, pages 151–158, 2003. 5, 6, 18, 19, 161

[69] T. M. Ghazalie and T. P. Baker. **Aperiodic servers in a deadline scheduling environment**. *Real-Time Syst.*, **9**(1):31–67, 1995. 90

[70] Steve Goddard and Xin Liu. **A Variable Rate Execution Model**. In *ECRTS*, pages 135–143, 2004. 90

[71] Qian Guangming. **An earlier time for inserting and/or accelerating tasks**. *Real-Time Systems*, 2009. 10, 133

[72] Arne Hamann and Rolf Ernst. **TDMA Time Slot and Turn Optimization with Evolutionary Search Techniques**. In *Proceedings of the 8th Design, Automation and Test in Europe (DATE)*, pages 312–317, 2005. 60

[73] Michael Gonzlez Harbour, Daniel Sangorrn, and Miguel Tellera de Esteban. **FRESCOR Delivarable D-AT2: Schedulability analysis techniques for distributed systems**, 2009. 10

[74] T.A. Henzinger and S. Matic. **An Interface Algebra for Real-Time Components**. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 253–266, April 2006. 155

[75] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. **Giotto: A Time-Triggered Language for Embedded Programming**. In *Proceedings of the First International Workshop on Embedded Software (EMSOFT)*, pages 166–184, 2001. 8

[76] R Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. **The pinwheel: a real-time scheduling problem**. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989. Vol.II: Software Track,*, pages 693–702, 1989. 5

[77] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzio. **Adaptive Dynamic Power Management for Hard Real-Time Systems**. In *the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 23–32, Washington D.C. U.S., 2009. 39, 47, 65, 66

[78] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzio. **Periodic Power Management Schemes for Real-Time Event Streams.** In *the 48th IEEE Conf. on Decision and Control (CDC)*, pages 6224–6231, Shanghai, China, 2009. 39, 40, 41, 60, 65, 66, 71

[79] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzio. **Adaptive Power Management for Real-Time Event Streams**. In *the 15th IEEE Conf. on Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 7–12, 2010. 39, 47, 65, 66

[80] Yu-Kai Huang, Ai-Chun Pang, and Hui-Nien Hung. **An Adaptive GTS Allocation Scheme for IEEE 802.15.4**. *IEEE Transactions on Parallel and Distributed Systems*, **19**(5):641–651, 2008. 153

[81] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. **Algorithms for Power Savings**. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 37–46, 2003. 38

[82] D. Isovic and G. Fohler. **Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints**. pages 207 –216, 2000. 90

[83] **International Technology Roadmap for Semiconductors 2009 Edition: System Drivers**. `http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_SysDrivers.pdf`. 37

[84] R. Jejurikar, C. Pereira, and R. Gupta. **Leakage Aware Dynamic Voltage Scaling for Real-time Embedded Systems**. In *Proceedings of the 41st ACM/IEEE Design Automation Conference (DAC)*, pages 275–280, 2004. 36, 37, 38

[85] Ravindra Jejurikar and Rajesh K. Gupta. **Procrastination Scheduling in Fixed Priority Real-time Systems**. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 57–66, 2004. 38

[86] Ravindra Jejurikar and Rajesh K. Gupta. **Dynamic Slack Reclamation with Procrastination Scheduling in Real-time Embedded Systems**. In *Proceedings of the 42nd ACM/IEEE Design Automation Conference (DAC)*, pages 111–116, 2005. 38

[87] Michael B. Jones. **Adaptive Real-Time Resource Management Supporting Composition of Independently Authored Time-Critical Services**. In *In Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 135–139, 1993. 2

[88] Gilad Koren and Dennis Shasha. **D^{over}; an optimal on-line scheduling algorithm for overloaded real-time systems**. In *IEEE Real-Time Systems Symposium*, pages 290–299, 1992. 90

[89] Anis Koubâa, Mário Alves, Eduardo Tovar, and André Cunha. **An implicit GTS allocation mechanism in IEEE 802.15.4 for time-sensitive wireless sensor networks: theory and practice**. *Real-Time Syst.*, **39**(1-3):169–204, 2008. 66, 153

[90] Anis Koubaa, Andre Cunha, and Mario Alves. **A Time Division Beacon Scheduling Mechanism for IEEE 802.15.4/Zigbee Cluster-Tree Wireless Sensor Networks**. In *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 125–135, Washington, DC, USA, 2007. IEEE Computer Society. 153

[91] P. Kulkarni, D. Ganesan, P. Shenoy, and Q. Lu. **SensEye: a multi-tier camera sensor network**. In *Proceedings of the 13th annual ACM international conference on Multimedia*, Singapore, 2005. 152

[92] LAN-MAN Standards Committee of the IEEE Computer Society. *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Press, 2006. 151, 153

[93] J. Y. Le Boudec and P. Thiran. *Network calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag New York, Inc., 2001. 22, 26, 67, 91, 105, 172, 176

[94] Yann-Hang Lee, Krishna P. Reddy, and C. M. Krishna. **Scheduling Techniques for Reducing Leakage Power in Hard Real-Time Systems**. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 105–112, 2003. 38

[95] John P. Lehoczky and S. Ramos-Thuel. **An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems**. In *IEEE Real-Time Systems Symposium*, pages 110–123, 1992. 7, 84

[96] John P. Lehoczky, Lui Sha, and Y. Ding. **The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior**. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989. 4, 14, 15, 16, 33, 34, 97

[97] John P. Lehoczky, Lui Sha, and Jay K. Strosnider. **Enhanced Aperiodic Responsiveness in Hard Real-Time Environments**. In *Proceedings of the IEEE Real-Time System Symposium (RTSS 1987)*, December 1997. 6, 16, 84

[98] J. Leung and J. W. Withehead. **On the complexity of fixed priority scheduling of periodic real-time tasks.** *Performance Evaluation*, **2**(4), 1982. 4, 14

[99] Caixue Lin and Scott A. Br. **Improving soft real-time performance through better slack reclaiming**. In *In Proc. of Real-Time Systems Symposium (RTSS*, page 314, 2005. 89

[100] Giuseppe Lipari and Sanjoy Baruah. **Greedy Reclamation of Unused Bandwidth in Constant-Bandwidth Servers**. *Real-Time Systems, Euromicro Conference on*, **0**:193, 2000. 89

[101] Giuseppe Lipari and Sanjoy K. Baruah. **Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems**. In *IEEE Real Time Technology and Applications Symposium*, pages 166–, 2000. 17, 90

[102] Giuseppe Lipari and Sanjoy K. Baruah. **A Hierarchical Extension to the Constant Bandwidth Server Framework**. In *IEEE Real Time Technology and Applications Symposium*, pages 26–, 2001. 17, 90

[103] Giuseppe Lipari, Enrico Bini, and Gerhard Fohler. **A Framework for Composing Real-Time Schedulers**. *Electr. Notes Theor. Comput. Sci.*, **82**(6), 2003. 19

[104] Giuseppe Lipari and Giorgio Buttazzo. **Scheduling Real-Time Multi-Task Applications in an Open System**. *Real-Time Systems, Euromicro Conference on*, **0**:0234, 1999. 17, 90

[105] Giuseppe Lipari, John Carpenter, and Sanjoy Baruah. **A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments**. *Real-Time Systems Symposium, IEEE International*, **0**:217, 2000. 90

[106] C. L. Liu and James W. Layland. **Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment**. *Journal of the ACM*, **20**(1):46–61, 1973. 3, 4, 5, 116

[107] J.W. S. Liu. *Real-Time Systems*. Kluwer Academic Publishers, Upper Saddle River, New Jersey, USA, 2000. 6, 85

[108] Jose' L. Lorente, Giuseppe Lipari, and Enrico Bini. **A Hierarchical Scheduling Model for Component-Based Real-Time Systems**. In *Proc. of IPDPS'06*, 2006. 18

[109] Spuri Marco and Buttazzo Giorgio. **Scheduling Aperiodic Tasks in Dynamic Priority Systems**. *Real-Time Systems*, **10**:179–210, 1996. 6, 7, 90, 98

[110] Shanmuga Priya Marimuthu and Samarjit Chakraborty Chakraborty. **A Framework for Compositional and Hierarchical Real-Time Scheduling**. *Real-Time Computing Systems and Applications, International Workshop on*, **0**:91–96, 2006. 7

[111] T. Martin and D. Siewiorek. **Non-ideal Battery and Main Memory Effects on CPU Speed-Setting for Low Power**. *IEEE Transactions on VLSI Systems*, **9**(1):29–34, 2001. 69

[112] Luca Marzario, Giuseppe Lipari, Patricia Balbastre, and Alfons Crespo. **IRIS: A New Reclaiming Algorithm for Server-Based Real-Time Systems**. *Real-Time and Embedded Technology and Applications Symposium, IEEE*, **0**:211, 2004. 89

[113] Slobodan Matic and Thomas A. Henzinger. **Trading End-to-End Latency for Composability**. In *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 99–110, Washington, DC, USA, 2005. IEEE Computer Society. 6

[114] Alexander Maxiaguine, Samarjit Chakraborty, and Lothar Thiele. **DVS for Buffer-constrained Architectures with Predictable QoS-energy Tradeoffs**. In *the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, pages 111–116, 2005. 38

[115] Alexander Maxiaguine, Simon Künzli, and Lothar Thiele. **Workload Characterization Model for Tasks with Variable Execution Demand**. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 21040, Washington, DC, USA, 2004. IEEE Computer Society. 23, 173, 177

[116] C. Mercer, R. Rajkumar, and J. Zelenka. **Temporal protection in real-time operating systems**. In *Real-Time Operating Systems and Software, 1994. RTOSS '94, Proceedings., 11th IEEE Workshop on*, pages 79–83, May 1994. 6, 7, 9, 84, 172

[117] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. **Processor Capacity Reserves for Multimedia Operating Systems**. Technical Report CMU-CS-93-157, Carnegie Mellon University, Pittsburg, May 1993. 5

[118] A. K. Mok. **FUNDAMENTAL DESIGN PROBLEMS OF DISTRIBUTED SYSTEMS FOR THE HARD-REAL-TIME ENVIRONMENT**. Technical report, Cambridge, MA, USA, 1983. 3, 5

[119] Aloysius K. Mok and A. K. Feng. **Towards Compositionality in Real-Time Resource Partitioning Based on Regularity Bounds**. In *RTSS'01, IEEE Computer Society*, 2001. 31

[120] Aloysius K. Mok and A. K. Feng. **A Model of Hierarchical Real-Time Virtual Resources**. In *RTSS'02, IEEE Computer Society*, pages 26–35, 2002. 6

[121] Aloysius K. Mok, Xiang (Alex) Feng, and Deji Chen. **Resource Partition for Real-Time Systems**. In *Real-Time Systems*, 2001. 5, 6, 18

[122] Chewoo Na, Yaling Yang, and Amitabh Mishra. **An optimal GTS scheduling algorithm for time-sensitive transactions in IEEE 802.15.4 networks**. *Comput. Netw.*, **52**(13):2543–2557, 2008. 153

[123] Vincent Nelis, Joel Goossens, and Bjorn Andersson. **Two Protocols for Scheduling Multi-mode Real-Time Systems upon Identical Multiprocessor Platforms**. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 151–160, Washington, DC, USA, 2009. IEEE Computer Society. 8

[124] Linwei Niu and Gang Quan. **Reducing Both Dynamic and Leakage Energy Consumption for Hard Real-time Systems**. In *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems (CASES)*, pages 140–148, 2004. 38

[125] Luís Nogueira and Luís Miguel Pinho. **Capacity Sharing and Stealing in Dynamic Server-based Real-Time Systems**. In *IPDPS*, pages 1–8, 2007. 90

[126] Luigi Palopoli, Luca Abeni, Fabio Conticelli, Marco Di Natale, and Giorgio Buttazzo. **Real-Time control system analysis: an integrated approach**. *Real-Time Systems Symposium, IEEE International*, **0**:131, 2000. 5

[127] Luigi Palopoli, Luca Abeni, Tommaso Cucinotta, and Sanjoy K. Baruah. **Weighted feedback reclaiming for multimedia applications**. In *In ESTImedia*, pages 121–126, 2008. 89

[128] P. Pedro and A. Burns. **Schedulability Analysis for Mode Changes in Flexible Real-Time Systems**. In *ECRTS*, pages 172–179, 1998. 8, 9, 106

[129] S. Perathoner, N. Stoimenov, and L. Thiele. **Reliable Mode Changes in Real-Time Systems with Fixed Priority or EDF Scheduling**. TIK Report 292, Computer Engineering and Networks Laboratory, ETH Zurich, ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-292.pdf, September 2008. 125

[130] Linh T. X. Phan, Insup Lee, and Oleg Sokolsky. **Compositional Analysis of Multi-mode Systems**. In *ECRTS '10: Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, pages 197–206, Washington, DC, USA, 2010. IEEE Computer Society. 10

[131] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. **Resource kernels: A resource-centric approach to real-time and multimedia systems**. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, Vol. 3310*, pages 150–164, 1998. 5

[132] S. Ramos-Thuel and J.P. Lehoczky. **On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems**. pages 160 –171, dec. 1993. 84

[133] J. Real and A. Crespo. **Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal**. *Real-Time Systems*, **26**(2):161–197, 2004. 8, 10, 133

[134] John Regehr and John A. Stankovic. **HLS: A Framework for Composing Soft Real-Time Schedulers**. In *IEEE Real-Time Systems Symposium*, pages 3–14, 2001. 6, 90

[135] Saowanee Saewong, Ragunathan (Raj) Rajkumar, John P. Lehoczky, and Mark H. Klein. **Analysis of Hierar hical Fixed-Priority Scheduling**. *Real-Time Systems, Euromicro Conference on*, **0**:173, 2002. 6

[136] Luca Santinelli, Mangesh Chitnis, Christian Nastasi, Fabio Checconi, Giuseppe Lipari, and Paolo Pagano. **A Component-Based Architecture for Adaptive Bandwidth Allocation in Wireless Sensor Networks**. In *IEEE Symposium on Industrial Embedded Systems (SIES)*, 2010. 153, 154, 156, 171, 173

[137] Luca Santinelli, Mauro Marinoni, Francesco Prosperi, Francesco Esposito, Gianluca Franchino, and Giorgio Buttazzo. **Energy-Aware Packet and Task Co-Scheduling for Embedded Systems**. In *International Conference On Embedded Software (EMSOFT)*, 2010. 66, 69, 71, 82

[138] Curt Schurgers, Vijay Raghunathan, and Mani B. Srivastave. **Modulation Scaling for Real-Time Energy Aware Packet Scheduling**. In *Global Telecommunications Conferance (GLOBECOMM 01)*, pages 3653–3657, San Antonio, Texas (USA), 2001. 66

[139] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. **Mode change protocols for priority-driven preemptive scheduling**. *Real-Time Systems*, **1**(3):243–264, 1989. 9

[140] Lui Sha, John P. Lehoczky, and Ragunathan Rajkumar. **Solutions for Some Practical Problems in Prioritized Preemptive Scheduling**. In *IEEE Real-Time Systems Symposium*, pages 181–191, 1986. 84, 110

[141] I. Shin and I. Lee. **Compositional Real-Time Scheduling Framework**. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)*, pages 57–67, December 2004. 6, 19, 31

[142] Insik Shin and Insup Lee. **Periodic Resource Model for Compositional Real-Time Guarantees**. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, page 2, Washington, DC, USA, 2003. IEEE Computer Society. 6, 16

[143] Insik Shin and Insup Lee. **A Compositional Framework for Real-Time Guarantees**. In *ASWSD*, pages 43–56, 2004. 18, 31

[144] Insik Shin and Insup Lee. **Compositional real-time scheduling framework with periodic model**. *ACM Trans. Embed. Comput. Syst.*, **7**(3):1–39, 2008. 6, 31, 70

[145] Youngsoo Shin, Daehong Kim, and Kiyoung Choi. **Schedulability-driven performance analysis of multiple mode embedded real-time systems**. In *DAC '00: Proceedings of the 37th Annual Design Automation Conference*, pages 495–500, New York, NY, USA, 2000. ACM. 8

[146] Aviral Shrivastava, Eugene Earlie, Nikil Dutt, and Alex Nicolau. **Aggregating Processor Free Time for Energy Reduction**. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, pages 154–159, 2005. 37

[147] B. Sprunt, L. Sha, and J. P. Lehoczky. **Aperiodic Task Scheduling for Hard Real-Time Systems**. *Journal of Real-time Systems*, 1989. 6, 7, 16, 84, 172

[148] Brinkley Sprunt. **Aperiodic Task Scheduling for Real-Time Systems**. Technical report, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, 1990. 3, 7, 84

[149] Brinkley Sprunt, John P. Lehoczky, and Lui Sha. **Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm**. In *IEEE Real-Time Systems Symposium*, pages 251–258, 1988. 84

[150] Marco Spuri, Giorgio C. Buttazzo, and Scuola Superiore S. Anna. **Efficient Aperiodic Service under Earliest Deadline Scheduling**. pages 2–11, 1994. 85, 98

[151] J. A. Stankovic, K. Ramamritham, M. Spuri, and G. C. Buttazzo. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. 4

[152] Liesbeth Steffens and Gerhard Fohler. **Resource Reservation in Real-Time Operating Systems- a joint industrial and academic position**, 2003. 2

[153] N. Stoimenov, S. Perathoner, and L. Thiele. **Reliable Mode Changes in Real-Time Systems with Fixed Priority or EDF Scheduling**. In *DATE*, 2009. 9, 124, 133

[154] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo. **Resource Adaptations with Servers for Hard Real-Time Systems**. TIK Report 292, Computer Engineering and Networks Laboratory, ETH Zurich, ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-320.pdf, September 2010. 179, 181

[155] Nikolay Stoimenov, Lothar Thiele, Luca Santinelli, and Giorgio Buttazzo. **Resource Adaptations with Servers for Hard Real-Time Systems**. In *International Conference On Embedded Software (EMSOFT)*, 2010. 11, 177, 189, 190

[156] Jay K. Strosnider, John P. Lehoczky, and Lui Sha. **The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments**. *IEEE Trans. Comput.*, **44**(1):73–91, 1995. 7, 16, 84

[157] Vishnu Swaminathan and C. Chakrabarti. **Energy-Conscious, Deterministic I/O Device Scheduling in Hard Real-Time Systems**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **22**(7):847–858, 2003. 37

[158] Vishnu Swaminathan and Krishnendu Chakrabarty. **Pruning-based, Energy-optimal, Deterministic I/O Device Scheduling for Hard Real-time Systems**. *ACM Transactions in Embedded Computing Systems*, **4**(1):141–167, 2005. 37

[159] L. Thiele, S. Chakraborty, and M. Naedele. **Real-time calculus for scheduling hard real-time systems**. In *ISCAS*, **4**, pages 101–104, 2000. 22, 23, 45, 48, 91, 155, 172, 176

[160] L. Thiele, E. Wandeler, and N. Stoimenov. **Real-Time Interfaces for Composing Real-Time Systems**. In *EMSOFT*, pages 34–43, 2006. 6, 22, 27, 41, 158

[161] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. **Real-time Interfaces for Composing Real-time Systems**. In *International Conference On Embedded Software (EMSOFT)*, pages 34–43, 2006. 48, 156

[162] Deepu C. Thomas, Sathish Gopalakrishnan, Marco Caccamo, and Chang-Gun Lee. **Spare CASH: Reclaiming Holes to Minimize Aperiodic Response Times in a Firm Real-Time Environment**. In *ECRTS*, pages 147–156, 2005. 90

[163] K. W. Tindell, A. Burns, and A. J. Wellings. **Mode changes in priority pre-emptively scheduled systems**. In *RTSS*, pages 100–109, 1992. 9

[164] J. Trdlička, M. Johansson, and Z. Hanzálek. **Optimal Flow Routing in Multi-hop Sensor Networks with Real-Time Constraints through Linear Programming.** In *12th IEEE International Conference on Emerging Technologies and Factory Automation*, pages –, Piscataway, 2007. Institute of Electrical and Electronic Engineers. 153

[165] Marisol Garcia Valls, Alejandro Alonso, and Juan A. de la Puente. **Mode Change Protocols for Predictable Contract-Based Resource Management in Embedded Multimedia Systems**. *Embedded Software and Systems, Second International Conference on*, **0**:221–230, 2009. 10

[166] E. Wandeler and L. Thiele. **Real-Time Interfaces for Interface-Based Design of Real-Time Systems with Fixed Priority Scheduling**. In *EMSOFT*, pages 80–89, 2005. 27

[167] E. Wandeler and L. Thiele. **Optimal TDMA Time Slot and Cycle Length Allocation**. In *ASP-DAC*, pages 479–484, 2006. 8, 110, 172, 190

[168] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. **Performance analysis of greedy shapers in real-time systems**. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 444–449, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. 105

[169] Ernesto Wandeler and Lothar Thiele. **Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling**. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 80–89, New York, NY, USA, 2005. ACM. 48, 56, 59, 156

[170] Ernesto Wandeler and Lothar Thiele. **Interface-Based Design of Real-Time Systems with Hierarchical Scheduling**. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 243–252, Washington, DC, USA, 2006. IEEE Computer Society. 8, 22, 92, 174, 178

[171] Ernesto Wandeler and Lothar Thiele. **Interface-Based Design of Real-Time Systems with Hierarchical Scheduling**. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 243–252, April 2006. 47

[172] Ernesto Wandeler and Lothar Thiele. **Real-Time Calculus (RTC) Toolbox**. http://www.mpa.ethz.ch/Rtctoolbox, 2006. Available from: `http://www.mpa.ethz.ch/Rtctoolbox`. 60, 174, 190

[173] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. **System Architecture Evaluation Using Modular Performance Analysis - A Case Study**. *Software Tools for Technology Transfer (STTT)*, **8**(6):649 – 667, October 2006. 39, 174

[174] Chuan-Yue Yang, Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. **System-Level Energy-Efficiency for Real-Time Tasks**. In *the 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 266–273, 2007. 39, 69

[175] F. Yao, A. Demers, and S. Shenker. **A Scheduling Model for Reduced CPU Energy**. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995. 37

[176] Yumin Zhang, Xiaobo Hu, and Danny Z. Chen. **Task Scheduling and Voltage Selection for Energy Minimization**. In *Proceedings of the 39th ACM/IEEE Design Automation Conference (DAC)*, pages 183–188, 2002. 37

[177] Baoxian Zhao and Hakan Aydin. **Minimizing expected energy consumption through optimal integration of DVS and DPM**. In *ICCAD '09: Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 449–456, New York, NY, USA, 2009. ACM. 66

[178] Jianli Zhuo and Chaitali Chakrabarti. **System-level Energy-efficient Dynamic Task Scheduling**. In *Proceedings of the 42nd ACM/IEEE Design Automation Conference(DAC)*, pages 628–631, 2005. 39, 69