SYSTEM-LEVEL ENERGY MANAGEMENT FOR REAL-TIME SYSTEMS

by

Vinay Devadas
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

[illegible]	Dr. Hakan Aydin, Dissertation Director
[illegible]	Dr. Fei Li, Committee Member
[illegible]	Dr. Robert Simon, Committee Member
Brian L. Mark	Dr. Brian Mark, Committee Member
[illegible]	Dr. Hassan Gomaa, Department Chair
[illegible]	Dr. Lloyd J. Griffiths, Dean, The Volgenau School of Engineering
Date: 7/20/11	Summer Semester 2011 George Mason University Fairfax, VA

System-Level Energy Management for Real-Time Systems

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Vinay Devadas
Master of Science
George Mason University, 2007
Bachelor of Science
Visveswaraiah Technological University, 2005

Director: Dr. Hakan Aydin, Professor
Department of Computer Science

Summer Semester 2011
George Mason University
Fairfax, VA

Table of Contents

List of Tables

List of Figures

Abstract

SYSTEM-LEVEL ENERGY MANAGEMENT FOR REAL-TIME SYSTEMS

Vinay Devadas, PhD

George Mason University, 2011

Dissertation Director: Dr. Hakan Aydin

Energy management has recently become one of the key dimensions in the design of real-time embedded systems. While early studies focused separately on individual energy management techniques targeting different system components, there is growing interest in *system-level* energy management frameworks that exploit multiple techniques simultaneously.

A primary objective of this dissertation is the integration of two well-known energy management techniques Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM). With DVS, the supply voltage and clock frequency of the processor can be scaled down at run-time, to save CPU energy at the expense of increased task response times. On the other hand, DPM targets reducing the energy consumption of idle off-chip system devices such as disk and memory modules, by transitioning them to their low-power sleep states. While effective system-level energy management mandates the use of both DVS and DPM, their integration poses several challenges. For instance, minimizing device energy requires running the processor at high clock frequencies to maximize the length of device idle intervals in order to apply DPM, but minimizing CPU energy involves lowering CPU clock frequencies.

This dissertation first illustrates how the DVS and DPM techniques can be integrated optimally for a real-time application potentially using multiple devices during execution. An exact characterization of the system-level energy as a function of the CPU frequency is provided. Using this characterization, an efficient static algorithm is designed to determine the CPU frequency and device transitioning decisions that minimize system-wide energy without violating the timing constraints.

Second, the integration of DVS and DPM for real-time applications that consist of multiple periodic tasks is considered. The problem of optimally using DPM for periodic real-time tasks, even in the absence of DVS, is formally shown to be NP-Hard in the strong sense. Then, a novel DPM framework called *device forbidden regions* is proposed and feasibility tests for both fixed- and dynamic-priority periodic real-time systems are developed. Using this framework as a building block, unified energy management frameworks that efficiently combine DVS and DPM at the system level are proposed.

Third, the problem of managing system-wide energy for periodic real-time tasks running on emerging chip-multiprocessor systems with global voltage and frequency constraint is addressed. Contributions made in this area include selecting the number of cores to execute the workload and managing the global frequency at run-time across all cores to reduce dynamic energy while meeting the task deadlines.

A final contribution of this dissertation is the competitive analysis of online real-time scheduling problems under a given hard energy constraint. Specifically, worst-case performance bounds that apply to any online algorithm are derived, when compared to an optimal algorithm that has the knowledge of the input sequence in advance. Focusing on uniform value-density preemptive execution settings, optimal online and semi on-line algorithms achieving the best competitive factors are proposed. A number of additional fundamental results are provided for non-uniform value density, non-preemptive, and DVS-enabled execution settings.

Chapter 1: Introduction

Real-time embedded systems have become increasingly important in numerous application domains such as automated control and manufacturing, robotics, telecommunication networks, multimedia and military systems. In these applications, *timeliness* is as crucial as the correctness of results produced. More recently, *energy-awareness* has been promoted to a prime design and operational objective in these systems. This is, in part, due to the emergence of systems that have to rely on limited battery power. Also, with growing power densities, energy management is a highly desirable feature from economic, environmental, and system reliability perspectives. In real-time embedded systems, energy management should be achieved without compromising the critical *temporal predictability* guarantees.

Energy management can be performed at the architecture, compiler, operating system and application levels. Over the past decade, the research community has made significant progress in the area of low-power system design [61,99]. On the industry side, the *Advanced Configuration and Power Interface (ACPI)* standard has moved power management to the operating system level by providing system calls for predictive shutdown of system components [139]. Two well-established and widely-used energy management techniques for real-time systems are *Dynamic Voltage Scaling* (DVS) and *Dynamic Power Management* (DPM).

With DVS [11, 101, 103, 109, 124], the CPU clock frequency and supply voltage can be adjusted dynamically on the fly. Due to the convex relationship between the CPU power consumption and processor frequency, DVS helps to significantly reduce processor dynamic energy consumption at the cost of increased task response times. On the other hand, DPM [23,38,40,88,120] was proposed to reduce the off-chip device (primarily I/O device and main memory) energy consumption by transitioning devices to *low-power (sleep)* states when not in use. While it has an intuitive appeal, a primary challenge with DPM is to ensure that the non-trivial energy overheads associated with device state transitions do not offset the energy savings obtained during the device idle intervals.

While DVS and DPM are both important and each alone presents non-trivial difficulties in real-time settings, there is a need for *unified* frameworks that contain both techniques as essential components for *system-level energy management.* The proper integration of DVS and DPM techniques poses several challenges due to the fact that aggressive DVS schemes lead to short device idle intervals that limit the effectiveness of DPM solutions – similarly, solutions with DPM as the primary focus may lead to excessive CPU power consumption. Thus, there exists a trade-off spectrum between DVS and DPM at the system level.

The chip multiprocessors (CMPs) that offer multiple processing cores on a single chip have quickly become prevalent in the computing landscape. Major chip makers (e.g., Intel, AMD, Sun) have now several CMP lines with 2, 4 or 8 cores [48, 77, 95, 144]. Further, extensive research activity is underway to build chips with potentially hundreds of cores (or, many-core systems [26, 94]). This development has important implications for real-time embedded applications that will execute on these high performance architectures. Energy management has been a very active research area in the recent past and one of the main motivating factors leading to CMP architectures was the unsustainable ever-increasing frequency and power density trends of traditional single-core architectures [26]. As a result, CMPs come equipped with a variety of advanced power management features (including DVS and multiple idle states (e.g. *Halt, Sleep, Off* states)) [114] and most comply with the ACPI standard [139] endorsed by the industry. Effective energy management of real-time embedded applications on CMPs through coordinated DVS and core transitions is a relatively new and unexplored area.

Online algorithms have to make run-time decisions without the knowledge of future input. Competitive analysis is a well-known technique in theoretical computer science to evaluate the performance of online algorithms [27]. In general, competitive analysis aims at deriving worst-case performance guarantees of online algorithms by comparing it against a clairvoyant scheme which knowns the future input in advance [27]. Energy-constrained systems are those in which energy is a *hard constraint* (i.e. these systems have a limited and fixed energy budget within which they have to operate). A primary objective in energy-constrained systems is to effectively and efficiently utilize the available energy while maximizing a certain performance metric. While the online real-time scheduling problem has been analyzed using the competitive analysis framework [17, 18, 20, 98], the competitive analysis of *energy-constrained online real-time scheduling* has remained an open problem.

1.1 Problem Statement

In this dissertation, the following four problems are addressed.

1.1.1 Optimal Integration of DVS and DPM for a Frame-based Real-time Application

Given a periodic real-time application with a worst-case execution time and deadline/period, running on a DVS- and DPM-enabled platform, and using a set of devices during execution, the problem of determining the CPU frequency and device transitioning decisions so as to minimize the overall system energy is considered.

1.1.2 System-level Energy Management for Periodic Real-time Tasks

This dissertation considers also the system-level energy management problem for real-time applications that consist of multiple periodic tasks. The problem has two main dimensions:

- Investigating the computational complexity and tractability of the problem.
- Developing unified system-level energy management frameworks that combine both DVS and DPM components by considering their interplay across multiple tasks.

When exploring this problem, both fixed- and dynamic-priority real-time systems are considered. The solution frameworks need to assume a general energy model where the

CPU and device power consumptions, as well as device break-even times and transition overheads (in terms of both energy and time) are accounted for.

1.1.3 Energy Management of Periodic Real-time Tasks on Chip-Multiprocessors

Assuming chip-multiprocessors with *Global DVS* feature (commonly found in many state-of-art multi-core processor lines including Intel Itanium 2, Intel i5 series, Intel i7 series, Intel Core Duo, IBM Power 6 and IBM Power 7 [90, 91, 140, 141, 143]) where all active cores are constrained to operate at the same voltage and frequency level, the goal is to first statically determine an energy-optimal number of processing cores and partition the workload upon these. Then a comprehensive framework for run-time dynamic energy management of the selected cores should be sought. The framework should involve both coordinated global frequency scaling across active cores and performing core power state transitions when appropriate.

1.1.4 Competitive Analysis of Energy-Constrained Real-time Scheduling

Another objective of this dissertation is to undertake a preliminary investigation of the competitive analysis of energy-constrained online real-time scheduling for underloaded uni-processor systems where energy is the main limiting factor to execute the workload. Targeting *value-based real-time systems* and preemptive EDF scheduling, the goal is to investigate upper bounds on the competitive factor that can be achieved by any online algorithm, and existence of online algorithms that can achieve these bounds for various fundamental real-time workload models.

1.2 Contributions

For the problem of the *optimal integration of DVS and DPM for frame-based real-time applications*, the following contributions are made:

- Formal analysis and an exact characterization of the interplay between DVS and DPM for a frame-based real-time application that uses several devices is provided. By using the results from this analysis, an $O(m \log m)$-time algorithm (where m is the number of devices used by the application) is proposed to determine the optimal processor frequency and device transitioning decisions to minimize the system-wide energy consumption. To the best of the author's knowledge, this is the first research effort to not only investigate the exact interplay between DVS and DPM, but to also provide a provably optimal solution to the system-level energy management problem by taking into account DVS/DPM-related issues and device transition overheads under the given energy model. The performance of the optimal scheme is evaluated over a wide spectrum of system/application parameters using real device and CPU specifications. It is shown that the optimal scheme yields significant energy gains when compared to the existing sub-optimal approaches.

- The optimal solution is extended to deal with workload variability. The extended algorithm requires the knowledge of average-case execution time and minimizes average-case system energy while still guaranteeing the timing constraints. Through simulations, it is shown that this extension helps achieve significant energy savings in the presence of variability in dynamic execution behavior of the application.

The following contributions are made towards the *system-level energy management problem for periodic real-time tasks*:

- Optimally solving the DPM problem for periodic real-time tasks (even without DVS) is shown to be NP-Hard in the strong sense, closing a long standing open problem.

- A novel DPM framework for periodic real-time tasks called *device forbidden regions (DFR)* is proposed. In the DFR approach, devices are associated with long device idle intervals that are frequently enforced at run-time to help transition devices to their low-power states. Developing an exact feasibility test for the DFR framework is shown to co-NP-Hard in the strong sense. Sufficient feasibility conditions are derived to integrate the DFR-based DPM approach with both fixed- and dynamic-priority real-time scheduling policies.

- Using the DFR framework as a building block, two unified system-level energy management frameworks *DFR-RMS* and *DFR-EDF* are developed for fixed- and dynamic-priority real-time systems, respectively. The unified frameworks have both DVS and DPM components and further integrate them by taking into account their interplay. In the static phase, system-wide energy-efficient DFR parameters and CPU frequencies are derived. The run-time routines extend the duration of device idle intervals by combining DFRs and predictive techniques and exploit the run-time slack for both DVS and DPM simultaneously. Experimental evaluations with realistic processor and device specifications indicate that the proposed frameworks outperform the state-of-art schemes by significant margins. Also, to allow for non-preemptable device accesses, resource access protocols are integrated to the framework.

The following contributions are made towards the *energy management of periodic real-time tasks on chip-multiprocessors* problem:

- The dissertation proposes and evaluates several effective schemes with varying levels of complexity and performance for the problem of selecting the optimal number of processing cores for the execution of the real-time workload. Experimental results indicate that by limiting the number of cores to execute the workload, substantial energy gains can be obtained.

- The time-dependent *global energy-efficient frequency* is introduced and quantified, characterizing the boundaries of effective DVS when the voltage and frequency can be only globally controlled on a chip multi-processor system. The dissertation shows how to re-compute the global frequency at scheduling points, by taking into account the *active* cores and characteristics of tasks that will run in parallel upon them during the next execution interval. Deterministic rules are provided to put an idle core to

Sleep state and temporarily eliminate dynamic power, without violating the timing constraints or incurring excessive state transition overhead.

- The *Coordinated Voltage and Frequency Scaling (CVFS)* algorithm is devised in order to determine the feasible frequency while satisfying the energy-efficient execution across all cores. An enhanced version *CVFS** that adapts to the actual workload conditions at run-time is also proposed. The feasibility of both algorithms is formally demonstrated and their effectiveness in reducing the dynamic energy consumption is shown through simulations.

For the *competitive analysis of energy-constrained real-time scheduling* problem, the following contributions are made:

- For uniform value density settings where the value of a job is proportional only to its execution time, an upper bound on the competitive factor that can be achieved by any online algorithm is derived, and further, a variant of EDF algorithm called *EC-EDF* that achieves this upper bound is developed. If the online algorithm has a priori information about the largest job size in the actual input instance, then it is shown that there is an optimal semi-online algorithm ($EC\text{-}EDF^*$) with a competitive factor of 0.5. Performances of online algorithms in non-idling and non-preemptive execution settings are also analyzed, separately.

- The results are extended to more general settings with non-uniform value densities where there need not be a one-to-one correspondence between the value of a job and its execution time. Competitive analysis in settings where the processor has the DVS capability is also addressed and a semi-online algorithm $EC\text{-}DVS^*$ is proposed. $EC\text{-}DVS^*$ uses the additional knowledge of both the largest job size in the actual input instance and the *maximum loading factor* β [16], [70]. It is shown that the competitive factor of $EC\text{-}DVS^*$ is 0.5 times that of the optimal semi-online algorithm in these settings. The problem is also investigated under the resource augmentation technique [100],[72].

1.3 Dissertation Organization

This dissertation is organized as follows. Chapter 2 gives the relevant background and related work in real-time systems, power management, and competitive analysis research areas. Chapter 3 presents the fundamental system models and assumptions. Chapter 4 addresses the optimal integration of DVS and DPM for a frame-based real-time application. In Chapter 5, first a novel DPM framework for periodic real-time tasks, called *device forbidden regions*, is introduced. Using this framework, unified system-level energy management frameworks for both fixed- and dynamic-priority systems are developed. Chapter 6 explores the problem of system-wide energy minimization for periodic real-time tasks on chip-multiprocessor systems. Chapter 7 presents the competitive analysis of energy-constrained real-time scheduling and Chapter 8 gives the concluding remarks.

Chapter 2: Background

2.1 Real-Time Systems

In real-time systems preserving the timing constraints is of paramount importance. The workload in real-time systems is usually modeled as a set of *tasks*, where a task represents a unit of computation. A real-time task is characterized by a *release time* representing the time it becomes available for execution, a *worst-case execution time* which is the *maximum* amount of CPU time it requires, and a *deadline* indicating the time by which it must complete.

Real-time tasks are categorized as either *aperiodic* or *periodic*. *Aperiodic* real-time tasks are executed only once. On the other hand, *periodic* tasks consist of a stream of task instances (or, simply *jobs*) which are invoked repetitively. Periodic tasks are characterized by a *period* which represents the *minimum* separation time between two consecutive job releases. Usually, the relative deadline of periodic task instances coincides with its period. Most real-time systems are periodic in nature with examples including multimedia systems, control systems, avionics, among others [83]. Real-time systems that consist of both periodic and aperiodic tasks are called *hybrid* systems. Tasks in real-time systems may also have *precedence constraints* which impose a partial order on their execution.

Real-time scheduling policies guarantee the temporal constraints of a real-time system and can be classified as *preemptive* or *non-preemptive* depending on whether a running task can be interrupted or not by another task at run-time.

Most real-time systems can be classified as *hard* or *soft*. A *hard* real-time system is one where the temporal constraints cannot be compromised as doing so would have serious (and sometimes catastrophic) consequences. Examples of hard real-time systems include safety-critical control systems. In hard real-time systems timing guarantees are provided based on worst-case scenarios. On the other hand, in a *soft* real-time system, violation of the temporal constraints does not lead to very severe consequences, but degrades the overall performance. Examples of soft real-time systems include multimedia applications and on-line transaction systems.

A schedule is said to be *feasible* if and only if all tasks start execution no earlier than their respective release times and complete execution no later than their respective deadlines, while satisfying the *precedence constraints* (if any). A scheduling policy for real-time tasks is said to be *optimal* if and only if it can generate a feasible schedule whenever at least one such schedule exists.

2.1.1 Previous Work on Hard Real-Time Systems

Real-time scheduling of periodic tasks is a well-established area with two well-known strategies: dynamic priority policies and fixed priority policies. With dynamic priority policies, different task instances (jobs) of the same task may have different priority levels at run-time. Earliest Deadline First (EDF) is a well-known dynamic priority policy. Among all the ready jobs, EDF dispatches the one with the earliest deadline. Let C_i and P_i denote the worst-case execution time and period of a periodic task T_i, respectively. The utilization of the periodic task T_i is defined as $U_i = \frac{C_i}{P_i}$. In one of the seminal works in real-time scheduling [84], Liu and Layland established that a task set consisting of n periodic tasks is schedulable by preemptive EDF if and only if the total utilization of the task set is no more than unity (i.e. $\sum_{i=1}^{n} U_i \leq 1$). By achieving 100% system utilization, EDF is an *optimal* real-time scheduling policy. Least-Laxity First (LLF) is another optimal dynamic priority policy for hard real-time systems, though its run-time overhead is quite high [92].

On the other hand, with fixed priority policies, all jobs of a particular task have the same priority level. A well-known example is Rate Monotonic Scheduling (RMS) wherein tasks with smaller periods have higher priority. In [84] it was shown that given n periodic tasks, RMS is guaranteed to schedule the task set in a feasible manner if the total utilization is no more than $n(2^{\frac{1}{n}} - 1)$. Later in [25], this result was extended and a less pessimistic sufficient condition for RM schedulability in the form of $\Pi_{i=1}^{n}(U_i + 1) \leq 2$ was derived. In [81], Lehoczky et al. provided an exact characterization for RM schedulability through a necessary and sufficient pseudo-polynomial schedulability test. The relative merits and demerits of dynamic and fixed priority systems are discussed in [32] for multiple dimensions including ease of implementation, overhead issues, system utilization, and resilience in overloaded settings.

Several scheduling policies exist for hybrid systems consisting of both periodic and aperiodic tasks. In these solutions, the key idea is to allocate a periodic process (called *server*) whose unique purpose is to execute aperiodic tasks. A certain time allocation (also called its *capacity* or *bandwidth*) is made for the server such that the feasibility of the periodic task set is not compromised. Several solutions for server capacity allocation and aperiodic task scheduling have been developed, including Polling Server [83], Deferrable Server [80], Sporadic Server [117], Constant Utilization Server [44] and Total Bandwidth Server [118].

[69] studies the problem of non-preemptive scheduling for periodic real-time tasks. Non-preemptive periodic hard real-time scheduling is shown to be NP-Hard in the strong sense. Further, its is also shown that EDF remains optimal even under non-preemptive settings provided that the scheduler is *work-conserving* or non-idling (i.e. the CPU cannot idle in the presence of ready jobs). Real-time tasks may also need to share system resources which are non-sharable and non-preemptable. In such cases, coordinated access to resources while guaranteeing system feasibility is necessary. Several resource access protocols such as Priority Inheritance Protocol [112] Priority Ceiling Protocol [112] and Stack Resource Policy [13] have been proposed to address this dimension.

Real-time scheduling on multiprocessor systems has been also extensively studied. In [93], Mok and Dertouzos showed that real-time scheduling algorithms that are optimal for uni-processor systems remain no longer optimal in multiprocessor environments. Further, multiprocessor scheduling is known to be a NP-Hard problem [56]. Well-known approaches to multiprocessor real-time scheduling are *global* and *partition-based* schemes. In *global* scheduling all tasks are stored in a global queue and the same number of highest priority tasks as the number of processors are selected for execution at run-time [7, 57, 78]. With global scheduling tasks can start execution on one processor and resume execution on another processor. In general, global scheduling based frameworks incur migration- and cache-affinity-related problems.

On the other hand, with *partitioned* scheduling, tasks are allocated to processors and each task executes only on its assigned processor [28,86,97]. Partitioned scheduling has low run-time overhead and reduces multiprocessor scheduling problem to a set of uniprocessor scheduling problems after task allocation. Optimal partitioning of tasks to processors is a well-known NP-Hard problem [56]. It is known that there exist task sets with total utilization slightly exceeding $\frac{k}{2}$, and not admitting a feasible partition on k processors [33]. However, in the average-case several bin-packing heuristics (First-Fit, Next-Fit and Best-Fit) perform reasonably well. A relative performance analysis of these heuristics is given in [28,86,97].

Recently, *semi-partitioned* scheduling was proposed [1–3] where most tasks are statically allocated to processors through partitioned scheduling, while a few tasks are split to sub-tasks and assigned to multiple processors. This improves the resource utilization and feasibility bounds compared to partition-based schemes, at the cost of increased run-time overhead.

2.1.2 Previous Work on Soft Real-Time Systems

For *underloaded* real-time systems, where a feasible schedule for the workload is known to exist, the preemptive Earliest Deadline First (EDF) algorithm is optimal in the hard real-time sense, meaning that it is guaranteed to meet all the deadlines even when it processes and schedules jobs as they arrive, without any knowledge of future release times [45]. A real-time system is said to be *overloaded*, if there does not exist a schedule where all jobs meet their deadlines. In these settings where deadline misses are unavoidable, the goal is typically to maximize a soft real-time performance metric which quantifies the quality degradation with increasing number of deadline misses. There are several well-known paradigms in soft real-time scheduling.

In *value-based scheduling*, each job is associated with a *value* which quantifies its contribution to the overall system performance. The value of the job is added to the overall performance metric (*cumulative,* or, *total* system value) if and only if it meets its deadline – no value is accrued for partial executions that are not completed before the task deadline [17,18,31]. The objective is to maximize the total value of jobs completed by their deadlines.

In *reward-based scheduling*, each task consists of *mandatory* and *optional* parts. While completing the *mandatory* parts of tasks is crucial for proper functioning of the system, execution of the *optional* parts is linked to the quality of the output. The quality of the output is expressed as a *reward* function of the length of the executed optional task segments. Thus, the scheduling problem is to complete the *mandatory* execution segments while maximizing the total reward from *optional* executions [43, 47, 51, 54].

The *weakly-hard real-time model* is motivated by the observation that for many real-time applications some deadline misses are acceptable as long as they are spaced distantly/evenly. Examples of such systems include multimedia and real-time communication. The general (m, k)-firm deadline model requires that each task meets at least m deadlines in *every* k consecutive instances [60, 104, 106].

The *rate adaptation model* (also known as the *elastic model*) provides flexibility to change task execution rates to provide different quality of service. This model can handle overload situation in a more flexible way by reducing task periods and provides an efficient method to handle system quality of service as a function of the current load. The basic idea in this framework is to model each task as a spring with a given rigidity coefficient and length constraints [29, 30].

2.2 Energy Management for Real-Time Systems

Many embedded devices are battery-operated and hence, have limited energy supply. Due to the growing demand for smaller devices with longer battery life, energy management has become one of the major goals in embedded systems research. Many applications running on power-limited systems (such as embedded controllers) are subject to timing constraints. As a result, the real-time and energy-aware operation is a highly desirable and sometimes critical feature of a real-time embedded system. Further, studies in [52,53] posit that power management is both crucial and necessary in large server systems and data centers for technical, financial, and environmental reasons. Leading companies such as IBM, Google, Microsoft and Sun have already initiated and moved towards *energy-efficient computing* (also called *Green Computing*), to reduce overall energy and cooling costs, and to minimize the "carbon footprint".

Dynamic Voltage Scaling [11,101,103,109,124] is a popular and widely-used technique for power management in real-time embedded systems. With the DVS technique, one can lower the supply voltage and operating clock frequency of the processor [124] to reduce CPU dynamic energy. Most commercially available modern processors, including recent multicore systems are equipped with DVS feature, such as Intel XScale, Transmeta Crusoe, Intel i-series multicore processors, AMD Barcelona and Sun Nehalem based multicore processors. Changing the processor voltage and frequency through DVS involves overheads both in terms of time and energy. However, such overheads are low (in the order of μsecs and μJoules [24]). Further, the technology trends indicate that these latencies are bound to decrease [24].

Research studies dealing with DVS capable processors consider two models: *continuous speed* and *discrete speed*. In the *continuous speed* model, the processor frequency f can be adjusted up to a certain maximum frequency level f_{max}, while in the *discrete speed* model the processor is provided with a finite set of k frequency values $(f_1, \ldots, f_k)$ in which it can operate. The *discrete speed* model is more practical, but the results derived assuming the *continuous speed* mode can be adapted to the *discrete speed* model in several ways [24, 67].

Dynamic Power Management [23, 38, 40, 88, 120] is another commonly used energy management technique, aiming at reducing energy consumption of off-chip devices such as main memory and I/O modules. Typical devices have an *active* state in which they process requests and at least one low-power *sleep* state. DPM involves transitioning devices to low-power states when not in use so as to reduce the device energy consumption [88]. Memory modules and I/O devices which consume significant energy have been the primary targets of DPM. However, non-trivial time and energy overheads are associated with each device state transition. As a consequence, transitioning devices to low-power states is energy-efficient only when the device idle interval is guaranteed to be greater than a certain threshold (typically called the *device break-even time*).

In real-time systems, meeting the timing constraints is of paramount importance. Application of DVS results in increased execution times when reducing the processor frequency for energy management. Similarly, on the DPM side, devices are associated with non-trivial transition times that could lead to potential deadline violations if a real-time application incurs delay while waiting for a transitioning device. Both these issues imply that provisions must be taken to avoid deadline misses when applying DVS and/or DPM techniques to minimize energy in real-time systems.

The trade-offs involved in the systems with both DVS and DPM features warrant a detailed analysis. The DVS technique primarily targets CPU energy minimization, while DPM policies aim at reducing device energy consumption. More recently the research community has become more focused on minimizing the *system-wide energy* that includes the energy consumed by both the CPU and system devices.

2.2.1 Previous Work on Energy Management for Uniprocessor Hard Real-Time Systems

CPU Energy Management

The problem of minimizing the CPU dynamic energy consumption while guaranteeing the timing constraints is known as the *RT-DVS* problem. For aperiodic tasks, Yao et al. [130] proposed a polynomial optimal static offline solution assuming tasks execute their worst-case workload. [64] provided heuristics for on-line scheduling of aperiodic tasks in the presence of periodic tasks. Non-preemptive power-aware scheduling was investigated in [63]. Slowing down the processor when a single task is available for execution was investigated in [113]. A static solution to the periodic model where tasks have different power characteristics is given in [9].

Most of the above solutions assume that tasks exhibit their worst-case workload. However, in practice the *actual* execution behavior of real-time tasks are typically far from worst-case [50]. As a consequence, at run-time, the unused CPU time (also called *slack*) due to task early completions can be used to further reduce the CPU energy through application of DVS. This dimension of reclaiming slack at run-time and using it for further reducing the CPU frequency is called *dynamic reclaiming.* Research efforts in [11, 101] studied this problem for the periodic task model and presented energy-efficient dynamic reclaiming policies which preserve the system's temporal constraints.

[11] suggested the *Generic Dynamic Reclaiming Algorithm* (GDRA) and *Aggressive Speed Adjustment* (AGR) policies. In GDRA, the slack of high priority tasks were transferred to low-priority tasks which allowed further reduction in CPU operational frequency at dispatch time. In the AGR scheme, the CPU frequency was set based on the execution history of tasks. In [101], the *Cycle-Conserving* algorithm was based on the concept of *dynamic utilization.* In this scheme, the system's dynamic utilization is updated at run-time based on tasks' actual execution times. This dynamic utilization value is then used to set the CPU operational frequency. [101] also proposed the *Look-Ahead* algorithm which was

based on the concept that in the presence of slack, it is typically better to start out at low frequencies and defer the use of high frequencies as much as possible.

[109] studies the problem of real-time power-aware scheduling in fixed-priority and periodic real-time systems. Static solutions assuming worst-case behavior and dynamic reclaiming policies were proposed. [103] studies the problem of *RT-DVS* for the sporadic task model. In [79], the authors present a dynamic reclaiming policy for the online scheduling of aperiodic tasks. In [132], the authors consider the problem of *RT-DVS* and dynamic reclaiming in the presence of shared resources and non-preemptable sections.

Most of the above studies apply DVS only at task dispatch times or completion times. This paradigm is referred to as *inter-task DVS*. On the other hand, *intra-task DVS* provides the flexibility of applying DVS during task execution. Intra-task DVS is usually effective only when probabilistic information about the probabilistic distribution of task's workload is available. A number of research efforts have analyzed the problem of applying intra-task DVS to hard real-time systems [87, 127]. The major limitations of intra-task DVS are high run-time overhead and the need for some kind of compiler support.

Device Energy Management

One of the primary difficulties associated with the use of DPM is to decide when to switch a device to a low-power state. DPM techniques can be classified as stochastic, predictive and timeout-based [23]. In real-time systems, the predictive DPM techniques are commonly used. The predictive techniques involve making accurate predictions about the next usage time of idle devices. As such, predicting the next device usage time is of critical importance in real-time systems. Under-estimations may lead to inefficient energy management (as devices would not be put to low-power states) and over-predictions may lead to potential deadline misses (due to the non-trivial transition delays).

The problem of applying DPM techniques to minimize device energy in hard real-time systems is known as the RT-DPM problem. A number of research groups recently tackled the RT-DPM problem. For example, [119, 121] present heuristic-based RT-DPM schemes that can be used with both EDF and RMS scheduling polices. However, the schemes consider only non-preemptive real-time task execution. In [120] the same group of authors present an offline scheme, called *Maximum Device Overlap (MDO)*, for preemptive task scheduling. MDO groups task executions such that tasks with common devices are executed next-to-next. This provides both long device usage and idle periods thus helping significantly reduce device energy consumption. MDO involves very high time complexity and it cannot be successfully adapted to dynamic/online settings where job release and execution times can vary considerably.

For such systems, the online dynamic power management becomes an imperative. The University of Nebraska-Lincoln Real-Time Systems research group exploited various aspects of this problem, mostly within the periodic EDF scheduling framework. In [38, 41], the authors proposed an online RT-DPM scheme *Conservative Energy-Efficient Device Scheduling* (CEEDS) based on next device usage predictions. In CEEDS, when a device is no longer in use, its next usage time is predicted as the minimum of the earliest next release times of all the tasks using that device. Though conservative, this prediction mechanism provides an effective tool for performing DPM at run-time. Further, CEEDS can be used in conjunction with DVS. In [40], the authors present a more sophisticated RT-DPM policy, *Energy Efficient Device Scheduling (EEDS)*, which exploits task and device slacks to create long device idle intervals, again for preemptive EDF. EEDS is a DPM-only scheme with no DVS component. In [39], EEDS is extended to include non-preemptive shared resources.

System-Level Energy Management

While DVS and DPM are popular techniques targeting energy minimization in CPU and external devices respectively, a comprehensive system-level energy management policy is likely to use both DVS and DPM. However, integrating DVS and DPM in a single framework poses several challenges. With DVS, low processor frequencies lead to low CPU dynamic energy consumption figures. However, this also results in elongated task execution times and shortened device idle intervals. This not only forces devices to remain in active state for longer periods but also limits the possibility of transitioning devices to *sleep* states (as shortened idle intervals will tend to be smaller than the device break-even times). On the other hand, running the processor at higher frequencies reduces the device energy and creates more opportunities for transitioning devices to *sleep* states, at the cost of increased CPU energy and transition energy. Thus, there is an intriguing trade-off spectrum covering the benefit/cost spaces of DVS and DPM techniques.

Recently, a number of research efforts addressed system-wide energy consumption issues for real-time embedded systems. In [71], the concept of *critical frequency* (or, *energy-efficient frequency*) was introduced. This stems from the observation that lowering the processor frequency below a certain threshold can have negative effects on the system-wide energy consumption. This is because while lower frequencies provide significant reduction in CPU energy, they also extend task execution times thereby increasing device energy as devices used by tasks will need to remain in active mode for prolonged time intervals. Critical frequency represents the CPU frequency below which the decrease in CPU energy is over shadowed by the increase in device energy.

The energy-efficient frequency is calculated by considering both the device energy and CPU energy consumed during task executions. Each task can potentially have a unique energy-efficient frequency, depending on the devices it uses during its execution. In [71], the authors provide a single policy to manage both processor leakage energy and device energy. In [138], the authors propose a dynamic task scheduling algorithm using the concept of critical frequency which minimizes the system-wide energy consumption.

In [10], the authors consider a generalized power model taking into account several factors such as workload characteristics, effective switching capacitance figures, and frequency-dependent and frequency-independent power components. For this generalized power model, the authors show how to derive the task-level energy-efficient frequencies and propose an $O(n^3)$ algorithm to optimally solve the system-wide energy minimization problem for n periodic hard real-time tasks. In [133], the authors address the energy minimization problem assuming a DVS processor with limited number of frequency values. Recently, other research groups addressed the CPU and memory energy minimization problems [116, 134].

While the concept of critical frequency helps mitigate the negative impacts of DVS on the system-wide energy, one major drawback of most system-level real-time energy management research efforts is that they either assume negligible device transition overheads or provide no DPM policies. A recent research effort that combines both DVS and DPM in the same framework while taking into account the device transition overheads and DPM related issues is given in [38], where the authors propose a practical system-level energy management heuristic called SYS-EDF for periodic real-time tasks and discrete model DVS capable processor. SYS-EDF performs DVS using the concept on energy-efficient speed scaling and uses the prediction based CEEDS policy for DPM decisions.

2.2.2 Previous Work on Energy Management for Multiprocessor and Multicore Real-Time Systems

Research studies on energy management for multiprocessor real-time systems are typically based on independent DVS capabilities of individual processors. In [12], the authors considered the problem of minimizing CPU dynamic energy with partitioned multiprocessor scheduling and EDF policy. They showed that the problem of energy-optimal partitioning is NP-Hard in the strong sense even when the total workload can fit on a single CPU. The same paper also indicated that more balanced partitions typically yield better energy savings and suggested the use of Worst-Fit Decreasing (WFD) partitioning scheme to balance the load. [4] re-considered the problem for fixed-priority systems and RMS policy.

Exploiting potential and actual early task completions have been another focus point for multiprocessor systems. In [135], the authors investigate dynamic reclaiming strategies for global scheduling of frame-based tasks on homogeneous multiprocessors. [37] provides a 1.13-approximation algorithm for the problem of partitioning tasks to minimize the expected energy consumption. In [128], the authors considered the problem of energy-efficient partitioning in heterogeneous multiprocessor platforms.

The increasing frequency and power density trends of traditional single-core architectures have led to multicore architectures with several processing units on a single chip. These multi-core architectures (also called *chip-multiprocessors (CMP)*) are provided with advanced features including DVS and multiple low-power states to facilitate effective power management.

The emerging CMP platforms have a number of unique traits which make the problem different from the multi-processor platforms. For example, while it is natural to have different voltage levels per CPU (hence, per-CPU DVS capability) in a multi-processor system, the tight coupling of cores on a single chip (CMP) implies that the per-core DVS feature would come with severe additional circuit complexity, stability, and power delivery problems [26, 62, 94]. In fact, in the state-of-the-art commercial CMP lines the processing cores share a common voltage level. A recent study [62], based on detailed VLSI circuit simulations, suggests that the potential energy gains of per-core DVS are likely to remain too modest for justifying the complicated design problems. [66] independently reaches the conclusion that per-core DVS's additional energy gains would be not be substantial for reasonably-balanced workload-to-core distributions. For the next-generation many-core systems, it is likely that only a small number of clusters/blocks each with several cores and independent voltage regulators, will be possible [26]. Independent and effective management of such clusters (or, the so-called voltage islands) would be the ultimate objective in these next-generation systems [26, 62, 105].

Energy management of real-time tasks on CMP under the global DVS constraint has started to attract the attention of the research community more recently. In [129], assuming a frame-based system where all tasks have the same deadline, the authors showed the problem is NP-Hard and provided a 2.371-approximation scheme for this simple task model. In [22], the authors proposed a power-aware scheduler for multicore systems executing a soft real-time workload. In [73] the authors consider a CMP system running a single real-time application modeled as a directed acyclic communication task graph (CTG). The authors effectively deploy two techniques to save energy: DVS to reduce the dynamic energy and power shutdown of the entire system to reduce static energy.

[110] considered the problem of energy-efficient scheduling for periodic hard real-time tasks on CMP systems. The authors proposed a scheme to re-partition tasks at run-time by resorting to task migrations, so as to create more balanced schedules that adapt to dynamic workload variability. Further, they also proposed a dynamic core scaling algorithm adjusting at run-time the number of active cores to reduce static power under the assumption that transitions between off and active states can be done instantaneously and with no additional overheads. However, in practice such transitions are rarely attractive or possible for periodic real-time applications. Moreover, they do not consider the frequency-independent component of dynamic power (effectively ignoring the energy-efficient frequency). Also, the overhead of frequent task migrations may be a concern in practice.

2.3 Competitive Analysis of Online Real-Time Scheduling

An algorithm is said to be *online* if it must make its decisions at run-time, without having any information about the future input. Design and analysis of online algorithms is a well-established field with direct applications in a wide range of areas such as load balancing, scheduling, circuit design, server performance evaluation, memory hierarchy design, search, portfolio selection, and revenue management [27]. In online settings, the performance of an algorithm is often assessed by comparing it to that of an optimal and clairvoyant algorithm that knows the entire input in advance. This framework, known as competitive analysis, is considered as a standard analysis and evaluation tool in Computer Science [27, 102].

An online scheduling algorithm is said to have a *competitive factor* r, $0 \leq r \leq 1$, if and only if it is guaranteed to achieve a cumulative value at least r times the cumulative value achievable by a clairvoyant algorithm on any finite input job sequence [17],[31],[75]. Formally, if an online algorithm $\mathcal{A}$ has a competitive factor of r, then over any finite input sequence $\mathcal{I}$, the following holds:

$$\frac{\textsf{Value of } \mathcal{A} \textsf{ over } \mathcal{I}}{\textsf{Value of Optimal Offline Algorithm over } \mathcal{I}} \geq \textsf{r}$$

An online algorithm is said to be competitive, if it has a constant competitive factor strictly greater than zero. In general, the higher the competitive ratio, the better performance guarantees provided by an online algorithm. As such, the competitive analysis technique aims to establish performance bounds that hold even under worst-case scenarios for online algorithms, when compared to an optimal clairvoyant algorithm [27],[102].

Another competitive analysis technique is to explore the impact of giving some (but still limited) information to online algorithms about the actual input sequence (actual job set), in an effort to improve its competitiveness. Such algorithms are known as *semi-online* algorithms [102]. For example, a priori information about the largest job size that will appear in the actual input sequence typically enables the design of semi-online algorithms with improved competitive ratio [102].

2.3.1 Previous Work on Competitive Analysis of Online Real-Time Scheduling

A significant body of research has been devoted to the analysis of online scheduling algorithms for overloaded real-time systems, where the goal is to maximize the total system value [17],[18],[20],[75]. In a seminal paper, Baruah et al. showed that no online algorithm can achieve a competitive factor greater than 0.25 [17], in an overloaded real-time system. This result holds for task systems with *uniform density* settings, where the value of a job is proportional to its execution time. For more general *non-uniform value density* settings, where different tasks may contribute different values per execution time, the bound is much smaller. Consider a firm real-time task system where the job J_i accrues k_i units of value per execution time (value density), if it completes by the deadline. Then, no online algorithm can achieve a competitive factor greater than $\frac{1}{(1+\sqrt{k})^2}$, where $k = \frac{max(k_i)}{min(k_i)}$ is the *importance ratio* obtained through the largest and smallest value densities in the task set [17]. Note that for $k = 1$ (uniform density settings), the bound evaluates to 0.25.

Koren and Shasha provided an optimal online algorithm D^{over} that achieves this upper bound [75]. The same authors also considered extensions to multiprocessor settings [76]. Several other studies addressed competitive online real-time scheduling for imprecise computation tasks [21], tasks with bounded slack factors [20], and tasks with given stretch metrics [98].

2.3.2 Previous Work on Competitive Analysis of Online Energy-Aware Real-Time Scheduling

Theoretical investigation of online energy-aware real-time scheduling under the competitive analysis framework has been addressed in several research efforts. In [130], the authors study the problem of minimizing the total energy used to complete all tasks subject to feasibility constraints. They give an offline greedy algorithm (*YDS*) that optimally solves the problem. [130] also proposes two online schemes, *Average Rate (AVR)* which runs each task in the optimal manner assuming that it is the only task in the system and *Optimal Available (OA)* which at any point of time schedules the unfinished work optimally under the assumption that no more tasks will arrive. While [130] presents the competitive analysis of *AVR*, the competitiveness of *OA* is investigated in [15]. In [15], the authors introduce an online algorithm *BKP* which has a better competitive factor compared to both *AVR* and *OA*. [34] studies the competitive analysis of energy-efficient real-time scheduling in overloaded conditions.

Chapter 3: Models and Assumptions

3.1 Processor Model

In this research, both uni-processor and emerging modern chip-multiprocessor systems are considered. The platforms under consideration may or may not have DVS capability. If DVS is not available, all tasks are executed at a constant frequency level. For DVS-enabled systems, it is assumed that the frequency (f) can be adjusted up to a maximum level f_{max}. For convenience, all frequency values are normalized with respect to f_{max}. *Inter-task DVS* is assumed throughout this dissertation. When the CPU is active and executing tasks, power consumption depends on the operational frequency level of the processor. On the other hand, when the CPU is idle and not executing tasks, it enters a *low-power (idle/sleep)* mode.

Chip-multiprocessors (CMPs) have several processing units (called cores) on a single chip. All available cores are divided into groups called clusters (or blocks) and each cluster is powered by an independent voltage regulator. In current technologies, all cores within a given cluster are typically constrained to operate at the same supply voltage.

3.2 Workload and Scheduling Model

This dissertation considers both aperiodic and periodic task models. For the periodic task model a task set $\psi = \{T_1 \ldots T_n\}$ with n independent periodic tasks is considered. P_i denotes the period of T_i. The relative deadline of T_i is assumed to be equal to its period and $T_{i,j}$ denotes the j^{th} instance of task T_i. A task instance is referred to as a *job*. C_i denotes the worst-case execution time of T_i at f_{max}. The *base utilization* of a periodic task set is given by $U_{tot} = \sum_{i=1}^{n} \frac{C_i}{P_i}$. The *hyperperiod* of the periodic task set is defined as the least common multiple (LCM) of all the task periods.

On DVS-enabled systems, it is assumed that task execution times scale linearly with CPU frequency. Thus, the worst-case execution time of T_i at frequency f is given by $C_i(f) = \frac{C_i}{f}$. In this dissertation, *preemptive* scheduling is considered. It is assumed that the task set ψ is feasible when executed at f_{max}. Preemption overheads are assumed to be negligible; if not, they can be incorporated into the tasks' worst-case execution times [83].

The aperiodic task model is considered in Chapter 7 at the beginning of which the related terminology is presented.

3.3 Device Model

The system is assumed to have a set of m devices represented by $\mathcal{D} = \{D_1 \dots D_m\}$. Each device is assumed to have (at least) two states: an *active* (working) state and a *sleep (low-power)* state. In accordance with the previous RT-DPM research [38, 40, 120, 121], throughout this research the *inter-task device scheduling* assumption is made. According to this assumption, all devices needed by a task may be forced to remain in *active* state when the task executes. As the exact times at which a running task generates a request for a device cannot be known in advance and device state transition delays are often significant, the inter-task device scheduling paradigm is considered realistic for energy modeling and minimization in real-time systems research [38, 40, 120].

The following parameters are associated with each device D_i:

- P_a^i: The device power consumption in *active* state
- P_s^i: The device power consumption in *sleep* state
- T_{sd}^i and T_{wu}^i: The device state transition times (from *active* to *sleep*, and from *sleep* to *active*, respectively)

- E_{sd}^i and E_{wu}^i: The device transition energy overheads (from *active* to *sleep*, and from *sleep* to *active*, respectively)

Let $T_{sw}^i = T_{sd}^i + T_{wu}^i$ and $E_{sw}^i = E_{sd}^i + E_{wu}^i$. Given that devices are associated with non-zero transition costs, the device *break-even time* B_i denotes the minimum length of the idle period which justifies a device transition from *active* to *sleep* state. B_{actual}^i denotes the minimum idle interval length during which keeping D_i in *active* state consumes the same amount of energy as transitioning D_i from *active* to *sleep* and back from *sleep* to *active*. Thus, $B_{actual}^i = \frac{E_{sd}^i + E_{wu}^i - T_{sw}^i . P_s^i}{P_a^i - P_s^i}$.

In other words, B_{actual}^i characterizes the minimum idle interval length for *energy-efficient device state transitions*. Further, the device idle interval should be long enough to allow the device transitions from *active* to *sleep*, and from *sleep* to *active* states, implying that device break-even times cannot be shorter than T_{sw}^i. Hence, the device break-even time B_i is given as $B_i = max(B_{actual}^i, T_{sw}^i)$ [40, 88], yielding:

$$B_i = max(\frac{E_{sd}^i + E_{wu}^i - T_{sw}^i \cdot P_s^i}{P_a^i - P_s^i}, T_{sd}^i + T_{wu}^i)$$

$\gamma_i \subseteq \mathcal{D}$ denotes the set of devices used by task T_i, while β_i denotes the set of tasks requiring device D_i for execution. Thus, $\beta_i = \{T_j \mid D_i \in \gamma_j\}$.

3.4 Energy Model

System energy consumption can be divided into static energy and dynamic energy components. The static power (P_{static}) is needed for purposes such as keeping the clock running, maintaining the basic circuits and keeping the devices in *sleep* states. Due to the periodic nature of real-time tasks and the significant delays involved in completely turning off CPU and other components, static energy is considered to be not manageable for uni-processor systems. However, on CMP platforms by keeping only a subset of the available cores active, one can effectively manage the static energy. Below the energy model for uni-processor systems is given. In Chapter 6, the model is extended to CMP platforms before presenting the research results in that direction.

All system devices and CPU contribute to the overall dynamic energy consumption (E_{system}) which is expressed as:

$$E_{system} = E_{cpu} + \sum_{i=1}^{m} E^{i}_{device}$$

E_{cpu} represents the energy consumed by the processor while executing the real-time tasks and E^{i}_{device} corresponds to the overall energy consumption due to a specific device D_i.

The processor dynamic power consumption P_{cpu} depends and the supply voltage V and the CPU clock frequency f. Specifically, $P_{cpu} = a \cdot V \cdot f^2$ where 'a' is the switching capacitance [124]. In DVS technique, V is adjusted alongside with f in linear fashion. Thus, P_{cpu} is modeled as a cubic function of its clock frequency, i.e. $P(f) = a \cdot f^3$ [11, 101, 103, 109, 124]. The energy $E_i(f)$ consumed in executing task T_i for C_i units at frequency f is thus given by: $E_i(f) = P_{cpu} \cdot \frac{C_i}{f} = a \cdot f^2 \cdot C_i$.

E^{i}_{device} includes three components:

- E^{i}_{fixed}: The energy consumed by D_i when *active* and in use by tasks. This component depends on the execution times of tasks using D_i.

- E^{i}_{trans}: The energy overhead involved in transitioning D_i between *active* and *sleep* states during execution.

- E^{i}_{mod}: The energy consumed by D_i when *active* and not in use. As a consequence of transition time/energy overheads, a device not in use may be forced to remain in *active* state when the estimated length of the idle interval is shorter than its break-even time.

Thus, the overall system dynamic energy consumption can be expressed as:

$$E_{system} = E_{cpu} + \sum_{i=1}^{m} (E^{i}_{fixed} + E^{i}_{mod} + E^{i}_{trans})$$

Chapter 4: Optimal Integration of DVS and DPM for a Frame-based Real-time Application

4.1 Introduction

This chapter considers the problem of optimally integrating DVS and DPM for a frame-based real-time application. Specifically, given a real-time application which uses a certain set of devices, the problem is to determine the optimal CPU clock frequency and device transitioning decisions to minimize the system-wide energy consumption. The real-time application has a worst-case execution time (WCET) of C units at maximum CPU frequency f_{max}. It is assumed to be invoked periodically, where the period coincides with the relative deadline P of the invocation. The interval $[(k-1) \cdot P, k \cdot P]$ is called the k^{th} *frame* of the execution [107, 126].

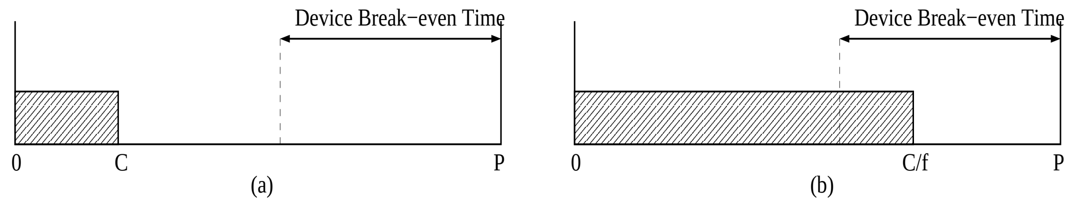


Figure 4.1: DVS and DPM trade-off

When using DVS, the completion time of the task ($\frac{C}{f}$) falls in the interval $[C, P]$ (Figure 4.1). This frequency assignment has obviously serious consequences for the applicability of DPM, and hence for overall system energy. The *slack* refers to the unused CPU time between the completion time of the application and the beginning of the next frame, at each invocation. Formally, the slack of the application at frequency f is given by $\delta(f) = P - \frac{C}{f}$.

In Figure 4.1(a), the application is executed at f_{max} resulting in maximum CPU energy consumption and minimum device energy consumption. This scenario maximizes slack: $\delta(f_{max}) = P - C$. As the CPU processing frequency decreases below f_{max}, the CPU energy decreases at the cost of simultaneously increasing device active energy because the devices required by the application are forced to remain in active state for longer periods of time. However, as long as the slack is larger than the device break-even time, the device can still be transitioned, but by incurring transition energy overhead. At certain low frequency values, DVS elongates application execution time to the extent the slack is no longer sufficient to transition the associated device in energy-efficient manner (Figure 4.1(b)). In that case, while device state transition energy disappears and the CPU energy is significantly reduced, device active energy consumption is maximized. This results in maximum device energy consumption over the frame. Thus, one can see an inherent trade-off between DVS and DPM at the system level.

While the existence of multiple devices with different power characteristics and break-even times complicate the problem, to better understand the DVS/DPM trade-offs, first a simplified model where the real-time application uses a single device is considered and several non-trivial observations are made that lead to the characterization of the exact interplay between DVS and DPM. Section 4.3 extends these results to the case with multiple devices. Section 4.4 extents the results to address dynamic workload variability and show how to minimize average-case energy consumption.

Due to the periodic nature of real-time execution, devices cannot be completely turned off at run-time; but they can be put to low-power (*sleep*) states whenever possible. As a result, a device D_i will always consume power at the rate of at least P_s^i. Thus, for simplicity, all power consumption rates for a device D_i are given in excess of P_s^i. In other words, the following transformations are applied: $P_a^i \leftarrow P_a^i - P_s^i$, $E_{sd}^i \leftarrow E_{sd}^i - (P_s^i \cdot T_{sd}^i)$, $E_{wu}^i \leftarrow E_{wu}^i - (P_s^i \cdot T_{wu}^i)$ and $P_s^i \leftarrow 0$. Notice that such a transformation does not change the original value of the break-even time B_i.

4.2 Single-Device Model

This section considers a model where the real-time application uses a single device. Using this simple model, the fundamental trade-offs involved in the imterplay of DVS and DPM are quantified. Also, an $O(1)$ algorithm is provided to calculate the frequency that optimizes the system-wide energy while taking into account the DVS/DPM interplay and device transition overheads. In Section 4.3, the results are extended to the general case with multiple devices.

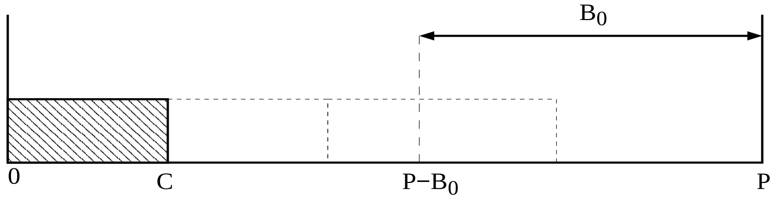


Figure 4.2: The break-even time and the impact of DVS

The exact characterization of the trade-offs between DVS and DPM is critical for system-wide energy minimization. Consider a real-time application with WCET of C units and frame length of P units, using a device D_0 with break-even time B_0. By adjusting CPU frequency, task completion time can be made to vary in the range $[C, P]$ (Figure 4.2). Let U denote the minimum frequency at which the task can still meet its deadline, that is, $U = \frac{C}{P}$ [24]. Further, the frequency which produces a slack of exactly B_0 units is denoted by f^*. In other words, $f^* = \frac{C}{P-B_0}$. Note that in order to transition D_0, the processor has to run the real-time frame-based application at a frequency no less than f^*.

Figure 4.3(a) shows the variations in device energy consumption (E_{device}) and CPU energy consumption (E_{cpu}) as a function of the processor frequency[1]. Note that E_{device} also includes device transition costs, when applicable. To start with, E_{cpu} increases with increasing frequency in a quadratic manner. However, E_{device} follows different patterns in two different regions. In Region A where $U \le f < f^*$, the device D_0 cannot be transitioned (because the slack is smaller than B_0) and it is forced to remain in *active* state

[1]For the purpose of presentation, Figure 4.3 is drawn assuming device break-even time $B = B_{actual}$. The formal analysis does not make such an assumption.

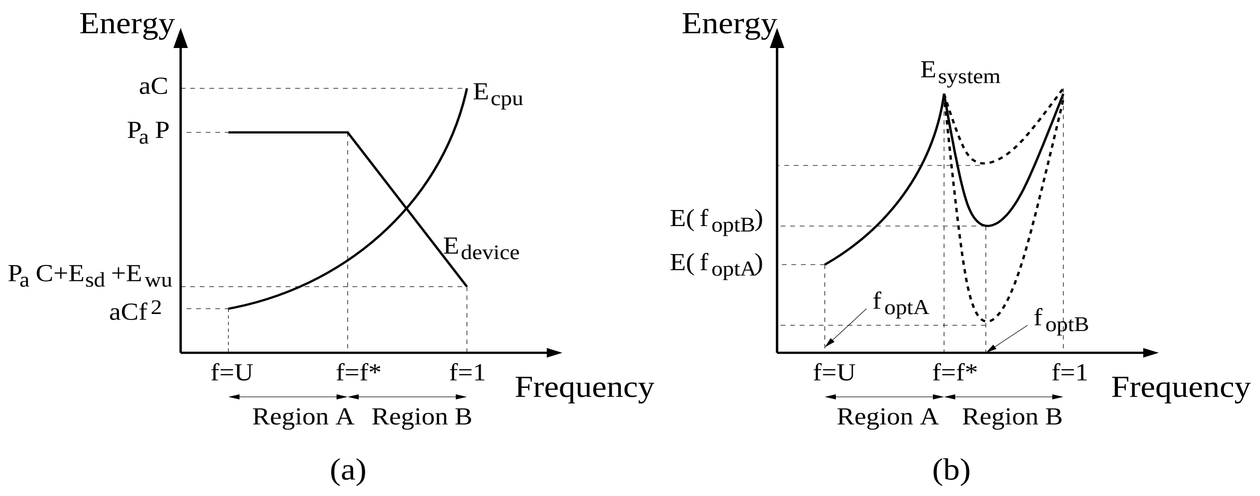


Figure 4.3: System Energy Consumption as a function of Processor Frequency

throughout the frame. As such, $E_{device} = P_a \cdot P$ is constant in Region A. In Region B where $f^* \leq f \leq f_{max}$, the device can be transitioned to *sleep* state. Further, as the frequency increases beyond f^* in Region B, the slack and hence, the length of the device sleep interval, increases. Thus, in Region B, the total device energy consumption during the execution of the application (E_{device}) decreases with increasing frequency.

Figure 4.3(b) shows the variation of the system energy consumption, that is, $E_{system} = E_{device} + E_{cpu}$, as a function of the frequency. E_{system} exhibits varying trends in Regions A and B. While E_{system} increases in Region A with increasing frequency, the local minimum of E_{system} in Region B can lie anywhere in the range $[f^*, f_{max} = 1]$. Also, as additional plots with dashed lines in Region B illustrate, the minimum value of E_{system} in that region can have quite different values.

It is worthwhile to compare these trends to the results of prior energy management studies. Region A is the spectrum where only the dynamic CPU power can be controlled. In fact, this was precisely the assumption of the early real-time DVS literature [11, 101], which effectively ignored Region B. Consequently, in Region A, the minimum frequency that guarantees system feasibility ($f = U$) is optimal. Region B is somewhat similar to the spectrum assumed by the recent system-level energy management papers [10, 71, 138], which considered the CPU and device energy figures at the same time. But, these papers neither accounted for energy transition overheads nor addressed the question of whether DPM is justifiable at run-time, given the length of actual idle intervals. As a result, these approaches ignored Region A. Thus, one really needs to consider both regions to analyze (and get full benefits of) DVS and DPM, simultaneously.

The local optimal frequencies that minimize E_{system} in Regions A and B are well defined. However, there is no a priori reason why global optimal frequency that minimizes E_{system} should lie in Region A or Region B. Depending on the relative power consumption rates of the device and CPU, the execution time of the application and the relative positions of U and f^*, the global optimal may be in either Region A or Region B. In fact, f_{optB}, which optimizes $E_{cpu} + E_{device}$ *and* transitions the device to *sleep* state (by incurring the transition energy) may possibly consume more system-wide energy compared to the frequency f_{optA}, which minimizes E_{cpu} and avoids device transition costs, without paying special attention to E_{device}. Consequently, exact evaluation and comparison of the local optimal values in Regions A and B is necessary in order to determine the global optimal.

4.2.1 System Energy Minimization in Region B

While the local optimal in Region A is straightforward to find, the one in Region B requires some elaboration. In Region B, all frequency values support energy-efficient device transitions and the device will be transitioned at the end of task execution. Thus, in Region B the system energy consumption can be expressed as:

$$\bar{E}(f) = (af^3 + P_a^0) \cdot \frac{C}{f} + (E_{sd}^0 + E_{wu}^0)$$

Observe that $\bar{E}(f)$ is a strictly convex function. $(E_{sd}^0 + E_{wu}^0)$ appears as a constant in $\bar{E}(f)$ and hence, the frequency f_{ee} that minimizes $\bar{E}(f)$ can be found by setting its derivative to zero. This gives:

$$f_{ee} = (\frac{P_a^0}{2a})^{1/3}$$

This is, as expected, numerically equal to the *energy-efficient speed* value given in [10], that did consider neither the transition energy, nor the DPM issues.

Remark 1. *An energy-efficient device state transition as assumed by the operation in Region B may not be possible by using the frequency f_{ee}, if f_{ee} lies outside the range $[f^*, f_{max}]$.*

Remark 2. *Even when f_{ee} is in the range $[f^*, f_{max}]$, in order to find the global optimal frequency, one still needs to compare the minimum energy consumption in Region B which* incurs a device transition overhead *with the minimum energy consumption in Region A which does* not *incur a transition overhead. This will be fully analyzed in Section 4.2.2.*

A strictly convex function with one variable has a single global optimal and its second-derivative is always positive. Hence, the convex nature of $\bar{E}(f)$ justifies the following two basic properties for any $\epsilon > 0$.

Property 1. $\forall f,\ f > f_{ee},\ \bar{E}(f_{ee}) < \bar{E}(f) < \bar{E}(f + \epsilon)$

Property 2. $\forall f,\ f < f_{ee},\ \bar{E}(f_{ee}) < \bar{E}(f) < \bar{E}(f - \epsilon)$

Let f_b denote the frequency that minimizes system energy in Region B. f_b is determined by considering 3 possible cases.

- **Case 1:** $f^* \leq f_{ee} \leq f_{max}$

  In this case, D_0 can be transitioned to *sleep* state at $f = f_{ee}$ as $\delta(f_{ee}) \geq B_0$. Also, there is no other frequency which can transition D_0 *and* yield better system energy consumption in Region B. Thus, $f_b = f_{ee}$.

- **Case 2:** $f_{ee} > f_{max}$

  From Property 2, the system energy in Region B is minimized when $f = f_{max}$. Thus, $f_b = f_{max}$.

- **Case 3:** $f_{ee} < f^*$

  This implies $\delta(f_{ee}) < B_0$ and D_0 cannot be transitioned in energy-efficient fashion at $f = f_{ee}$. Thus, in an effort to transition the device the frequency needs to be increased beyond f_{ee} and towards f^*, which represents the first instance when the device can be transitioned. From Property 1, the system energy in Region B is minimized when $f = f^*$. Hence, $f_b = f^*$.

Based on the analysis above, f_b can be expressed as:

$$f_b = max(f^*, min(f_{ee}, f_{max}))$$

Note that, this formulation covers the three cases examined for energy minimization in Region B and also restricts f_b to the range $[f^*, f_{max}]$. Since $0 \leq B_0 < P$, $f^* \geq U$ and it follows that $f_b \geq U$. Thus, f_b preserves the system feasibility as well.

4.2.2 Finding the Global Optimal

The preceding analysis showed that U and f_b are the local optimal values in Regions A and B, respectively. But, depending on system parameters, the global optimal frequency that minimizes system energy may be in either of these two regions. The example below illustrates this fact.

Illustrative Example 1: Consider a real-time application with $C = 10$ and $P = 42$. Let D_0 have the parameters $P_a^0 = 0.5$, $E_{sd}^0 = E_{wu}^0 = 5$ and $T_{sd}^0 = T_{wu}^0 = 10$. Assume switching capacitance $a = 1$. From the data it can be seen that $B_0 = 20$ and $f_{ee} = 0.63$. Observe that $\delta(f_{ee}) > B_0$. Hence, it turns out that the device can be effectively put to *sleep* state and run at $f = f_{ee}$. However, $E(U = \frac{10}{42}) = 21.57$ and $E(f_b = f_{ee}) = 21.91$. This shows that, for the given settings, despite the fact that the device can be transitioned at f_{ee}, from the system energy point of view it is better to keep D_0 in *active* state throughout the frame and run the application at $f = U$. This is shown through Case 1 in Figure 4.4. In the same example, changing $E_{sd}^0 = E_{wu}^0 = 1.25$ and $T_{sd}^0 = T_{wu}^0 = 5$ gives $B_0 = 10$. With these new

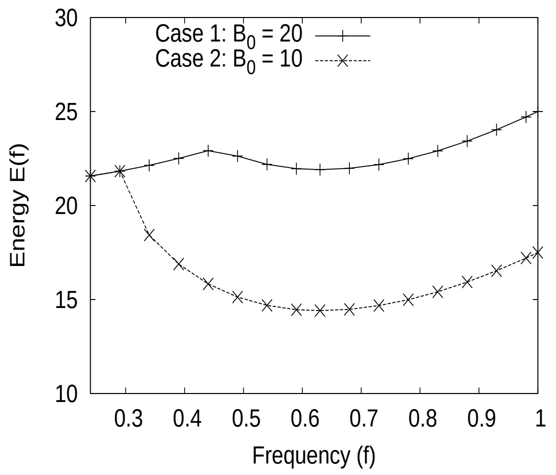


Figure 4.4: Example illustrating the position of global optimal

parameters, one can verify that $E(f_{ee}) < E(U)$ as shown in Case 2 of Figure 4.4.

Thus, to determine the global optimal, it is essential to consider the local optimal values in both Regions A and B and compare them. Determining $E(U)$, $E(f_b)$ and comparing them are all constant time operations. Hence, the overall complexity of the algorithm to determine the global optimal (the frequency that minimizes the total system energy) is $O(1)$.

A final observation is in order about the relative ordering of B^0_{actual} and T^0_{sw}, whose maximum was defined as the break-even time B_0. If $B_0 = B^0_{actual}$, since $\delta(f^*) = B_0 = B^0_{actual}$, the following inequality holds:

$$E(U) \leq E(f^*) \leq E(f^* + \epsilon)$$

This implies that, if $f_b = f^*$, then for sure $f = U$ is the global optimal and a comparison between $E(U)$ and $E(f_b = f^*)$ is not required. However, the above inequality may not hold when $B_0 = T^0_{sw} > B^0_{actual}$. In this case, nothing can be said about the relative ordering of $E(U)$ and $E(f^*)$, as illustrated by the following example.

Illustrative Example 2: Let $C = 5$, $P = 19$, $P^0_a = 0.25$, $T^0_{sd} = T^0_{wu} = 5$ and $E^0_{sd} = E^0_{wu} = 0.625$. For the given data, $B_0 = T^0_{sw} = 10$ and $f^* = \frac{5}{9}$. Assume $a = 1$. It can be verifed that

$E(f^*) = 5.04 < E(U) = 5.096$. By setting $E^0_{sd} = E^0_{wu} = 1$ it can be verified that B_0 and f^* still remain the same. However, in these new settings, $E(f^*) > E(U)$.

4.2.3 Experimental Evaluation

This section performs an experimental evaluation using the actual CPU and device specifications taken from [40] and [127]. The CPU power consumption rate at the maximum processing frequency (f_{max}) is modeled after Intel XScale at 1.6 Watts [127]. A real-time application with a frame length of $44ms$ and using IBM Microdrive during its execution is considered. IBM Microdrive has the following device specifications [40]: $P_a = 1.3W$, $P_s = 0.1W$, $P_{sd}(P_{wu}) = 0.5W$, $T_{wu}(T_{sd}) = 12ms$. Based on these parameters, the break-even time of the device is computed to be $24ms$. Observe that if the worst-case execution time C of the real-time application at the maximum speed is greater than $20ms$, then the device can never be transitioned to *sleep* state as there is not enough slack to justify the transition. Thus, when $C > 20$, the problem of system-wide energy minimization reduces to minimizing CPU energy only and running the CPU at $f = U$ is optimal. Hence, C is only varied from 2-$20ms$ in steps of $2ms$. For each distinct utilization value, three schemes are compared:

- *OPT*: Optimal scheme from Section 4.2.2.

- *Aggressive Slow-Down (AG-SD)*: Run the processor at the lowest frequency $f = \frac{C}{P} = U$ that can still meet the deadline. In this scheme, the devices are never transitioned to *sleep* state. The frequency $f = U$ minimizes the *CPU dynamic energy consumption only* [11, 101].

- *Device-Aware Slow-Down (DA-SD)*: This scheme is based on the concept of energy-efficient speed [10, 71, 138] and is adopted from [10], where the authors propose an optimal solution to the system-wide energy minimization problem *ignoring DPM issues and device transition overheads.* The energy-efficient speed (denoted by f_{ee}) is computed as the speed that minimizes the system energy, by considering only CPU

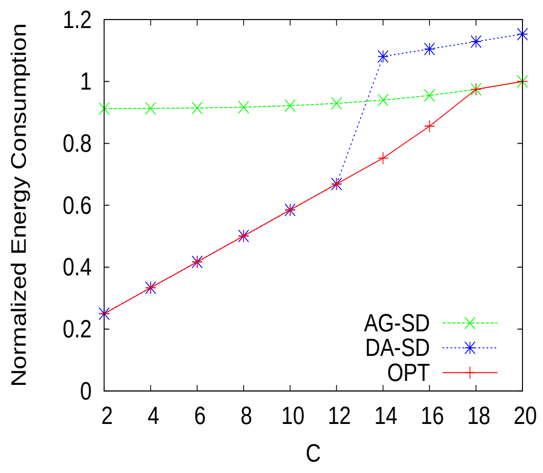


Figure 4.5: Impact of the worst-case execution time C

energy and device *active* energy consumption. Since f_{ee} can be less than the system utilization, to preserve the feasibility, the task is executed at $f = max(U, f_{ee})$. If the device can be transitioned at $f = f_{ee}$ then it is transitioned; else the device remains in *active* state throughout the frame.

Figure 4.5 shows the effect of varying worst-case execution time. The values are normalized with respect to *AG-SD* when $C = 20$. For $C \leq 18$, the system energy benefits from running the processor at frequencies higher than U since the gain in device energy consumption overshadows the loss in CPU energy. Hence, $f = f_b$ is optimal in this region. On the other hand, for $C > 18$, at high frequencies the loss in CPU energy starts to overshadow the gain in device energy. Consequently, running at frequencies higher than U starts to hurt system energy and $f = U$ is the optimal in this spectrum.

It can be seen that in the interval where $C \leq 12$, *OPT* follows *DA-SD* while for $C \geq 18$, *OPT* follows *AG-SD*. However, for $12 < C < 18$ *OPT* follows neither *AG-SD* nor *DA-SD*. In this interval $f_b = f^*$ and *OPT* represents $E(f^*)$ which is optimal in this spectrum. Notice that for this example, the optimal frequency that minimizes the system-wide energy changes from $f_{opt} = f_{ee}$ to $f_{opt} = f^*$ and finally to $f_{opt} = U$ with increasing utilization values. As f_{opt} transitions from f_{ee} to f^*, the device can no longer be put to *sleep* state at $f = f_{ee}$. Thus, running at $f = f_{ee}$ consumes more system energy compared to $f = U$, explaining the sharp increase in *DA-SD* at $C = 14$. The energy optimal scheme *OPT* avoids the sub-optimal performances of *AG-SD* and *DA-SD* at low and high utilization values, respectively.

Note that there is an interval $[12, 18]$ where f_{opt} is neither U nor f_{ee}, but f^*, in the above results. Thus, the optimal scheme (*OPT*) is more than just determining at every point the better frequency in the set $\{U, f_{ee}\}$. In other words, there are regions where both of these well-known frequencies fail to minimize the system-wide energy consumption as in these regions the optimal frequency f^* differs from both U and f_{ee}.

4.3 Multiple-Device Model

This section generalizes the solution to the case of multiple devices. The real-time application is assumed to use m different devices $\{D_1 \ldots D_m\}$ during its execution. Each device D_i has its own parameters (P_a^i, P_s^i, E_{sd}^i, E_{wu}^i, T_{sd}^i, T_{wu}^i) and is associated with the corresponding break-even time denoted by B_i. First, the problem is formally defined.

Problem Statement: *Given a frame-based real-time application using m different devices, determine the CPU frequency and device transitioning decisions so as to minimize the system-wide energy consumption.*

Since each device can be put to *sleep* state at the end of the execution, or remain in *active* state until the end of the frame, at first, it seems that there are 2^m possibilities that need to be examined. If true, this would imply an exponential-time algorithm. By careful analysis, some important properties of the optimal solution are established, which enables the design of an $O(m \log m)$-time algorithm.

Let R_{opt} denote the response time of the real-time application in the optimal solution. It is known that $R_{opt} \in [C, P]$. Without loss of generality, the break-even times are arranged in non-decreasing order, i.e. $0 < B_1 < \ldots < B_m < d - c$. With this ordering, the range $[C, P]$ is divided into $m + 1$ intervals $\{[C, (P - B_m)] \ldots [(P - B_{i+1}), (P - B_i)] \ldots [(P - B_1), P]\}$ denoted by $\{I_m \ldots I_0\}$, respectively (Figure 4.6). If two devices have the same break-even

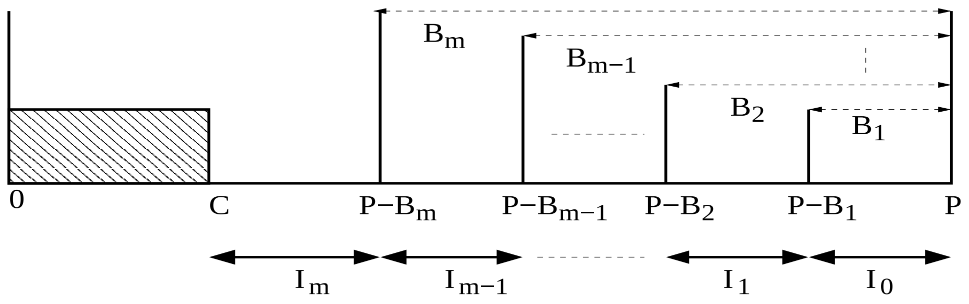


Figure 4.6: The ordering of the break-even times

time, $B_m = P - C$, or $B_1 = 0$, then there will be less than $m + 1$ intervals. The same analysis can then be performed on this reduced interval set.

For convenience, the analysis will be divided into two steps that will eventually lead to an $O(m \log m)$-time algorithm.

- *Step*1: For each of these intervals, assuming that R_{opt} lies in that interval, the set of devices to be transitioned at the end of task execution are determined which helps to evaluate the exact system-wide energy consumption function. The exact form of the system-wide energy in an interval is formulated, laying ground for energy minimization in that interval.

- *Step*2: The analysis in *Step*1 is narrowed down to at most $m + 2$ cases that have to be examined. By comparing the energy consumption figures of these $m + 2$ cases, an optimal solution is determined.

Next, the full analysis for the above mentioned two steps are presented. By ordering the devices in non-decreasing order of break-even times, *Step*1 can be addressed as follows. If R_{opt} belongs to interval I_i, then devices $\{D_{i+1} \ldots D_m\}$ cannot be transitioned as the idle time is smaller than their break-even times. However, all devices $\{D_1 \ldots D_i\}$ can and should be transitioned. This is because if any device in the set $\{D_1 \ldots D_i\}$ is not transitioned to *sleep* state at the end of task execution, then by transitioning that device, one could effectively reduce device energy consumption and hence obtain a schedule with reduced system energy consumption. Based on this, one can infer that if $R_{opt} \in I_m$, all m devices will be transitioned. Similarly, if $R_{opt} \in I_0$, then none of the devices will be transitioned.

Based on the interval to which R_{opt} belongs, one can characterize the devices that should be transitioned to *sleep* states at the end of task execution. With this information, a characterization for the system energy consumption function when $R_{opt} \in I_i$ is provided. The number of devices transitioned to *sleep* states and hence the exact form of the system energy consumption remains the same as R_{opt} varies within a given interval and changes only when R_{opt} transitions between intervals. Let $E_i(f)$ represent the system energy consumption when $R_{opt} \in I_i$. Specifically,

$$E_i(f) = (af^3 + \sum_{j=1}^{i} P_a^j) \cdot \frac{C}{f} + \sum_{j=i+1}^{m} P_a^j \cdot P + \sum_{j=1}^{i} (E_{sd}^j + E_{wu}^j)$$

Notice that in the formulation of $E_i(f)$, devices $\{D_1 \ldots D_i\}$ are transitioned while devices $\{D_{i+1} \ldots D_m\}$ are kept in *active* state throughout the frame. For uniformity, the lower and upper limits of interval I_i are denoted by LL_i and UL_i, respectively. That is, $LL_0 = P - B_1$, $UL_0 = P$, $LL_m = C$, $UL_m = P - B_m$, and, $LL_i = P - B_{i+1}$; $UL_i = P - B_i$, $(i = 1 \ldots m - 1)$. The problem of minimization of E_i is formalized by enforcing that the response time of the task falls in interval I_i. This leads to the following constrained convex optimization problem for I_i, denoted by OPT_i.

$$\text{minimize} \quad E_i(f) \tag{4.1}$$

$$\text{subject to} \quad -\frac{C}{f} + LL_i \leq 0 \tag{4.2}$$

$$\frac{C}{f} - UL_i \leq 0 \tag{4.3}$$

Constraints (4.2) and (4.3) make sure that the response time of the application does not fall outside the range of interval I_i to which R_{opt} is assumed to belong.

Proposition 1. *If the frequency f is the solution to the optimization problem OPT_i, then, $U \leq f \leq f_{max}$.*

Proof. If $f > f_{max}$, from both (4.2) and (4.3) it follows that the response time at $f > f_{max}$ is in the range $[LL_i, UL_i]$. Since $C = LL_m$, this implies $\frac{C}{f_{max}+\epsilon} \geq C = \frac{C}{f_{max}}$, which is a contradiction.

Similarly, if $f < U$, from both (4.2) and (4.3) it follows that the response time at $f < U$ is in the range $[LL_i, UL_i]$. Now, since $UL_0 = d$, this implies $\frac{C}{U-\epsilon} \leq P = \frac{C}{U}$, which is again a contradiction. $\square$

Let f_i be the value that sets the derivative of $E_i(f)$ to zero. This gives:

$$f_i = \sqrt[3]{\frac{\sum_{j=1}^{i} P_a^j}{2a}}$$

Lemma 1. *If f_i satisfies conditions (4.2) and (4.3) then it is the solution to OPT_i. Else, in the solution to OPT_i either $f = \frac{C}{LL_i}$ or $f = \frac{C}{UL_i}$.*

Proof. By definition, the response time of the application in OPT_i must be in interval I_i (i.e. in the interval $[LL_i, UL_i]$). As a consequence, it follows that the frequency at which the application is executed is in the range $[\frac{C}{LL_i}, \frac{C}{UL_i}]$. In other words, if $f \in [\frac{C}{LL_i}, \frac{C}{UL_i}]$ then conditions (4.2) and (4.3) will be satisfied.

Since $E_i(f)$ is strictly convex, it is minimized at f_i. Thus, if $f_i \in [\frac{C}{LL_i}, \frac{C}{UL_i}]$ then it is the solution to OPT_i. On the other hand, if $f_i \notin [\frac{C}{LL_i}, \frac{C}{UL_i}]$ then due to convexity of $E_i(f)$ either $f = \frac{C}{LL_i}$ or $f = \frac{C}{UL_i}$ is the solution to OPT_i [89]. $\square$

Observe that when $I_i = I_m$, f_m is found equal to f_{ee}, which is the traditional energy-efficient frequency for a task using m devices derived by ignoring DPM issues [10]. Assuming that f_{ee} satisfies the response time constraints for interval I_m, an interesting observation at this point is that f_{ee} is *only* the local optimal solution for the interval I_m.

While Lemma 1 solves *Step*1 of the analysis, the following Corollary connects *Step*1 and *Step*2.

Corollary 1. *If the response time under frequency f_i lies outside the interval I_i ($\forall i = 0 \ldots m$) then in the optimal solution, $R_{opt} \in \{C, (P - B_m), \ldots, (P - B_1), P\}$.*

Corollary 1 states that if for all the $(m+1)$ intervals, I_i, f_i does not satisfy the conditions (4.2) and (4.3) of the optimization problem OPT_i, then in the optimal solution the response time of the application is limited to the set $\{C, (P - B_m), \ldots, (P - B_1), P\}$. That is, if the given conditions hold, in the optimal solution, the slack of the application should be exactly equal to 0, $P - C$ or one of the break-even times $\{B_i\}$.

If $R_{opt} = P - B_i$, technically it falls in two intervals and one may tend to think that there is a need for evaluating two cases: one in which D_i is not transitioned (as part of the interval I_{i-1}) and another is which D_i is transitioned (as part of interval I_i). However, as the following observation states, one of these possibilities is never worse than the other.

Observation 1. *If $R_{opt} = P - B_i$, then the device D_i can be transitioned without increasing the overall energy consumption.*

Observation 1 follows from the fact that B_i is defined as $max\{B^i_{actual}, T^i_{sw}\}$. If $B_i = B^i_{actual} > T^i_{sw}$, then by definition of B^i_{actual}, transitioning or not transitioning the device results in the same energy consumption when the application completes at $t = P - B_i = P - B^i_{actual}$. On the other hand, if $B_i = T^i_{sw} > B^i_{actual}$, and the application completes at $t = P - B_i$, it leaves a slack strictly larger than B^i_{actual} and transitioning the device reduces the energy consumption. Hence, in either case, the device D_i can be transitioned without increasing the energy consumption when $R_{opt} = P - B_i$.

Observation 1 implies that there are at most $m + 2$ cases that need to be examined to determine the optimal solution. Let EC_{opt} denote the set of energy consumption values obtained by evaluating the final $m+2$ cases. An interesting question is whether there exists a pattern among these final $m+2$ cases that can be further exploited by convex optimization techniques. Unfortunately, the answer is negative.

Observation 2. *The relative ordering of the* $(m+2)$ *values in the set* EC_{opt} *does not exhibit a special pattern.*

The following observation justifies the above observation.

Illustrative Example 3: Consider a real time application with $C = 10$ and $P = 30$. The application uses four devices $D_1(P_a^1 = 0.2, B_1 = 5)$, $D_2(P_a^2 = 0.15, B_2 = 10)$, $D_3(P_a^3 = 0.5, B_3 = 15)$ and $D_4(P_a^4 = 0.4, B_4 = 17)$. Assume $a = 1$ and $T_{sw} \leq B^i_{actual}$ for all devices. $B^i_{actual} = \frac{E^i_{sd}+E^i_{wu}}{P^i_a}$, $i = 1 \ldots 4$. In these settings, it can be verified that $f_4 = 0.855$, $f_3 = 0.752$, $f_2 = 0.559$ and $f_1 = 0.464$. Notice that $\forall i$, $\frac{C}{f_i} \in I_i$. In interval I_0, setting $f = U$ is the best choice as no devices are transitioned to *sleep* states. With the above data it can be verified that $E(f = U) = 38.611$, $E(f = f_1) = 38.963$, $E(f = f_2) = 38.886$, $E(f = f_3) = 38.958$ and $E(f = f_4) = 38.730$.

Notice how the interval-optimal energy consumption $E_i(f)$ first increases, next decreases, then increases before decreasing once again, as one moves from the first candidate frequency f_1 to f_2, f_3 and f_4. This shows that the optimal energy consumption values of the final $m+2$ cases need not to have a well-defined relationship (such as convexity) which can be exploited by optimization techniques. Hence, it is indeed necessary to evaluate and compare the $m+2$ candidate cases for the optimal solution.

4.3.1 Computing the Optimal Frequency Efficiently

Based on the above characterizations an $O(m \log m)$ algorithm is formulated, given in Figure 4.7, to find the optimal frequency for the multiple-device model. As an implication of Observation 2, it is necessary to compare the best energy consumptions obtained by assuming $R_{opt} \in I_i$, $i = 0 \ldots m$ to obtain the global optimal. From Lemma 1 and Observation 1, in every interval I_i, $i \neq m$, if $f = f_i$ does not satisfy the response time constraints

```
Function Optimal frequency:
1   Set $P_{ON} = \sum_{i=1}^{m} P_a^i$
2   Set $P_{OFF} = 0$
3   Set $E_T = 0$
4   Set $E_0 = aCU^2 + P_{ON} \cdot P$
5   Set $E_{best} = E_0$
6   Set $f_{best} = U$
7   for $i = 1$ to $m$
8       Set $P_{ON} = P_{ON} - P_a^i$
9       Set $P_{OFF} = P_{OFF} + P_a^i$
10      Set $E_T = E_T + E_{sd}^i + E_{wu}^i$
11      Set $f_i = \sqrt[3]{\frac{P_{OFF}}{2a}}$
12      if $(\frac{C}{f_i} \in I_i)$ then $f = f_i$
13      else $f = \frac{C}{UL_i} = \frac{C}{P-B_i}$
14      Set $E_i = (af^3 + P_{OFF}) \cdot \frac{C}{f} + P_{ON} \cdot P + E_T$
15      if $(E_i < E_{best})$
16          $E_{best} = E_i$
17          $f_{best} = f$
18      end if
19  end for
20  Set $E_{m+1} = (af^3 + P_{OFF}) \cdot \frac{C}{f} + P_{ON} \cdot P + E_T$
21  if $(\frac{C}{f_m} \notin I_m$ and $E_{m+1} < E_{best})$
22      Set $E_{best} = E_{m+1}$
23      Set $f_{best} = f_{max}$
24  end if
25  return $f_{best}$
```

Figure 4.7: Algorithm to Compute the Optimal Frequency (Multiple-Device Case)

then it is sufficient to evaluate and compare energy consumption at $f = \frac{C}{UL_i}$. The algorithm begins assuming that the optimal solution is in I_0 and $f = U$ minimizes system energy (lines 4-6). The E_{best} variable holds the minimum system energy consumption value encountered so far and the f_{best} holds the corresponding frequency. In lines 7-19, cases where the optimal response time of the application is assumed to belong to each of the remaining m intervals $I_1 \ldots I_m$ are considered. For each such interval the value of f_i is computed. Based on whether or not f_i satisfies the response time constraints energy consumption at either $f = f_i$ or $f = \frac{C}{UL_i}$ is compared with E_{best}. For interval I_m, if f_m does not satisfy the response time constraints then it is necessary to evaluate and compare energy consumption at $LL_m = C$, as C does not act as an upper limit to any interval. In lines 20-24, this final comparison is performed. At the end, f_{best} holds the optimal value of f that minimizes system energy consumption.

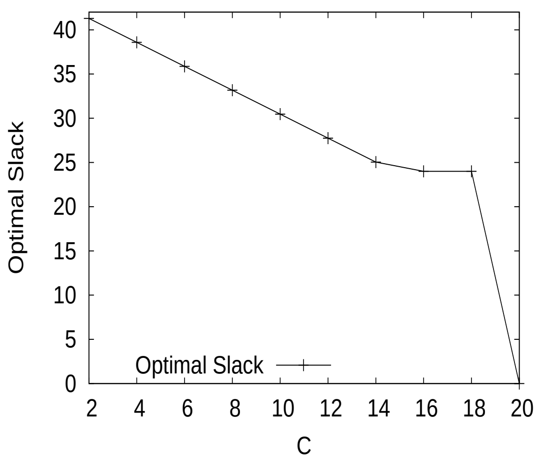
Time Complexity: Sorting the devices based on break-even times requires $O(m \log m)$ time. The algorithm performs a constant-time comparison in every interval and there are at most $m + 1$ intervals. Thus, the time complexity of the algorithm is $O(m \log m)$.

4.3.2 Experimental Evaluation

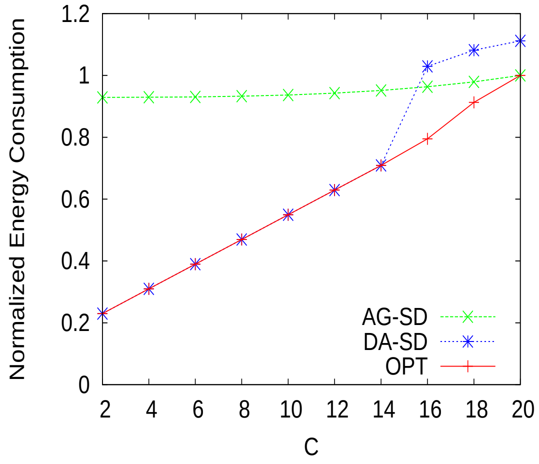
The experimental methodology in this section is an extension of the one described in Section 4.2.3. Again, a real-time application with frame length of $44ms$ is considered. The application now uses three devices during its execution: IBM Microdrive, Realtek Ethernet Chip and Simple Tech Flash Card whose device specifications are given in Table 4.1 [40]. As before, the CPU power consumption rate at the maximum processing frequency is modeled after Intel XScale at $1.6W$ [127].

Device	P_a, P_s, $P_{wu}(P_{wu})$	$T_{wu}(T_{wu})$	Break-even Time
Realtek Ethernet Chip	0.19, 0.085, 0.125(W)	10(ms)	20(ms)
IBM Microdrive	1.3, 0.1, 0.5(W)	12(ms)	24(ms)
SST Flash SST39LF020	0.125, 0.001, 0.05(W)	1(ms)	2(ms)

Table 4.1: Device Specifications



(a) Optimal Slack as a function of C



(b) Relative performance of schemes as a function of C

Figure 4.8: Experimental evaluation for multiple device model

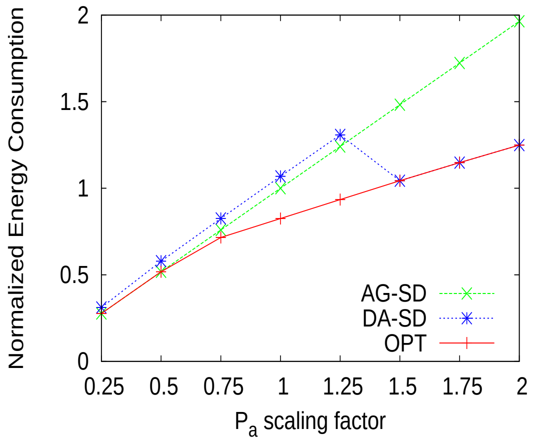
First, the effect of worst-case execution time on both optimal system slack and optimal system energy consumption is considered (Figures 4.8(a) and (b)). Figure 4.8(a) shows the optimal slack $(P - R_{opt})$ which minimizes system-wide energy as a function of C. Figure 4.8(b) shows the relative performance of the three schemes. The energy values are normalized with respect to AG-SD when $C = 20ms$. The optimal slack decreases uniformly with increasing utilization in the range $2 \leq C \leq 14$. In this interval, $f = f_{ee}$ is optimal, and the OPT scheme follows DA-SD. For values in the range $14 < C < 20$, OPT is significantly different from both DA-SD and AG-SD. During this period, one or more devices cannot be transitioned at $f = f_{ee}$, which explains the sharp increase in DA-SD at $C = 16$. The step-like behavior of the optimal slack is also a consequence of the optimal frequency shifting from $f = f_{ee}$ to an intermediate value between U and f_{ee}. Note that depending on the power characteristics of devices and frame length, the sharp increase in DA-SD scheme may also occur at an earlier stage than the one shown in figure. In such cases, the advantage of the proposed optimal scheme is even more pronounced.

In models minimizing the system-wide energy while ignoring device transition overheads and DPM related issues, *DA-SD* scheme was shown to be optimal assuming it satisfies feasibility constraints [10]. Observe that *AG-SD* outperforms *DA-SD* in the spectrum $C \geq 16$ in Figure 4.8(b). Due to device transition overheads, as mentioned before, transitioning devices at $f = f_{ee}$ is not always possible. When $C = 16$, IBM Microdrive cannot be transitioned at f_{ee} and remains *active* throughout the frame significantly increasing device energy consumption. The CPU power consumption rate is significantly high compared to that of Flash Card and Ethernet Chip. Thus, with IBM Microdrive in *active* state throughout the frame, the CPU energy savings in scheme *AG-SD* dominate the device energy savings obtained by transitioning Flash Card and Ethernet Chip in *DA-SD*. This is the reason why *AG-SD* outperforms *DA-SD*.

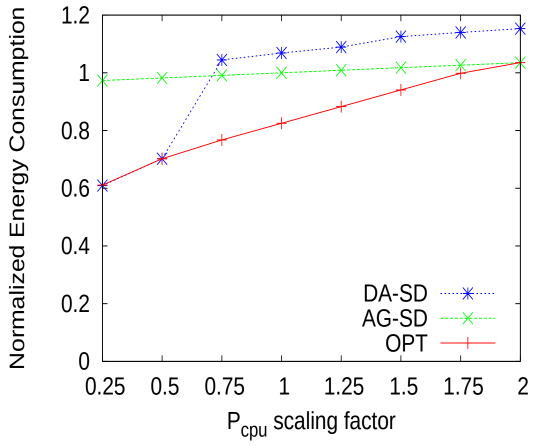
Finally, at $C = 20$, *OPT* follows *AG-SD*. Observe that for the devices considered $T^i_{sw} > B^i_{actual}$. As a result, the device break-even time, defined as $max(T^i_{sw}, B^i_{actual})$, is dominated by the device transition times. Thus, even at $C = 20$, the system has enough slack to potentially transition all three devices energy-efficiently. However, when $C = 20$, there is no device transitioning decision, involving transitioning at least one device to *sleep* state, which can reduce device energy consumption to an extent that it overshadows the increase in CPU energy by running the processor at frequencies higher than $f = U$. Thus, *AG-SD* is optimal at $C = 20$.

While the above experiments were based on device/processor parameters taken from [40,127], in the following experiments the sensitivity of the results are analyzed with respect to power characteristics of the system components by scaling up and scaling down the device/processor parameter values given in [40,127]. Figure 4.9 shows the impact of varying device, processor and application characteristics. In these experiments $C = 16ms$.

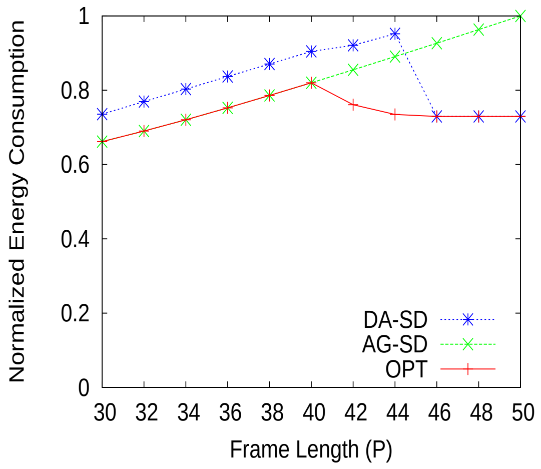
Figures 4.9(a) and (b) show the impact of varying device and processor power characteristics, respectively. In Figure 4.9(a) the active power of all devices are multiplied by a certain scaling factor and re-compute device break-even times while keeping processor characteristics the same. On the contrary, in Figure 4.9(b) the processor power consumption at the maximum frequency is multiplied by a certain scaling factor while keeping device



(a) Impact of device power scaling



(b) Impact of processor power scaling



(c) Impact of frame length

Figure 4.9: Impact of variations in device, processor and application characteristics

characteristics the same. At each scaling point the system energy consumption of all three schemes are evaluated. All energy values in Figures 4.9(a) and (b) are normalized with respect to scaling factor of 1 (i.e. the original device/processor parameters).

In Figures 4.9(a) and (b) it is worth observing that there is a well-defined region where

OPT's energy savings differ from those of *AG-SD* and *DA-SD*. At lower P_a scaling factors and higher P_{cpu} scaling factors, the processor energy consumption overshadows device energy consumption and dominates system energy. As such, in these regions *AG-SD* outperforms *DA-SD*. On the contrary, at higher P_a scaling factors and lower P_{cpu} scaling factors device energy consumption is dominant, hence *DA-SD* outperfors *AG-SD*. Thus, *DA-SD* suffers from performance degradation in settings where CPU energy dominates, while in settings where device energy dominates the performance of *AG-SD* significantly degrades. As is evident in Figures 4.9(a) and 4.9(b) OPT maintains a robust performance at all scaling factors and is not susceptible to performance degradation due to changes in power characteristics of system components.

Figure 4.9(c) shows the impact of varying the frame length (P). P is varied between $30ms$ to $50ms$ in steps of $2ms$. The energy values are normalized with respect to that of *AG-SD* at $P = 50$. With increasing values of P the energy consumption of *AG-SD* increases as devices are forced to remain in *active* state for prolonged periods of time. For the same reason in the range $30 \leq P < 46$ the energy consumption of *DA-SD* increases with increasing values of P. In this range there is not enough slack to create an effective and energy-efficient device transition with which *DA-SD* can outperform *AG-SD*. However, when $P = 46$, such a transition does occur and hence *DA-SD* outperforms *AG-SD* in the range $46 \leq P \leq 50$. Notice that in the region $40 < P < 46$ *OPT* differs from both *AG-SD* and *DA-SD*. Finally, the sudden decrease in energy consumption of *OPT* at $P = 42$ is due to the fact that the frame length becomes large enough to allow for transitioning additional devices energy-efficiently.

4.4 Workload Variability

While Section 4.3 has developed a provably optimal solution to minimize the system energy consumption for a deterministic workload by assuming worst-case execution behavior, frequently, the actual workload in real-time applications exhibits significant variability [50]. In fact, exploiting workload variability to minimize CPU energy through reclaiming is a heavily explored problem in DVS research [11,101,109]. In the presence of such variability in run-time execution behavior, the optimal solution derived in Section 4.3 becomes sub-optimal and pessimistic. In variable workload settings, even though the application's *actual* workload may not be known precisely in advance, some stochastic information about the run-time execution behavior of the application can be determined [11,35,59,82,87,127]. This section, following [11,82], shows how information about the *average-case* execution behavior of the application, if known, can be used to extend the framework to minimize the *average-case system energy* consumption, while still providing deterministic guarantees to meet the deadline.

Note that the hard deadline constraint will impose an absolute lower bound on the CPU frequency, occasionally limiting the efficacy of energy optimization with average-case execution time. The details of the approach are now given.

Let C' denote the average-case execution time of the real-time application (under maximum frequency). Let R_{exp} represent the response time of the application in the solution that minimizes the average-case energy. Thus, by definition $R_{exp} \in [C', P]$. As in Section 4.3, the interval $[C', P]$ can be divided into $m+1$ intervals $I_0 \ldots I_m$. All interval values are the same as in Section 4.3 except for the lower limit of I_m which is C' as opposed to C (the worst-case execution time). Thus, R_{exp} belongs to one of the $m+1$ intervals $I_0 \ldots I_m$.

Further, following the same reasoning as in Section 4.3, it can be seen that if $R_{exp} \in I_i$, in the solution that minimizes the average-case energy, all devices $\{D_1 \ldots D_i\}$ can and must be transitioned to *sleep* state. On the other hand, devices $\{D_{i+1} \ldots D_m\}$ cannot be transitioned in energy-efficient fashion and will remain in *active* state over the interval $[0, P]$. As such, if $R_{exp} \in I_i$ then the average-case system-wide energy consumption $E'_i(f)$ is given by:

$$E'_i(f) = (af^3 + \sum_{j=1}^{i} P_a^j) \cdot \frac{C'}{f} + \sum_{j=i+1}^{m} P_a^j \cdot P + \sum_{j=1}^{i} (E_{sd}^j + E_{wu}^j)$$

Now, one can again construct $(m+1)$ constrained optimization problems by enforcing

$R_{exp} \in I_i \; i = 0 \ldots m$. Formally, the constrained optimization problem OPT_i^* is defined as:

$$\text{minimize} \quad E_i'(f) \tag{4.4}$$

$$\text{subject to} \quad -\frac{C'}{f} + LL_i' \leq 0 \tag{4.5}$$

$$\frac{C'}{f} - UL_i' \leq 0 \tag{4.6}$$

$$U \leq f \leq f_{max} \tag{4.7}$$

LL_i' and UL_i' are the lower and upper limits of interval I_i respectively. Recall from Section 4.2 that $U = \frac{C}{P}$ represents the minimum frequency which guarantees feasibility under worst-case workload. Though the objective is to minimize the average-case system energy, it is still important to meet the application deadline under worst-case execution behavior. The constraint (4.7) in OPT_i^* helps to meet that objective.

Let F_i denote the solution to OPT_i^*. Further, let F_i' denote the solution to the same optimization problem, but without the constraint (4.7). By repeating the analysis in Section 4.3 one can verify Propositions 2 and 4, below. Proposition 3 is a consequence of convexity.

Proposition 2. *$F_i' = f_i$ if $\frac{C'}{f_i} \in I_i$ where $f_i = \sqrt[3]{\frac{\sum_{j=1}^{i} P_a^j}{2a}}$. Otherwise, either $F_i' = \frac{C'}{UL_i'}$ or $F_i' = \frac{C'}{LL_i'}$.*

Proposition 3. *If $U \leq F_i' \leq f_{max}$ then $F_i = F_i'$. Otherwise, if $F_i' < U$ then $F_i = U$ else $F_i = f_{max}$.*

Proposition 4. *A solution to the problem of minimizing the average-case energy can be obtained in $O(m \log m)$ time by comparing the OPT_i^* solutions (F_i), $i = 0 \ldots m$.*

4.4.1 Experimental Evaluation

To evaluate the performance of the new scheme under workload variability, a series of experiments are conducted. As in Section 4.3.2, a real-time application with a frame length of $44ms$, using three devices: IBM Microdrive, RealTek Ethernet Chip and Simple Tech Flash Card is considered. The actual execution time of the application is determined using normal distribution with mean $\frac{BCET+WCET}{2}$ and standard deviation $\rho_0 = \frac{WCET-BCET}{6}$, where BCET and WCET are the best-case and worst-case execution times of the application under maximum frequency, respectively. The specific mean and standard deviation values coincide with those used in [11, 59, 74, 82, 113], and guarantee that 99.7% of the execution times fall in the range $[BCET, WCET]$[2]. The results presented are the average from $1,000,000$ experiments. Also, the corresponding confidence intervals at 99% confidence level are reported. The following three schemes are compared:

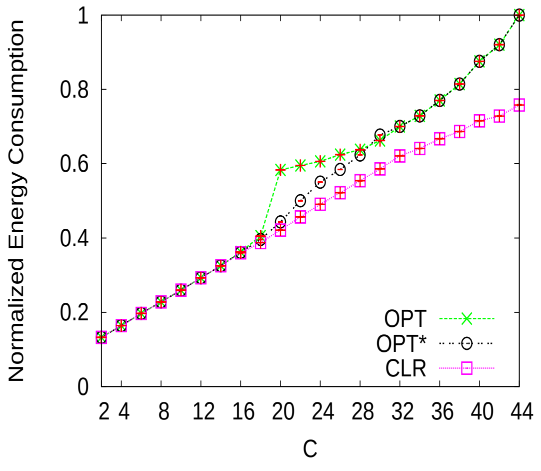
- OPT which is the optimal solution from Section 4.3 assuming worst-case execution behavior. However, when the application completes in a given frame, the amount of actual slack until the next invocation is computed and all devices that can be transitioned in energy-efficient fashion are put to low-power states.

- OPT^* which minimizes the average-case system energy by assuming average-case execution time ($\frac{BCET+WCET}{2}$).

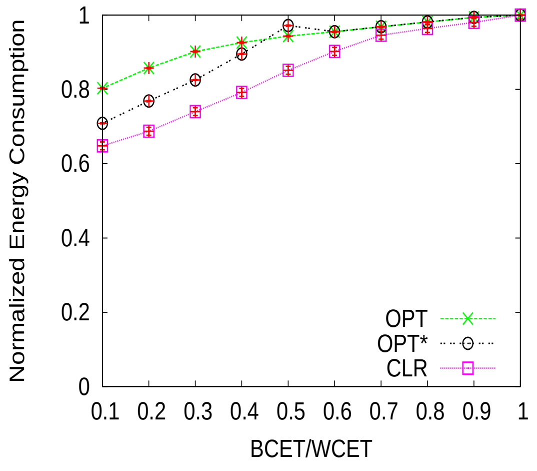
- CLR which is the clairvoyant scheme that knows the actual execution times in advance and uses this information to derive optimal CPU operation frequency and device transitioning decisions. While CLR is not a practical scheme, it is used as a yardstick algorithm yielding the lower bound on system energy consumption.

Figure 4.10(a) shows the comparison of the schemes as a function of the worst-case execution time of the application (C). The best-case to worst-case execution time ratio is fixed at 0.2. The results are normalized with respect to OPT at $C = 44$. When C is in the range

[2]If the randomly generated execution time exceeds $WCET$, which happens with the probability 0.3%, that value is not considered in the experiments.



(a) Relative performance of schemes as a function of C

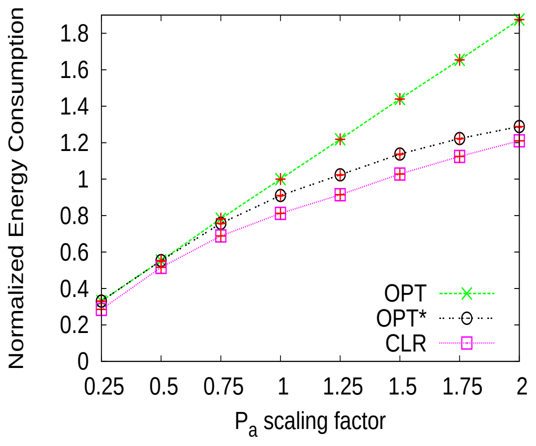


(b) Relative performance of schemes as a function of $\frac{BCET}{WCET}$

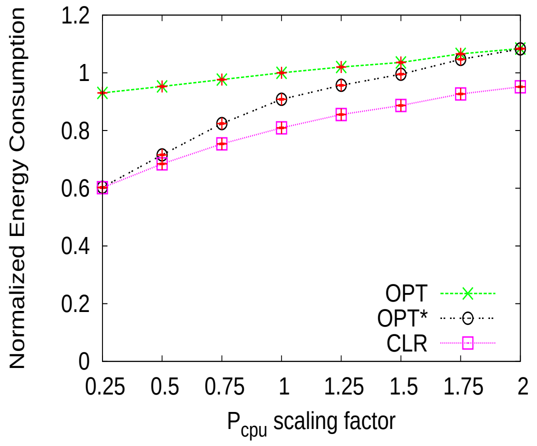
Figure 4.10: Experimental evaluation under dynamic workload variability

$[18, 30]$, the benefits of OPT^* over OPT are evident. In this range, the frequency determined by OPT^* is more energy-efficient towards dynamic workload variability compared to that of OPT. At low C values $[0, 18]$ there is more slack in the system and hence more DPM opportunities; thereby all schemes perform the same. For C values in the range $[30, 44]$, the high worst-case and average-case execution times severely limit device transitions while calculating the CPU frequency. As a consequence, both OPT and OPT^* perform the same. However, CLR which uses actual workload information performs significantly better.

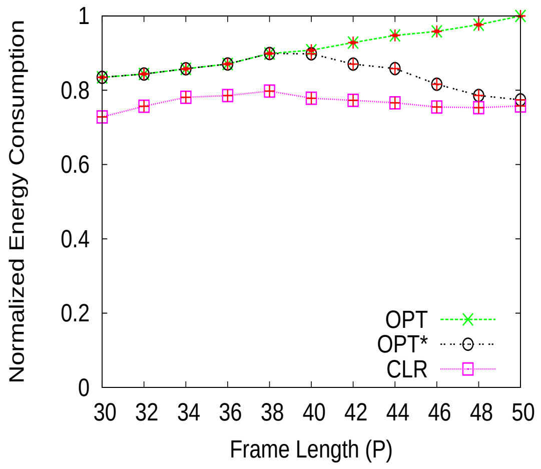
Figures 4.10(b) shows the impact of varying $\frac{BCET}{WCET}$ ratio at $C = 24$. The results are normalized with respect to OPT at $\frac{BCET}{WCET} = 1$. One can see that as $\frac{BCET}{WCET}$ ratio decreases, OPT^* performs better compared to OPT. This is because with more variations in run-time behavior of the application, the accuracy of OPT^* in estimating actual workload increases compared to OPT and thus the frequency determined by OPT^* yields lower system energy consumption.



(a) Impact of device power scaling



(b) Impact of processor power scaling



(c) Impact of frame length

Figure 4.11: Impact of variations in device, processor and application characteristics

Figure 4.11 shows the impact of power and application characteristics. In these experiments $C = 24$ and $\frac{BCET}{WCET} = 0.2$. The experiment methodology is the same as described for Figure 4.9 in Section 4.3.2. While OPT deviates significantly from CLR at high P_a and low P_{cpu} scaling factors, OPT^* remains close to the CLR, a fact showing its robustness for various device and processor power characteristics (Figures 4.11(a) and (b)). Similar trends can also be seen in Figure 4.11(c) where OPT deviates significantly from CLR at high P values, while the performance of OPT^* gets closer to CLR as P increases. This is because at large frame lengths, the importance of being able to accurately estimate actual workload information (and use it to determine CPU frequency while accounting for DPM issues) translates to significant system-wide energy savings.

4.5 Chapter Summary

This chapter addressed the problem of system-wide energy minimization for a frame-based real-time application through a novel approach. The system-level energy model considered both DVS- and DPM-related issues and accounted for device transition overheads. With this general model, the exact interplay between DVS and DPM is formally characterized. By deriving useful properties from this characterization, an $O(m \log m)$-time algorithm is formulated (where m is the number of devices) to determine the CPU frequency and device transitioning decisions to minimize the system-wide energy. Through experimental evaluations using real device parameters the potential benefits of the proposed optimal scheme are demonstrated. Also, the solution is extended to address the workload variability in order to minimize the average-case energy consumption assuming the knowledge about average-case execution time.

Chapter 5: System-level Energy Management for Periodic Real-time Tasks

5.1 Introduction

Most real-time embedded applications, such as those used in control systems, consist of multiple tasks which are periodic in nature. While DVS solutions for periodic real-time tasks are well-established, DPM solutions are less mature. Further, the problem of developing system-wide energy management frameworks for periodic real-time tasks that unify DVS and DPM is mostly an unexplored problem. This chapter investigates the complexity of the RT-DPM problem and develops a new DPM framework called *Device Forbidden Regions*. Using this novel framework, unified system-level energy management frameworks for both fixed-priority and dynamic-priority periodic task systems are developed by combining DVS with DPM, and accounting for their non-trivial interplay.

5.2 Dynamic Power Management for Real-Time Systems

One of the major challenges in online real-time dynamic power management is to make sure that the device sleep intervals are longer than the corresponding break-even times, to guarantee the energy efficiency. In theory, the scheduler can attempt to order the execution of tasks in such a way that the device usage and sleep intervals are grouped together to the extent it is possible. Theorem 1, whose proof is given in the Appendix A, indicates that solving the problem of minimizing device energy consumption for general periodic tasks (even in the absence of DVS) is intractable, closing an open problem. First, the device energy minimization problem for real-time tasks is formally defined.

RT-DPM: Given a set of real-time tasks and a set of devices with known power characteristics, find the feasible schedule that minimizes the total device energy consumption.

Theorem 1. *RT-DPM for general periodic tasks is NP-Hard in the strong sense.*

This result indicates that even a pseudo-polynomial time optimal algorithm for the problem is unlikely, unless $P = NP$.

5.2.1 Next Device Usage Predictions

One solution to the RT-DPM problem is to commit to a given scheduling policy (such as RMS or EDF) and then perform device transitions by trying to predict the next device usage time of device D_i at time t (denoted by $NDUT_i(t)$), at run-time. The device can be put to sleep state with confidence if the difference $(NDUT_i(t) - t)$ is larger than the break-even time B_i of D_i, as long as $NDUT_i(t)$ does not overestimate the actual next device usage time.

Unfortunately, the stringent timing constraints of real-time tasks, the variability in actual execution times, and release time jitters make a precise prediction impossible. Further, even if all this information is known to a clairvoyant scheduler, performing an exact online response time analysis for various task instances during the hyperperiod (to figure out when a task in need of D_i will be dispatched next) is not computationally feasible [83].

One conservative but safe solution is to compute $NDUT_i(t)$ by considering the current time, the tasks in ready queue, and the earliest release time of any future job that requires D_i. Hence, if the ready queue contains a task that requires D_i, $NDUT_i(t)$ will be simply t; otherwise it will be the earliest next release time of any instance of a task in β_i (set of tasks requiring device D_i, Section 3.3). This is indeed the main principle of the *Conservative Energy-Efficient Device Scheduling (CEEDS)* algorithm, proposed by Cheng and Goddard [38, 41]. When de-activating a device, CEEDS also schedules an activation time (or, *UpTime*) by considering typically non-negligible re-activation delays and NDUT. At this activation-time the device starts transitioning from low-power to active state in order to be ready to serve requests at proper times.

5.2.2 Dynamic Device Power Management through Forbidden Regions

DFR-based approach to real-time DPM problem is based on the sleep intervals which are explicitly and periodically enforced for each device. These special intervals are called *device forbidden regions*. A separate forbidden region FR_i is defined for each device D_i. The forbidden region FR_i has a pre-determined duration $\Delta_i \geq B_i$; as a result, D_i can be safely put to the sleep state during its forbidden region.

Further, a minimum separation time (or, period) Π_i is associated with FR_i, meaning that two consecutive "activations" (or, "enforcements") of a given FR_i should be separated by at least Π_i time units. Under certain circumstances, it may be more beneficial to delay the next activation of a forbidden region. In other words, the forbedden regions may be activated in "sporadic" manner. During execution, the next "earliest release" (activation) time of FR_i is denoted by rf_i.

An extremely important implication of inserting enforced device forbidden regions to real-time schedules is that none of the tasks using D_i can be dispatched while FR_i is active. However, other tasks (i.e. those in $\psi - \beta_i$) can still execute. In other words, the tasks in β_i (set of tasks that use D_i during execution) are effectively "blocked" by FR_i when the latter is active.

Naturally, a task using two different devices D_i and D_j would be blocked whenever FR_i or FR_j is active. Determining Δ_i and Π_i values to guarantee the feasibility of the real-time task set and at the same time to maximize energy savings is a non-trivial problem. These issues are addressed in Sections 5.3.1 and 5.3.2, respectively. Table 5.1 gives a summary of the variables relevant for the DFR framework presented in this chapter.

The DFR scheme does not exclude the usage of CEEDS, or for that matter, any other "prediction-based" online DPM algorithm. In fact, the feasibility analysis that is presented in Section 5.3.1 treats each forbidden region FR_i as a high priority periodic task delaying the execution of tasks in β_i. Consequently, when the system is able to shutdown the device

FR_i	Forbidden region associated with device D_i
Δ_i	Duration of forbidden region FR_i
Π_i	Period of forbidden region FR_i
γ_i	Set of devices used by task T_i
β_i	Set of tasks using device D_i
nfr_i	Earliest next enforcement time for forbidden regions FR_i
act_i	Reactivation timer for device D_i
P_a^i	Active power of device D_i
P_s^i	Sleep power of device D_i
$T_{sd}^i(T_{wu}^i)$	State transition times for device D_i
$E_{sd}^i(E_{wu}^i)$	State transition energy for device D_i
B_i	Breakeven time of device D_i
P_i	Period of task T_i
C_i	Worst-case execution time of task T_i (under f_{max})

Table 5.1: Notations

D_i using the standard prediction techniques (e.g. through CEEDS), the next activation time rf_i of FR_i can be “delayed” without affecting feasibility. This, in turn, will help to save the “bandwidth” of the forbidden region and prolong the sleep interval in the future. Similarly, when a scheduled activation time of FR_i corresponds to a time instant when D_i is actively used by the running task, FR_i should be postponed to a later time, to avoid unnecessary transitions. Clearly, in all these cases, the activation of all subsequent forbidden regions of the same device will be delayed by the same amount.

The following example illustrates the principles of DFR. Consider a harmonic task set (where task periods are multiples of each other) with three real-time tasks T_1, T_2 and T_3 and the following parameters: $C_1 = C_2 = C_3 = 1000, P_1 = 2000, P_2 = 4000$, and $P_3 = 8000$. The devices D_1 and D_2 are used by the tasks T_1 and T_2, respectively. For these two devices, $B_1 = 990, T_{sd}^1 = T_{wu}^1 = 495, B_2 = 20$ and $T_{sd}^2 = T_{wu}^2 = 10$. Assume all tasks are released at $t = 0$ and all devices are initially in *active* state. Note that since the task set is harmonic the schedules generated by both RMS and EDF are the same [83]. Hence, one can assume either policy for the example considered. The schedule generated by DFR is shown in Figure 5.1a. In this example, the forbidden region durations and separation times are selected as $\Delta_1 = \Delta_2 = 1000$ and $\Pi_1 = \Pi_2 = 4000$. It is assumed that the first scheduled enforcements

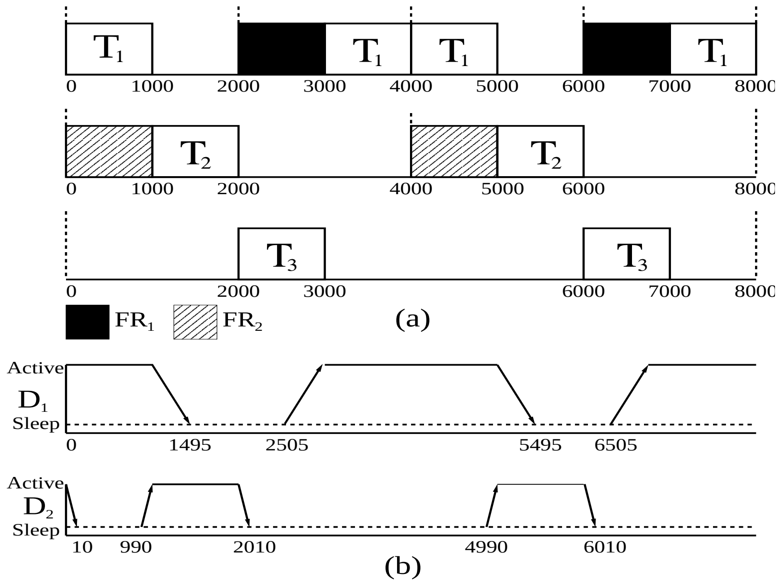


Figure 5.1: DFR Schedule

of both forbidden regions occur at $t = 0$.

At time 0, T_1 is dispatched. FR_1 is postponed as D_1 is already active and in use by the current job. FR_2 is enabled: D_2 is put to sleep state since it is currently not in use and its next activation time is set to $t + \Delta_2 - T^2_{wu} = 990$. At time 1000, T_1 completes and T_2 is dispatched. T_2 finds D_2 in active state. Also, FR_2 ends and rf_2 is set to $t+\Pi_2-\Delta_2 = 4000$. D_1 is put to sleep since $NDUT_1$ is found to be greater than B_1 through CEEDS prediction mechanism. In this case, one can further postpone FR_1 to preserve its "bandwidth" for the future. Thus, the activation time of D_1 is set to $NDUT_1 - T^1_{wu} = 1505$.

At $t = 1505$, the predicted $NDUT_1$ is 2000 and FR_1 is pending. At this point, DFR run-time management system forcefully starts FR_1 at $t = 2000$. Observe that this allows D_1 to remain in sleep state without compromising the timing constraints. At $t = 2000$, FR_1 is enabled and the activation time of D_1 is set to 2505. In the mean time, D_2 is shutdown as $NDUT_2 > B_2$. The activation time of D_2 is set to 3990. T_3 is dispatched. At time 3000, FR_1 is disabled and rf_1 is set to 6000. D_1 completes its transition to the active state and T_1 preempts T_3 upon arrival. At time 3990, DFR postpones again the activation of D_2 as rf_2 is 4000 which is the same as $NDUT_2$. Observe how DFR prolongs the idle intervals and prevents D_2 from remaining in active state unnecessarily. The rest of the schedule can be obtained in a similar way. Figure 5.1b shows the device states and transitions for the DFR algorithm. As a result, with DFR, D_1 remains in *sleep* state for a total of 2020 units while D_2 sleeps for 5950 units. One can verify that with CEEDS for the same example D_1 and D_2 have 40 and 3960 units of total sleep time respectively. The significant gains of DFR are evident in this example.

5.3 DFR-RMS: Integrated System-Level Energy Management Policy for Fixed-Priority Systems

In this section, a DFR-based RT-DPM policy called *DFR-RMS* for fixed-priority systems and RMS policy is developed. Further, *DFR-RMS* is integrated with existing DVS policies and its effectiveness in reducing system energy is shown.

5.3.1 RMS Schedulability Analysis for DFRs

Time Demand Analysis (TDA) technique [81] is a well-established methodology to assess the feasibility of a periodic real-time task set scheduled by the preemptive RMS policy. It provides a sufficient and necessary condition for the schedulability and is generalized to various settings with aperiodic servers, precedence constraints, and task blocking times [8, 83]. TDA relies heavily on the *critical instant* concept, where the response time of a job is maximum when it is released simultaneously as all high-priority tasks [84]. Let $hp(i)$ denote all tasks with priority higher than that of a given periodic task T_i. The time demand function of T_i (denoted by $w_i(t)$) is defined as: $w_i(t) = C_i + \sum_{T_j \in hp(i)} \lceil \frac{t}{P_j} \rceil \cdot C_j$.

Theorem 2. [81, 83] *A set of fixed-priority periodic independent real-time tasks with relative deadlines equal to the periods is feasible if and only if* $\forall\ i,\ \exists\ t,\ 0 \leq t \leq P_i,\ w_i(t) \leq t$, *under critical instant phasing.*

The complexity of TDA is $O(\frac{P_{max}}{P_{min}} \cdot n^2)$, where P_{max} and P_{min} are the largest and smallest periods in the task set respectively [83]. The exact characterization of the critical instant for *DFR-RMS* is non-trivial. Multiple forbidden regions that overlap in time will cause a total interference which is less than their total duration, and a given forbidden region FR_j may be dynamically postponed (for example, when D_j is in active use at the release time of FR_j). Further, the tasks in $hp(i)$ that typically delay the execution of T_i may be themselves delayed by various forbidden regions. Nevertheless, it is still possible to use the traditional TDA technique by conservatively predicting (i.e. over-estimating) the interference of high priority tasks in $hp(i)$ and the forbidden regions, on a given task.

Let $w_i^{FR}(t)$ be the time-demand function of T_i in the *DFR-RMS* algorithm, such that the total interference on T_i from tasks in $hp(i)$ and forbidden regions is maximized.

Proposition 5. *In* DFR-RMS*, for every task T_i and every t in the range $[0, P_i]$, the following holds:*

$$w_i^{FR}(t) \leq w_i^{max}(t) = C_i + \sum_{T_j \in hp(i)} \lceil \frac{t}{P_j} \rceil \cdot C_j + \sum_{j \in \gamma_i} \lceil \frac{t}{\Pi_j} \rceil \cdot \Delta_j$$

Proof. The right hand side of the expression in Proposition 5 overestimates the actual maximum interference on T_i by assuming that:

- the task T_i is released at the same time as all tasks in $hp(i)$,
- none of the tasks in $hp(i)$ is delayed by any forbidden regions until T_i completes, and,
- all forbidden regions FR_j such that $D_j \in \gamma_i$ are activated at $t = 0$, and each of these forbidden regions is treated as a high priority preemptive task invoked at the maximum frequency (i.e. every Π_j time units).

Note that the last assumption above effectively ignores the 'overlap effect' of forbidden regions on T_i: if two forbidden regions D_j and D_k used by T_i overlap for x time units, then their total interference would be $\Delta_j + \Delta_k - x$, and not $\Delta_k + \Delta_j$ as assumed. However, it is fairly difficult to precisely characterize the aggregate impact of such overlaps – though it is certain that the worst-case interference for T_i occurs when there are no such overlaps at all. Hence, the proposition holds. $\square$

Corollary 2. *A set of periodic tasks can be feasibly scheduled by DFR-RMS if* $\forall\ i,\ \exists\ t,$ $0 \le t \le P_i,\ w_i^{max}(t) \le t.$

Note that just like the original TDA algorithm, it is necessary and sufficient to evaluate $w_i^{max}(t)$ at every period boundary in interval $[0,\ P_i]$. Hence, the overall complexity of the extended TDA is $O(\frac{max\{P_{max}, \Pi_{max}\}}{P_{min}}(n+m)^2)$.

5.3.2 Determining Forbidden Region Parameters for DFR-RMS

Before DFR scheme can be used with a given task set, the duration (Δ_i) and period (Π_i) of each forbidden region FR_i needs to be determined. While a fundamental requirement is to ensure the feasibility of the real-time task set, another major objective is to optimize energy savings with the selected Δ_i and Π_i values.

Intuitively, the longer Δ_i (beyond B_i), the higher energy savings for device D_i. Similarly, as Π_i decreases, the number of forbidden region instances of FR_i gets higher, increasing the overall sleep time. Also, considering that the energy savings of D_i during a sleep interval is proportional to the difference $(P_a^i - P_s^i)$, one can define the *Expected Energy Savings (EES)* of D_i as:

$$EES_i = \frac{\Delta_i - B_i}{\Pi_i}(P_a^i - P_s^i) \tag{5.1}$$

In order to perform an efficient search for the best $\{\Delta_i, \Pi_i\}$ values, one needs to first establish some lower and upper bounds on these quantities.

Bounding Δ_i: Observe that having a forbidden region duration $\Delta_i \le B_i$ is not helpful: this stems from the very definition of break-even time. Similarly, it is easy to see that Δ_i cannot exceed $P_j - C_j$ (the maximum allowable laxity for T_j) for any task T_j that uses D_i. Doing otherwise may result in a deadline miss for T_j, in case that an instance of T_j is released immediately after the activation of FR_i. Hence, $B_i < \Delta_i \leq \min_{T_j \in \beta_i} (P_j - C_j)$.

Bounding Π_i: Consider the quantity $z_i = \max_{j \in \beta_i}\{P_j\}$. When evaluating the feasibility of any task in β_i through the TDA in Section 5.3.1, the quantity $\lceil \frac{t}{\Pi_i} \rceil = 1$ for any $\Pi_i \geq z_i$. Hence, if a given task T_x is infeasible with a certain $\Pi_i = z_i$ value ($D_i \in \gamma_x$), then it is guaranteed to remain infeasible for any $\Pi_i > z_i$. Recalling that increasing Π_i does not help to improve the energy savings either, one can obtain $\Pi_i \leq \max_{j \in \beta_i}\{P_j\}$.

Define the utilization U_{D_i} of a given device D_i as the total utilization of tasks in β_i. Observe that the ratio $\frac{\Delta_i}{\Pi_i}$ for given forbidden region FR_i cannot exceed $1 - U_{D_i}$, since this is the maximum amount of time during which D_i can be in sleep state. Combining all this, $\frac{\Delta_i}{1-U_{D_i}} \leq \Pi_i \leq \max_{T_j \in \beta_i}\{P_j\}$.

Given the expected energy savings (EES_i) formula (Equation (5.1)), a reasonable approach is to treat the devices with large EES_i values with high priority in the search process, to increase the potential energy savings. Moreover, for a given D_i, the Δ and Π ranges can be scanned at equi-distant points to assess the feasibility and EES_i with the given values. Simulation results show that evaluating EES_i for 10-15 equi-distant candidates Δ_i and Π_i values gives significant energy savings while keeping the running time at acceptable levels.

The resulting *Greedy FR Assignment* algorithm works as follows. In each iteration, for every device D_i not associated with an FR, a (Δ_i,Π_i) pair that gives the best EES_i is computed while committing to all forbidden region assignments made in the previous iterations and maintaining the feasibility. Among the newly assigned FRs, the forbidden region of the device expected to provide the maximum energy savings (highest EES_i) is incorporated to the system. The algorithm ends either when no new FR assignment can be made or FRs have been associated with all system devices.

Complexity: At each iteration, the complexity of finding the best (Δ_i, Π_i) pairs is still $O(\frac{max\{P_{max},\Pi_{max}\}}{P_{min}}(n+m)^2)$, given the constant number of candidate points evaluated in the range. Hence, the overall complexity of this static algorithm is $O(\frac{max\{P_{max},\Pi_{max}\}}{P_{min}}m(n+m)^2)$, where P_{max} and P_{min} represent the maximum and minimum periods in the task set.

Greedy FR Assignment Algorithm

```
• Set $Z = \{D_1, \ldots, D_m\}$
• Set $W = \emptyset$
• Repeat
    – for each device $D_i$ in $Z$, compute ($\Delta_i$,$\Pi_i$) pair that gives best
      $EES_i$ while committing to $\Delta$, $\Pi$ values already in $W$ and main-
      taining the feasibility.
    – Commit to ($\Delta_j$, $\Pi_j$) pair for $D_j$ that gives the maximum EES
    – Set $W = W \cup D_j$
    – Set $Z = Z - D_j$
• Until there is at least one $D_i \in Z$ for which an FR can be assigned
```

5.3.3 DFR-RMS Run-Time Management Routines

This subsection presents the actions performed by DFR-RMS at important scheduling points and gives the pseudo-codes of the online routines to implement those actions. There are three important scheduling points for the DFR-RMS framework corresponding to job arrival, job completion and device re-activation events (Figure 5.2).

In the DFR-RMS framework a ready job J_i is said to be *eligible* for execution if all the devices required by J_i are in *active* state. If one or more devices required by J_i are not *active* (due to enforced forbidden regions), then J_i is said to be *blocked* from execution. The ready queue is partitioned to two queues Q_r and Q_b such that at any given time, Q_r has all the eligible ready jobs and Q_b has all the blocked jobs. The scheduler always executes the highest priority job in Q_r.

At job arrival time, the new job is inserted to either Q_r or Q_b based on its eligibility for execution. If the new job is both eligible and has the highest priority among all eligible jobs, then the scheduler dispatches it for execution. Similarly, at job completion time, the next highest priority eligible job is picked for execution, provided the Q_r is not empty. In

Job J_k arrives at time t:

```
1  if $\exists D_i$:  $D_i \in \gamma_k$ AND $D_i$ not active
2   insert $J_k$ to $Q_b$
3  else
4   insert $J_k$ to $Q_r$
5   Scheduler()
```

Job J_k completion at time t:

```
1  $J_h = null$
2  if $Q_r$ not empty
3   $J_h =$ job with highest priority in $Q_r$
4   Dispatch($J_h$)
5  else
6   SleepStateTransitions($\mathcal{D}$)
```

Re-activation timer interrupt at time t:

```
1  for i = 1 to m
2   if ($act_i = t$)
3     if ($NDUT(D_i,t) > act_i + T^i_{wu}$)
4      $act_i = NDUT(D_i,t) - T^i_{wu}$
5     else
6      if $nfr_i \leq NDUT(D_i,t)$
7       $act_i = NDUT(D_i,t) + \Delta_i - T^i_{wu}$
8       $nfr_i = NDUT(D_i,t) + \Pi_i$
9      else
10      Transition $D_i$ to active state
11 for each job $J_i \in Q_b$
11  if all $D_j \in \gamma_i$ active
12   Move $J_i$ to $Q_r$
13 Scheduler()
```

Figure 5.2: The DFR-RMS Run-time Actions

both cases, if possible, devices that are no longer in use are energy-efficiently transitioned to their respective *sleep* states using function *SleepStateTransitions()* in Figure 5.3.

Each device D_i is associated with a re-activation timer act_i that indicates the time at which a device in *sleep* state needs to start transitioning to *active* state. At the event corresponding to device re-activation, first an effort is made to delay device activation by extending device idle intervals. To this extent, next device usage times are re-computed to capture changes, if any, from their previously computed values and forbidden regions are enforced, if possible to do so. If device idle intervals cannot be successfully extended, then the device is transitioned to *active* state.

When a device in *sleep* state transitions to *active* state, it can potentially unblock one or more jobs in Q_b. All such jobs that become eligible as a consequence of device *active* state transitions are moved from Q_b to Q_r. Based on the priority of the jobs moved to Q_r, the currently running job J_{curr} may be preempted.

Figures 5.3 and 5.4 give the online routines needed to perform the actions in Figure 5.2. In Figure 5.3, function *SleepStateTransitions($\mathcal{S}$)* is used to transition a set of unused devices $\mathcal{S}$ to *sleep* state. In doing so, for each idle device one needs to predict next device usage times. This is achieved using the function $NDUT(D_i, t)$ that gives the next device usage time of D_i at time t. $NDUT(D_i, t)$ is based on the predictive RT-DPM policy CEEDS given in [41] and approximates next device usage times using task release times and device reactivation times (the earliest time an idle device will be used is when a job requiring the device arrives to the system or a blocked job requiring the device becomes eligible for execution).

At time t, in transitioning idle device D_i to *sleep* state, *SleepStateTransitions($\mathcal{S}$)* first tries to see if forbidden region FR_i can be enforced. If so, FR_i is enforced at $t' = NDUT(D_i, t)$ which effectively combines the inherent device idle interval $[t, t']$ captured by the predictive RT-DPM component of DFR-RMS (function $NDUT(D_i, t)$) and the device idle interval $[t', t' + \Delta_i]$ due to the FR enforcement. This creates a long contiguous device idle interval $[t, t' + \Delta_i]$ for D_i. The reactivation timer act_i of D_i and the next FR_i

```
Notations:
1 $t$:  Current time
2 $J_{curr}$:  Currently running job
3 $\mathcal{D}$:  Set of $m$ system devices
4 $nfr_i$:  Earliest next enforcement of $FR_i$
5 $act_i$:  Re-activation timer for device $D_i$
6 $Q_r$:  Ready job queue
7 $Q_b$:  Blocked job queue

NDUT($D_i$,$t$)
1  for each $T_j \in \beta_i$
2   if a job of $T_j$ in $Q_r$ or $Q_b$
3     $z_j$ = max($act_k$)  $D_k \in \gamma_j$
4     $z_j$ = max($t$, $z_j$)
5   else
6     $z_j$ = next job release time of $T_j \geq t$
7  return min($z_j$)  $T_j \in \beta_i$

SleepStateTransitions($\mathcal{S}$)
1  for each $D_i \in \mathcal{S}$ in active state
2   if $nfr_i \leq t$
3     $act_i = NDUT(D_i, t) + \Delta_i - T^i_{wu}$
4     $nfr_i = NDUT(D_i, t) + \Pi_i$
5     Transition $D_i$ to sleep state
6   else if $NDUT(D_i, t) - t > B_i$
7     $act_i = NDUT(D_i, t) - T^i_{wu}$
8     Transition $D_i$ to sleep state
```

Figure 5.3: The DFR-RMS Run-time Routines

```
Dispatch($J_i$)
1  SleepStateTransitions($\mathcal{D} - \gamma_i$)
2  Set $J_{curr} = J_i$
3  Dispatch $J_{curr}$

Scheduler()
1  $J_h$ job with highest priority in $Q_r$
2  if $J_{curr} \neq J_h$
3   if $J_{curr} \neq null$
4    Preempt $J_{curr}$
5   Dispatch($J_h$)
```

Figure 5.4: The DFR-RMS Run-time Routines (Continued)

enforcement nfr_i are updated appropriately.

Functions *Dispatch()* and *Scheduler()* in Figure 5.4 deal with selecting the highest priority eligible job in Q_r for execution and dispatching it. The *Dispatch()* function makes a call to *SleepStateTransitions()* in an effort to transition unused devices to sleep states.

Run-time Complexity: At either of the events corresponding to job arrival/completion and device re-activation, at most m device transitioning decisions are made. For each device transition, the computational overhead is constrained by next device usage predictions (function $NDUT(D_i, t)$) which takes $O(n)$ time. Thus, the overall run-time complexity of the framework is $O(mn)$ per decision point. Since the number of system devices m is typically small, the complexity of the framework for practical purposes can be treated as $O(n)$.

Extensions to DVS Settings with Dynamic Slack Reclaiming: The DFR algorithm can be used in DVS settings with any feasible task speed assignment. Further, the SDRA algorithm [10] is extended to fixed-priority settings for reclaiming. In SDRA, at dispatch time a job computes its 'earliness' by considering the slack of all higher priority jobs. Earliness represents the total slack available in the system from higher priority tasks that the current job can reclaim. Earliness is computed through the α-queue, which represents the canonical schedule in which all jobs take their worst-case execution time. All jobs, upon arrival, insert their worst-case execution times (under the nominal speed) to the α-queue. Jobs in the α-queue are ordered based on task priorities. A consequence of DFR algorithm is that, at dispatch time, care must be taken not to consider blindly all higher priority tasks in the α-queue: Some of these higher priority tasks may be currently blocked by forbidden regions. Reclaiming from such a task might indeed lead to potential deadline misses. Therefore, in DFR settings, the earliness is computed by considering only the higher priority jobs that have completed their execution.

5.3.4 Experimental Evaluation

This section gives the performance evaluation of the DFR scheme through simulations. A discrete event simulator was implemented in C for evaluation purposes. The results shown are from 1000 randomly generated synthetic task sets, each containing 20 periodic tasks. The periods of the tasks were randomly chosen to be in the interval [25ms, 1300ms] which corresponds to the period range observed in example real-time applications [85]. Task utilizations are generated randomly using uniform distribution and task worst-case execution times are calculated as the product of task periods and task utilizations. The energy consumption of the task set over the hyperperiod (LCM) was recorded. Each task is assumed to use 0-2 devices, determined randomly from the device specification list shown in Table 5.2 and adopted from [40]. The CPU power consumption at maximum frequency is considered to be 1.6W and modeled after the Intel Xscale at [127].

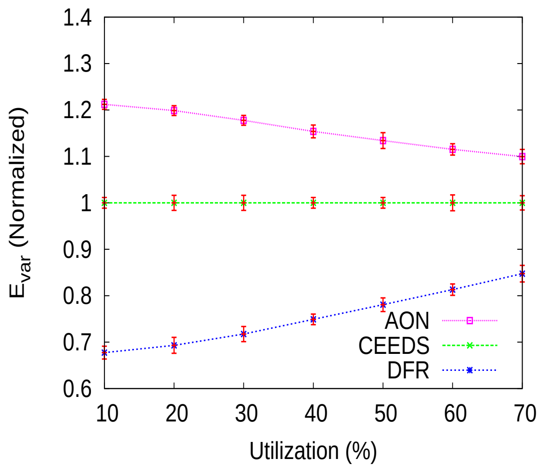
Device	P_a, P_s, $P_{wu}(P_{wu})$	$T_{wu}(T_{wu})$
Realtek Ethernet Chip	0.19, 0.085, 0.125(W)	10(ms)
MaxStream wireless module	0.75, 0.005, 0.1(W)	40(ms)
IBM Microdrive	1.3, 0.1, 0.5(W)	12(ms)
SST Flash SST39LF020	0.125, 0.001, 0.05(W)	1(ms)
Simple Tech Flash Card	0.225, 0.02, 0.1(W)	2(ms)
Fijitsu 2300AT Hard disk	2.3, 1.0, 1.5(W)	20(ms)

Table 5.2: Device Specifications

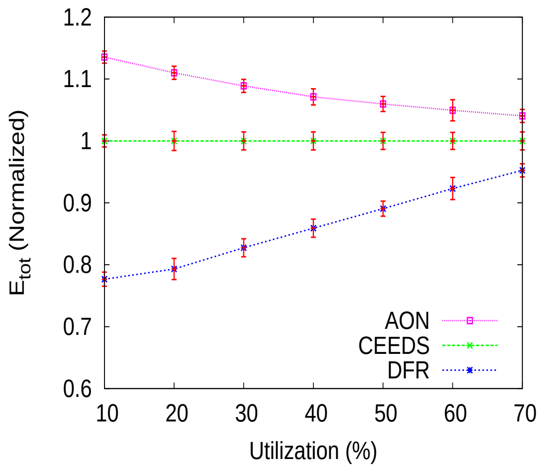
The performance of three schemes are compared: *Always On (AON)* where the devices remain in active state throughout the simulation (no dynamic power management), *CEEDS* [40] (adapted to RMS settings) and *DFR-RMS* in which devices are put to prolonged sleep intervals at run-time through device forbidden regions. For each scheme, the mean energy consumption and the corresponding confidence intervals at 99% confidence level are reported.

Let $E_{var} = \sum_{i=1}^{m}(E^i_{mod} + E^i_{trans})$. E_{var} represents the energy consumption of all devices when in *active* state but not in use (E_{mod}) and device transition energy (E_{trans}) over all devices. Observe that DPM algorithms do not have a direct impact on E_{fixed} (device energy when in *active* state and in use), but only on E_{var}, which can be reduced to the extent the algorithm is successful in increasing the length of sleep intervals and reducing unnecessary transitions. As such, when evaluating the energy consumption, both the total device variable active energy (E_{var}) and the total system energy (E_{tot}) are presented separately. While E_{var} shows the effectiveness of the DPM schemes, E_{tot} values show their impact on overall system energy.

First, settings where the tasks exhibit their worst-case workload are considered. The impact of utilization on the schemes are shown. The task set utilizations are scaled from 0.2 to 0.7. Figure 5.5 shows the relative performance of the DPM schemes, on settings where all jobs run at the maximum CPU speed (no DVS). All results are normalized with respect to the energy consumption of CEEDS. At lower utilization values, *DFR-RMS* effectively postpones device activations using forbidden regions. This helps in achieving prolonged sleep periods and clustered active periods where the device is in continuous use. As a result, *DFR-RMS* reduces E_{var} significantly: E_{var} gains are in the range of 15% to 32% when compared to CEEDS. *DFR-RMS* scheme successfully reduces E_{tot} by 5% to 22%, as well. At high utilization values, the device idle intervals are strongly constrained by device usage patterns, periods of tasks and break-even times of devices. These factors reduce the gains obtained by *DFR-RMS* scheme at high utilizations. Notice that even at 70%



(a) Energy consumption E_{var}



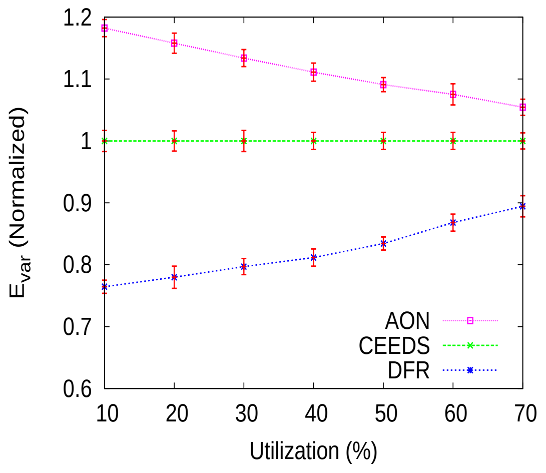
(b) Energy consumption E_{tot}

Figure 5.5: Energy consumption without DVS

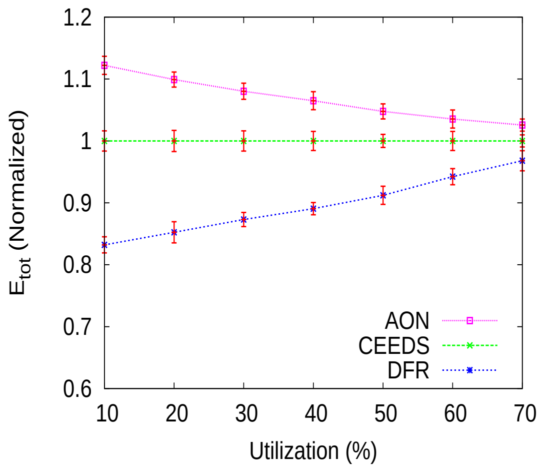
utilization it is possible to schedule some forbidden regions and obtain gains. This is due to the fact that schedulability of the task set is checked through time demand analysis given in Section 5.3.1 and forbidden regions are successfully able to exploit the CPU idle times.

The above experiments are repeated for DVS settings (Figure 5.6). An energy-efficient speed threshold for each task is computed through the technique given in [10]. Then, task level speed assignments are computed by the algorithm in [10] with feasibility bound set to the Liu-Layland bound for 20 tasks (approximately 70%). The results are similar showing *DFR-RMS*'s gains on DVS settings as well. Even though the relative performance of *DFR-RMS* is the same in both settings, the absolute E_{tot} values for DPM schemes with DVS are lower than those without DVS, showing the positive impact of DVS.

The next experiment shows the impact of varying device characteristics on DPM schemes in general. In this setup, CEEDS and *DFR-RMS* are compared. DVS is enabled and nominal speed values are assigned as above. The system utilization is fixed at 40%. For a given task set and a device usage pattern, first the values of E_{var} and E_{tot} are recorded for the P_a values given in [40]. Then, for the same task set and device usage pattern, the

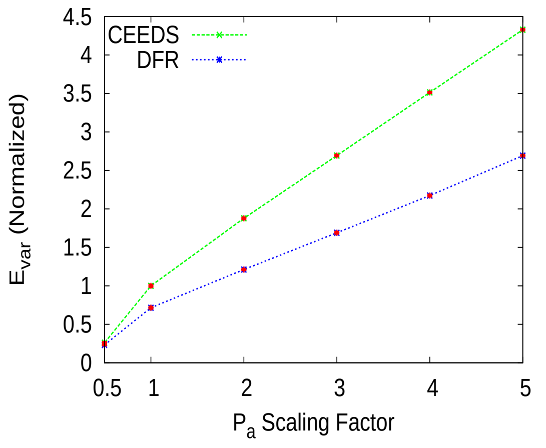


(a) Energy consumption E_{var}

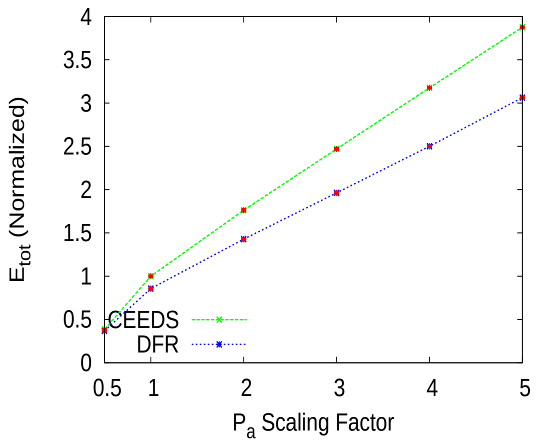


(b) Energy consumption E_{tot}

Figure 5.6: Energy consumption with DVS



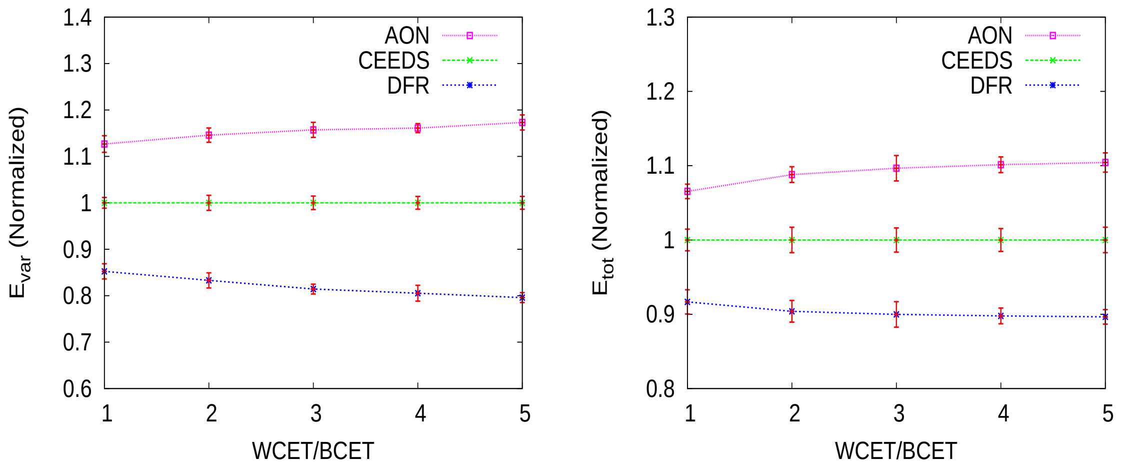
(a) Energy consumption E_{var}



(b) Energy consumption E_{tot}

Figure 5.7: Energy consumption with P_a scaling

experiment is repeated by scaling up/down P_a for all devices and re-computing the break-even times. Figure 5.7 shows the results normalized with respect to energy consumption of the task set with P_a for all devices taken from figure [40], i.e. when the scaling factor is one.



(a) Energy consumption E_{var} (b) Energy consumption E_{tot}

Figure 5.8: Energy consumption with reclaiming enabled

Note that, CEEDS forces a device to be in active state at the predicted next device usage time, however the device may not be used at this point. With *DFR-RMS* scheme, DFR postponements and forceful DFR starts at next device usage times help prolong to extend sleep intervals. This makes DFR scheme more tolerant to P_a scaling. Thus the gains of DFR scheme increase significantly as P_a is scaled up, indicating that DFR would be even more applicable in systems with significant power consumer devices.

Next, the effect of reclaiming on DFR scheme is considered (Figure 5.8). Again, the utilization is set to 40%. The actual workload is varied randomly between best case execution time (BCET) and worst-case execution time (WCET) of the task. The ratio of $\frac{WCET}{BCET}$ is varied from 1 to 5. SDRA [10] is used for reclaiming. As mentioned in Section 5.3.3, a consequence of DFR algorithm is that it is possible to reclaim slack only from higher priority tasks that have completed execution in the α-queue. This somewhat limits the gains for DFR. In fact, the relative performance of CEEDS and DFR appear to be mostly uniform throughout the spectrum.

5.4 DFR-EDF: A Unified Energy Management Framework for Dynamic-Priority Systems

In this section, the DFR approach is integrated with dynamic-priority systems and EDF policy. By combining DVS with DPM while taking into account the interplay between different energy management techniques as well as power characteristics of the system components a unified energy management framework, *DFR-EDF*, is developed.

5.4.1 EDF Schedulability Analysis with DFRs

While the potential benefits of DFRs for enhancing the effectiveness of DPM are clear, it is imperative to make sure that all task instances meet their deadlines under various DFR activation patterns. In this subsection, first the exact feasibility analysis of a real-time periodic task set under preemptive EDF policy in the presence of DFRs is formally shown to be co-NP-Hard in the strong sense. Then a sufficient schedulability condition for a set of periodic tasks scheduled with preemptive EDF is derived, in the presence of device forbidden regions with given duration (Δ) and period (Π) parameters. Later, Section 5.4.2 addresses the problem of determining Δ and Π parameters.

Theorem 3. *Given a task set ψ and a set of forbidden regions ϕ, deciding on the feasibility of ψ under preemptive EDF is co-NP-Hard in the strong sense.*

The proof of Theorem 3 can be found in Appendix B.

Definition 1. Υ_k *represents* all *the forbidden regions associated with* all *the devices used by the* k smallest period tasks.

Formally, if tasks are sorted in non-decreasing order of periods, Υ_k represents the set containing all the forbidden regions associated with devices $\cup_{i=1}^{k}\gamma_i$.

Theorem 4. *Given a set of periodic tasks $\psi = \{T_1 \dots T_n\}$ arranged in non-decreasing order of periods and a set of forbidden regions $\phi = \{(\Delta_1, \Pi_1) \dots (\Delta_m, \Pi_m)\}$, the periodic task set ψ can be scheduled by EDF in feasible manner if,*

$$\sum_{i\in\Upsilon_k}(\frac{\Delta_i}{\Pi_i}+\frac{\Delta_i}{P_k})+\sum_{j=1}^{k}\frac{C_j}{P_j}\leq 1 \quad k=1\dots n$$

The proof of Theorem 4 can be found in Appendix C.

Complexity: The schedulability test has n iterations and each iteration takes $O(m+n)$-time. Therefore, the overall complexity is $O(mn+n^2)$. Since the number of devices is typically much smaller than the number of tasks, the complexity of the static feasibility test can be considered as $O(n^2)$.

In DVS settings with variable processing frequency, the following corollary holds as a consequence of Theorem 4.

Corollary 3. *Given a set of periodic tasks $\psi=\{T_1\dots T_n\}$ arranged in non-decreasing order of periods, and a set of forbidden regions $\phi=\{(\Delta_1,\Pi_1)\dots(\Delta_m,\Pi_m)\}$, the periodic task set ψ can be scheduled by EDF in a feasible manner at the processing frequency f if,*

$$\sum_{i\in\Upsilon_k}(\frac{\Delta_i}{\Pi_i}+\frac{\Delta_i}{P_k})+\sum_{j=1}^{k}\frac{C_j}{f\cdot P_j}\leq 1 \quad k=1\dots n$$

5.4.2 Determining System Parameters for Effective Integration of DVS and DPM

To integrate DVS and DPM, *DFR-EDF* relies on a *power management configuration* $\mathcal{C}$ uniquely determined by a processing frequency f and a set of forbidden regions $\phi=\{(\Delta_1,\Pi_1)\dots(\Delta_m,\Pi_m)\}$. Section 5.4.1 provided an efficient test to decide whether the task set ψ is feasible under EDF with a given forbidden region set ϕ. Obviously, it is equally important to determine the power management configuration $\mathcal{C}$ that yields best energy savings, as there are typically many configurations that preserve the feasibility.

However, such a decision is not trivial. Typically, DVS and DPM components tend to favor power management configurations with opposing features. Reducing the processing frequency (to favor DVS) will scale up task execution times and hence seriously limit the DPM opportunities. On the other hand, configurations that favor DPM will create long idle intervals – but these typically require high CPU frequencies and will also need to take into account the device transition overheads. In fact, in the light of Theorem 1 (which indicates that RT-DPM is intractable even in the absence of DVS), one can safely state that an exact and efficient solution is unlikely. The *DFR-EDF* framework takes a more direct approach: it exploits several features of forbidden regions and known properties of DVS solutions to quantify the energy savings one can expect, given a power management configuration $\mathcal{C}$. Hence, the static phase of *DFR-EDF* includes an iterative procedure to decide on the run-time power configuration.

An inspection of Corollary 3 reveals that as more FRs are assigned to the system, the minimum frequency that guarantees feasibility will typically increase. Thus, there is a trade-off between assigning additional FRs to decrease device energy consumption and the resulting increase in processor energy consumption. *DFR-EDF* incrementally assigns FRs to the system. At each step, it evaluates the expected benefit (in terms of energy savings) of assigning an FR for another system device separately, and commits to the one which appears most promising. The process of assigning new FRs to the power management configuration stops when it is expected that by doing so the overall energy consumption will increase (due to, typically, excessive CPU energy consumption).

In the formal description of procedure, ΔE^i_{sys} denotes the expected change in system energy consumption as a consequence of adding a new forbidden region, FR_i, to an already assigned forbidden region set ϕ. Let ΔE^i_{device} and ΔE^i_{cpu} denote the expected decrease in device energy and the expected increase in processor energy respectively, due to the additional forbidden region FR_i during a hyperperiod. ΔE^i_{sys} can be expressed as:

$$\Delta E^i_{sys} = \Delta E^i_{device} - \Delta E^i_{cpu}$$

If $\Delta E^i_{sys} > 0$ then the system is expected to benefit from the additional forbidden region

FR_i. On the other hand, if $\Delta E^i_{sys} < 0$ then the system energy is likely to increase due to the addition of FR_i.

As a consequence of forbidden region enforcements, the devices are transitioned to *sleep* states. Each FR_i enforcement provides a potential device energy saving of $\Delta_i \cdot (P^i_a - P^i_s)$ and a device state transition energy overhead of E^i_{sw}. During a hyperperiod H, there can be at most $\lfloor \frac{H}{\Pi_i} \rfloor$ FR_i enforcements. As such, ΔE^i_{device} can be approximated as,

$$\Delta E^i_{device} = \lfloor \frac{H}{\Pi_i} \rfloor \cdot (\Delta_i \cdot (P^i_a - P^i_s) - E^i_{sw})$$

Let fa and fb denote the minimum frequencies that guarantee the feasibility of task set ψ with FR sets ϕ and $(\phi \cup FR_i)$, respectively. Let P_{cpu} denote the power consumption of the processor at maximum frequency. Recall that U_{tot} represents task set utilization under maximum processor frequency ($f_{max} = 1$). As a consequence of adding FR_i, the system is forced to switch to fb from frequency fa to guarantee feasibility. Clearly, $fb \geq fa$. Thus, ΔE^i_{cpu} can be quantified as,

$$\Delta E^i_{cpu} = H \cdot U_{tot} \cdot P_{cpu} \cdot (fb^2 - fa^2)$$

Finally, for a given FR_i, the range of possible Δ_i and Π_i values should be provided to make the algorithm's search component complete. Clearly, Δ_i must be no shorter than the corresponding device break-even time and cannot exceed the maximum laxity of a task using the corresponding device. Hence, $B_i \leq \Delta_i \leq \min\limits_{j | D_i \in \gamma_j} (P_j - C_j)$. Also, the ratio $\frac{\Delta_i}{\Pi_i}$ must not exceed $(1 - U_{D_i})$, where U_{D_i} is the total utilization of tasks using device D_i. Thus, $\frac{\Delta_i}{1 - U_{D_i}} \leq \Pi_i \leq H$.

In determining the energy-minimal system parameters FRs assignments are made one at a time, evaluating the benefit of such an assignment at each stage. An additional FR is only incorporated to the system if it seems to be beneficial from the system-wide energy

Notations and Assumptions:

ϕ: Set of FRs

f_{nom}: CPU frequency

H: Hyperperiod of task set

P_{cpu}: CPU power consumption

U_{tot}: Task set utilization

ψ sorted: $P_1 \leq \ldots \leq P_n$

DetermineEMPRoutine():

```
1  $\mathcal{Z} \leftarrow \{D_1, \ldots, D_m\}$
2  $\phi \leftarrow \emptyset$
3  $f_{nom} \leftarrow MinSchedulableFreq()$
4  repeat
5   $\forall D_i \in \mathcal{Z}$, $\Delta E^i_{sys} \leftarrow -1$
6   for each $D_i$ in $\mathcal{Z}$ do
7     ($E^i_{sys}$, $(\Delta_i, \Pi_i)$, $\varphi_i$) $\leftarrow BestSysFR(D_i)$
9     $E^{i*}_{sys} \leftarrow \max_{i \in \mathcal{Z}} (\Delta E^i_{sys})$
10    if($E^{i*}_{sys} > 0$)
11      $\phi \leftarrow \phi \cup (\Delta_{i^*}, \Pi_{i^*})$
12      $Z \leftarrow Z - D_{i^*}$
13      $f_{nom} \leftarrow \varphi_{i^*}$
14    endif
15  endfor
16 until ($\mathcal{Z} = \emptyset \quad OR \quad \Delta E^{i*}_{sys} \leq 0$)
17 $\mathcal{C} \leftarrow (f_{nom}, \phi)$
```

Figure 5.9: Determining Energy-Minimal System Parameters

```
**MinSchedulableFreq()**:
1 for $h = 1$ to $n$
2   $\alpha \leftarrow \sum_{j=1}^{h} \frac{C_j}{P_j}$
3   $\beta \leftarrow 1 - \sum_{i \in \Upsilon_h} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_h})$
4   $F_h \leftarrow \frac{\alpha}{\beta}$
5 endfor
6 return max($F_1, \ldots, F_n$)
**BestSysFR($D_i$)**:
1  for each of $N$ $(\Delta_j, \Pi_j)$ pairs do
2    $\phi \leftarrow \phi \cup (\Delta_j, \Pi_j)$
3    $\varphi_j \leftarrow MinSchedulableFreq()$
4    $\phi \leftarrow \phi - (\Delta_j, \Pi_j)$
5    if $\varphi_j \leq 1$
6      $\Delta E^i_{device} \leftarrow \lfloor \frac{H}{\Pi_j} \rfloor \cdot \Delta_j \cdot (P^i_a - P^i_s)$
7      $\Delta E^i_{cpu} \leftarrow H \cdot U_{tot} \cdot P_{cpu} \cdot (\varphi_j^2 - f_{nom}^2)$
8      $\Delta E^j_{sys} \leftarrow (\Delta E^i_{device} - \Delta E^i_{cpu})$
9    else
10     $\Delta E^j_{sys} \leftarrow -1$
11 endfor
12 $\Delta E^i_{sys} \leftarrow max(\Delta E^1_{sys}, \ldots, \Delta E^N_{sys})$
13 return ($E^i_{sys}$, ($\Delta_i$, $\Pi_i$), $\varphi_i$)
```

Figure 5.10: Determining Energy-Minimal System Parameters (continued)

perspective. With the addition of each FR the CPU operating frequency is also updated appropriately. This FR assignment process stops when no more FRs can be incorporated into the system in an system-wide energy-efficient manner that also preserves the feasibility of the task set.

DetermineEMPRoutine() given in Figure 5.9 determines the power management configuration $\mathcal{C}$ that consists of a set of FRs and CPU frequency f. At any time, ϕ keeps track of the FRs assigned to the system and f_{nom} represents the corresponding CPU frequency. *DetermineEMPRoutine()* requires two procedures *MinSchedulableFreq()* and *BestSysFR()*.

MinSchedulableFreq() determines the minimum CPU frequency f that guarantees the feasibility of task set ψ with FR set ϕ. *MinSchedulableFreq()* is implemented in n iterations by running the test provided in Corollary 3 and recording the minimum feasible frequency at each step. The maximum frequency value recorded through n iterations is the minimum value that guarantees the feasibility (as using a lower frequency would violate the feasibility condition for at least one iteration). If this maximum frequency exceeds $f_{max} = 1$ then ψ is not schedulable with ϕ. Note that *MinSchedulableFreq()* has the same complexity as the feasibility test provided in Section 5.4.1 ($O(mn + n^2)$).

When *BestSysFR()* is called for a specific device D_i it scans the range of Δ_i and Π_i for (a fixed number of) N equi-distant points and determines the (Δ_i, Π_i) pair and processor frequency φ_i that maximizes ΔE^i_{sys} while committing to the current FRs (set ϕ) and maintaining the feasibility.

Specifically, for device D_i, the range of Δ_i and Π_i is divided into $\lceil \sqrt{N} \rceil$ equi-distant points each. Let DS and PS denote this set of $\lceil \sqrt{N} \rceil$ Δ_i and Π_i points respectively. For each of the N possible combinations (Δ, Π), where $\Delta \in DS$ and $\Pi \in PS$, the minimum schedulable frequency, f_{sched}, is calculated to potentially incorporate the forbidden region (Δ, Π) for D_i. Then, ΔE_{sys} is computed based on the already assigned system FRs, the new FR (Δ, Π) and f_{sched}. Among the N pairs, the one that yields the best ΔE_{sys} value is considered. The complexity of *BestSysFR()* is $O(N \cdot (mn + n^2))$.

The *DetermineEMPRoutine()* procedure starts with an empty FR set and incrementally assigns FRs one at a time. At each stage, for each device D_i not associated with an FR (set $\mathcal{Z}$), a (Δ_i, Π_i) pair and processor frequency φ_i that maximizes ΔE^i_{sys} is calculated using the *BestSysFR()* routine. Following this, the most promising FR and corresponding processor frequency combination (i.e. the one with the largest positive ΔE^i_{sys}) is recorded as the new best configuration (ϕ, f_{nom}). This iterative process stops either when all devices are assigned with FRs, no more FRs can be assigned due to feasibility, or assigning additional FRs does not improve the system energy (i.e. $\Delta E^i_{sys} \leq 0$ for all devices in $\mathcal{Z}$).

Complexity: *DetermineEMPRoutine()* has at most m iterations. Each iteration invokes the *BestSysFR()* routine once for every device not associated with an FR. Thus, the complexity of each iteration is $O(mN(mn+n^2))$ making the overall complexity $O(m^2N(mn+n^2))$. Though this complexity is technically pseudo-polynomial, extensive simulations show that scanning no more than 50 equi-distant candidate Δ_i and Π_i values is typically sufficient to determine system parameters that yield significant energy savings for task sets of different sizes. Further, the number of devices in the system is typically small compared to the number of tasks. Consequently, in practice the complexity can be seen as quadratic in the number of tasks, which is affordable for a routine invoked only once in static analysis phase.

5.4.3 Online Components

Section 5.4.2 provided the details of the static component of *DFR-EDF*, which generates a power management configuration $\mathcal{C}$ with forbidden regions and a CPU frequency. This section presents the details of its online (dynamic) component. Such an on-line component is needed because:

- On the DPM side, one needs to decide at run-time when and for how long devices can be put to *sleep* state. This involves predicting when an idle device will be needed by a task in the future and managing DFRs. DFR management involves frequently enforcing and occasionally postponing the run-time activation of DFRs. DFR management is coupled with predictive DPM policies to create long contiguous device idle intervals during which system devices can be put to *sleep* state in energy-efficient manner.

- On the DVS side, one needs to calculate at run-time the energy-efficient frequency threshold below which DVS affects system-wide energy negatively. This energy-efficient frequency threshold is calculated as a function of the power characteristics of both the CPU and the active device set. The CPU frequency is never reduced below this threshold.
- Tasks rarely exhibit their worst-case workloads and leave unused CPU time (or slack). At run-time one needs to reclaim and manage the slack from task early completions. Further, one needs to exploit this slack using both DVS and DPM techniques to further reduce system energy consumption.

Online DPM through DFRs

Section 5.4.2 presented rules to determine the parameters of forbidden regions. Specifically, the length (or duration) Δ_i and period (or minimum separation time) Π_i of forbidden region FR_i were statically determined. Though, in theory, DFRs can be enforced strictly periodically, at run-time DFR enforcements are ensured not to interfere with the execution of a running job. In other words, the execution of a job cannot be preempted by DFR enforcements. This is accomplished by postponing DFRs enforcements as required. Thus, FR_i associated with device D_i is enforced at run-time only if the currently running job does not require D_i or the CPU is idle (i.e. FR_i is enforced only if D_i is idle and not in use). Once FR_i is enforced at run-time it has a duration of exactly Δ_i units.

The performance of the DPM component can be further improved by incorporating run-time prediction mechanisms, allowing enforcement of idle intervals even longer than the statically-determined FR durations. To see this, first observe that the pre-determined period Π_i of a given forbidden region FR_i gives the minimum separation time of two consecutive enforcements of FR_i: delaying the next enforcement of FR_i cannot have a negative impact on feasibility. This gives a powerful opportunity to enhance the DPM effectiveness at run-time. Specifically, when a device is idle, the next enforcement of its DFR can be postponed and aligned exactly with the device next usage time as estimated using prediction mechanisms. This gives the device a longer contiguous device idle interval during which it can yield higher energy savings. Figure 5.11 illustrates this principle.

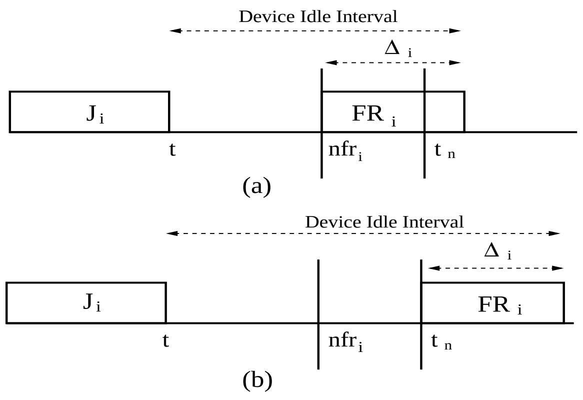


Figure 5.11: Postponements of DFRs

At time t, job J_i using device D_i completes and D_i is no longer in use. At this time let t_n be the estimated next device usage time determined through prediction mechanisms. Also, let $nfr_i < t_n$ be the earliest time in the future FR_i can be enforced. As shown in Figure 5.11(a), if FR_i is enforced at nfr_i, D_i effectively has an idle interval $[t, nfr_i + \Delta_i]$. However, if by postponing and aligning the next enforcement of FR_i at t_n, D_i would have a long contiguous idle interval $[t, t_n + \Delta_i]$ (Figure 5.11(b)). One can observe that by postponing the enforcement of FR_i to t_n, the overlap between the guaranteed device idle interval $[t, t_n]$ and the device idle interval of Δ_i units provided by FR_i is minimized. Thus, overall sleep time of D_i increases. Note that such an online optimization is possible only when $nfr_i \leq t_n$ holds.

As seen by the above example, combining FR enforcements with predictive DPM policies clearly increases the overall effectiveness of the DPM component. As such, the ability to accurately predict the inherent device idle intervals in the schedule is still important for the *DFR-EDF* framework. In the following, first the rules for predicting next device usage times (NDUT) by improving on the bounds given by CEEDS predictive policy [41] is given. Following this rules are provided for transitioning devices using DFR enforcements in conjunction with the improved predictive policy.

Predicting next device usage times: At time t, let C_k^r denote the remaining worst-case execution time of job J_k under the frequency assigned by the static routine. Let $\mathcal{HP}_k$ denote the set of jobs in the ready queue having higher priority than any unfinished job of T_k. In the worst-case feasible schedule (where all jobs present their worst-case workload), a given task T_k cannot start to execute until time $t + \sum_{j \in \mathcal{HP}_k} C_j^r$. Further, if an instance of T_k is not currently ready, then one needs to wait at least until its earliest next release time $R_k(t)$. Hence, in either case, at time t, T_k's next dispatch time cannot occur earlier than $max(R_k(t), t + \sum_{j \in \mathcal{HP}_k} C_j^r)$. Thus, at time t, D_i used by T_k can be put to *sleep* state for a maximum of $\mathcal{V}_i^k = max(R_k - t, \sum_{j \in \mathcal{HP}_k} C_j^r)$ time units without causing a deadline miss for T_k.

Interestingly, the *DFR-EDF* framework creates further opportunities to put a device to *sleep* state by exploiting the run-time information about idle intervals currently enforced for other devices. Specifically, note that when a device D_j is explicitly put to *sleep* state (e.g. through an *FR*), a task T_k using both D_j <u>and</u> another device D_i is guaranteed not to generate a request for D_i as well, while D_j remains in *sleep* state. This follows from the inter-task device scheduling paradigm, where all the devices of a given task must be ready before it can start to execute. In other words, the existence of a task T_k using both D_i and D_j during its execution, makes the *sleep* intervals of D_i and D_j inter-dependent at run-time.

Let n_j denote the end of a currently enforced idle interval for device D_j. At time t, a task T_k is guaranteed not to use any device $D_i \in \gamma_k$ for at least $(n_j - t)$ time units, if there exists another device $D_j \in \gamma_k$ for which an idle interval is already enforced until time n_j. Considering all other devices, one can see that $D_i \in \gamma_k$ cannot be used by T_k for a maximum of $\mathcal{W}_i^k = \ max(n_j - t), \ \ j | D_j \in (\gamma_k - D_i)$ time units.

Combining the two factors, namely the interference of high priority jobs and impact of already enforced sleep intervals it can be seen that at time t, $D_i \in \gamma_k$ can be safely put to *sleep* state for $\delta_i^k(t) = max(\mathcal{V}_i^k, \mathcal{W}_i^k)$ time units without causing a deadline miss for T_k. The maximum time D_i can be put to *sleep* state without affecting the feasibility can be determined by iterating over all tasks using D_i and taking the minimum.

Proposition 6. *A given device D_i can be put to sleep state at time t for $\delta_i(t) = min(\delta_i^k(t)), \quad k|D_i \in \gamma_k$ time units without compromising the system feasibility.*

Device state transitions: Rules for transitioning devices and performing effective DPM are now given. Two *actions* DEACTIVATE and ACTIVATE are defined. The DEACTIVATE action tries to transition all devices not in use to *sleep* state. For each device put to *sleep* state, a device re-activation time is set which indicates the time at which the sleeping device must start transitioning back to *active* state. Device re-activations can be handled by the operating system through timers. Each device is associated with a re-activation timer whose value is appropriately set while transitioning the corresponding device to *sleep* state. When the re-activation timer of a device expires, the ACTIVATE *action* tries to postpone its re-activation time if possible to do so, else starts transitioning the device to *active* state.

The *DeviceCntrlRoutine()* given in Figure 5.12 handles device transitioning decisions by combining $\delta_i(t)$ with the run-time enforcements of FRs. *DeviceCntrlRoutine()* is invoked at scheduling points corresponding to job completion/dispatch times with *action* set to DEACTIVATE and at device re-activation times with *action* set to ACTIVATE.

Let T_{wu}^i refer to the time delay involved in transitioning device D_i from *sleep* to *active* state. Due to T_{wu}^i, a device D_i transitioned to *sleep* state at time t and having expected next usage time $t_n > t$ must be re-activated at time $t_n - T_{wu}^i$. This will ensure D_i will be *active* and ready to service requests at time t_n.

J_h represents either the currently running job or the job to be dispatched at time t ($J_h = null$ when the processor idles). For each device D_i, nfr_i represents the earliest next

Notations:

t: current time

$\mathcal{S}$: Set of devices in *sleep* state

$\mathcal{A}$: Set of devices in *active* state

```
DeviceCntrlRoutine(action, $J_h$):
if(action = DEACTIVATE)
  $\forall D_i | D_i \in \mathcal{A}$
    if($D_i \notin \gamma_h$ && $t + \delta_i(t) \geq nfr_i$)
      $act_i \leftarrow t + \delta_i(t) + \Delta_i - T^i_{wu}$
      $nfr_i \leftarrow t + \delta_i(t) + \Pi_i$
      $\mathcal{S} \leftarrow \mathcal{S} \cup D_i$
      $\mathcal{A} \leftarrow \mathcal{A} - D_i$
      Start $D_i$ transition to sleep
  $\forall D_i | D_i \in \mathcal{A}$
    if($D_i \notin \gamma_h$ && $\delta_i(t) > B_i$)
      $act_i \leftarrow t + \delta_i(t) - T^i_{wu}$
      $\mathcal{S} \leftarrow \mathcal{S} \cup D_i$
      $\mathcal{A} \leftarrow \mathcal{A} - D_i$
      Start $D_i$ transition to sleep
if(action = ACTIVATE)
  $\forall D_i | (D_i \in \mathcal{S}$ && $act_i = t)$
    if($t + \delta_i(t) \geq nfr_i$)
      $act_i \leftarrow t + \delta_i(t) + \Delta_i - T^i_{wu}$
      $nfr_i \leftarrow t + \delta_i(t) + \Pi_i$
    else if($t + \delta_i(t) > act_i + T^i_{wu}$)
      Set $act_i \leftarrow t + \delta_i(t) - T^i_{wu}$
    else
      $\mathcal{S} \leftarrow \mathcal{S} - D_i$
      $\mathcal{A} \leftarrow \mathcal{A} \cup D_i$
      Start $D_i$ transition to active
```

Figure 5.12: DFR-EDF DPM Component

enforcement time of FR_i. If D_i is not associated with an FR, nfr_i is set to ∞. Also, as explained above due to device transition delays, each D_i has a reactivation timer act_i representing the time at which it must start transitioning to *active* state. $\mathcal{S}$ and $\mathcal{A}$ denote the set of devices in *sleep* and *active* states respectively at the current time t. At system start time nfr_i for all assigned FRs in the system is set to 0 and all devices are assumed to be in *active* state.

Let $\mathcal{A}' \subseteq \mathcal{A}$ denote the set of *active* devices not in current use. When invoked with *action* set to DEACTIVATE *DeviceCntrlRoutine()* tries to transition all devices in $\mathcal{A}'$. First devices in $\mathcal{A}'$ that are associated with an FR are considered. Every such device D_i has an inherent device idle interval $[t, t + \delta_i(t)]$. It is assumed that $\delta_i(t)$ is computed as explained in Proposition 6. Where possible this device idle interval is extended by postponing and aligning the next enforcement of FR_i exactly at $t + \delta_i(t)$ thus creating a long contiguous sleep interval $[t, t + \delta_i(t) + \Delta_i]$ for D_i (Lines 2-8).

Next both the set of devices in $\mathcal{A}'$ for which FR postponements were not possible and the set of devices in $\mathcal{A}'$ that have no associated FRs are considered. For every such device a transitioning decision is made based on the information, $\delta_i(t)$, estimated by the prediction mechanism (Lines 9-14). Note that by handling the DPM of devices for which FRs can be enforced prior to handling the DPM of other devices, *DeviceCntrlRoutine()* helps use the information of enforced FRs in determining $\delta_i(t)$.

Notice that $\delta_i(t)$ may increase with new job arrivals due to the workload of newly arriving high priority jobs. Thus, when *DeviceCntrlRoutine()* is invoked with *action* set to ACTIVATE at time t, among the devices that need to be transitioned to *active* state only those whose sleep intervals cannot be extended are transitioned. The remaining devices can remain in *sleep* state and their reactivation timers are updated accordingly based on their respective extended intervals. Also, FRs are enforced if possible to further elongate device sleep periods (Lines 16-25).

Complexity: The run-time complexity of *DeviceCntrlRoutine* is mainly constrained by the computation overhead involved in computing $\delta_i(t)$ for all system devices at time t. Next it is shown how this can be done in $O(n \log n)$ time using appropriate data structures. Before invoking *DeviceCntrlRoutine* at a given scheduling point some pre-processing is done and two temporary data structures: *c-array* and *v-array* are constructed. These data structures are destroyed at the end of *DeviceCntrlRoutine*.

c-array is constructed using the ready queue. Each node in the *c-array* has two associated values: deadline (d) and *cumulative interference* (CI). The cumulative interference on a job represents the total interference from the remaining worst case execution times of all high priority jobs currently in the ready queue. If the i^{th} node in the ready queue contains job J_k, then C_k^r will be denoted as x_i (i.e. $x_i = C_k^r$). The *c-array* is constructed in the following way: for each entry k in the ready queue create an entry in the *c-array* with the corresponding deadline and cumulative interference $CI_k = \sum_{j=1}^{k} x_j$. Notice that the *cumulative interference* of the *c-array* nodes can be recursively defined as:

$$CI_1 = x_i$$

$$CI_i = CI_{i-1} + x_i$$

Thus, building the *c-array* takes $O(n)$ time and further its entries are sorted in order of deadlines (since they are constructed using the ready queue).

The *v-array* is constructed using the *c-array*. The i^{th} node in the *v-array* represents the interference for the unfinished job instance of T_i from high priority jobs currently in the ready queue. This value can be obtained by performing a binary search like interpolation on the *c-array* using as key the deadline of the unfinished job instance. As such, computing value for each node of the *v-array* takes $O(\log n)$ time making the total complexity of creating the *v-array* $O(n \log n)$.

Once pre-processing is done and the *v-array* is constructed, computing $\mathcal{V}_i^k$ takes constant time while computing $\mathcal{W}_i^k$ takes $O(m)$ time. Therefore, computing $\delta_i(t)$ takes $O(n + mn)$ time. Since there are at most m devices computing all $\delta_i(t)$ values takes $O(mn + m^2 n)$ time.

Since m is usually small, for all practical purposes this complexity can be considered $O(n)$. Thus, the overall complexity of *DeviceCntrlRoutine* is constrained by the pre-processing phase and is $O(n \log n)$.

Dynamic Voltage Scaling

Many DVS schemes adopt a nominal (default) frequency f_{nom} [11,101,109] that is statically computed. In *DFR-EDF* framework, f_{nom} is determined using the *DetermineEMPRoutine()* routine as described in Section 5.4.2. It has been well-established that aggressive slowdown will affect system energy negatively and hence, at run-time, the frequency should not be lowered below a certain energy-efficient threshold [10, 41, 71]. *DFR-EDF* adopts the system energy-efficient scaling technique given in [41]. In that technique, an energy-efficient frequency threshold (f_{thres}) is calculated *dynamically* based on the power characteristics of both the processor and the devices in *active* state, at job dispatch times. The processor frequency is never reduced below this threshold.

Specifically, consider a DVS-enabled CPU where there are k discrete frequency levels $l_1, \ldots, l_k = f_{max}$. Then, f_{thres} at time t is calculated as the frequency $f \in \{l_1, \ldots, l_k\}$ that minimizes $\bar{E}(f)$ given by the expression:

$$\bar{E}(f) = (P_{cpu} + \sum_{D_i \in \mathcal{A}} (P_a^i - P_s^i)) \cdot \frac{1}{f} \tag{5.2}$$

In the above expression, $\mathcal{A}$ represents the *active* device set at time t and P_{cpu} denotes the CPU power consumption at maximum CPU frequency. Thus, $\bar{E}(f)$ approximates the total system energy consumption per unit of execution at frequency f. Since in practice both the number of system devices and the number of discrete CPU operation modes are relatively small, [41] provides offline pre-processing methods to determine f_{thres} in constant time during execution.

At run-time, the dispatch frequency of a job is calculated as $max(f_{nom}, f_{thres})$. This ensures both the feasibility and dynamic energy-efficient frequency constraints are met.

Generalized Slack Reclaiming

As many real-time tasks complete early at run-time, detecting and reclaiming unused CPU time (slack) has been a major tool for dynamic DVS schemes [11,101,132]. These schemes typically incorporate mechanisms to decide on the amount of slack that can be reclaimed without compromising the feasibility, before the CPU frequency is reduced. Dynamic slack can be also exploited by DPM techniques to increase the length of device idle intervals by delaying task executions [40]. *DFR-EDF* has a built-in mechanism that can be used to keep track of and reclaim dynamic slack, for DVS or DPM. However, possibly the most novel aspect of *DFR-EDF*'s generalized reclaiming mechanism is that, whenever possible, it allows the use of the same dynamic slack for *both* DVS and DPM (i.e. for both lowering the processor frequency and increasing the idle intervals of devices).

First a motivation example is given to illustrate this novel aspect. Figure 5.13 shows the schedule of three ready jobs with decreasing priority at time t under worst-case workload. Each job requires a worst-case execution time of 10 units under their nominal frequency assumed to be $f_{max} = 1$. Jobs J_1 and J_3 use device D_1 with break-even time 12 units, while J_2 uses no device. D_1 has no associated FR and is in *active* state at time t.

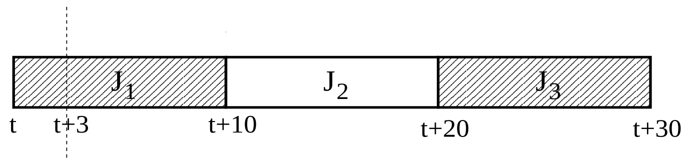


Figure 5.13: Reclaiming for both DVS and DPM

Assume J_1 completes early at time $t_1 = t + 3$, creating a dynamic slack of 7 units. At time t_1, J_2 will be dispatched and D_1 is no longer in use. Thus, the system needs to make a decision on whether or not to transition D_1 and decide on the processor frequency for executing J_2. Notice that at time t_1, if the dynamic slack is ignored, the device idle interval length of D_1 is predicted to be 10 units (the worst-case execution time of J_2) which is smaller than the break-even time of D_1. Hence, D_1 would be kept in *active* state by a naive predictive DPM policy. However, J_3 can start as late as $t + 20$ without violating system feasibility. Thus, by taking into account the dynamic slack of 7 units, one can transition D_1 at time t_1, as delaying its usage until $t + 20$ for 17 units (which is greater than the break-even time of D_1) would still keep the system feasible. Also, J_2 (dispatched at time t_1) can use the same dynamic slack of 7 units to lower its dispatch frequency to $\frac{10}{17}$. Thus, the dynamic slack of 7 units from J_1's early completion has supported both DVS and DPM.

The details of *DFR-EDF's* generalized slack monitoring and reclaiming mechanism are now provided. To keep track of unused run-times a technique similar to that used in Dual Speed Dynamic Reclaiming Algorithm (DSDR) [132] is adopted. When a job completes, its remaining (unused) run-time is added to a data structure called the *slack-queue*. Each element in the *slack-queue* has two components, one indicating the remaining run-time of the job and the other indicating its deadline. The *slack-queue* is maintained in non-decreasing order of deadlines. At time t, the remaining run-time of job J_i is denoted by rrt_i. At the release time of J_i, rrt_i is set to $\frac{C_i}{f_{nom}}$. At time t, when J_i is being dispatched, let $\mathcal{H}$ denote the set of elements in *slack-queue* with deadlines no greater than that of J_i. The slack available to J_i at time t due to early completions is given by:

$$slack_i = \sum_{j \in \mathcal{H}} rrt_j \tag{5.3}$$

Note that the remaining run-times of jobs and hence the *slack-queue* change with time and need to be updated accordingly. Below are the rules for updating job run-times and the *slack-queue*.

- When job J_i executes, as long as $slack_i > 0$ it consumes run-time from the head of the *slack-queue*. When $slack_i = 0$, J_i consumes its own run-time rrt_i.
- During an idle processor cycle, run-time is consumed from the head of the *slack-queue* (provided the *slack-queue* is non-empty).

- When the run-time of an element in the *slack-queue* is completely depleted it is removed from the *slack-queue*.

Slack Reclaiming for DVS: Recall from Section 5.4.3 that C_i^r represents the remaining worst-case execution time of J_i under f_{nom}. Further, from Section 5.4.3, when a job is being dispatched at time t, the frequency is not reduced below the energy-efficient frequency threshold (f_{thres}). Thus, J_i with available slack $slack_i$ is dispatched at frequency:

$$f = max(\frac{C_i^r}{slack_i + rrt_i}, f_{thres}) \tag{5.4}$$

The first component of Eq (5.4) is adopted from [132] where it is formally proved that this frequency does not violate the system timing constraints. Since f in Eq (5.4) is no less than its first component, it preserves system feasibility.

Slack Reclaiming for DPM: The slack that is available for a specific job at dispatch time can be also used to improve the estimation of the maximum time a device can idle without compromising the feasibility. Specifically, recall from Section 5.4.3 that $\mathcal{V}_i^k$ was defined as the maximum time D_i can remain in *sleep* state at time t without causing a deadline miss for T_k and was computed through the total remaining workload of ready jobs in $\mathcal{HP}_k$ (jobs with higher priority than any unfinished job of T_k). Notice that the very same principle can be applied to re-define $\mathcal{V}_i^k$ in a more precise manner: since in a pessimistic scenario the job of T_k would have to be delayed until all high priority jobs complete with their worst-case workload, delaying T_k during the unused run-times of such completed jobs (i.e. effectively keeping D_i in *sleep* state) would not hurt its feasibility. Let $slack_k$ denote the sum of remaining run-times from all completed jobs that is available to the earliest unfinished job instance of T_k. $\mathcal{V}_i^k$ is updated as:

$$\mathcal{V}_i^k = max(R_k(t) - t, slack_k + \sum_{j \in \mathcal{HP}_k} C_j^r)$$

```
Event: Release of job $J_i$
1 $rrt_i \leftarrow \frac{C_i}{f_{nom}}$
2 Insert $J_i$ in Ready Queue
3 Scheduler()
Event: Completion of job $J_i$
1 Insert $J_{curr}$ to slack queue
2 $J_{curr} = null$
3 if (ready queue = null)
4   DeviceCntrlRoutine(DEACTIVATE, null)
5 else
6   Scheduler()
7 end if
Event: Scheduled time to start device re-activation
1 DeviceCntrlRoutine(ACTIVATE, null)
Event: Completion of device re-activation
1 Scheduler()
```

Figure 5.14: DFR-EDF Run-Time Adjustments

```
GetNextEligibleJob()
1 $\mathcal{E} \leftarrow null$
2 for each $J_i$ in Ready Queue
3  if $D_i$ active $\forall D_i \mid D_i \in \gamma_i$
4    $\mathcal{E} \leftarrow \mathcal{E} \cup J_i$
5 end for
6 return highest priority job in $\mathcal{E}$
Dispatch($J_i$)
1 DeviceCntrlRoutine(DEACTIVATE, $J_i$)
2 $f_{thres} \leftarrow f$ that minimizes Eq (5.2)
3 $slack_i \leftarrow \sum_{J_k \in \mathcal{H}} rrt_k$ (from Eq (5.3))
4 $f \leftarrow max(\frac{C_i^r}{slack_i + rrt_i}, f_{thres})$ (from Eq (5.4))
5 $J_{curr} = J_i$ /* Update current job */
6 Dispatch $J_{curr}$ at smallest $l_i \geq f$
Scheduler()
1 $J_n \leftarrow GetNextEligibleJob()$
2 if ($J_n \neq null$ AND $J_n \neq J_{curr}$)
3  if ($J_{curr} \neq null$)
4    Preempt $J_{curr}$
5  end if
6  Dispatch($J_n$)
7 end if
```

Figure 5.15: DFR-EDF Run-Time Adjustments (continued)

One can create an additional *c-array* like data structure (as explained in Section 5.4.3) but using the *slack-queue*. Further, an additional *v-array* like structure can be created using this *c-array* build from the *slack-queue* to compute $slack_k$ in constant time. Since, building these additional structures takes no more than $O(n \log n)$ time (Section 5.4.3), it does not increase the run-time complexity of the pre-processing phase and hence that of *DeviceCntrlRoutine*.

Figures 5.14 and 5.15 summarize the main principles behind the online component of the DFR-EDF framework. Four events are defined corresponding to release of a job, completion of a job, scheduled time at which a device in *sleep* state must start transitioning back to *active* state and the time at which such a device re-activation completes. Figures 5.14 and 5.15 gives the rules and routines to be called at each of these events. J_{curr} in the Figures represent the current job in execution.

Routine *GetNextEligibleJob()* returns the current highest priority eligible ready job for execution. A ready job is said to be eligible for execution if all its devices are in *active* state. The *Scheduler()* uses the *GetNextEligibleJob()* routine to determine which ready job to execute. In addition to invoking the *Scheduler()* at events corresponding to job completion/preemption, it is also necessary to do so when a device in *sleep* state returns to *active* state as this may make a higher priority job eligible for execution, hence necessitating a preemption.

The *DeviceCntrlRoutine()* is called with option DEACTIVATE at job completion/dispatch events, to try transition to *sleep* state, the devices that are unnecessarily *active*. Similarly, at device re-activation times *DeviceCntrlRoutine()* is called with option ACTIVATE to begin transitioning appropriate devices back to *active* state.

The initial run-time of jobs are set at their release times (Line 1, under job release event) and jobs are inserted to the slack queue upon completion (Line 1, under job completion event). The slack queue is assumed to be updated according to the rules described previously.

5.4.4 Experimental Evaluation

The experimental evaluation settings are the same as those described previously in Section 5.3.4. As before, device specifications are as listed in Table 5.2 and the CPU power consumption at maximum frequency is modeled after the Intel Xscale processor at $1.6W$ [127]. In these experiments, the processor is assumed to operate at 10 discrete normalized frequency levels $[0.1, 0.2, \ldots, 1]$. In addition to *DFR-EDF*, four well-known energy management algorithms are implemented:

- *UNI-DVS* (Uniform DVS) is the CPU energy management scheme based on [11]. *UNI-DVS* scales the execution times of all tasks uniformly by setting the CPU frequency to the minimum value that guarantees system feasibility (i.e. all jobs are dispatched at frequency equal to system utilization). There is no DPM component in *UNI-DVS* and all devices kept in *active* state at all times.

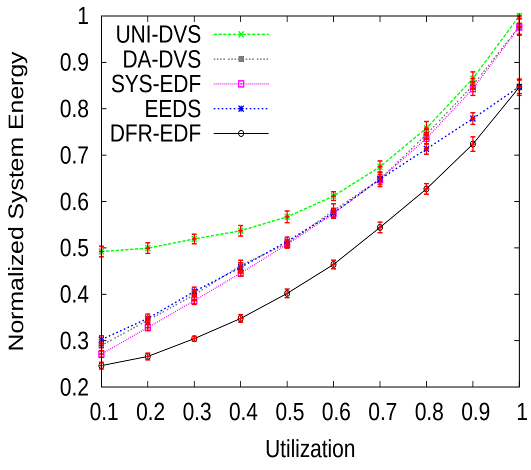
- *UNI-DVS* (Uniform DVS) is the CPU energy management scheme based on [11]. *UNI-DVS* scales the execution times of all tasks uniformly by setting the CPU frequency to the minimum value that guarantees system feasibility (i.e. all jobs are dispatched at frequency equal to system utilization). There is no DPM component in *UNI-DVS* and all devices kept in *active* state at all times.

- *EEDS* (Energy-Efficient Device Scheduling) scheme, adopted from [40], is a state-of-the-art DPM-only scheme for dynamic priority systems and EDF scheduling. There is no DVS component in *EEDS* and it is designed for systems where the device power dominates over that of the CPU.

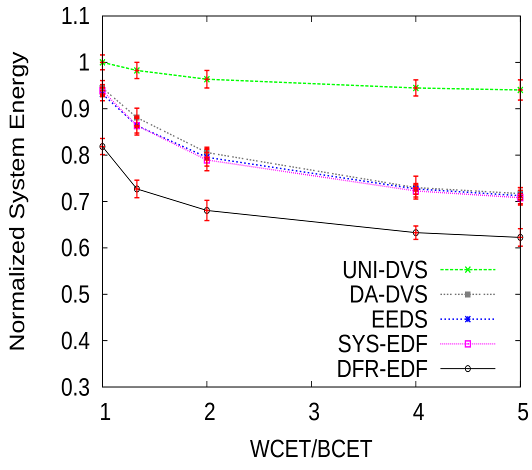
- *DA-DVS* (Device-Aware DVS) represents the schemes that exploit the concept of energy-efficient scaling. The DVS algorithm was adopted from [10] which gives the optimal task level slowdown factors by taking into account device energy-efficient frequency thresholds. Though [10] does not consider the DPM-related issues, for completeness a DPM component was incorporated to *DA-DVS*. *DA-DVS* performs device transitions based on the CEEDS (Conservative Energy Efficient Device Scheduling) algorithm given in [41].

- *SYS-EDF* is a system-level energy management scheme with both DVS and DPM components [41]. *SYS-EDF* performs DVS using the concept of energy-efficient scaling and has a simple prediction-based DPM component that is applied at run-time.

In all schemes, when there is a sufficiently long processor idle interval, the CPU is transitioned to its low-power state. CPU state transitions are assumed to incur energy overheads, but have negligible latencies. In line with Intel Xscale [96, 127] figures, the CPU idle time threshold is set to $10ms$, CPU transition energy overhead to $0.8mJ$ and CPU idle power to $0.08W$. The mean energy consumption and the corresponding confidence intervals at 99% confidence level are reported.



(a) Impact of system utilization



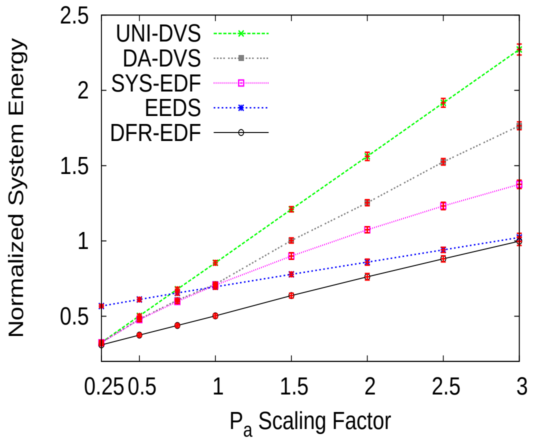
(b) Impact of workload variability

Figure 5.16: Impact of System Utilization and Workload Variability

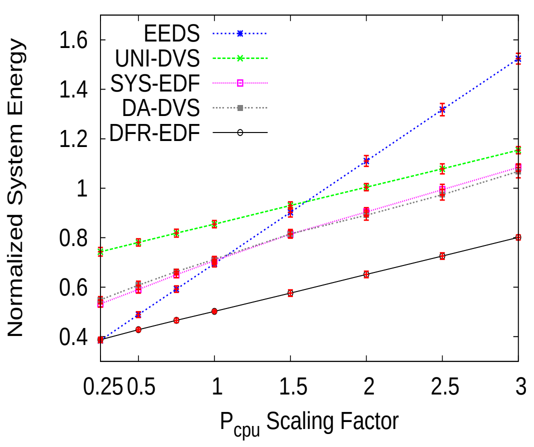
Figure 5.16(a) shows the relative performance of these schemes as a function of system utilization with worst-case workloads. All energy values are normalized with respect to the energy consumption of *UNI-DVS* at 100% system utilization. *DFR-EDF* provides clear gains over all schemes throughout the entire spectrum. *UNI-DVS* performs poorly compared to other schemes since it does not have a DPM component. Among the four schemes with both DVS and DPM components, the performance of *SYS-EDF* and *DA-DVS* quickly degrades at high utilization values, whereas *DFR-EDF* maintains its high performance. This is because, with increasing utilization, the system has to use high frequencies and hence becomes increasingly dependent on DPM (rather than DVS) for energy management. Further, the DPM policy of *SYS-EDF* and *DA-DVS* is a relatively simple lookahead-based prediction scheme.

DFR-EDF with its sophisticated DPM component effectively minimizes device energy at high utilization values which translate to significant system energy savings. In fact, notice that even *EEDS* with its more comprehensive DPM approach is able to outperform *SYS-EDF* and *DA-DVS* at increasing utilization values when the latter's processor energy gains become less significant. By judicially combining DVS and DPM, *DFR-EDF* outperforms all other schemes by margins of up to 20%.

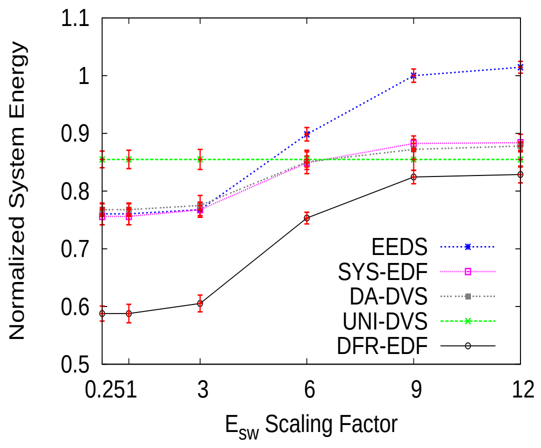
Figure 5.16(b) shows the relative performance of schemes under variability in the actual workload. This variability is controlled by modifying the worst-case to best-case execution time ratio, $\frac{WCET}{BCET}$. The actual workload of each job is determined randomly at its arrival time, according to a uniform probability distribution between $BCET$ and $WCET$. Clearly, the higher the $\frac{WCET}{BCET}$ ratio, the larger the dynamic slack that can be used for additional energy savings. In these experiments, the system utilization is fixed at 60% and all energy values are normalized with respect to energy consumption of *UNI-DVS* at $\frac{WCET}{BCET} = 1$. Notice that with increasing $\frac{WCET}{BCET}$ ratio the energy consumption of all schemes decreases, but relative improvements after a certain threshold become marginal. This is because with ample slack, with DVS, tasks tend to run at energy-efficient frequency thresholds. Similarly, for DPM, the idle interval of each device is naturally bounded by the periods of tasks using it. As explained in Section 5.4.3, whenever possible *DFR-EDF* utilizes the same dynamic slack for both DVS and DPM.



(a) Impact of Device Power



(b) Impact of CPU Power



(c) Impact of Device Transition Power

Figure 5.17: Impact of System Parameters

Figures 5.17(a), 5.17(b) and 5.17(c) show the impact of varying power characteristics of system components. In these experiments the system utilization is fixed at 60% and $\frac{WCET}{BCET} = 1$. In Figure 5.17(a), for each taskset and device usage pattern, the device active power (P_a) of all devices is multiplied by a certain scaling factor and recompute the device break-even times while keeping the processor power the same. That is, the higher the scaling factor, the more dominant the device power. Similarly, in Figure 5.17(b), the processor power consumption at the maximum frequency (P_{cpu}) is multiplied while keeping the device power characteristics the same. And, in 5.17(c) the device transition power is multiplied, which in turn effects device transition energy (E_{sw}), while keeping all other parameters the same.

In all these experiments, for each scaling factor the energy consumed by all schemes are recorded. All energy values are normalized with respect to scaling factor of one (i.e. the original device/processor parameters as in [40, 127]).

With decreasing P_a scaling factors or increasing P_{cpu} scaling factors, processor energy consumption becomes more dominant and thus the role of DVS proves more crucial compared to that of DPM in minimizing the system energy. Thus, at low P_a scaling factors and high P_{cpu} scaling factors *EEDS* performs worse compared to all other schemes. In fact, there are regions in Figures 5.17(a) and 5.17(b) where *EEDS* performs worse than *UNI-DVS* which does not have a DPM component. Similarly, with increasing P_a scaling factors or decreasing P_{cpu} scaling factors, device energy consumption becomes more dominant and thus DPM becomes critical for minimizing system energy. Thus, in these regions, *EEDS* outperforms both *SYS-EDF* and *DA-DVS*. However, due its inherent design features that exploit DVS and DPM in a synergistic way, *DFR-EDF maintains a robust performance throughout all scaling factors* – unlike other schemes, *DFR-EDF* does not suffer from excessive degradations in settings where CPU or device power dominates.

In Figure 5.17(c), the energy consumption of all schemes, except *UNI-DVS* which has no DPM component, increases with E_{sw}. *EEDS* which is an aggressive DPM policy is most sensitive to E_{sw} scaling performing significantly worse than all other schemes at high E_{sw} values. *SYS-EDF* and *DA-DVS* end up performing atleast as worse as *UNI-DVS* (with no DPM policy) at high E_{sw} values. By accounting for device energy overheads in determining the energy-efficient system configuration (Section 5.4.2), *DFR-EDF* outperforms all schemes in the entire spectrum.

5.4.5 Integrating Resource Access Protocols to DFR-EDF

While certain I/O operations can be interrupted, others are inherently non-interruptable. As such, in practice, it may not always be possible to preempt all system components and resources at arbitrary times. For such non-preemptable serially re-usable resources, mutual exclusion must be enforced at run-time.

To enforce mutual exclusion, several mechanisms are proposed including semaphores and mutex locks [13, 83, 112]. Without loss of generality, one instance of each non-sharable resource in assumed the system. This implies that *binary semaphores* can be used to enforce non-preemptive execution while holding the resources. Specifically, there is a unique and distinct semaphore S_i associated with each non-preemptable serially re-usable resource D_i. Some resources may be still usable in preemptive fashion. Without loss of generality resources $D_1, \ldots, D_h$ are assumed to be used in non-preemptive fashion; but resources $D_h, \ldots, D_m$ can be used preemptively $(1 \leq h \leq m)$. As a result, before using a non-preemptable serially re-usable resource D_i $(i \leq h)$, the requesting task has to execute *wait(S_i)* operation and it must release it by executing *signal(S_i)* [83]. The initial value of all binary semaphores are set to zero.

While the above solution effectively guarantees mutual exclusion on non-preemptable serially re-usable resources, it introduces the well-known priority inversion problem. A high priority job (J_h) is blocked from execution by a low priority job (J_l) which is holding a non-sharable resource also required by J_h. While J_h is blocked by J_l, a medium priority job J_m with priority greater than J_l, but smaller than J_h, arrives to the ready queue. J_m needs no resources and hence is able to successfully preempt J_l and execute. Execution of J_m in this scenario indirectly extends the blocking time of J_h. As a consequence of this priority inversion problem, the high priority job J_h can potentially suffer from indefinite blocking. This unbounded priority inversion problem can lead to potential deadline violations in real-time systems. Several resource access protocols such are Stack Resource Policy (SRP) and Priority Ceiling Protocol (PCP) have been proposed to eliminate the unbounded priority inversion problem [13, 83, 112].

The following describes how to incorporate *SRP* to *DFR-EDF* framework, assuming that each non-preemptable serially re-usable resource is accessed through *wait()* and *signal()* operations on the corresponding binary semaphores.

The main idea behind *SRP* is to allow job executions only when all resources required by the job are available. Thus, once a job begins execution it will not be blocked again until it completes. This way, *SRP* ensures that job blocking times are bounded. In *SRP*, each task T_i has a priority level and preemption level.

- *Priority level* of task T_i, denoted as p_i, is assigned either statically or dynamically and is dictated by the real-time scheduling policy.

- Each task T_i is statically assigned a *preemption level* based on its relative deadline. Smaller the relative deadline of a task, higher it's preemption level. All jobs of a task share the same preemption level as that of the task. Preemption level of job J is denoted by $pl(J)$.

A job J_a is allowed to preempt a job J_b only if $pl(J_a) > pl(J_b)$. To solve the unbounded priority inversion problem, *SRP* uses the concept of *resource ceiling* and *system ceiling*.

Resource ceiling of resource D_i, denoted by cl_i, represents the highest preemption level of jobs that could be blocked on D_i. If D_i is available $cl_i = 0$. Otherwise,

$$cl_i = max\{pl(J) \mid D_i\ blocks\ J\}$$

System ceiling is a time-varying quantity that is defined as $cl_{sys} = max(cl_1, \ldots, cl_m)$.

In SRP a job J can preempt/start execution only when the following condition holds:

- J has the highest priority among all ready jobs.

- Preemption level of J is greater than the current system ceiling (i.e. $pl(J) > cl_{sys}$).

Direct application of *SRP* in *DFR-EDF* settings is not sufficient. To illustrate this fact, consider the following example scenario. At time t, there are no ready jobs and a DFR is enforced for resource D_1. The DFR activation for D_1 will be in effect in the interval $[t, t_1]$. Now at time t_2 $(t < t_2 < t_1)$, job J_1 using resources D_1 and D_2 arrives. J_1 is blocked from execution until time t_1 by the FR enforced for D_1. Now, assume another job J_2 using resources D_2 and D_3 arrives to the system at time t_3 $(t_2 < t_3 < t_1)$. At time t_3, both resources required by J_2 are available and the rules of *SRP* as described above do not prevent the execution of J_2 at time t_3. However, if J_2 executes beyond t_1 (the time at which FR activation for D_1 will end), it can block the execution of J_1 (due to shared resource D_2), thereby extending the blocking time of J_1.

As seen from the above example, with DFR's, the system must be careful in executing a low priority job J_i during the duration of a forbidden region activation FR_j, even if all resources needed by J_i are currently available. This is because, allowing such an execution can potentially lead to unbounded priority inversion, if J_i is also subsequently preempted by other jobs. The solution to this problem is based on the observation that an active DFR effectively blocks a job J_i which may arrive during its enforcement. Hence the resource ceiling definition must be updated accordingly to address unbounded priority inversion problem. The definition of resource ceiling is modified as follows:

Resource ceiling of resource D_i, denoted by cl_i, represents the highest preemption level of jobs that could be blocked on D_i. If D_i is available and has no FR enforced $cl_i = 0$. Otherwise, $cl_i = max\{pl(J) \mid D_i \text{ blocks } J\}$.

The definition of system ceiling and scheduling rules remain the same as described above.

Note that with this modification, in the above example, when an FR is enforced for D_1 at time t, the system ceiling is raised to at least the resource ceiling of D_1 (i.e. $cl_{sys} \geq cl_1 \geq pl(J_1)$). This prevents J_2 from executing at time t_3 as $pl(J_2) < pl(J_1) \leq cl_{sys}$.

For each task T_i, b_i is defined to be the maximum resource blocking time. b_i is calculated exactly the same way it is done with *SRP* [13]. In a scenario where both preemptable and non-preemptable resources are present, the value attributed by preemptable resources in the calculation of b_i is assumed to be zero.

Theorem 5. *Given a set of periodic tasks $\psi = \{T_1 \dots T_n\}$ arranged in non-decreasing order* *of periods, and a set of forbidden regions* $\phi = \{(\Delta_1, \Pi_1) \dots (\Delta_m, \Pi_m)\}$*, the periodic task set* ψ *can be scheduled by* DFR-EDF *and* DFR-SRP *if,*

$$\frac{b_k}{P_k} + \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{f \cdot P_j} \leq 1 \quad k = 1 \dots n$$

The full proof of Theorem 5 is given in Appendix D.

5.5 Chapter Summary

This chapter proposed a novel device forbidden region (DFR) based approach to RT-DPM. Using the DFR approach system-level energy management frameworks were developed, *DFR-RMS* for fixed priority systems and *DFR-EDF* for dynamic priority systems. The device energy minimization problem for general periodic tasks (even in the absence of DVS) was formally shown to be NP-Hard in the strong sense. Also, it was formally proved that developing an exact feasibility test for preemptive EDF scheduling and DFRs is co-NP-Hard in the strong sense. Sufficient feasibility tests were developed for a set of periodic real-time tasks scheduled with RMS and EDF in the DFR framework. Using this test, an algorithm was provided to determine the DFR configuration and processing frequency for efficient integration of DPM and DVS in a single framework. Further, online components were developed to integrate predictive DPM policies and use slack reclaiming for both DVS and DPM simultaneously at run-time. Simulation results indicate that DFR-RMS and DFR-EDF offers significant energy gains over state-of-the-art energy management schemes. Finally, the well known Stack Resource Policy was modified for use in conjunction with DFR-EDF to handle non-preemptable serially re-usable resources.

Chapter 6: Energy Management of Periodic Real-time Tasks on Chip-Multiprocessors

6.1 Introduction

The focus of this chapter is the effective system-level energy management of periodic real-time tasks on a set of homogenous processing cores that share the same supply voltage and frequency. The proposed framework assumes partitioned scheduling based on preemptive EDF policy on each core.

The global DVS model poses a number of challenges that only very recently started to attract attention [73,110,129]. To start with, under the global voltage/frequency constraint, the core with the maximum load at a specific time becomes the main deciding factor in the overall CMP energy consumption [110, 129]. This suggests the importance of load balancing [66, 110, 129]. However, the frequency-independent power characteristics (which limit the efficacy of DVS) may vary over time, complicating the problem. In addition, while parallelism in general helps to save energy [12,37,135], the increasing core-level static power trends effectively limits the energy-efficient parallelism level: for light workloads, it can be more energy-efficient to use only a subset of cores (and put others to low-power states), as opposed to keeping all cores in *active* state [26, 94, 114].

This chapter addresses two primary problems in the energy management on CMP platforms. The first one is energy-efficient core activation and task allocation: to determine, for a given workload, the energy-optimal number of cores to activate and further, the mapping of tasks to active cores. The other problem is the run-time dynamic energy management through coordinated DVS and core idle state transitions.

6.2 CMP System Model

6.2.1 Processor Model

The system has a set of n periodic real-time tasks $\psi = \{T_1 \dots T_n\}$ that are partitioned upon m homogeneous processing cores $\mathcal{C}_1 \dots \mathcal{C}_m$. ψ_i denotes the subset of tasks allocated to core $\mathcal{C}_i$. As before, T_i is defined by the parameters C_i and P_i. The load on core $\mathcal{C}_i$ is defined as the total utilization of tasks allocated to $\mathcal{C}_i$, namely, $\sigma_i = \sum_{T_j \in \psi_i} U_j \leq 1$.

6.2.2 Power Model

As in [73, 110, 129], the CMP platform is assumed to have the Global DVS feature, that is, the voltage can be adjusted for all active cores uniformly, along with the frequency (up to an upper bound f_{max}).

ACPI defines an *active* state in which the core executes instructions. The exact power profile in *active* state (defined as state $\mathcal{C}_0$ in ACPI) will consist of *static* and *dynamic* power figures. In the *active* state, by using the power model from [10, 105, 137], the power consumption of a core $\mathcal{C}_i$ executing task T_j is modeled as:

$$P_i(t) = P_{static} + a_j V^2 f + P_{ind}^j$$

where $a_j V^2 f$ and P_{ind}^j represent the *frequency-dependent* and *frequency-independent* components of active power, respectively. V denotes the supply voltage and f denotes the CPU clock frequency. a_j is the effective switching capacitance of T_j. Note that the values of a_j and P_{ind}^j depend on the characteristics of the task T_j executing on core $\mathcal{C}_i$ at a given time.

In Global DVS settings, all *active* cores are inherently constrained to operate at the same supply voltage and frequency level [73, 110, 129]. Given the almost-linear relationship between the supply voltage and frequency, the power consumption of the *active* core $\mathcal{C}_i$ at time t can be expressed as:

$$P_i(t) = P_{static} + a_j f^3 + P_{ind}^j \tag{6.1}$$

The *aggregate* power consumption of all the cores varies with time and is a function of individual core states and the global operating frequency of all *active* cores. Let H be the hyperperiod of the task set ψ. The energy consumption of the system over the interval $[0, H]$ is given as:

$$E = \int_0^H \sum_{i=1}^{m} P_i(t)\, dt$$

When a core is not executing any instructions, it may be put in one of the various idle states [139]. Each idle state has a different power consumption characteristic; as a general rule, the lower power consumption in a given idle state, the higher the time and energy overheads involved in returning to the active state. While the exact number of idle states varies from architecture, this work assumes the existence of at least the following three fundamental states that are supported by most modern multicore systems (including Intel Core 2 Duo, Intel Xeon, most recent Intel i7-Nehalem lines):

- *Halt* state: In this state, the execution of instructions is halted and the core clocks are gated, resulting in significant reduction in dynamic power. The core can return to *active* state almost instantaneously ($\approx 10ns$) [77, 139]. The power consumption on core $\mathcal{C}_i$ in the *halt* state is given by $P_i = P_{static} + P_0$, where P_0 is the reduced dynamic power.

- *Sleep* state: Here, further, the Phase Locked Loops (PLLs) are gated and $L1$ cache contents are invalidated. In this state, dynamic power is eliminated thus making P_{static} the only component of power consumption. However, this saving in power consumption comes at the cost of addition overheads compared to the *halt* state.

Returning to *active* state may require a few hundred microseconds and involves non-trivial energy overheads [95, 139].

- *Off* state: Here, the core voltage is reduced to very low levels, to make even the static power consumption negligible. CPU context is not preserved and returning to *active* state involves significant time and energy overheads [139]. Intel's new *i7* architecture achieves this very low energy consumption through *power gating* feature [77].

Table 6.1 provides a summary of the additional notations used in this chapter.

$\mathcal{C}_i$	Core i
P^i_{ind}	Frequency-independent power of task T_i
a_i	Effective switching capacitance of task T_i
$f_{ee}(t)$	Global energy-efficient frequency
σ_i	Load on $\mathcal{C}_i$
$\sigma(t)$	Maximum load among all active cores
σ^*_i	Effective load on $\mathcal{C}_i$
$\sigma^*(t)$	Maximum effective load among all active cores
ψ_i	Set of tasks allocated to $\mathcal{C}_i$

Table 6.1: Notations

6.3 Global Energy-Efficient Frequency

Existing DVS studies for uni-processor systems established that the frequency-independent dynamic power (P_{ind}) implies the existence of a energy-efficient frequency (also called critical frequency) threshold below which DVS is no longer effective from the system-level energy point of view [10,71,137]. This is because, with decreasing frequency, the gains in frequency-dependent dynamic energy can be offset by the excessive increase in frequency-independent dynamic energy after some point. Further, as different tasks may have different power characteristics, the energy-efficient frequency is task-dependent [10]. In [137], the value of the energy-efficient frequency for task T_i with effective switching capacitance a_i and frequency-independent power P^i_{ind} was given as $\sqrt[3]{\frac{P^i_{ind}}{2 \cdot a_i}}$.

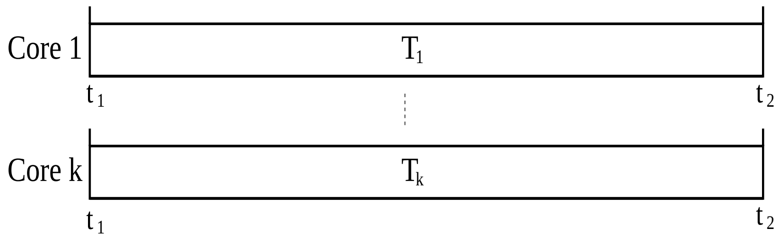


Figure 6.1: $T_1, \ldots, T_k$ running in parallel

However, in CMP platforms, with the unique voltage and frequency constraint, this concept needs to be re-visited. Consider $k \leq m$ *active* cores, where core $\mathcal{C}_i$ executes task T_i. Let the set of tasks $T(t) = \{T_1 \ldots T_k\}$ run in parallel from t_1 to t_2 as shown in Figure 6.1.

During this concurrent execution, T_i completes c_i cycles of its workload. If f denotes the global operating frequency in interval $[t_1, t_2]$, the total dynamic energy consumption in this interval is given by:

$$E' = \sum_{i=1}^{k}(a_i f^3 + P_{ind}^i) \cdot \frac{c_i}{f} = \sum_{i=1}^{k}(a_i f^2 + \frac{P_{ind}^i}{f}) \cdot c_i$$

It can be easily verified that E' is a strictly convex function of f. Thus, setting the first derivative of E' to zero gives the *global energy-efficient frequency* threshold for the k *active* cores at time t as:

$$f_{ee}(t) = \sqrt[3]{\frac{P_{ind}(t)}{2 \cdot a(t)}} \tag{6.2}$$

where $P_{ind}(t) = \sum_{i=1}^{k} P_{ind}^i$ and $a(t) = \sum_{i=1}^{k} a_i$.

Note that the global energy-efficient frequency level is potentially different from the energy-efficient frequency levels of tasks executing in parallel. In other words, global energy management may mandate the use of frequency levels that are below individual tasks' energy-efficient frequency thresholds.

Remark 3. *The global energy-efficient frequency threshold depends on the frequency-dependent active power and effective switching capacitance of the set of currently executing tasks on all* active *cores. Since the set of tasks executing in parallel changes with time, the global energy-efficient frequency threshold is time-dependent.*

Consequently, at the scheduling points that correspond to job completion, dispatch and preemption events, the global energy-efficient frequency should be re-computed. This operation will take at most $O(m)$ time at each scheduling point.

Remark 4. *The timing constraints of the task set may require using a frequency-level higher than $f_{ee}(t)$ at time t. The time-dependent global energy-efficient frequency level indicates a lower bound that should not be violated even if timing constraints allow.*

6.4 Components of Coordinated Power Management

Effective and coordinated power management of multiple processing cores to execute a given workload involves two main dimensions: statically making core activation and task-to-core allocation decisions, and dynamically managing the activated cores. Note that since a partitioned-based approach is assumed, the allocation of the periodic tasks to cores is done statically and run-time migration of tasks is not considered.

6.4.1 Energy-efficient Core Activation and Task Allocation

In general, the number of available processing cores (m) may be greater than the minimum number of cores upon which the given real-time workload can be scheduled in feasible manner. While the early studies that exclusively focused on dynamic power [4,12] suggested using all processing elements in parallel whenever possible, ever-increasing static power figures [26, 94] renders such an approach infeasible.

The power consumption of a given core can be minimized (in fact, effectively eliminated through techniques such as power gating in Intel i7 architecture [77]) when it is put to *off* state (Section 6.2). In *active, halt* and *sleep* states, the static power would be consumed continuously. This is because the periodic nature of the real-time application and significant time/energy overheads associated with transitions to/from *off* state make dynamically putting a core to *off* state at run-time an unrealistic option. As a result, instead of activating a core with light workload (with corresponding static energy consumption), it would be preferable to move that workload to other cores when possible. Obviously, a correlated and major issue is to perform task allocation on the selected cores to preserve feasibility and prepare favorable initial conditions for run-time management of dynamic energy.

Thus, the offline phase can be seen as an integrated component that decides on task-to-core allocations while keeping an eye on total (i.e. static+dynamic) potential energy consumption. The $k \leq m$ cores selected by this phase will be activated and then will be managed by the run-time component. The remaining $(m-k)$ cores are put to *off* state with negligible power consumption.

6.4.2 Run-time Power Management of Active Cores

The run-time management of the selected $k \leq m$ cores involves the use of Global Voltage Scaling as well as selectively putting some cores to *halt* and *sleep* states (Section 6.2) to reduce dynamic energy. To start with, the global frequency level that determines the dynamic power consumption at time t is decided by the highest performance level required by any core in *active* state at time t (Equation (6.1)). This requires both closely monitoring the workload conditions on all cores and exploiting the available idle states whenever possible. As an example, if the core that requires highest performance level (e.g. highest frequency level for the feasibility of its workload) is put to *halt* or *sleep* state temporarily, the frequency can be reduced to the next highest performance level required by any of the remaining cores during that interval. In addition, putting any core to *halt* and in particular *sleep* states have the potential of reducing dynamic energy consumption for all the cores through reducing the global energy-efficient frequency as well (Section 6.3).

Sections 6.5 and 6.6 give a detailed discussion of these two fundamental dimensions. Since solving to the problem of energy-efficient core activation depends on some important dynamic energy consumption approximation formulas that are driven by the results of Section 6.5, first that component is considered.

6.5 Run-time Coordinated Power Management

This section assumes that $k \leq m$ cores are selected for the execution of the periodic workload and that task-to-core allocations are already performed by the static phase.

6.5.1 Exploiting Core Idle States at Run-time

In general, any of the k cores can be occasionally put to *halt* and *sleep* states when they have no ready task to execute, with corresponding gains in dynamic energy at the related core. While transitioning to *sleep* state provides higher dynamic energy savings, more significant time and energy overheads associated with that transition requires a more careful evaluation (Section 6.2). In fact, there exists a minimum length of idle interval, denoted by $\mathcal{I}_{thres}$, that justifies transitioning a core to *sleep* state [73]. Thus, an idle core can be put to *sleep* state in energy-efficient manner if and only if its predicted length of idle interval is no less than $\mathcal{I}_{thres}$.

To preserve the feasibility of the workload, a simple scheme is provided to compute the predicted length of the idle interval. The earliest time in future an idle core will have to execute a task is constrained by the earliest next release time among all jobs allocated to that core. This value provides a safe lower bound on the minimum length of idle interval and hence can be used for making safe core state transitioning decisions. Let nrt_j denote the earliest time in future a job of task T_j may be released. Then, at time t, the minimum length of idle interval for an idle core $\mathcal{C}_i$ is given as:

$$\delta_i(t) = min(nrt_j) - t, \quad j|T_j \in \psi_i$$

where ψ_i denotes the set of tasks allocated to $\mathcal{C}_i$. An idle core $\mathcal{C}_i$ will be transitioned to *sleep* state if and only if $\delta_i(t) \geq \mathcal{I}_{thres}$. Following this, a timer is set to appropriately start transitioning $\mathcal{C}_i$ back such that it will be *active* and ready to execute jobs at time $t + \delta_i(t)$ which marks the end of its idle interval. On the other hand, if $\delta_i(t) < \mathcal{I}_{thres}$ then $\mathcal{C}_i$ is simply put to *halt* state, which involves negligible transition overheads [139]. Finally, note that the run-time overhead of making this decision is constrained by the complexity of computing $\delta_i(t)$. On each core, one can always update the information about the next earliest job release time in the future ($min(nrt_j)$) at job release times in $O(1)$ time. Thus, core state transition decisions can be done in constant-time.

Remark 5. *Core state transitions to* halt *or* sleep *states not only help reduce power at the core-level but may potentially provide additional savings for the entire system, since the global energy-efficient frequency may be effectively reduced.*

6.5.2 Coordinated Voltage and Frequency Scaling (CVFS) Algorithm

Recall from Section 6.3 that running all the *active* cores at $f_{ee}(t)$ at all times minimizes the dynamic energy. However, obviously, this does not necessarily guarantee the feasibility of the workload. Since $f_{ee}(t)$ is time-dependent, computing the optimal feasible frequency $f(t) \geq f_{ee}(t)$ to minimize energy in the long-run poses great challenges. Hence, a more direct but efficient approach is taken.

The feasibility on each *active* core $\mathcal{C}_i$ is guaranteed as long as its operational frequency is no smaller than is total load [11, 101], i.e. $f(t) \geq \sigma_i$ preserves the feasibility on core $\mathcal{C}_i$. Let $\sigma(t)$ denote the largest load value among all *active* cores, i.e.

$$\sigma(t) = max(\sigma_i), \quad i|\mathcal{C}_i \textit{ is active}$$

CVFS consists in setting $f(t) = max(\sigma(t), f_{ee}(t))$ to preserve the feasibility of all *active* cores without violating the energy-efficient frequency constraint. Recall that the scheduling points and core state transitions that can potentially change the set of simultaneously executing tasks may have an impact on the global energy-efficient frequency threshold $f_{ee}(t)$. Thus, $f(t)$ needs to be re-computed at these important events. Note that the new value of

$f(t)$ can be evaluated in time $O(m)$ at each scheduling point.

6.5.3 *CVFS**: Adapting to Dynamic Load Conditions

CVFS is based on using the static load values of active cores at run-time. The load $\sigma_i = \frac{C_i}{P_i}$ corresponds to the worst-case utilization of the task set ψ_i on the core $\mathcal{C}_i$. While this is a safe approach, there are potential benefits of computing the instantaneous load σ_i^*, which may differ from σ_i for two reasons:

- Some jobs may not take their worst case cycles and complete early. Due to this unused CPU time, in some intervals, the instantaneous load may be less than σ_i.
- Due to the constraints imposed by $f_{ee}(t)$ and global voltage/frequency, a given core may be forced to execute at frequency levels higher than what is necessary to preserve its own feasibility. Hence, its remaining workload may be lower than σ_i in some intervals.

The algorithm *CVFS** is based on maintaining a reasonably accurate estimation of the core-level instantaneous loads σ_i^* and reducing the frequency below what is suggested by *CVFS* when the conditions allow.

Exploiting task early completions. In this direction, the well-known *cycle conserving EDF (cc-EDF)* algorithm (which is originally proposed for uni-processor systems [101]) is extended to multicore environments with global energy-efficient frequency awareness.

Specifically, for each task T_j on core $\mathcal{C}_i$, $u_j(t)$ is defined as its effective load at time t. Let wcc_j denote the worst-case number of CPU cycles required by task T_j at the maximum CPU frequency f_{max}. Formally, $wcc_j = C_j \cdot f_{max}$. Then, the rules to update u_j on core $\mathcal{C}_i$ are given as [101]:

- When a job of T_j is released, u_j is reset to $\frac{C_j}{P_j}$.
- When a job of task T_j released at time r completes after executing $acc_j \leq wcc_j$ cycles, it has effectively consumed acc_j CPU cycles in the interval $[r, r + P_j]$. Thus, the

effective load of T_j over this interval is $\frac{acc_j}{f_{max} \cdot P_j}$ and not necessarily its worst-case workload $\frac{C_j}{P_j}$. Hence, when a job of T_j completes, u_j is set to $\frac{acc_j}{f_{max} \cdot P_j}$.

Given this, the instantaneous effective load σ_i^* on $\mathcal{C}_i$ is defined as $\sigma_i^* = \sum\limits_{T_j \in \psi_i} u_j$. Also, let $\sigma^*(t)$ be the maximum effective load at time t among all *active* cores, i.e.

$$\sigma^*(t) = max(\sigma_i^*), \quad i|\mathcal{C}_i \ is \ active$$

σ_i^* is updated on $\mathcal{C}_i$ at events corresponding to job completions and job arrivals. Now consider the frequency assignment where at time t, all *active* cores are executed at frequency $f(t)$ given by:

$$f(t) = max(\sigma^*(t), f_{ee}(t)) \tag{6.3}$$

Proposition 7. *At any time t, executing core $\mathcal{C}_i$ at $f(t) = \sigma^*(t)$ preserves the feasibility of task set ψ_i.*

Proof. The feasibility of task set ψ_i is preserved as long as the operating frequency $f(t)$ on core $\mathcal{C}_i$ at time t, satisfies the constraint $f(t) \geq \sigma_i^*$. This follows from the correctness of *cc-EDF* [101]. Since $\sigma^*(t) \geq \sigma_i^*$, executing core $\mathcal{C}_i$ at $f(t) \geq \sigma^*(t) \geq \sigma_i^*$ preserves feasibility of task set ψ_i. □

Corollary 4. *At any time t, executing all* active *cores at $f(t) = max(\sigma^*(t), f_{ee}(t))$ preserves the overall feasibility.*

Refining the load estimation. Since all *active* cores are constrained to run at the same global frequency, typically many cores will operate at a processing frequency higher than the level necessary to guarantee the timing constraints of their remaining workload. This fact can be exploited to further refine the estimate of σ_i^*, providing additional energy savings. The basic principle is given below:

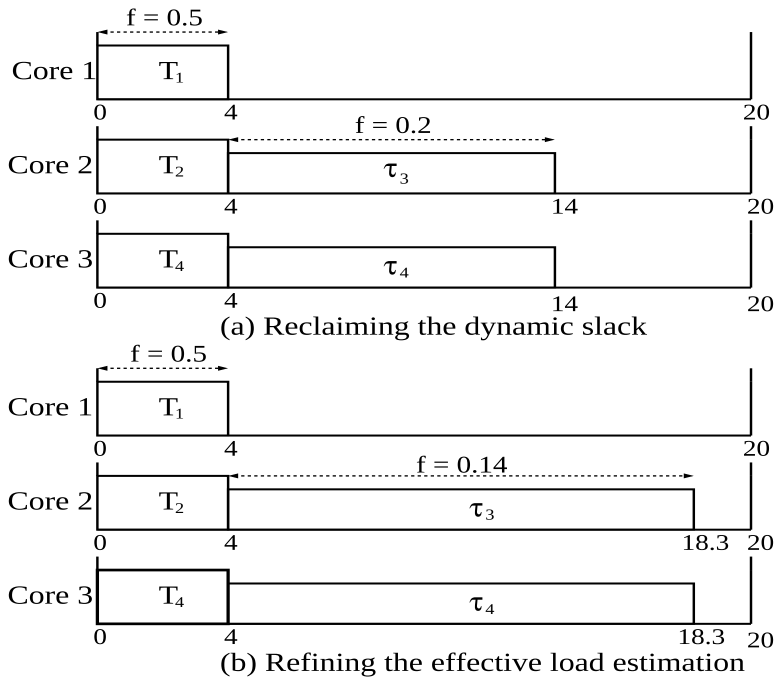


Figure 6.2: An example with four tasks and three cores

On core $\mathcal{C}_i$, the execution at a frequency higher than σ_i may be seen as equivalent to executing a smaller workload at speed σ_i. In other words, speeding up a task on core $\mathcal{C}_i$ can potentially be considered as an event of early task completion, even if tasks exhibit their worst-case workload.

The following example illustrates this point. Figure 6.2 shows a CMP with three cores. $\mathcal{C}_1$ has one task T_1 $(C_1 = 10, P_1 = 20)$. $\mathcal{C}_2$ has two tasks T_2 $(C_2 = 2, P_2 = 20)$ and T_3 $(C_3 = 2, P_3 = 20)$. $\mathcal{C}_3$ has one task T_4 $(C_4 = 4, P_4 = 40)$. Thus, the initial effective loads on the cores are given as $\sigma_1^* = 0.5$, $\sigma_2^* = 0.2$ and $\sigma_3^* = 0.1$. It is assumed that $f_{max} = 1.0\ GHz$. For simplicity, assume $a = 1$ and $P_{ind} = 0$ for all tasks. Also, the actual execution of T_1 on $\mathcal{C}_1$ at f_{max} is limited to 2 units while all other tasks exhibit their worst-case behavior.

At time $t = 0, T_1, T_2$ and T_4 are dispatched on $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ respectively at $f = 0.5\ GHz$. At time $t = 4$, T_1 and T_2 complete. Observe that at this point, the slack reclaiming rules that are previously provided would make no change to the effective load of $\mathcal{C}_2$ as T_2 took its worst-case workload. Thus, if one estimated the effective load on a core using the previous rules, then at $t = 4$, T_3 and T_4 would be dispatched at $f = 0.2\ GHz$ as shown in Figure 6.2(a).

However, observe that in the interval $[0, 4]$, T_2 executed at $f = 0.5Ghz$, which is higher than 0.2 GHz which is sufficient to maintain the feasibility of the workload on $\mathcal{C}_2$. Since at $f = 0.2\ GHz$, T_2 would have executed only 0.8×10^9 cycles in the interval $[0, 4]$, one can potentially see the completion of T_2 as an early completion at $f = 0.2\ GHz$. Hence, at $t = 4$, σ_2^* can be set to $\frac{0.8}{20} + \frac{2}{20} = 0.14$. Thus, at $t = 4$, both T_3 and T_4 would be dispatched at $f = 0.14\ GHz$ as shown in Figure 6.2(b), increasing the energy savings.

The following describes how to update σ_i^* at run-time according to these principles. Without loss of generality, assume a job of T_j executes on core $\mathcal{C}_i$ in p contiguous execution intervals, denoted by $\{e_1 \ldots e_p\}$. During each contiguous execution e_k, let T_j consume at_k units of CPU time at frequency f_k. For each e_k, one can compute the workload (ac_k) T_j would have completed at frequency σ_i in at_k time units, by setting $ac_k = at_k \cdot \sigma_i$. The cumulative workload completed by T_j corresponding to the contiguous execution sequence $\{e_1 \ldots e_p\}$ is given by:

$$c_j = \sum_{k=1}^{p} ac_k = \sum_{k=1}^{p} (at_k \cdot \sigma_i)$$

The operating system can keep track of and update c_j for each task T_j appropriately at task preemption and completion points. Thus, the rules to update u_j can be re-defined (refined) as:

- When a job of T_j is released, set $u_j = \frac{C_j}{P_j}$.
- When a job of task T_j completes, set $u_j = \frac{c_j}{f_{max} \cdot P_j}$.

Figure 6.3 shows the pseudo-code for *CVFS**. The function *AdjustFrequency()* recomputes the global energy-efficient frequency threshold based on Equation (6.2) in Section 6.3

```
Function AdjustFrequency():
1  Set $p = 0$
2  Set $a = 0$
3  for each active core $\mathcal{C}_i$ on which $T_j$ currently runs
4      Set $p = p + P_{ind}^j$
5      Set $a = a + a_j$
6      Set $\sigma_i^* = \sum_{T_k \in \psi_i} u_k$
7  Set $\sigma^*(t) = max(\sigma_i^*),\ i|\mathcal{C}_i\ is\ active$
8  Set $f_{ee}(t) = \sqrt[3]{\frac{p}{2 \cdot a}}$
9  Set $f(t) = max(\sigma^*(t), f_{ee}(t))$

At job arrival of $T_j$ on core $\mathcal{C}_i$:
1  Set $u_j = \frac{C_j}{P_j}$
2  AdjustFrequency()

At state transition of core $\mathcal{C}_i$ to active state:
1  AdjustFrequency()

At job completion of $T_j$ on core $\mathcal{C}_i$:
1  Set $u_j = \frac{c_j}{f_{max} \cdot P_j}$
2  if ready queue is empty
3      Set $\delta_i(t) = min(nrt_k) - t,\ k|T_k \in \psi_i$
4      if $\delta_i(t) \geq \mathcal{I}_{thres}$
5          Transition $\mathcal{C}_i$ to sleep state
6          Set timer to transition $\mathcal{C}_i$ back
7      else
8          Transition $\mathcal{C}_i$ to halt state
9  AdjustFrequency()
```

Figure 6.3: The pseudo-code of *CVFS**

and the maximum effective load $\sigma^*(t)$ among all *active* cores. The new global frequency $f(t)$ is then easily calculated by taking the maximum.

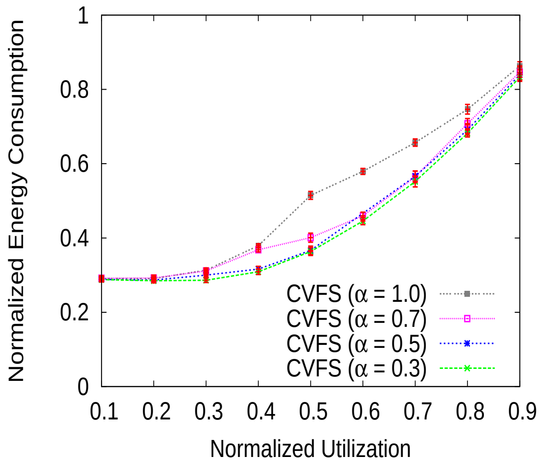
An event corresponding to either job arrival or completion may change u_j, which in turn may trigger changes in the effective load of $\mathcal{C}_i$ and hence $f(t)$. Also, as mentioned before, events corresponding to job completions, job preemptions and core state transitions have the potential to change $f_{ee}(t)$ and hence $f(t)$. Thus, at these events the *AdjustFrequency()* function is called.

Since core-level power state transitioning decisions can be made in $O(1)$ time (Section 6.5.1), the complexity of *CVFS** is determined by the complexity of *AdjustFrequency()* function. Observe that the value of σ_i^* on each core $\mathcal{C}_i$ can be updated at job completion and job arrival events and kept track of in constant time. Thus, when *AdjustFrequency()* is called σ^* and $f_{ee}(t)$ can be re-computed in $O(m)$ time. Hence, the overall run-time complexity of *CVFS** is $O(m)$ at each scheduling point.

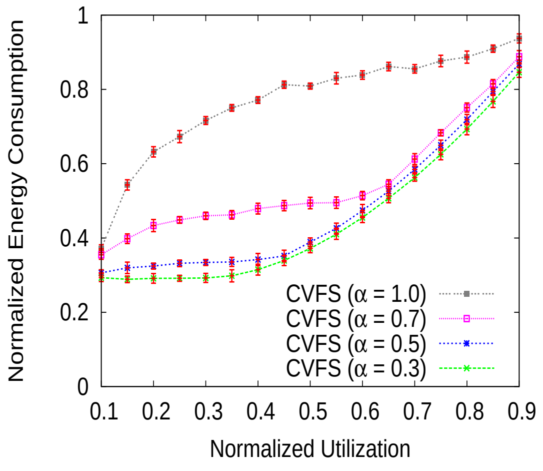
6.5.4 Experimental Evaluation

This section evaluates the performance of the algorithms with the help of a discrete-event simulator. For 2- and 8-core systems, synthetic task sets each with 20 and 50 tasks, respectively are generated. P_{ind}^i values were randomly chosen in the range $[0, 0.2]$. The power consumption at f_{max} on each core is assumed to be $1W$. Power consumption of cores is sleep state is assumed to be $5mW$. The break-even time for core transitions to sleep state is taken as $1ms$ and the energy overhead associated with such transitions is assumed to be $0.5mJ$. Task periods were generated randomly in the interval $[25ms, 1300ms]$. Task utilizations are randomly generated using uniform distribution and task worst-case execution times are computed as the product of task utilizations and task periods.

The reported energy consumption values are normalized with respect to the base scheme which executes all tasks at f_{max} at all times (no power management). Previous studies on energy minimization on multi-processor systems [4,110] showed that the maximum allowable utilization of a task (denoted as α) is an important parameter for performance. As a



(a) 2 cores



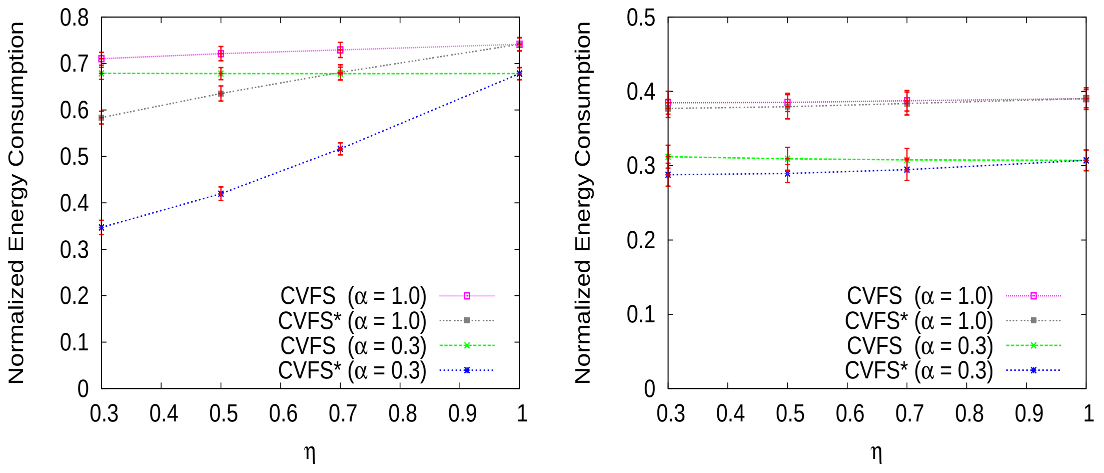
(b) 8 cores

Figure 6.4: Impact of Utilization

result, the impact of this task utilization factor α is also investigated. In the experiments, normalized utilization refers to the quantity $\frac{U_{tot}}{m}$, where m is the number of cores on which the workload is executed. For each normalized utilization and α pair, 1000 task sets are generated; the data points in the plots reflect the average of these runs. Also, the confidence intervals at 99% confidence level are reported. The results show only dynamic energy consumption; the static power that is continuously consumed by all active cores is not included.

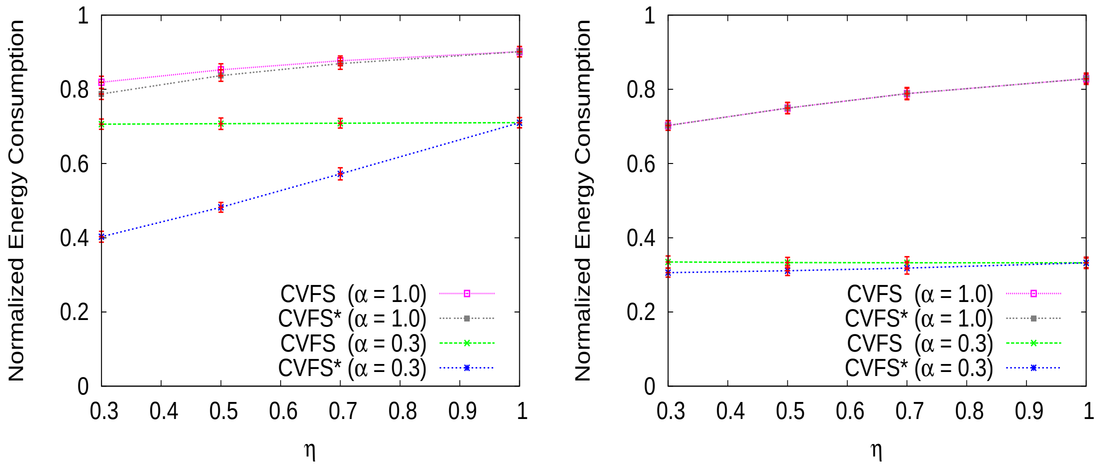
First, the behavior of *CVFS* over the normalized utilization spectrum. In these experiments, all tasks complete their worst-case workload. Task allocation to m cores is done using Worst-Fit-Decreasing (WFD) heuristic which is known to generate better-balanced partitions [4, 12].

Figures 6.4(a) and (b) show the impact of normalized system utilization on CMP with $m = 2$ cores and $m = 8$ cores for various α values, respectively. It can be seen that the *CVFS* scheme provides significant overall system energy savings. With increasing normalized utilization values, the gains of *CVFS* decrease as high frequency is often needed to meet



(a) normalized utilization 0.8 (b) normalized utilization 0.4

Figure 6.5: Impact of workload variability (2 cores)



(a) normalized utilization 0.8 (b) normalized utilization 0.4

Figure 6.6: Impact of workload variability (8 cores)

the feasibility constraints. With increasing α values, the partitions created by WFD have a higher σ (maximum load among all cores) value. Since σ is one of the factors constraining $f(t)$ (Section 6.5.2), with increasing α values the relative gains of *CVFS* tend to decrease.

Figures 6.5 and 6.6 show the impact of workload variability on the schemes. η is defined as the ratio of average-case execution cycles (acc) to worst-case execution cycles(wcc) and use it to model the notion of dynamic workload variability. The lower the η ratio, the more the actual workload deviates from the worst case workload. For a specific value of η, the actual execution cycles are generated randomly using normal distribution with mean acc and standard deviation $\frac{wcc-acc}{6}$.

Figures 6.5(a) and 6.6(a) show the impact of varying η for 2-core and 8-core systems respectively, with normalized utilization fixed to a high value (0.8). With decreasing η, the gains of *CVFS** over *CVFS* are prominent, in particular for the case where $\alpha = 0.3$. This is due to the run-time effective load adjustments of *CVFS** which provides additional DVS opportunities and hence better energy savings compared to *CVFS*. For the same utilization value, higher α values tend to create more unbalanced partitions relative to lower α values. In CMP systems where all cores are constrained to operate at the same frequency, this limits the opportunities to exploit dynamic workload variability. As such, the gains of *CVFS** over *CVFS* decreases with increasing α values.

Figures 6.5(b) and 6.6(b) show the impact of varying η for 2-core and 8-core systems respectively, with normalized utilization fixed to a low value (0.4). In this case, irrespective of α values, the gains provided by *CVFS** over *CVFS* are rather small. Recall that $f(t)$ is constrained by σ and the *global energy-efficient frequency threshold* $f_{ee}(t)$ (Section 6.5.2). At low normalized utilization values, $f(t)$ is predominantly constrained by $f_{ee}(t)$, hence the run-time adaptations of *CVFS** do not provide significant benefits compared to *CVFS*.

6.6 Energy-Efficient Core Activation and Task Allocation

This section elaborates on the important dimension of the problem introduced in Section 6.4.1, namely selecting $k \leq m$ number of cores for the execution of the workload to minimize expected energy while preserving the feasibility through a proper task allocation on these k cores. Clearly, since determining feasibility of a workload on a fixed number of processors is NP-Hard in the strong sense, one cannot hope for an efficient and optimal solution to this problem.

Recall from Section 6.5.2 that at any given time the unique frequency for all *active* cores is given as $f(t) = max(\sigma, f_{ee}(t))$. Hence, starting with initial task allocations (partitions) that are reasonably balanced and adjusting these to maintain a balance between static and dynamic power consumptions is a promising approach. In fact, minimizing the maximum load among cores is also in line with existing multiprocessor and multicore energy management results [4,12,110]. Among task allocation heuristics, the Worst-Fit Decreasing (WFD) algorithm is known to typically yield well-balanced partitions where the maximum load on any core is small [4,12]. Assuming that the tasks are already sorted in non-increasing order according to their utilization values, WFD allocates tasks one by one to the core with the least load at a time. For this specific problem, WFD is equivalent to the well-known List Scheduling Algorithm (LST) where independent tasks each with a given size in the range $[0, 1]$ are partitioned to m CPUs each with unit capacity. The result in [58] implies that the maximum load among all cores generated by LST (and equivalently WFD in the settings under investigation) is no more than $\frac{4}{3}$ times that of the optimal. As a result, the first step of the proposed framework will consist in generating an initial partition on all m cores through WFD, before transforming this initial schedule into a final and a more energy-efficient partition with possibly a smaller number of active cores.

Having an efficient mechanism to evaluate the expected energy consumption of a given partition in static phase is an important component of the approach. Let $\mathcal{P}_k$ be a feasible partitioning of task set ψ to $k \leq m$ cores. Since only k cores have tasks allocated to them, the remaining $(m - k)$ cores can be put to *off* state. Thus, the static energy consumption of partition $\mathcal{P}_k$ is given as:

$$E_s(\mathcal{P}_k) = k \cdot P_{static} \cdot H$$

Since the global unique frequency at time t, $f(t)$, is time-dependent and further depends on the set of tasks executing in parallel at any given time, it is very difficult, if not impossible, to have an accurate figure for the dynamic energy consumption of the task set ψ, in advance. The dynamic energy consumption of $\mathcal{P}_k$ is estimated by calculating the weighted average value of $f(t)$ in the interval $[0, H]$, where H is the hyperperiod. Let F_{ee} denote the weighted average of all $f_{ee}(t)$ values in the interval $[0, H]$. F_{ee} is approximated as:

$$F_{ee} = \sqrt[3]{\frac{P^*_{ind}}{2 \cdot a^*}}$$

where, $P^*_{ind} = \sum_{i=1}^{n}(U_i \cdot P^i_{ind})$ and $a^* = \sum_{i=1}^{n}(U_i \cdot a_i)$. Recall from Equation (6.2) that the global energy-efficient frequency at any given time is determined by the ratio of P^i_{ind} and a_i values of tasks. Hence, it is natural to expect that tasks with large utilization values will have a higher contribution to F_{ee} on the average. Given this, the weighted average of all $f(t)$ values in the interval $[0, H]$ can be approximated as:

$$F = max(\sigma, F_{ee})$$

The expected dynamic energy consumption of task set ψ over partition $\mathcal{P}_k$ is then calculated as:

$$E_d(\mathcal{P}_k) = \sum_{i=1}^{n}(a_i F^3 + P^i_{ind}) \cdot \frac{U_i}{F} \cdot H$$

Notice that different partitions may produce different F values and thus have different expected dynamic energy consumptions. The total expected energy consumption of $\mathcal{P}_k$ is:

$$E_{exp}(\mathcal{P}_k) = E_s(\mathcal{P}_k) + E_d(\mathcal{P}_k) \tag{6.4}$$

Next, three schemes are presented for determining the number of active cores.

Sequential-Search (SS) Algorithm. The minimum number of cores necessary to execute a workload with total utilization U_{tot} in feasible manner is $\lceil U_{tot} \rceil$. *SS* exhaustively considers every possible k in the range $[\lceil U_{tot} \rceil, m]$ and for each such k it generates a partition $\mathcal{P}_k$ using WFD. If $\mathcal{P}_k$ is a feasible partition then the algorithm computes the expected energy consumption of $\mathcal{P}_k$ using Equation (6.4). The k value corresponding to the partition with the least E_{exp} is returned. Figure 6.7 gives the pseudo-code.

```
1 for each $k$ in the range $[\lceil U_{tot} \rceil, m]$ do
2     Determine partition $\mathcal{P}_k$ using WFD
3     if $\mathcal{P}_k$ feasible
4         Compute $E_{exp}(\mathcal{P}_k)$
5 Select $k$ corresponding to partition with $\min(E_{exp})$
```

Figure 6.7: Algorithm *SS*

Complexity Issues: *SS* has at most m iterations. In each iteration the algorithm has to execute worst-fit decreasing (which takes $O(n \log m)$ time) and calculate E_{exp} from Equation (6.4) (which takes $O(n)$ time). Thus, the overall complexity of *SS* is $O(nm \log m)$.

Greedy Load Balancing (GLB) Algorithm. *GLB* invokes WFD only once on all m cores. Working on the resulting partitioning, *GLB* tries to free the least loaded core, by simply moving all tasks on the least loaded core to the second least loaded core, if and only if doing so preserves the feasibility of the workload and does not increase the expected energy consumption, computed through (6.4). The algorithm is re-invoked iteratively for the remaining cores until such a block move of tasks is no longer possible.

Before giving the formal pseudo-code the following two variables are introduced: $\mathcal{P}_k(\psi_i)$ representing the set of tasks allocated to core $\mathcal{C}_i$ in partition $\mathcal{P}_k$ and $\mathcal{P}_k(\sigma_i)$ denoting the load on $\mathcal{C}_i$ in $\mathcal{P}_k$. Figure 6.8 gives the pseudo-code.

Complexity Issues: *GLB* invokes WFD once on all m cores (which takes $O(n \log m)$ time). Following this, *GLB* has at most m iterations (Lines 3-11) where calculating E_{exp} takes $O(n)$ time and re-arranging the position of the second least loaded core, after moving the workload from the least loaded core to it, can be done in $O(\log m)$ time. Thus, the

```
1   $\mathcal{P}_m$ = Partition obtained through WFD on $m$ cores
2   $k = min(m, n)$
3   while $(k > 1)$
4       $src$ =index of the core with minimum load
5       $des$ =index of the core with second minimum load
6       if $(\sigma_{src} + \sigma_{des} > 1)$ return $\mathcal{P}_k$
7       $\mathcal{P}_{k-1} = \mathcal{P}_k - (\mathcal{P}_k(\psi_{src}), \mathcal{P}_k(\sigma_{src}))$
8       Set $\mathcal{P}_{k-1}(\psi_{des}) = \mathcal{P}_{k-1}(\psi_{des}) \cup \mathcal{P}_k(\psi_{src})$
9       Set $\mathcal{P}_{k-1}(\sigma_{des}) = \mathcal{P}_{k-1}(\sigma_{des}) + \mathcal{P}_k(\sigma_{src})$
10      if $(E_{exp}(\mathcal{P}_{k-1}) \geq E_{exp}(\mathcal{P}_k))$ return $\mathcal{P}_k$
11      Set $k = k - 1$
12  return $\mathcal{P}_k$
```

Figure 6.8: Algorithm *GLB*

overall complexity of *GLB* is $O(n \log m + m \log m + mn) = O(mn)$.

Threshold-based Load Balancing (TLB) Algorithm. *TLB* is similar to *GLB* but does not use the expected energy formula given in Equation (6.4), to improve efficiency. Instead, *TLB* uses the concept of *load threshold*, wherein a partition is accepted by *TLB* as long as the minimum load on any core is no smaller than a pre-defined threshold value. This threshold value should be carefully chosen by the system designer to reflect an appropriate balance between static and active power consumptions. Similar to *GLB*, *TLB* first invokes WFD once on all m cores and then iteratively tries to free the least loaded core, by simply moving all tasks on it to the second least loaded core, if and only if the minimum load is smaller than the pre-defined threshold and doing so preserves the feasibility of the workload. After such a move, the algorithm is iteratively re-invoked on the new set of active cores. Figure 6.9 gives the pseudo-code.

Complexity Issues: Assuming $n \geq m$, WFD takes $O(n \log m)$ time. Following this there are at most m iterations (Lines 3-11) and in each of these iterations re-arranging the position of the second least loaded core takes $O(\log m)$ time making the total complexity

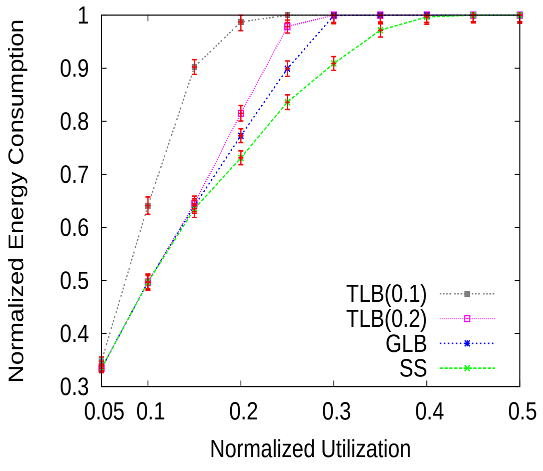
```
1   $\mathcal{P}_m$ = Partition obtained through WFD on $m$ cores
2   $k = min(m, n)$
3   while $(k > 1)$
4       $src$ =index of the core with minimum load
5       $des$ =index of the core with second minimum load
6       if($\sigma_{src} > threshold$ or $\sigma_{src} + \sigma_{des} > 1$)
7           return $\mathcal{P}_k$
8       $\mathcal{P}_{k-1} = \mathcal{P}_k - (\mathcal{P}_k(\psi_{src}), \mathcal{P}_k(\sigma_{src}))$
9       Set $\mathcal{P}_{k-1}(\psi_{des}) = \mathcal{P}_{k-1}(\psi_{des}) \cup \mathcal{P}_k(\psi_{src})$
10      Set $\mathcal{P}_{k-1}(\sigma_{des}) = \mathcal{P}_{k-1}(\sigma_{des}) + \mathcal{P}_k(\sigma_{src})$
11      Set $k = k - 1$
12  return $\mathcal{P}_k$
```

Figure 6.9: Algorithm *TLB*

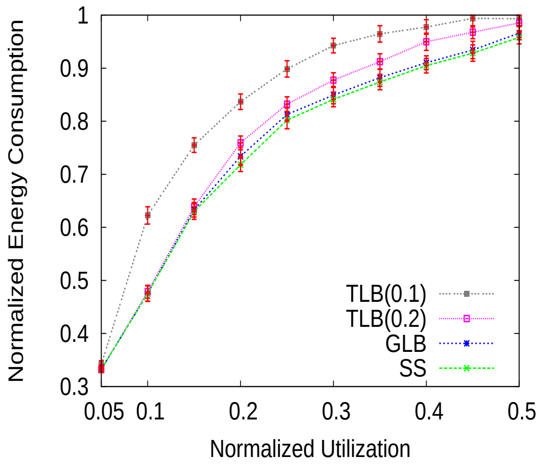
$O(n \log m)$. On the other hand, if $n < m$, WFD takes $O(n)$ time, there are only n iterations (Lines 3-11) each taking $O(\log n)$ time which gives a complexity of $O(n \log n)$. Thus, the overall complexity can be expressed as $O(n \log(min(m, n)))$.

6.6.1 Experimental Evaluation

In this section, the performance of algorithms *SS*, *GLB* and *TLB* are compared. The simulation methodology is parallel to the one described in Section 6.5.4. The static power consumption of cores is taken as $0.1W$. Results for 4 and 8 cores are shown in Figures 6.10 and 6.11 respectively, the trends of which are similar. Results are shown with $\alpha = 0.3$ and $\alpha = 1.0$. For the *TLB* scheme, results are shown with threshold values 0.1 and 0.2 that performed best in the experiments. For each core, P_{static} was set to 10% of CPU dynamic energy consumption at f_{max} [73]. In these experiments, the worst-case workload for each task is considered and once a partitioning of tasks to $k \leq m$ cores is decided by the algorithms, the task set is executed with *CVFS* scheme and the energy consumption over the hyperperiod is recorded. All energy consumption values are normalized with respect to

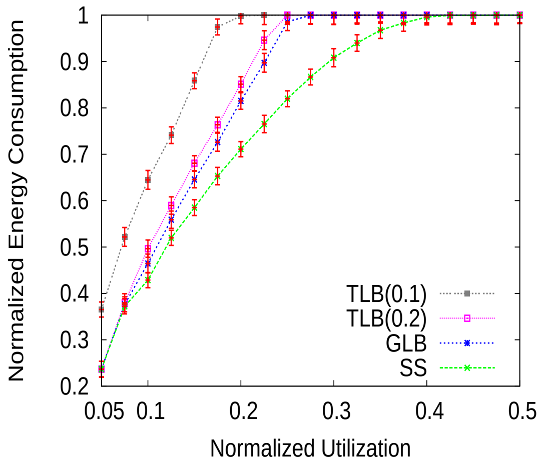


(a) $\alpha = 0.3$

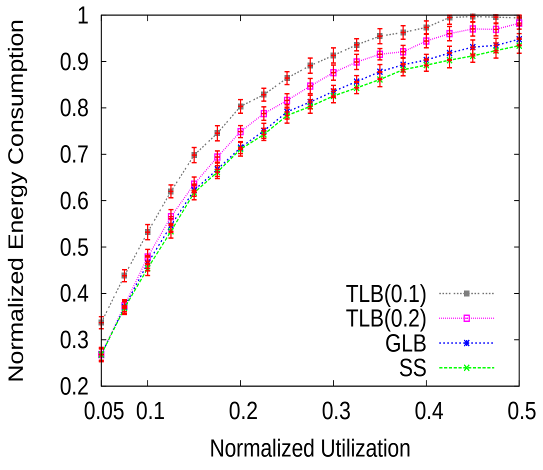


(b) $\alpha = 1.0$

Figure 6.10: Comparing *SS*, *GLB* and *TLB* (4 cores)



(a) $\alpha = 0.3$



(b) $\alpha = 1.0$

Figure 6.11: Comparing *SS*, *GLB* and *TLB* (8 cores)

the scheme which uses all m available cores to execute the workload.

In decreasing order of performance with respect to energy savings, the algorithms can be arranged as: *SS*, *GLB*, *TLB* (with threshold 0.2) and *TLB* (with threshold 0.1). However, the better the system energy savings of a scheme, the more its computational complexity. For the *TLB* scheme, the energy benefits are sensitive to the threshold value. In the simulations it was observed that a threshold of 0.2 provides energy benefits that are comparable to that provided by *SS*, which has high execution overhead.

At low utilization values the gains provided by the schemes are significant (easily exceeding 50%). With increasing utilization values, the dynamic energy consumption of the workload dominates static energy and hence the number of cores activated to execute the workload in energy-efficient and feasible manner approaches m. Thus, as utilization increases the benefits of schemes decrease. In fact, when the normalized utilization $\frac{U_{tot}}{m}$ exceeds 0.5, all the schemes are forced to activate all m cores to enforce feasibility or avoid excessive dynamic power that can result from using less number of cores at high frequencies.

It can also be seen that the benefits of the schemes decrease much quickly at lower α values compared to higher α values (Figure 6.10(b) and 6.11(b)). This is because, with large α values, WFD tends to generate more unbalanced initial partitions [12]. This results in more chances for finding cores with light workloads in the WFD partition; the workloads on these cores can then be transferred to the ones with high load, enabling them to switch to *off* state.

Finally, one can also see that at low normalized utilization values, the benefits of schemes are more pronounced in the case with 8 cores. This is due to the fact that with higher number of cores the opportunities to minimize static energy by turning off cores also increase, in particular for low utilization values.

6.7 Chapter Summary

This chapter considered the problem of system energy minimization of periodic real time tasks executing on CMP platforms with partitioning and global DVS capability. Considering a generalized power model, an expression for the *global energy-efficient frequency* that depends on the set of tasks executing in parallel, was derived. Two schemes *CVFS* and

*CVFS** were proposed to exploit global DVS and core-level idle states. *CVFS** has the additional capability of adapting to workload variations at run-time. Also, the problem of determining the optimal subset of cores to execute the workload with low static power while preserving feasibility through an appropriate task allocation was investigated and three techniques were suggested for this purpose. Experimental evaluations verified the effectiveness of the proposed solutions to reduce the system energy on CMP platforms.

Chapter 7: Competitive Analysis of Energy-Constrained Real-Time Scheduling

7.1 Introduction

This chapter investigates the impact of energy constraint on the competitiveness of on-line real-time scheduling in settings where there is sufficient time but insufficient energy to complete all the workload. Upper bounds on the competitive factor achievable by on-line/semi-online algorithms are derived. In addition, on-line/semi-online algorithms are designed and their competitive factors are obtained. In this investigation, various fundamental models are considered, including uniform value density, non-uniform value density, constrained execution environments such as non-preemptive/non-idling settings and platforms with DVS capability. Also, the competitiveness of energy-constrained online real-time scheduling is investigated under the resource augmentation framework.

7.2 Terminology and Assumptions

This chapter makes the *underloaded energy-constrained system assumption* where the input instance ψ is feasible in real-time sense; that is, there must exist a feasible schedule where all the jobs in ψ meet their deadlines if the energy constraint is not taken into account.

To address the problem, first non-DVS settings are considered (DVS-enabled systems will be considered in Section 7.6) consisting of a uni-processor system having limited energy budget of E units. The system's total energy consumption during its operation cannot exceed this allowance. It is assumed that executing a job consumes one unit of energy per time unit. The system's energy consumption in idle state (i.e. when it is not executing any job) is negligible.

Real-time jobs enter and leave the system during its operation. ψ denotes the finite input sequence of jobs that arrive to the system during its operation. Preemptive scheduling is assumed. Each job J_i is represented by the tuple (r_i, e_i, d_i). Here r_i and d_i are the release time and deadline of the job J_i, respectively. For the sake of clarity, e_i is used to denote the size of the job, indicating both its execution time and *energy* requirements (since execution per unit time requires unit energy). The laxity of the job J_i is given by $d_i - (r_i + e_i)$. Since ψ is finite and one unit of execution requires one unit of energy, similar to [5],[6] $E_b = \sum\limits_{J_i \in \psi} e_i$ is defined to be the minimum energy needed to complete all the jobs in ψ.

Each job is associated with a *value* that is proportional to its execution time. *Value-based real-time systems* where a job that successfully completes execution by its deadline contributes its value to the system are considered; no value is obtained from the executions of jobs that miss their deadlines [17],[75]. The analysis starts with the *uniform value density* model [17],[18],[20],[75], where the job's value is equal to its size (e_i). The analysis is also extended (in Section 7.5) to the more general non-uniform value density model. Note that the off-line problem of maximizing the total system value can be easily shown to be NP-Hard even for the simple case where all jobs have the same deadline, by slightly modifying the intractability proof given in [108].

e_{max} is defined as the upper bound on the size of any job that can be introduced to the system. It is assumed that $e_{max} \leq E$, since no algorithm can process jobs with energy requirement greater than E. Observe that this definition implies that a job with size exactly e_{max} may or may not appear in the actual input instance.

The minimum processing time (and energy requirement) of any job in the system is denoted by δ. That is, the size of any job is no smaller than δ units[1]. E and e_i (for each job J_i) are assumed to be expressed as exact multiples of δ. Observe that this assumption guarantees that the number of jobs introduced to the system is polynomial with respect to the energy constraint E. However, the ratio $\frac{\delta}{E}$ may be arbitrarily low.

[1]As energy and execution requirements are expressed as real numbers, δ is assumed to be a real number strictly greater than zero.

Before proceeding with the formal analysis, a specific job arrival pattern is introduced that has proven very useful in deriving competitive factors in online real-time scheduling [17],[75]. Specifically, a set of *sequential jobs* with size 'x' are introduced to the system at time t, if they have the following characteristics:
(1) all jobs in the set have size x,
(2) the first job is released at time t with deadline $t + x$, and,
(3) each subsequent job is released at the deadline of the previous job in the set.
Hence, all the jobs in the sequence have zero laxity. Note that a similar pattern of *sequential jobs* were crucial in obtaining the well-known competitive factor bound of 0.25 for overloaded real-time systems [17],[75].

7.3 Basic Results

It is well-known that preemptive EDF achieves the best possible performance without the knowledge of future job release times, if the system has sufficient energy to execute the entire workload (which is assumed to be feasible in real-time sense).

Proposition 8. *If $E \geq E_b$ then preemptive EDF has a competitive factor of* 1*.*

Proof. Follows from the optimality of preemptive EDF in underloaded real-time environments [45]. □

However, in underloaded environments with scarce energy, preemptive EDF turns out to be a poor online algorithm:

Lemma 2. *If $E < E_b$, then preemptive EDF is non-competitive (i.e. it cannot guarantee a non-zero competitive factor).*

Proof. Figure 7.1 shows how to construct an instance using n jobs ($J_1 \dots J_n$), with decreasing deadlines and increasing release times, for which preemptive EDF cannot guarantee a non-zero value. Let c and D be two positive numbers such that $c \leq e_{max}$ and $D \gg c$.

All jobs have size of c units. Also, $r_1 = 0$ and $d_1 = D$. Given this, the release times and deadlines of any two successive jobs are related by the following equations,

$$r_i = r_{i-1} + c - \delta$$

$$d_i = d_{i-1} - \delta$$

Observe that when EDF is about to finish executing a job, another job with a shorter deadline is released. EDF preempts the currently running job in favor of the job with earlier deadline. Continuing this way, while executing J_n, for some $n \geq 1$, EDF depletes its entire energy budget at time $t = E$. Notice that at time $t = E$, EDF has n pending jobs and zero remaining energy. Since preemptive EDF does not execute any job to completion in this instance, its total value is zero. $\square$

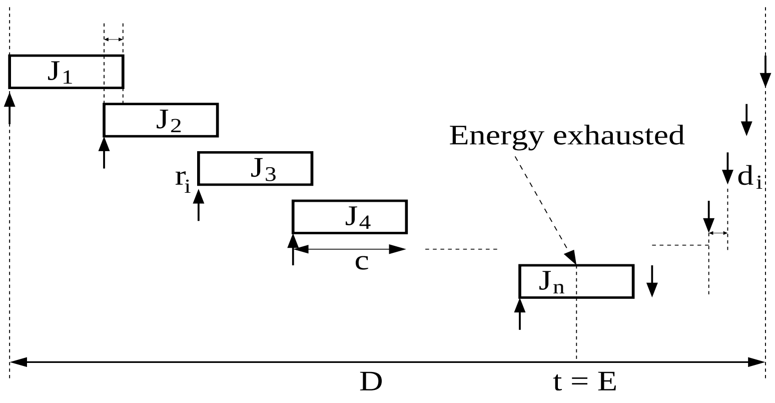


Figure 7.1: The worst-case instance for preemptive *EDF*

An interesting question involves the upper bound on the performance of any online algorithm in energy-constrained settings, in worst-case scenarios. In establishing such upper bounds, the competitive analysis technique typically makes use of the so-called adversary method ([18], [27],[75],[102]). In this proof technique, the adversary generates an initial input job sequence, observes the online algorithms' behavior, and then decides on what further jobs should be released. This process is repeated a finite number of times. At some point (which must comply with the problem specification), the adversary announces the optimal schedule that it would generate for that input sequence – and a bound on the competitive factor is established by comparing it to the schedule selected by the online algorithm. Theorem 6 establishes this upper bound as a function of the energy budget E and the upper bound on maximum job size e_{max}.

Theorem 6. *In energy-constrained underloaded settings, no online algorithm can achieve a competitive factor greater than* $\frac{E-e_{max}}{E}$.

See Appendix E for a full proof.

Theorem 6 implies that, the upper bound on the competitive factor depends heavily on the ratio $\frac{e_{max}}{E}$: the higher this ratio, the lower the best achievable competitive factor. For example, if $\frac{e_{max}}{E} = 1/3$, then, no algorithm can achieve more than 2/3 of the total value of a clairvoyant algorithm. However, as $\frac{e_{max}}{E} \rightarrow 1$, the upper bound approaches zero, implying that no online algorithm can be competitive.

Further, for a given workload, as the system's initial energy budget E is reduced, the best achievable competitive factor quickly decreases. Similarly, for a given energy budget E, as the upper bound on job size e_{max} for the workload increases, the best achievable competitive factor quickly decreases. Nevertheless, it will be shown next that an online algorithm that is able to achieve this upper bound indeed exists.

7.3.1 Algorithm *EC-EDF*

In this subsection, an optimal online algorithm called *EC-EDF* is developed for energy-constrained real-time scheduling in underloaded settings. *EC-EDF* uses an admission test to admit new jobs that arrive at run-time. Specifically, if the newly-arriving job J fails to pass the admission test, it is discarded (i.e. never executed). In case that it passes the test, the new job J is added to the set $\mathcal{C}$ of admitted jobs. When a job completes execution it is removed from set $\mathcal{C}$.

EC-EDF effectively commits to all the admitted jobs, in the sense that, as formally shown below, it guarantees their timely completion without violating the system's energy budget limits. Further, all the admitted jobs are scheduled according to the well-known preemptive EDF policy.

The admission test uses the following relatively simple rule to decide whether the new job J arriving at time t can be admitted or not: *J is admitted if and only if the system's remaining energy budget at time t, E^r, is sufficient to fully execute J and the remaining workload of all the pending admitted jobs (i.e. the remaining workload in set $\mathcal{C}$).*

Let E^r and e_i^r represent the remaining energy with the system and the remaining execution time of job J_i at time t respectively. It is assumed that both E^r and e_i^r are properly updated at run-time. Hence, formally, a job J with size e arriving at time t is admitted to the system if and only if :

$$E^r \geq e + \sum_{J_i \in \mathcal{C}} e_i^r$$

The following example illustrates the behavior of *EC-EDF*. Consider a system with $E = 100$. The following four jobs constitute the input sequence: $J_1(0, 20, 200)$, $J_2(10, 30, 190)$, $J_3(25, 75, 150)$ and $J_4(85, 15, 120)$. Figure 7.2 shows the schedules generated by *EC-EDF*, *EDF* and the clairvoyant algorithm along with the total values obtained by each. In each schedule, the unshaded jobs are those that complete before the corresponding algorithm depletes its energy budget, while the shaded jobs are those that fail to do so.

At time $t = 0$, *EC-EDF* admits J_1 and dispatches it ($\mathcal{C} = \{J_1\}$). At $t = 10$, J_2 with higher priority than J_1 arrives. *EC-EDF* admits J_2 as there is enough remaining energy to execute both J_2 and the pending workload of admitted jobs, $\mathcal{C} = \{J_1\}$ (i.e. $E^r \geq e_2 + e_1^r$). *EC-EDF* updates set $\mathcal{C}$ to $\{J_1, J_2\}$.

Notice that at $t = 25$ when the largest job J_3 in the set arrives the remaining energy is sufficient to execute it. However *EC-EDF* does not admit J_3 as with the remaining energy of 75 units, the system cannot execute J_3 and the pending workload of admitted jobs, $\mathcal{C} = \{J_1, J_2\}$ (i.e. $E^r < e_3 + e_1^r + e_2^r$).

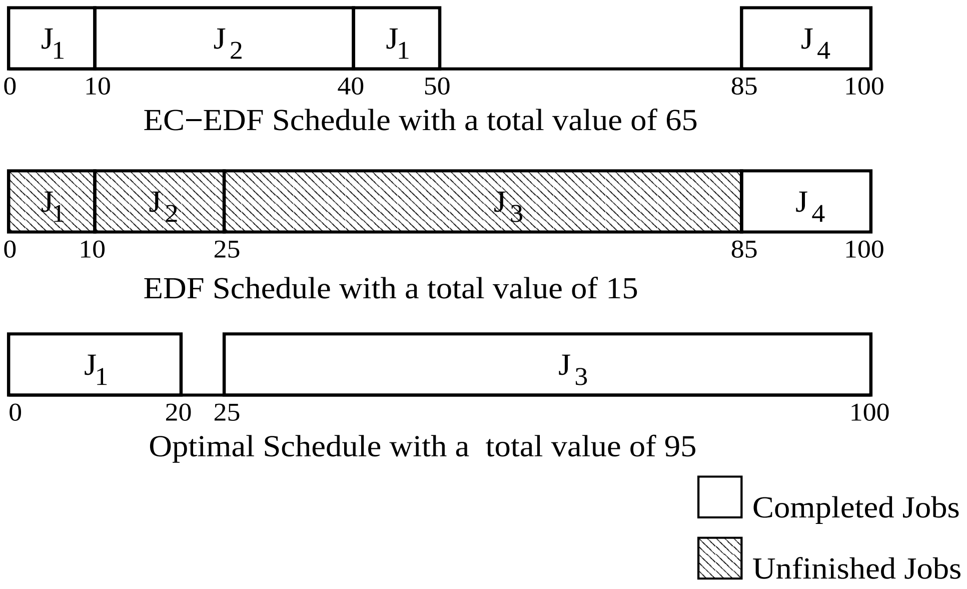

J1
J2
J1
J4
0
10
40
50
85
100
EC−EDF Schedule with a total value of 65
J1
J2
J3
J4
0
10
25
85
100
EDF Schedule with a total value of 15
J1
J3
0
20
25
100
Optimal Schedule with a total value of 95
Completed Jobs
Unfinished Jobs

Figure 7.2: Schedules Generated by *EC-EDF*, *EDF* and *Optimal*

At $t = 85$, *EC-EDF* admits and executes J_4 to completion. Thus, by executing jobs J_1, J_2 and J_4 to completion, *EC-EDF* gathers a total value of 65. It is straightforward to verify that EDF gets a total value of only 15. The clairvoyant algorithm knowing, the future job sizes and arrival patterns, skips certain jobs and idles as necessary, making a total value of 95 as shown in Figure 7.2.

Proposition 9. EC-EDF *guarantees the completion of all the admitted jobs before their deadlines, without violating the system's energy budget limits.*

Proof. Assume there exists a non-empty subset $\mathcal{C}'$ of the admitted jobs that cannot be completed in a timely manner without violating the energy budget limits. It will shown by contradiction that $\mathcal{C}'$ cannot exist.

Recall that by assumption, the input sequence ψ is guaranteed to be feasible from timing constraints point of view. Thus, $\mathcal{C}' \subset \psi$ is also feasible. Further, *EC-EDF* schedules the admitted jobs using preemptive EDF which is known to be optimal in underloaded conditions. Hence, if $\mathcal{C}'$ exists then *EC-EDF* must run out of energy before completing the jobs it admitted. Since the admission rule of *EC-EDF* ensures that there is always enough remaining energy in the system to meet the computational demand of all pending admitted jobs along with the newly admitted job, a contradiction is reached here. Hence, $\mathcal{C}'$ cannot exist. $\square$

Theorem 7. EC-EDF *has a competitive factor of* $\frac{E-e_{max}}{E}$.

Proof. First, note that if $E_b \leq E$, *EC-EDF* reduces to traditional EDF and can execute all the jobs in the input sequence ψ (which is assumed to be feasible), achieving a competitive factor of one. Thus, the adversary must create a feasible input sequence ψ such that $E_b > E$. Under this condition, eventually *EC-EDF* will be forced to discard a job due to energy limitations.

Let J_d be the first job discarded by *EC-EDF* at time t. At time t, let ψ' denote the set of jobs admitted by *EC-EDF* before discarding J_d and let E' denote the total workload (and energy demand) of the jobs in ψ'. From Proposition 9, *EC-EDF* is guaranteed to complete the job set ψ' in a timely manner without violating the system energy budget limits. Thus, *EC-EDF* guarantees a total value of E'.

As a consequence of J_d being discarded at time t, $E' + e_d > E$. Since $e_d \leq e_{max}$, this implies $E' > E - e_{max}$. Thus, the total value obtained by *EC-EDF* is at least $E - e_{max}$, while the adversary can make at most a value of E. Thus, *EC-EDF* has a competitive factor of $\frac{E-e_{max}}{E}$. $\square$

Note here that in settings where $E \geq E_b$, *EC-EDF* is able to finish all the jobs in the input sequence thanks to the optimality of *EDF*, achieving a competitive factor of 1 under that condition. That is, regardless of the relationship between E and E_b, *EC-EDF* yields the best competitive factor.

7.3.2 A Semi-online Algorithm with a Constant Competitive Factor

It is occasionally possible to improve the competitive ratio of online algorithms by providing some limited information about the actual input sequence. For example, online scheduling algorithms typically benefit from information about the maximum job size, sum of job processing times or job size patterns in the actual input [49],[102]. The online algorithms that exploit this type of limited information about the actual input are called semi-online and their design and analysis have been recently attracting increasing interest [49],[102].

Even for the problem under consideration, the knowledge of the largest job size in the actual input instance turns out to be very helpful. For the uniform value density model where the value of a job is equal to its execution time, the largest size job in the input instance is also the most valuable job in the set. Below, the algorithm *EC-EDF** that achieves a competitive factor of 0.5 using this information, is described. Further, it will be shown that with only the additional knowledge of the largest job size, no semi-online algorithm can perform better than *EC-EDF**, demonstrating its optimality.

Let $e_l \leq E$ represent the largest job size in the actual input sequence. First the rules for *EC-EDF** are given. *EC-EDF** exploits the information about e_l: in semi-online settings, the fact that at least one job of size e_l will be part of the input sequence is guaranteed by definition. *EC-EDF** compares e_l to the available energy budget E. If $e_l > \frac{E}{2}$, *EC-EDF** simply waits for the largest size job, e_l, and executes it. Since $E \geq e_l > \frac{E}{2}$, the value of *EC-EDF** is no less than $\frac{E}{2} + \delta$. The adversary can gather a value of at most E. On the other hand, if $e_l \leq \frac{E}{2}$, *EC-EDF** schedules jobs using *EC-EDF*. By definition, no job size in the actual input sequence can be larger than e_l. Thus, from Theorem 7, the competitive factor for *EC-EDF* becomes $\frac{E-e_l}{E}$. Further, given the constraint $e_l \leq \frac{E}{2}$, the lower bound on competitive factor of *EC-EDF* and hence that of *EC-EDF** reduces to 0.5. Thus, in either case, *EC-EDF** makes at least half of the value of the adversary.

Lemma 3. *EC-EDF** *has a competitive factor of* 0.5*.*

Lemma 4. *With only the additional knowledge of the largest job size, no semi-online algorithm can achieve a competitive factor greater than* 0.5*.*

Proof. The proof involves a specific scenario in which the adversary effectively limits the total value of any semi-online algorithm to half of its value. Let $E = k_1 \cdot \delta$ and $e_l = k_2 \cdot \delta$. Where, k_1 and k_2 are positive integers such that $k_1 \geq k_2$. The proof is similar to that of

Theorem 6. The adversary first releases $(k_1 - k_2 + 1)$ *sequential jobs* of size δ (note that here $k_2 = \frac{e_l}{\delta}$ and not $\frac{e_{max}}{\delta}$ as in Theorem 6). As mentioned in proof of Theorem 6, let m be the number of jobs not executed by the online algorithm $\mathcal{A}$. Again, there are 3 cases.

Cases 1, 2 and $3B$ are similar to those in Theorem 6, except that at the very end the adversary now releases a job of size e_l instead of e_{max}. Therefore, repeating the corresponding arguments from Theorem 6, the competitive factor of $\mathcal{A}$ for these cases is given by $\frac{E-e_l}{E}$.

Case $3A$ corresponds to k_2 jobs of size δ having been released and $\mathcal{A}$ having executed $0 \le m' < m$ of these jobs. Repeating the corresponding arguments of Theorem 6, notice that $\mathcal{A}$ has accrued a value of $(k_1 - k_2 + 1 + (m' - m)) \cdot \delta$ and has a remaining energy budget of $(k_2 - (m' - m + 1)) \cdot \delta$, while the adversary has accrued a total value of E and has no energy budget left. Also, observe that in the input sequence leading to and terminating in Case $3A$, a job of size e_l has not been released by the adversary. Thus, at the end of Case $3A$, the adversary releases a job of size e_l that $\mathcal{A}$ can execute provided it has enough energy budget left to execute it (i.e. if $(m' - m + 1) \le 0$). Thus, the maximum value that $\mathcal{A}$ can accrue is $e_l + (k_1 - k_2 + 1 + (m' - m)) \cdot \delta$ and its competitive factor is no better than

$$\frac{e_l + (k_1 - k_2 + 1 + (m' - m)) \cdot \delta}{E} \ge \frac{e_l}{E}$$

From the above 3 cases the competitive factor of $\mathcal{A}$ can be expressed as $\max(\frac{E-e_l}{E}, \frac{e_l}{E})$. Since, $0 \le e_l \le E$, one has $\max(\frac{E-e_l}{E}, \frac{e_l}{E}) \ge 0.5$. Thus, no semi-online algorithm can achieve a competitive factor greater than 0.5. $\square$

Corollary 5. *Among semi-online algorithms that have only the additional knowledge of the largest job size, EC-EDF* is optimal.*

In online real-time scheduling in overload conditions, the best achievable competitive factor was shown to be 0.25 [17] and further, there exists an algorithm D^{over} with this competitive factor [75]. In contrast, in online real-time scheduling with hard energy constraints for underloaded settings, with the knowledge of the largest job size in the actual input sequence, the best achievable competitive factor is 0.5 which is twice as large as that in overloaded settings. Further, the algorithm *EC-EDF** achieves this competitive factor.

7.4 Competitive Analysis for Non-idling and Non-preemptive Scheduling Algorithms

The main results in preceding sections were derived under the assumption that preemption was allowed and that, if needed, the online algorithm could leave the CPU idle in the presence of ready jobs. In this section, the implications of relaxing each of these assumptions are investigated separately.

7.4.1 Non-Idling Execution Settings

In real-time scheduling theory, a scheduling algorithm is said to be idling at time t, if there is a pending job and the algorithm is not executing any job. The algorithms that never idle (i.e. *non-idling* or *work-conserving* algorithms) have practical importance and were studied in the literature [69],[83]. In what follows, a basic competitive analysis of *non-idling* online real-time scheduling algorithms in energy-constrained settings is considered.

Note that the traditional definition of idling, as given above, is not quite adequate for energy-constrained settings, as an algorithm should not be forced to execute a pending job requiring more than the remaining energy, or waste energy by executing a job that is guaranteed to miss its deadline. Thus, in an effort to address non-idling scheduling issues, first the *idling* concept is formally re-defined for energy-constrained settings.

Definition 2. *In energy constrained settings, an algorithm is considered to be idling if and only if all of the following four conditions hold at a specific time instant t:*

- *The processor is not executing any job,*
- *There is a pending (ready) job J_i with remaining execution time/energy requirement $e_i^r > 0$.*

- *$e_i^r \leq d_i - t$,*
- *$e_i^r \leq E^r$, where E^r is the system's remaining energy budget.*

Given this definition, a non-idling algorithm is the one that does not idle at any time t. Observe that, according to these definitions, *EC-EDF* is a non-idling algorithm, while *EC-EDF** is an idling algorithm. Section 7.3.1 already shows that *EC-EDF* is the best non-idling algorithm that can be developed against a potentially idling adversary.

The power of idling for the adversary was crucial in deriving the competitive factor bound in Theorem 6. The main motivation behind this subsection is to investigate if there can exist an non-idling algorithm better than *EC-EDF* against an adversary that cannot idle either. Hence, as opposed to the previous sections, the adversary cannot idle in the analysis presented in this subsection.

Lemma 5. *No non-idling online algorithm can achieve a competitive factor greater than $\frac{E-e_{max}}{E}$ against a non-idling adversary.*

Proof. Let $E = k_1 \cdot \delta$ and $e_{max} = k_2 \cdot \delta$. Where, k_1 and k_2 are positive integers such that $k_1 \geq k_2$. In the execution scenario that will establish the bound, the adversary first introduces $(k_1 - k_2)$ *sequential jobs* each of size δ. By executing these jobs, at time $t = (k_1 - k_2) \cdot \delta$, both the online algorithm $\mathcal{A}$ and the adversary have accrued a total value of
$(k_1 - k_2) \cdot \delta = E - e_{max}$ and have a remaining energy budget of e_{max} units. At time t, the adversary introduces the following 2 jobs: $J_1(t, \delta, t+\delta)$ and $J_2(t, 2\delta, t+5\delta)$. $\mathcal{A}$ has two choices.

Case 1: $\mathcal{A}$ executes J_1. In this case, the adversary executes J_2 and waits until $t_1 = t+5\delta$. Notice that since $\mathcal{A}$ scheduled J_1, it will have to also schedule J_2 after finishing J_1, since it cannot idle. On the other hand by executing J_2, the adversary has effectively created an idle interval for itself by missing the deadline of J_1. At t_1, $\mathcal{A}$ has accrued a value of $E - e_{max} + 3\delta$ and has a remaining energy budget of $e_{max} - 3\delta$ units. On the other hand, at time t_1, the adversary has accrued a value of $E - e_{max} + 2\delta$ and has a remaining energy budget of $e_{max} - 2\delta$ units. At t_1, the adversary introduces a job with size $e_{max} - 2\delta$ that

$\mathcal{A}$ cannot execute. By executing this job the adversary accrues a total value of E. The competitive factor is $\frac{E-e_{max}}{E}$.

Case 2: $\mathcal{A}$ executes J_2. In this case, the adversary executes J_1 followed by J_2. When $\mathcal{A}$ finishes J_2 at time $t + 2\delta$, the adversary has finished executing only half of J_2. At this stage the adversary introduces job $J_3(t + 2\delta, 2\delta, t + 4\delta)$. Note that, $\mathcal{A}$ is forced to execute J_3 (as it cannot idle). The adversary, by continuing with execution of J_2, misses J_3 and again creates an idle period for itself from $t + 3\delta$ to $t + 5\delta$. At time $t1 = t + 5\delta$, $\mathcal{A}$ has accrued a value of $E - e_{max} + 4\delta$ and has a remaining energy budget of $e_{max} - 4\delta$ units. On the other hand, at time t_1, the adversary has accrued a value of $E - e_{max} + 3\delta$ and has a remaining energy budget of $e_{max} - 3\delta$ units. At t_1, the adversary introduces a job with size $e_{max} - 3\delta$ that $\mathcal{A}$ cannot execute. By executing this job the adversary accrues a total value of E. The competitive factor is $\frac{E-e_{max}}{E}$. $\square$

In the non-idling model, the knowledge of the largest job size in the actual input sequence does not help, as the algorithm $\mathcal{A}$ cannot idle and wait for it, which was crucial for achieving a competitive idling semi-online algorithm *EC-EDF**. In the non-idling model, the adversary can introduce jobs such that when the largest size job is released, the remaining energy budget of $\mathcal{A}$ is not sufficient to execute it.

Corollary 6. EC-EDF *is also optimal against an adversary that cannot idle.*

7.4.2 Non-Preemptive Execution Settings

This subsection restricts analysis to non-preemptive online scheduling algorithms. Non-preemptive execution have several attractive features, such as ease of implementation, low run-time overheads, implicit exclusive access to shared resources, among others [83]. As such, non-preemptive scheduling algorithms have been studied in the realm of real-time scheduling literature [69],[83]. This subsection presents basic results for settings where both the online algorithm $\mathcal{A}$ and the adversary schedule jobs through non-preemptive algorithms.

Lemma 6. *There does not exist a competitive non-preemptive online algorithm, irrespective of the ratio* $\frac{e_{max}}{E}$.

Proof. In the scenario that enforces this upper bound, the adversary introduces a job of size 2δ and deadline $e_{max} + 2\delta$. If the online algorithm $\mathcal{A}$ idles until $t = e_{max} + 2\delta$ and skips this job, the adversary continues the pattern. $\mathcal{A}$ will have to execute one of these jobs before the cumulative workload released by the adversary reaches E or more. When $\mathcal{A}$ executes a job (say at time t), the adversary releases a job of size e_{max} with zero laxity at time $t + \delta$. The non-preemptive algorithm $\mathcal{A}$ gathers a value of 2δ, while the adversary gathers a value of e_{max}. By repeating the same pattern a finite number of times, the algorithm $\mathcal{A}$ can be made technically non-competitive, regardless of the ratio $\frac{e_{max}}{E}$. □

Notice that with non-preemption, while the online algorithm is executing a job of size X for execution, the adversary can release a job of size $Y \gg X$ and no laxity, causing excessive value loss for the algorithm. This was the key to the instance given in the proof of Lemma 6. As a consequence, with the knowledge of the largest job size (say e_l) in the input instance, no (potentially) idling non-preemptive semi-online algorithm can achieve a competitive factor greater than $\frac{e_l}{E}$ and the algorithm that achieves this bound simply waits for the largest job size, e_l. The ratio $\frac{e_l}{E}$ may be arbitrarily small if $e_l < \frac{E}{2}$. However, if $e_l \geq \frac{E}{2}$, this semi-online algorithm has a competitive factor of 0.5.

7.4.3 Non-idling and Non-preemptive Execution Settings

In general, non-preemptive real-time scheduling is known to be NP-Hard in the strong sense, even in off-line settings [69]. However, non-idling EDF is optimal among all non-idling non-preemptive policies [69]. Lemma 7 below establishes a fundamental result for non-preemptive, non-idling algorithms. Lemma 7 is derived for settings where the workload is feasible under non-idling and non-preemptive scheduling.

Lemma 7. *No non-preemptive non-idling online algorithm can achieve a competitive factor greater than* $\frac{E-e_{max}}{E}$ *and non-preemptive* EC-EDF *achieves this competitive factor.*

Proof. The proof is very similar to that of Lemma 5. However, in Case 2 where $\mathcal{A}$ executes J_2, introduce the third job J_3 at time $t+\delta$ (i.e. $J_3(t+\delta, 2\delta, t+4\delta)$). Observe that this allows feasibility of J_1, J_2 and J_3 under non-preemptive non-idling scheduling. Now, by repeating the arguments of Lemma 5 the upper bound on performance of any online non-preemptive and non-idling algorithm can be found as $\frac{E-e_{max}}{E}$. Since non-preemptive non-idling EDF is an optimal real-time scheduling algorithm in the sense that it can generate a feasible schedule whenever another non-preemptive non-idling algorithm can do so [69], the non-preemptive *EC-EDF* is found as the optimal non-idling non-preemptive algorithm. $\square$

It is also straightforward to verify that the additional knowledge of the largest job size in the actual input sequence does not help design better non-preemptive non-idling online algorithms.

7.5 Non-Uniform Value Densities

In value-based real-time execution model, each job is associated with a value proportional to its execution time. The *value density* of a job is its value divided by its execution time. The ratio of the largest value density to the smallest value density is called *importance ratio* [17],[75]. In uniform density settings (Section 7.3), the *importance ratio* is one and hence the value of the job is equal to its size. This section undertakes the competitive analysis in the more general non-uniform density settings.

In these settings, the value of a job J_i with value density k_i is given by $k_i e_i$. Let k_{min} and k_{max} denote the smallest and largest value densities that can be associated with any job. Without loss of generality, it is assumed that $k_{min} = 1$ and thus the ratio of $\frac{k_{max}}{k_{min}}$ is simply k_{max}. Observe that in comparison to uniform density settings, the largest size job is no longer guaranteed to be the most valuable job.

Theorem 8. *In non-uniform value settings where $k_{max} > 1$, no online algorithm can achieve a competitive factor greater than $\frac{1}{(k_{max})^{\frac{e_{max}}{E}}}$.*

The full proof of Theorem 8 is provided in Appendix F.

Note that the upper bound established by Theorem 8 is not tight when the ratio $\frac{k_{max}}{k_{min}}$ is close to one. This can be verified by setting $k_{max} = 1 + \Delta \approx 1$, where Δ is very small, and repeating the input instance given in the proof of Theorem 6 with the following two changes: (1) All the released jobs have value density k_{max}. (2) At the very end, in every branch, the adversary releases a single job of size δ with value density $k_{min} = 1$. In this specific modified sequence the upper bound can easily be verified to be $\frac{k_{max}(E-e_{max})}{k_{max}E} = \frac{E-e_{max}}{E}$, while for the given parameters the upper bound suggested by Theorem 8 is close to 1, irrespective of the ratio $\frac{e_{max}}{E}$. However, as k_{max} increases the upper bound established by Theorem 8 decreases in exponential fashion. Investigating the tightness of the upper bound established by Theorem 8 when the ratio $\frac{k_{max}}{k_{min}}$ is large is an open problem.

Also, note that the upper bound on performance established by Theorem 8 holds with or without the knowledge of the largest job size in the input sequence. This can be verified by the job release pattern given in the proof of Theorem 8. Thus, a priori knowledge of the largest job size does not help design better online algorithms in non-uniform value density settings. Lemma 8 establishes the competitive factor of the semi-online algorithm, *EC-EDF**, which uses the additional knowledge of largest job size.

Lemma 8. *Algorithm EC-EDF* has a competitive factor of* $\frac{1}{2k_{max}}$.

Proof. Let $e_l \leq E$ represent the largest job size in the input sequence. If $e_l > \frac{E}{2}$, *EC-EDF** waits for the largest job of size e_l and is guaranteed a value of $k_{min}(\frac{E}{2} + \delta) = \frac{E}{2} + \delta$. The adversary can make at most $k_{max}E$. The competitive factor is $\frac{1}{2k_{max}}$. On the other hand, if $e_l \leq \frac{E}{2}$, *EC-EDF** follows *EC-EDF* which would guarantee a value of at least $k_{min}(E - e_l + \delta)$ even when executed on jobs with the minimum value density $k_{min} = 1$. Again, the adversary can make a total value of at most k_{max} E. Thus, the competitive factor is found as the minimum value of $\frac{E-e_l+\delta}{k_{max}\,E}$ which is $\frac{1}{2k_{max}}$ (obtained when $e_l = \frac{E}{2}$). As a result, in both cases, the competitive factor is $\frac{1}{2k_{max}}$. □

It is also interesting to consider the upper bound on the competitive factor of any non-idling algorithm (compared against an idling adversary) in these settings. Theorem 9 below establishes this bound.

Theorem 9. *No non-idling online algorithm can achieve a competitive factor greater than* $min(\frac{E-e_{max}}{E+(k_{max}-1)e_{max}}, \frac{1}{2k_{max}})$ *against a potentially idling adversary.*

Proof. Two constants $c_1 = \lfloor \frac{E}{e_{max}} \rfloor$ and $c_2 = E - (c_1 \cdot e_{max})$ are defined. First, the adversary introduces a workload W_1 of $E - e_{max} + \delta$ units using $n \geq 1$ jobs, each with value density $k_{min} = 1$. All jobs in W_1 have zero laxity and are released back to back, one at a time (i.e. the release time of the i^{th} job coincides with the deadline of the $(i-1)^{th}$ job and the deadline of the i^{th} job is its release time plus execution time). At least one job in W_1 will be of size exactly c_2. At the end of execution of the workload W_1 the non-idling algorithm $\mathcal{A}$ has exhausted $E - e_{max} + \delta$ units of energy and gathered a value of $E - e_{max} + \delta$. Two cases are distinguished:

Case 1: Assume $e_{max} \leq \frac{E}{2}$

At this stage, the adversary introduces another workload W_2 of $c_1 e_{max}$ units using c_1 jobs of size e_{max} and value density k_{max} (as with W_1 all jobs in W_2 have zero laxity and are released back to back). $\mathcal{A}$ cannot execute any job in W_2 irrespective of the job deadlines. The adversary gathers a value of $c_1 k_{max} e_{max} + c_2$ by executing all jobs in workload W_2 and the job of size c_2 in workload W_1. The value of $\mathcal{A}$ is $E - e_{max} + \delta$. If $c_2 = 0$, the competitive factor is $\frac{E-e_{max}}{k_{max}E}$. If $c_2 \neq 0$, by setting $c_2 = \delta$ the competitive factor is minimized and in that case, is bounded by $\frac{E-e_{max}}{k_{max}E}$. Thus, either way the competitive factor is no more than $\frac{1}{2k_{max}}$ (obtained by setting $e_{max} = \frac{E}{2}$).

Case 2: Assume $e_{max} > \frac{E}{2}$

In this case $c_1 = 1$ and $c_2 = E - e_{max}$. The adversary has two different strategies to reduce the competitive factor as much as possible depending on the value of k_{max} (similar to Case 1 above). First, the adversary can follow the exact same sequence given in Case 1 above by restricting the largest job size in the actual instance to $\frac{E}{2}$. In this case, the competitive factor, as explained in Case 1, would be $\frac{1}{2k_{max}}$.

Second, the adversary can introduce another workload, W_2, of $c_1 e_{max}$ units using c_1 jobs of size e_{max} and value density k_{max}. $\mathcal{A}$ cannot execute any job in W_2 irrespective of their deadlines. The adversary gathers a value of $c_1 k_{max} e_{max} + c_2$ by executing all jobs in workload W_2 and the job of size c_2 in workload W_1. The competitive factor would then be $\frac{E-e_{max}}{k_{max}e_{max}+E-e_{max}}$.

From Cases 1 and 2 one can conclude that the competitive factor is constrained either by $\frac{1}{2k_{max}}$ or by $\frac{E-e_{max}}{k_{max}e_{max}+E-e_{max}}$. Further, since the ordering between these two terms is dependent on the values of e_{max} and k_{max}, no non-idling algorithm can achieve a competitive factor greater than $min(\frac{E-e_{max}}{E+(k_{max}-1)e_{max}}, \frac{1}{2k_{max}})$. □

Next, it will be shown that *EC-EDF* is the optimal non-idling algorithm in non-uniform value density settings as well.

Lemma 9. EC-EDF *has a competitive factor of* $min(\frac{E-e_{max}}{E+(k_{max}-1)e_{max}}, \frac{1}{2k_{max}})$.

Proof. Let t represent the time instance at which *EC-EDF* discards a job for the first time. Let the execution requirement of this job discarded by *EC-EDF* be e_d. Since $e_d \leq e_{max}$, at time t, *EC-EDF* has accepted a workload of at least $E - e_d + \delta$ and from Proposition 9, *EC-EDF* is guaranteed to make a value of at least:

$$V = k_{min}(E - e_d + \delta) = E - e_d + \delta$$

Thus, the total value accrued by *EC-EDF* is $V + \Delta V$, where $\Delta V \geq 0$.

If V^* represents the total value accrued by the adversary then $0 \leq V^* \leq k_{max}E$. Thus the competitive factor can be defined as $\frac{V+\Delta V}{V^*}$. However, in non-uniform value density settings, it is not necessary that ΔV is minimized (i.e. $\Delta V = 0$) when V^* is maximized (i.e. $V^* = k_{max}E$); which was the case in uniform value density settings. Thus, when jobs have different value densities, there is a non-trivial relation between V^* and ΔV. Two cases are distinguished.

Case 1: $0 < e_d \leq \frac{E}{2}$

In this case the maximum value the adversary can make without increasing the value accrued by *EC-EDF* beyond V is $k_{max}E$; which is also the maximum value the adversary can make in non-uniform value density settings. This can be achieved by releasing a workload consisting of jobs with total execution requirement e_d and value density k_{max} after time t. While *EC-EDF* has not enough energy left to execute any of these jobs, the adversary can execute all these jobs making a value of at most $k_{max}E$. Thus, in this case $\Delta V = 0$ and $V^* = k_{max}E$ giving a competitive factor of $\frac{E-e_d}{k_{max}E}$ which is minimized at $e_d = \frac{E}{2}$ to $\frac{1}{2k_{max}}$.

Case 2: $e_{max} \geq e_d > \frac{E}{2}$

In this case the maximum value the adversary can make without increasing the value accrued by *EC-EDF* beyond V is:

$$V_1^* = E - e_d + \delta + k_{max}e_d < k_{max}E$$

In this case the competitive factor is minimized at $e_d = e_{max}$ and is given by:

$$\frac{E - e_{max}}{E + (k_{max} - 1)e_{max}}$$

On the other hand, if the adversary intends to make a value $V_2^* > V_1^*$ then $\Delta V > 0$, and the following observation holds.

Observe that, if V_2^* is maximized (i.e. $V_2^* = k_{max}E$) then $\Delta V \geq (2e_d - E)$. As a consequence, if $V_2^* = k_{max}E$ then $V + \Delta V \geq e_d + \delta$. In this case the competitive factor is minimized at $e_d = \frac{E}{2} + \delta$ and is given by $\frac{1}{2k_{max}}$.

From Cases 1 and 2, the competitive factor of *EC-EDF* is:

$$min(\frac{E - e_{max}}{E + (k_{max} - 1)e_{max}}, \frac{1}{2k_{max}})$$

□

Corollary 7. *EC-EDF is an optimal non-idling algorithm in non-uniform value settings.*

The following remarks conclude this subsection.

- The bound of $(k_{max})^{-\frac{e_{max}}{E}}$ holds in general for any online algorithm (idling or non-idling). The knowledge of the largest job size does not help improve the upper bound on competitive factor achievable by any online algorithm.
- Again, the competitive factor of the algorithm heavily depends on the ratio $\frac{e_{max}}{E}$. When $e_{max} \leq \frac{E}{2}$, the best achievable competitive factor is $\frac{1}{\sqrt{k_{max}}}$. As $\frac{e_{max}}{E} \to 1$, the best achievable competitive factor reduces to $\frac{1}{k_{max}}$.
- The upper bound established in Theorem 8 is not tight when the ratio $\frac{k_{max}}{k_{min}}$ is close to one. For larger values of $\frac{k_{max}}{k_{min}}$, investigating the tightness of this upper bound is an interesting open problem that deserves further research.

7.6 DVS settings

This section derives an upper bound on the competitive factor of DVS-enabled online algorithms. A DVS-capable processor whose frequency can be varied in the range $[f_{min}, f_{max}]$ is considered. Without loss of generality, it is assumed that $f_{max} = 1$. Power consumption of the processor at frequency f is modeled as a convex function $P(f) = af^{\alpha}$, where a is a constant characterized by the processor parameters[2] and $2 \leq \alpha \leq 3$. Thus, at frequency f, the time and energy required to execute a job with processing time x are given by $\frac{x}{f}$ and $E(f) = P(f) \cdot \frac{x}{f} = f^{\alpha-1}x$, respectively.

[2]In this section it is assumed that $a = 1$. Recently there have been research efforts addressing system energy models, where it was shown that lowering frequency below a certain threshold (called energy-efficient speed) may adversely affect overall system energy consumption [10]. The upper bound results can easily be adapted to these system-level energy models by enforcing arbitrarily low energy-efficient speed levels.

In line with the previous sections the *underloaded energy-constrained system assumption* is made: in DVS settings, there should exist a feasible schedule for all jobs in ψ, when the processor executes all jobs at frequency f_{max} and the energy constraints are not taken into account.

Notice that with DVS, the energy required to execute a job depends on the frequency at which the job is executed. As such, there is no longer one-to-one correspondence between job execution times and energy requirements. By taking this into account, in this section, a job J_i is represented by $J_i(r_i, C_i, D_i, e_i)$, where r_i is its release time, C_i is its execution time at frequency f_{max} (also referred to as workload of J_i), D_i is its relative deadline and e_i is its minimum energy requirement. The minimum energy requirement e_i is computed by assuming the minimum frequency $\frac{C_i}{D_i}$ that would allow the timely completion of J_i before its deadline. That is, $e_i = E(\frac{C_i}{D_i}) = (\frac{C_i}{D_i})^{\alpha-1} \cdot C_i$.

Note that the uniform density model is assumed throughout this section and the value of a job J_i is assumed to be equal to C_i (execution time at f_{max}). That is, executing a job at a low frequency reduces the energy consumption but does not affect its value. Before proceeding, some basic definitions and existing results are given that will be instrumental in the analysis.

Definition 3. *Let $l(t_1, t_2)$ denote the total amount of workload of jobs with release times at or later than t_1 and deadlines at or earlier than t_2. The effective loading factor $h(t_1, t_2)$ over an interval $[t_1, t_2]$ is defined as $h(t_1, t_2) = \frac{l(t_1,t_2)}{t_2-t_1}$.*

Definition 4. *The absolute effective loading factor (or simply the loading factor) β is the maximum effective loading factor over all intervals $[t_1, t_2]$: $\beta = max(h(t_1, t_2))$,*
$0 \leq t_1 < t_2$.

Theorem 10. (from [16],[70]) *A set of real-time jobs can be scheduled in feasible manner (by preemptive EDF) if and only if $\beta \leq 1$.*

Given the loading factor β, if the processor executes all jobs at constant frequency

$f = max(f_{min}, \beta)$, then the new loading factor β' (increased due to the reduced frequency) would be $\frac{\beta}{f}$. Further, one can easily verify that β' would still not exceed 1.0 under that condition [130]. Thus, running jobs at frequency $f = max(f_{min}, \beta)$ preserves the system feasibility (without the energy constraint).

Proposition 10. *A DVS algorithm that runs at constant speed $f \geq max(f_{min}, \beta)$ cannot make a total value $> \frac{E}{f^{\alpha-1}}$.*

Proposition 10 can be justified by observing that with DVS, to deplete e units of energy, the system will have to execute a workload of $\frac{e}{f^{\alpha-1}}$ at constant frequency f. Thus, with e units of energy, a maximum value of $\frac{e}{f^{\alpha-1}}$ can be made by running the processor at constant speed f. Note that this implies that with DVS the system is able to achieve a value greater than E. In settings where both the online algorithm and adversary have DVS, the pre-knowledge of β can potentially provide some advantage to the online algorithm. Theorem 11 characterizes this result.

Theorem 11. *Assuming $\beta > f_{min}$ where $0 < \beta \leq 1$,*
(i) Without the knowledge of β, there is no online DVS algorithm with a competitive factor greater than $f_{min}^{\alpha-1}$.
(ii) With the knowledge of β, there is no online DVS algorithm with a competitive factor greater than $(\frac{f_{min}}{\beta})^{\alpha-1}$.

Proof. **Case 1: Assume β is unknown to the algorithm**
Consider the following instance. The adversary sets $\beta = 1$ and introduces a job $J_1(0, E, E, E)$ (Notice the minimum energy requirement for J_1 is $e_1 = E$). Clearly, if the online algorithm $\mathcal{A}$ does not execute J_1, it will miss its deadline. The adversary executes J_1 and releases no more jobs. The value of $\mathcal{A}$ is zero, while that of the adversary is E.

If $\mathcal{A}$ executes J_1, then, at time $t = E$, the adversary introduces $J_2(E, \frac{E}{k^{\alpha-1}}, \frac{E}{k^{\alpha}}, E)$, where $f_{min} \leq k \leq 1$. Observe that J_2 can be executed at frequency $\frac{\frac{E}{k^{\alpha-1}}}{\frac{E}{k^{\alpha}}} = k$. By skipping

J_1, the adversary executes J_2 gathering a value of $\frac{E}{k^{\alpha-1}}$. Thus, the competitive factor is $\frac{E}{\frac{E}{k^{\alpha-1}}} = k^{\alpha-1}$. Since the minimum possible frequency is f_{min}, by setting $k = f_{min}$, the adversary can force an upper bound of $(f_{min})^{\alpha-1}$ on the competitive factor.

Case 2: Assume β is known to the algorithm

The pattern in Case 1 can be repeated with a slight modification. J_1 is given with parameters $J_1(0, \frac{E}{\beta^{\alpha-1}}, \frac{E}{\beta^{\alpha}}, E)$. $\mathcal{A}$ is forced to execute J_1 at frequency β to guarantee a non-zero total value. At that point, the adversary introduces J_2 with the following parameters $J_2(E, \frac{E}{f_{min}^{\alpha-1}}, \frac{E}{f_{min}^{\alpha}}, E)$. Observe that J_2 can be executed by the adversary at frequency f_{min}, yielding a value of $\frac{E}{f_{min}^{\alpha-1}}$. Further, since $f_{min} < \beta$, the processor loading factor can be easily shown to be β, satisfying the assumption. $\mathcal{A}$ has a value of $\frac{E}{\beta^{\alpha-1}}$ and thus the competitive factor is bounded by $(\frac{f_{min}}{\beta})^{\alpha-1}$. $\square$

The case where $\beta \leq f_{min}$ is relatively simple: consider the algorithm *EC-EDF* using a constant speed f_{min}. Notice that this algorithm does never spend more energy than required while processing jobs, as the system limitations do not allow processing below f_{min}. Also, due to the same constraint, f_{min} is the least possible frequency with which the adversary can process jobs. As such, this case can be shown to be equivalent to the non-DVS case (Section 7.3), where both the online algorithm and the adversary had to process jobs at the same (constant) speed. Thus, all the results of Section 7.3 apply.

As a consequence of Theorem 11, the upper bound on competitive factor approaches zero as $f_{min} \to 0$ and $\beta \to 1$.

7.6.1 Semi-online Algorithm $EC\text{-}DVS^*$

This subsection presents a semi-online algorithm $EC\text{-}DVS^*$ that uses the knowledge of both β and the maximum job size in the input instance. $EC\text{-}DVS^*$ is a variant of *EC-EDF** for DVS settings where $\beta > f_{min}$. First, a variant of *EC-EDF* called *EC-DVS* is described which will later be used by $EC\text{-}DVS^*$.

The *EC-DVS* algorithm uses an admission test similar to that of *EC-EDF* and admits a new job to the system if and only if there is enough remaining energy to completely execute both the new job and the remaining workload from all pending admitted jobs. All admitted jobs are executed in EDF order and at a constant speed β.

Observe that in executing a workload of W units, *EC-DVS* spends $\beta^{\alpha-1} \cdot W$ units of energy. Thus, formally, at time t, when the system has remaining energy budget of E^r units and remaining workload of R units from all pending admitted jobs, *EC-DVS* admits a newly arriving job $J_i(t, C_i, D_i, e_i)$ if and only if

$$E^r \geq \beta^{\alpha-1} \cdot (C_i + R)$$

Lemma 10. *If the largest job size in the input instance C_l is such that $C_l \leq \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$ then* EC-DVS *has a competitive factor of $\frac{1}{2}(\frac{f_{min}}{\beta})^{\alpha-1}$*

Proof. If *EC-DVS* rejects a job J_i with execution time C_i, then this implies that *EC-DVS* has depleted (or will eventually deplete upon completely executing the remaining workload of pending admitted jobs) at least $E' = E - \beta^{\alpha-1}C_i + \delta$ units of energy. In other words, *EC-DVS* has admitted a workload of at least $\frac{E'}{\beta^{\alpha-1}}$ units.

The following property of *EC-DVS* can be easily verified in the lines of Proposition 9: *EC-DVS* executes all admitted jobs to completion before their respective deadlines without violating the system's energy constraints. Thus, *EC-DVS* guarantees a value of at least $\frac{E'}{\beta^{\alpha-1}}$. Since $C_i \leq C_l \leq \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$, the following holds:

$$\frac{E'}{\beta^{\alpha-1}} = \frac{E - \beta^{\alpha-1}C_i + \delta}{\beta^{\alpha-1}} \geq \frac{E - \beta^{\alpha-1}C_l + \delta}{\beta^{\alpha-1}}$$

$$\frac{E'}{\beta^{\alpha-1}} \geq \frac{E - (\beta^{\alpha-1} \cdot \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})) + \delta}{\beta^{\alpha-1}}$$

$$\frac{E'}{\beta^{\alpha-1}} \geq \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$$

Since the optimal algorithm can make a value of at most $\frac{E}{f_{min}^{\alpha-1}}$, the competitive factor of *EC-DVS* is $\frac{1}{2}(\frac{f_{min}}{\beta})^{\alpha-1}$. □

The *EC-DVS** algorithm is based on *EC-DVS* and has the following rules. If the maximum job size in the input instance C_l is such that $C_l > \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$ then *EC-DVS** just waits for the job J_l with execution time C_l and executes it at speed $f = \frac{C_l}{D_l}$, where D_l is the deadline of J_l. Since the processor cannot run at a frequency lower than f_{min} and the minimum execution requirement of any job cannot exceed the system energy budget E, f is guaranteed to be in the range $[f_{min}, \beta]$. On the other hand, if $C_l \leq \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$ then *EC-DVS** follows the rules of *EC-DVS*.

Lemma 11. *EC-DVS** *has a competitive factor of* $\frac{1}{2}(\frac{f_{min}}{\beta})^{\alpha-1}$.

Proof. Let C_l denote the maximum job size in the input instance.

Case 1: If $C_l > \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$ then *EC-DVS** makes a value of at least C_l while the optimal online algorithm can make at most $\frac{E}{f_{min}^{\alpha-1}}$. Thus, the competitive factor of *EC-DVS** is $\frac{1}{2}(\frac{f_{min}}{\beta})^{\alpha-1}$.

Case 2: If $C_l \leq \frac{1}{2}(\frac{E}{\beta^{\alpha-1}})$ then *EC-DVS** follows *EC-DVS* and thus has a competitive factor of $\frac{1}{2}(\frac{f_{min}}{\beta})^{\alpha-1}$ (Lemma 10).

From Cases 1 and 2, *EC-DVS** has a competitive factor of $\frac{1}{2}(\frac{f_{min}}{\beta})^{\alpha-1}$. □

Lemma 12 below establishes the upper bound on the best achievable competitive factor by any semi-online algorithm that uses only the knowledge of β and the largest job size in the input instance.

Lemma 12. *No semi-online algorithm with the knowledge of β and the largest job size in the input instance can achieve a competitive factor greater than $(\frac{f_{min}}{\beta})^{\alpha-1}$.*

Proof. The adversary constructs an instance of back-to-back jobs (J_{i+1} released at the deadline of J_i) with the following parameters: $C = \delta$, $D = \frac{\delta}{f_{min}^{\alpha-1}}$ and $e = f_{min}^{\alpha-1} \cdot \delta$. This sequence of jobs ends either when the total workload released by the adversary reaches $\frac{E}{f_{min}^{\alpha-1}}$ or $\mathcal{A}$ executes one of these jobs at time $t < \frac{E}{f_{min}^{\alpha-1}}$. Both of these cases will be considered separately. Let C_l denote the maximum job size in the input instance.

- Case A: The total workload released by the adversary reaches $\frac{E}{f_{min}^{\alpha-1}}$. In this case, the adversary introduces job $J(\frac{E}{f_{min}^{\alpha-1}}, C_l, \frac{C_l}{\beta}, \beta^{\alpha-1}C_l)$. Observe that job J ensures the loading factor of the input instance created by the adversary is β. Since the online algorithm $\mathcal{A}$ has not executed any job (and hence not depleted any energy budget) until the release time of job J, it can execute job J and accrue a value no more than C_l. On the other hand, executing all jobs except job J, the adversary makes a total value of $\frac{E}{f_{min}^{\alpha-1}}$. Thus, the competitive factor is $f_{min}^{\alpha-1} \cdot \frac{C_l}{E}$. The minimum energy requirement of C_l must not exceed E, hence: $\beta^{\alpha-1}C_l \leq E$. Thus, setting $C_l = \frac{E}{\beta^{\alpha-1}}$ the competitive factor is bounded by $(\frac{f_{min}}{\beta})^{\alpha-1}$.

- Case B: $\mathcal{A}$ executes one of the jobs at time $t < \frac{E}{f_{min}^{\alpha-1}}$. In this case, $\mathcal{A}$ has depleted $f_{min}^{\alpha-1} \cdot \delta$ units of energy and accrued a value of δ. At this stage the adversary introduces job $J(t, C_l, \frac{C_l}{\beta}, E)$. Observe again that job J ensures the instance created by the adversary has a loading factor β. By executing J at frequency β the adversary makes a value of C_l. The competitive factor is $\frac{\delta}{C_l}$, where $C_l \leq \frac{E}{\beta^{\alpha-1}}$. Thus the competitive factor in this case can be arbitrarily small.

From Cases A and B above, no semi-online algorithm with the knowledge of β and the largest job size in the input instance can achieve a competitive factor greater than $(\frac{f_{min}}{\beta})^{\alpha-1}$. □

Corollary 8. *The competitive factor of EC-DVS^* is* 0.5 *times that of the optimal semi-online algorithm in these settings.*

7.7 Resource Augmentation

While the competitive analysis characterizes performance guarantees in the worst-case scenarios, some recent efforts exploited alternative means to quantify the performance of online algorithms. *Resource augmentation* technique, introduced by [100] and popularized by [72], is such a framework. With resource augmentation, the online algorithm is given additional resources compared to the adversary in an effort to compensate for the lack of knowledge about the future. For example, the online algorithm may run on a faster processor [72], or it may have access to additional CPUs [19]. The following analysis shows how resource augmentation can help significantly improve the performance of *EC-EDF*, especially when $\frac{e_{max}}{E}$ is close to one.

First, the implications of providing the online algorithm *EC-EDF* with additional energy is explored. Specifically, if the adversary possesses an energy budget of E units, then *EC-EDF* is assumed to have an initial energy of $(1 + x)E$ units, where $x > 0$. From Theorem 7 it is known that given an initial energy of E, *EC-EDF* guarantees a value of at least $E - e_{max}$. Thus, with $(1 + x)E$ units of initial energy, *EC-EDF* guarantees a value of at least $(1 + x)E - e_{max}$. The competitive factor is $1 + x - \frac{e_{max}}{E}$. Hence, if $x = \frac{e_{max}}{E}$ then *EC-EDF* has a competitive factor of 1.

Proposition 11. *The online algorithm* EC-EDF *achieves a competitive factor of* 1 *compared to an adversary with E units of energy budget, if it is allocated $(E + e_{max})$ units of energy.*

Further, $\frac{e_{max}}{E} \leq 1$, hence:

Corollary 9. *If* EC-EDF *is provided twice as much energy as the clairvoyant adversary* $\mathcal{C}_{adv}$*, it becomes at least as powerful as* $\mathcal{C}_{adv}$.

The following describes a practical way to effectively give more energy to *EC-EDF*. Specifically, the *EC-EDF* scheduler is augmented with the knowledge of the absolute loading factor β, and a DVS-capable processor. It will be shown that *EC-EDF* can successfully compete with a clairvoyant adversary without DVS feature. With this resource augmentation, *EC-EDF* always executes all jobs at speed $f = \beta$. This modified *EC-EDF* is referred to as *β-EC-EDF*. Observe that since the processor always runs at constant speed β, to deplete e units of energy, the processor must execute $\frac{e}{\beta^{\alpha-1}}$ units of workload. Thus, the initial energy budget of *β-EC-EDF* is effectively $\frac{1}{\beta^{\alpha-1}}$ times that of the adversary. Following Theorem 7, *β-EC-EDF* guarantees a value of $\frac{E}{\beta^{\alpha-1}} - e_{max}$.

Proposition 12. β-EC-EDF *has a competitive factor of* $\frac{E-\beta^{\alpha-1}e_{max}}{\beta^{\alpha-1}E}$.

The following is a consequence of Proposition 12 and the inequality $e_{max} \leq E$:

Corollary 10. *If* $\beta \leq (\frac{1}{2})^{\frac{1}{\alpha-1}}$, β-EC-EDF *is as powerful as a clairvoyant adversary without DVS.*

7.8 Chapter Summary

This chapter undertook a preliminary study of competitive analysis for energy-constrained online real-time scheduling in underloaded settings. An optimal algorithm *EC-EDF* was proposed which achieves the best possible performance guarantee obtainable by any online algorithm. Further, by assuming the knowledge of the largest job size, an optimal semi-online algorithm *EC-EDF** was proposed, which has a competitive factor of 0.5. Extending the analysis, fundamental results were provided for various models and settings including those of non-uniform value density and DVS.

Chapter 8: Conclusions

This chapter presents a summary of the main results obtained in this dissertation research, and then identifies a number of open problems for potential future work.

8.1 Summary of the Dissertation's Contributions

Energy management remains one of the fundamental design goals in real-time embedded systems. Dynamic Voltage Scaling (DVS) allows the processor supply voltage and clock frequency to be adjusted, thereby providing opportunities for reducing CPU dynamic energy. Dynamic Power Management (DPM) technique transitions device, when not in use, to low-power states and helps reduce the device energy. Both DVS and DPM techniques have been extensively applied for energy management in real-time systems. Recently there has been a growing interest in managing the system-wide energy, which mandates combining both DVS and DPM in a unified framework. Below, the specific contributions of this dissertation in this area are summarized.

8.1.1 Optimal Integration of DVS and DPM for a Frame-based Real-time Application

- A provably optimal algorithm OPT was developed to minimize the system-wide energy consumption. OPT is based on an exact characterization of the interplay between DVS and DPM.
- The dissertation extended the algorithm OPT to address dynamic variations in task execution times. The proposed algorithm OPT^* minimizes average-case system energy using the knowledge of average-case execution time. OPT^* guarantees the deadline of the application under worst-case execution workload.

8.1.2 System-level Energy Management of Periodic Real-time Tasks

- The RT-DPM problem for periodic tasks was shown to be NP-Hard in the strong sense, even in the absence of DVS. This result implies that for periodic real-time systems an exact and efficient solution to the system-level energy management problem is highly unlikely.

- The dissertation has developed two unified system-level energy management frameworks, *DFR-RMS* and *DFR-EDF*, that target the most widely deployed static- and dynamic-priority periodic real-time applications, respectively. A critical building block for these frameworks is the novel RT-DPM concept: *Device Forbidden Regions (DFR)*. With DFRs, long device idle intervals (called forbidden regions (FRs)) are periodically inserted to the schedule at run-time. Every FR is associated with a device, which is put to low-power mode during FR activations thereby reducing its energy. The proposed frameworks integrate DVS and DPM to minimize the system-wide energy. Through simulations the advantages of the DFR framework over other state-of-art schemes were shown.

8.1.3 Energy Management of Periodic Real-time Tasks on Chip-Multiprocessors

- One of the contributions of this dissertation is the introduction and characterization of global energy-efficient frequency, which varies over time with the number of active cores and the subset of tasks running in parallel upon them.

- Two schemes *CVFS* and *CVFS** were proposed for run-time dynamic energy management through coordinated DVS and core transitions among processing units. *CVFS** improves over *CVFS* by effectively exploiting slack from task early completions to save more energy.

- The dissertation also investigated the problem of determining the optimal subset of cores to execute the workload with low static power while preserving feasibility through an appropriate task allocation; and suggested three techniques (*SS, GLB, TLB*) for this purpose with varying effectiveness and run-time complexities.

- Through experimental evaluations the effectiveness of the proposed schemes in reducing system energy on CMP platforms was verified.

8.1.4 Competitive Analysis of Energy-Constrained Real-time Scheduling

- The dissertation carried out competitive analysis of energy-constrained real-time scheduling by investigating several models including online settings with no knowledge of future input, semi-online settings with partial (limited) knowledge of future input, uniform density settings where the value of a job is directly proportional to its execution time, non-uniform density settings where jobs accrue different value for unit time execution, and DVS-enabled systems.

- For the uniform value density settings, an optimal online algorithm *EC-EDF* was developed. Further, a constant competitive optimal semi-online algorithm $EC\text{-}EDF^*$ was also proposed.

- For the non-uniform value density settings, the dissertation derived an upper bound for the best achievable competitive factor by any online algorithm. Also, the competitive factor of $EC\text{-}EDF^*$ in these settings was investigated.

- The dissertation also analyzed various models including non-preemptive, non-idling and DVS settings, and derived a number of fundamental results.

8.2 Future Work

8.2.1 Energy-Efficient Real-Time Scheduling on Chip Multiprocessors

The focus of the work in this dissertation was the effective system-level energy management of a set of processing cores that share the same supply voltage and frequency (a single voltage island). In contrast, some recent processors consist of multiple voltage islands.

Investigating system-level energy management issues on such platforms is an open avenue. Also, investigating device power management and the problem of integrating DVS and DPM on CMP platforms are interesting open directions. Analyzing the potential of global scheduling and quantifying the impact of task migration on energy consumption are other open directions.

8.2.2 Joint Temperature and Power Management for Real-Time Tasks

In recent years, the exponential increase in power density of the processors has resulted in large energy consumption. As a consequence, processors have extensively been subject to overheating which has also reduced the system's reliability. Due to the exponential dependency of circuit reliability on operating temperature, it is predicted that over 50% of the electronic failures are temperature-related [131]. Given the interdependency between temperature and power, the high leakage power in modern multi-core architectures further aggravates the problem. For all of the above reasons, temperature management has become an important design goal.

Existing works on temperature management of hard real-time systems have typically focused on minimizing peak temperature [55,123]. In these and other existing works [14,115] the relationship between temperature and power is modeled using Fourier's Law of heat conduction. The thermal parameters are calculated by the RC thermal model.

An interesting and open problem to investigate is energy minimization subject to temperature constraints. Specifically, given a real-time scheduling policy $\mathcal{A}$, determine the scheduling frequency and voltage levels for task executions that minimize energy consumption while not violating both the thermal constraints and feasibility of the system.

Some recent CMP systems including AMD Opteron have the *Global Voltage Variable Frequency* feature. Here, though cores share the same supply voltage they can independently manage their own operating frequencies. While this will not significantly help in energy management (since voltage is not scaled), it may have an impact on temperature management. Investigating issues in temperature management for such platforms is an open problem.

8.2.3 Energy Constrained Weakly Hard Real-Time Scheduling on Multi-core Systems

Weakly hard real-time systems are motivated by the fact that for certain application domains, some distantly separated deadline misses are acceptable. Multimedia, real-time communication and embedded systems control are few examples of such applications. For these systems, the general (m, k)-firm deadline model has been proposed where the system degradation (or quality of service) is acceptable as long as every task misses no more than m deadlines in every k consecutive instances.

In the recent past, there has been an ever-growing demand for devices with superior quality of service and extended battery life. This has led to a proliferation in battery operated (hence energy constrained) devices. The soft real-time nature of the workload expected to run on these devices initially motivated the study of energy constrained weakly hard real-time uni-processor systems [5]. With the emerging multi-core platforms expected to dominate the embedded market, it would be worthwhile to re-visit the problem for chip-multi-core processors and propose efficient and effective heuristics.

Appendix A: Proof of Theorem 1

To show that the RT-DPM problem for periodic tasks is NP-Hard in the strong sense, the corresponding *decision* problem PRT-DPM is shown to be NP-Hard in the strong sense [56].

PRT-DPM: Given a periodic task set $\psi = \{T_1 \dots T_n\}$, where each task T_i uses a subset of devices in $\mathcal{D} = \{D_1 \dots D_m\}$, is there a feasible schedule that consumes at most X units of device energy in one hyperperiod ?

The given *PRT-DPM* problem is shown to be NP-Hard in the strong sense by reduction from the *3-Partition* problem. The *3-Partition* problem is defined below and known to be NP-Hard in the strong sense [56].

3-Partition: Given a set $\mathcal{S} = \{s_1 \dots s_{3m}\}$ of $3m$ positive integers, an integer B where $\frac{B}{4} < s_i < \frac{B}{2}$ $(i = 1 \dots 3m)$ and $\sum\limits_{i=1}^{3m} s_i = mB$, can $\mathcal{S}$ be partitioned into m disjoint subsets $\mathcal{S}_1 \dots \mathcal{S}_m$ such that the sum of elements in each subset is exactly B ?

Note that the *3-Partition* instance admits a "YES" answer if and only if there are exactly three elements in $\mathcal{S}_i$, $i = 1 \dots m$, such that $\sum\limits_{j|s_j \in \mathcal{S}_i} s_j = B$.

Given an instance of the *3-Partition* problem construct an instance of the *PRT-DPM* problem as follows:

- The system has a set of $3m+1$ periodic tasks $\psi = \{T_0 \dots T_{3m}\}$ with relative deadlines equal to periods and a set of $3m+1$ devices $\mathcal{D} = \{D_0 \dots D_{3m}\}$. The first instances of all tasks are released at time $t = 0$.
- Tasks $T_1 \dots T_{3m}$ are referred to as *normal* tasks (*N-tasks*). Each *N-task* T_i has $C_i = s_i$, $P_i = 3mB$ and uses a unique device D_i. The devices used by *N-tasks* are called *non-critical* devices.

- Task T_0 is referred to as *special* task (*S-task*). The *S-task* has $C_0 = B$, $P_0 = \frac{3}{2}B$ and using device D_0. D_0 is called the *critical* device.

- Device characteristics of *non-critical* device D_i $(i = 1 \dots 3m)$ are given as:

  - $P_a^i = 1, \quad P_s^i = 0, \quad E_{sw}^i = \frac{B}{2}, \quad T_{sw}^i = 3mB - s_i$
  - From the above one can deduce $B_{actual}^i = \frac{B}{2}$. Notice that $T_{sw}^i > B_{actual}^i$ as long as $m \geq 1$. Thus, $B_i = max(T_{sw}^i, B_{actual}^i) = 3mB - s_i$.

- Device characteristics of *critical* device D_0 are ginen as:

  - $P_a^0 = 3m + 1, \quad P_s^0 = 0, \quad E_{sw}^0 = \frac{B}{2}(3m + 1), \quad T_{sw}^0 = B$
  - From the above one can deduce $B_{actual}^0 = \frac{B}{2}$ and $B_0 = max(T_{sw}^0, B_{actual}^0) = B$.

- The value of X in the *PRT-DPM* instance is set to:

$$X = 2mB \cdot P_a^0 + m \cdot E_{sw}^0 + \sum_{i=1}^{3m}(s_i \cdot P_a^i + E_{sw}^i) = \frac{15}{2}m^2B + 5mB$$

A high level map to the proof is given.

- First, it is shown that a feasible schedule having device energy consumption of X units or less has necessarily a specific pattern where the *S-task* is scheduled at the beginning and end of every two consecutive execution periods (i.e. if this pattern is violated the schedule consumes more than X units of device energy).

- Then, it is further shown that a schedule can achieve device energy consumption of X units if and only if the *N-tasks* can be scheduled non-preemptively and in the same sequence in every hyperperiod.

- Having established that a feasible schedule with device energy consumption of at most X units is possible only with the specific execution pattern as described above, it is finally shown that the *PRT-DPM* instance admits a "YES" answer if and only if the corresponding *3-Partition* instance admits a "YES" answer.

Before proceeding some basic properties that are important to the proof are given. The hyperperiod of the given task set is $3mB$. The system utilization U is:

$$U = \frac{B}{\frac{3}{2}B} + \sum_{i=1}^{3m} \frac{s_i}{3mB} = \frac{2}{3} + \frac{mB}{3mB} = 1$$

Since $U = 1$, there exists a feasible schedule and the time-line is fully utilized in every hyperperiod (i.e. there is no time instant in any hyperperiod where the CPU is idle).

Since $T^i_{sw} > B^i_{actual}$, $P^i_s = 0$ and $B_i = T^i_{sw}$ for all D_i, $i = 0 \ldots 3m$, the following property follows.

Property 3. *For each device D_i, the energy consumed by keeping it* active *over an idle interval of length exactly B_i units exceeds the energy consumed in transitioning it back and forth over the same interval, i.e.* $B_i \cdot P^i_a > (E^i_{sw} + P^i_s(B_i - T^i_{sw}) = E^i_{sw})$

As a consequence of Property 3, transitioning the considered devices over idle intervals of length exactly equal to their corresponding break-even times is *beneficial* for energy savings of every device.

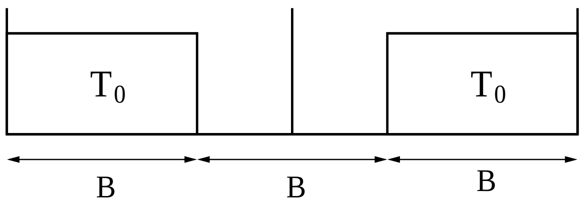


Figure A.1: Maximum Contiguous Idle Interval for Critical Device

Due to the periodic nature of tasks, the maximum contiguous idle interval a device can have is bounded by twice the *laxity* of the tasks using the device in any feasible schedule. Figure A.1 shows that the maximum length of idle interval the *critical device* D_0 can have is equal to B units. This maximum idle interval is obtained by scheduling the *S-task* at two far extremes of consecutive hyperperiods.

Property 4. *All device idle intervals for the critical device D_0 are of length B units or less.*

Consider a schedule where the *S-task* is scheduled at the beginning of every odd-numbered period and at the end of every even-numbered period as shown in Figure A.2. A schedule satisfying this property is called a *$\mathcal{Z}$-schedule.* Notice that since the periods of *N-tasks* is equal to the hyperperiod $3mB$, and preemption is allowed, there always exists a feasible $\mathcal{Z}$-schedule.

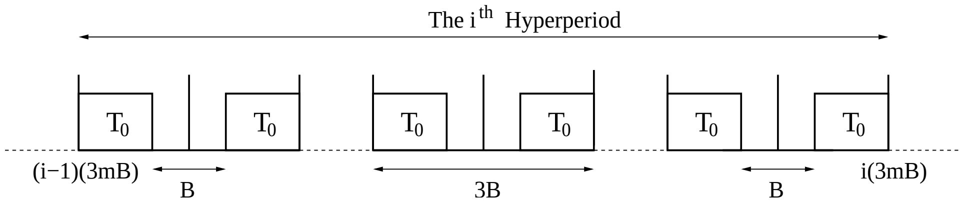


Figure A.2: $\mathcal{Z}$-Schedule

Property 5. *In a $\mathcal{Z}$-schedule, the* critical device *D_0 has exactly m idle intervals, each of length B.*

Without loss of generality energy consumption during the i^{th} hyperperiod is considered. The total device energy consumption of a schedule S during the i^{th} hyperperiod is expressed as:

$$E_d(S) = E_c(S) + E'_c(S) + E_{nc}(S) + E'_{nc}(S)$$

Above, $E_c(S)$ denotes the energy consumed by *critical* device D_0 when *active* and in use. $E'_c(S)$ denotes the aggregate energy consumption of D_0 over *all* of its *idle* periods during one hyperperiod (including transition costs when appropriate).

Similarly, $E_{nc}(S)$ represents the energy consumed by *all non-critical* devices when *active* and in use. Finally, $E'_{nc}(S)$ indicates the aggregate energy consumption of *all non-critical* devices over *all* of their respective *idle* periods during one hyperperiod (including transition costs when appropriate).

Lemma 13. *If a feasible schedule does not satisfy the $\mathcal{Z}$-schedule property then its total device energy consumption is greater than X units.*

Proof. As a consequence of Property 4, the *critical* device D_0 can have at most m device idle intervals of length B in one hyperperiod. Let S' be a schedule having $0 \le k < m$ device idle intervals of length B for D_0. By definition, S' is not a $\mathcal{Z}$-schedule.

Each hyperperiod has $2m$ instances of *S-task* and hence D_0 is *active* and in use for $2mB$ units. Thus, $E_c(S') = 2mB \cdot P_a^0$.

This gives a total idle period of $3mB - 2mB = mB$ units for D_0 during the hyperperiod. Since S' has k idle intervals of length B, D_0 is energy-efficiently transitioned k times, each of these transitions consume E_{sw}^0 units of energy and once transitioned, D_0 consumes no energy during the idle interval ($P_s^i = 0$). Finally, over device idle intervals that are smaller than $B_0 = B$, D_0 is kept in *active* state. Thus, $E_c'(S') = (m-k)B \cdot P_a^0 + k \cdot E_{sw}^0$. Each *non-critical* device D_i is *active* and in use for s_i units in one hyperperiod. Thus, $E_{nc}(S') = \sum_{i=1}^{3m} s_i \cdot P_a^i$.

The total device energy consumption of schedule S' during the i^{th} hyperperiod is:

$$E_d(S') = E_c(S') + E_c'(S') + E_{nc}(S') + E_{nc}'(S')$$

Note that $E_{nc}'(S') \ge 0$. Assume $E_d(S') \le X$. It will be shown by contradiction that this is impossible. Since, $E_d(S') \le X$,

$$E_d(S') \le 2mB \cdot P_a^0 + m \cdot E_{sw}^0 + \sum_{i=1}^{3m} (s_i \cdot P_a^i + E_{sw}^i)$$

$$(m-k)B \cdot P_a^0 + k \cdot E_{sw}^0 + E'_{nc}(S') \leq m \cdot E_{sw}^0 + \sum_{i=1}^{3m} E_{sw}^i$$

Note that for every device in the PRT-DPM instance, $E_{sw}^i = B_{actual}^i \cdot P_a^i$. By substituting this value,

$$(m-k)(B - B_{actual}^0) \cdot P_a^0 + E'_{nc}(S') \leq \sum_{i=1}^{3m} (B_{actual}^i \cdot P_a^i)$$

$$(m-k)\frac{B}{2} \cdot P_a^0 + E'_{nc}(S') \leq \frac{B}{2} \cdot 3m$$

This inequality cannot hold because even if the left hand side was minimized by setting $k = m-1$ which is its maximum value and $E'_{nc}(S') = 0$ respectively, $P_a^0 = 3m+1 \leq 3m$, giving a contradiction. $\square$

As a consequence of Lemma 13, any schedule having device energy consumption of X units or less must be a $\mathcal{Z}$-schedule. The energy consumed by *all* devices in a schedule S satisfying the $\mathcal{Z}$-schedule property during the i^{th} hyperperiod can be expressed as:

$$E_d(S) = 2mB \cdot P_a^0 + m \cdot E_{sw}^0 + \sum_{i=1}^{3m} s_i \cdot P_a^i + E'_{nc}(S)$$

Notice that $E_d(S) \leq X$ if and only if $E'_{nc}(S) \leq \sum_{i=1}^{3m} E_{sw}^i$. Lemma 14 below characterizes how to schedule *N-tasks* in a schedule S satisfying the $\mathcal{Z}$-schedule property to enforce that $E'_{nc}(S) \leq \sum_{i=1}^{3m} E_{sw}^i$ hold.

Lemma 14. *In a $\mathcal{Z}$-schedule, during the i^{th} hyperperiod, $E'_{nc} \leq \sum_{i=1}^{3m} E_{sw}^i$ if and only if* N-tasks *can be scheduled non-preemptively and exactly in the same sequence as in hyperperiods $(i-1)$ and $(i+1)$.*

Proof. One can classify the device idle intervals of *non-critical* devices as *enclosed intervals* and *extended intervals.* An *enclosed interval* is a device idle interval that starts and ends within the same hyperperiod. On the other hand, an *extended interval* is a device idle interval that starts in one hyperperiod and ends in the consecutive hyperperiod. Figure A.3 shows *enclosed* and *extended* intervals for a device used by task T_k. As a consequence of Lemma 13, *S-task* is scheduled at the beginning and end of every hyperperiod in a $\mathcal{Z}$-schedule. Thus, in a $\mathcal{Z}$-schedule

(1) Every *non-critical* device has exactly one *extended interval* for every two consecutive hyperperiods and the length of this *extended interval* is no less than $2B$ (Figure A.3).

(2) For a *non-critical* device D_i, the maximum length of an *enclosed interval* is $3mB - s_i - 2B < B_i = 3mB - s_i$ (Figure A.3). Thus, the *non-critical* devices cannot be transitioned energy-efficiently over any of their respective *enclosed intervals.* Observe that a *non-critical* device will have an *enclosed interval* if and only if the corresponding *N-task* using it is executed in a preemptive manner within one hyperperiod.

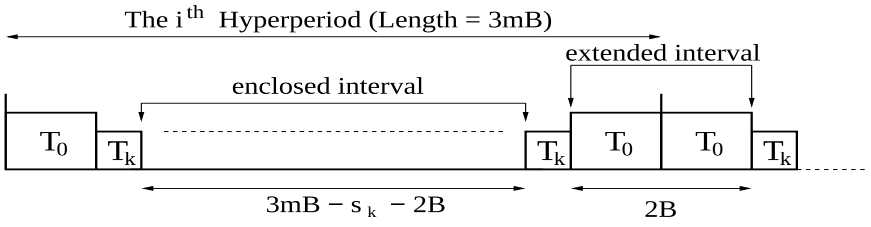


Figure A.3: Extended and Enclosed Intervals of N-tasks

By proving the implications in both directions Lemma 14 will be established.

Claim A: In a $\mathcal{Z}$-schedule, during the i^{th} hyperperiod, if *N-tasks* are scheduled non-preemptively and exactly in the same sequence as in hyperperiods $(i-1)$ and $(i+1)$ then $E'_{nc} \leq \sum_{i=1}^{3m} E^i_{sw}$.

Consider a $\mathcal{Z}$-schedule S^* where, during the i^{th} hyperperiod, *N-tasks* are scheduled non-preemptively and in the exact same sequence as in hyperperiods $(i-1)$ and $(i+1)$.

As a consequence of non-preemptive execution, the schedule S^* has no *enclosed intervals*

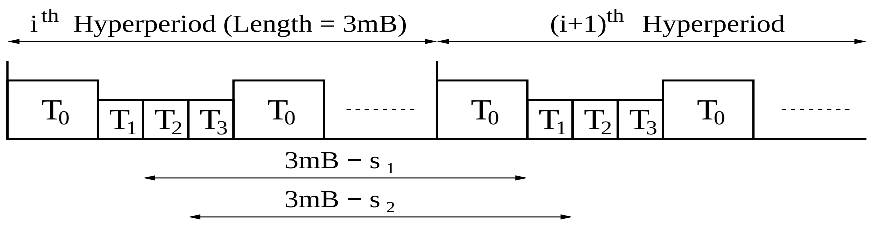


Figure A.4: Scheduling to Minimize Device Energy of the PRT-DPM Instance

over the i^{th} hyperperiod. Further, notice that in a $\mathcal{Z}$-schedule, when *N-tasks* are executed in the same sequence over consecutive hyperperiods a and b, the lengths of *extended intervals* across a and b are exactly equal to their respective device break-even times (Figure A.4).

Since tasks are executed in the same sequence in hyperperiods $(i-1)$ and i, each *non-critical* device D_i is energy-efficiently transitioned after the completion of task T_i in $(i-1)^{th}$ hyperperiod. Further, D_i returns to *active* state only at the *exact* start time of T_i in i^{th} hyperperiod. As a consequence, *non-critical* devices consume no *active* energy in the i^{th} hyperperiod until their first usage (i.e. start time of T_i in i^{th} hyperperiod).

Also, tasks are executed in the same sequence in hyperperiods i and $(i+1)$. Thus, each *non-critical* device D_i is energy-efficiently transitioned after completion of T_i in i^{th} hyperperiod. Transition energy costs for device transitions in the i^{th} hyperperiod are accounted in $E'_{nc}(S^*)$. Thus, $E'_{nc}(S^*)$ is expressed as, $E'_{nc}(S^*) = \sum_{i=1}^{3m} E^i_{sw}$.

Claim B: In a $\mathcal{Z}$-schedule, during the i^{th} hyperperiod, if $E'_{nc} \leq \sum_{i=1}^{3m} E^i_{sw}$ then *N-tasks* are scheduled non-preemptively and in the exact same sequence as in hyperperiods $(i-1)$ and $(i+1)$.

First observe that since system utilization $U = 1$ and $B_i = 3mB - s_i$ $(i = 1 \ldots 3m)$, if all *N-tasks* are *not* executed in the same sequence during consecutive hyperperiods a and b then there exists at least one *non-critical* device D_k whose *extended interval* length $\mathcal{I}_k$ across a and b is such that $2B \leq \mathcal{I}_k < B_k$.

The claim will be proved considering 3 cases and reaching a contradiction for each.

Case 1: Assume $E'_{nc} \leq \sum_{i=1}^{3m} E_{sw}^{i}$ and *N-tasks* are scheduled:

(1) Non-preemptively

(2) Not in the same sequence during hyperperiods i and $(i+1)$

Let S_{nr} be a schedule satisfying the above conditions (Case 1). In S_{nr}, there exist $k \geq 1$ *non-critical* device(s) that could not be energy-efficiently transitioned after their respective last usage in i^{th} hyperperiod. Without loss of generality let these k devices have indices $1 \ldots k$.

For device D_j $(j = 1 \ldots k)$, let $[t_1^j, t_2^j]$ denote the *extended interval* across hyperperiods i and $(i+1)$. Let $\mathcal{I}_j^1$ denote the length of interval $[t_1^j, iH]$ which represents portion of $[t_1^j, t_2^j]$ confined in i^{th} hyperperiod. As a consequence of Lemma 13, $\mathcal{I}_j^1 \geq B$.

Notice:

$$E'_{nc}(S_{nr}) \geq \sum_{j=1}^{k} \mathcal{I}_j^1 \cdot P_a^j + \sum_{i=k+1}^{3m} E_{sw}^i = \sum_{j=1}^{k} (\mathcal{I}_j^1 - B_{actual}^j) \cdot P_a^j + \sum_{i=1}^{3m} E_{sw}^i$$

$B_{actual}^j = \frac{B}{2}$ and $\mathcal{I}_j^1 \geq B$ implies $\mathcal{I}_j^1 - B_{actual}^j > 0$ $(j = 1 \ldots k)$. Therefore, $E'_{nc}(S_{nr}) > \sum_{i=1}^{3m} E_{sw}^i$ which is a contradiction.

Case 2: Assume $E'_{nc} \leq \sum_{i=1}^{3m} E_{sw}^{i}$ and *N-tasks* are scheduled:

(1) Non-preemptively

(2) In the same sequence during hyperperiods i and $(i+1)$

(3) Not in the same sequence during hyperperiods $(i-1)$ and i

Let S_{nl} be a schedule satisfying the above conditions (Case 2). There exists in S_{nl}, $k \geq 1$ *non-critical* device(s) that could not be energy-efficiently transitioned after their respective last usage in $(i-1)^{th}$ hyperperiod. As a result, these k devices will consume $\beta > 0$ units of energy from the start of i^{th} hyperperiod to the point of their respective first usage in it. Since all tasks in hyperperiods i and $(i+1)$ are executed in the same sequence, *non-critical* devices are energy efficiently transitioned after their last usage in hyperperiod i. Thus, $E'_{nc}(S_{nl}) = \beta + \sum_{i=1}^{3m} E^i_{sw} > \sum_{i=1}^{3m} E^i_{sw}$ which is a contradiction.

Case 3: Assume $E'_{nc} \leq \sum_{i=1}^{3m} E^i_{sw}$ and *N-tasks* are scheduled preemptively in the i^{th} hyperperiod

Let S_p be a schedule where one or more *N-tasks* are preempted in the i^{th} hyperperiod. S_p contains at least one *enclosed interval* for some *non-critical* device D_k. Let $\alpha > 0$ be the energy consumed in keeping *non-critical* device(s) *active* over their respective *enclosed* intervals.

Further, since $U = 1$ and $B_i = 3mB - s_i$ $(i = 1 \dots 3m)$, irrespective of the order of execution across hyperperiods,
(i) There exists in S_p, $k \geq 1$ *non-critical* device(s) that could not be energy-efficiently transitioned after their respective last usage in i^{th} hyperperiod. Similar to Case 1, one needs to account for the energy consumed by these device(s) after their respective last usage in i^{th} hyperperiod. Without loss of generality let these k devices have indices $1 \dots k$. Also, as described in Case 1, for device D_j $(j = 1 \dots k)$, $\mathcal{I}^1_j$ denotes the length of the extended interval confined in i^{th} hyperperiod.
(ii) Further, there also exists in S_p, $l \geq 1$ *non-critical* device(s) that could not be energy-efficiently transitioned after their respective last usage in $(i-1)^{th}$ hyperperiod. Similar to Case 2, let $\beta > 0$ account for the energy consumed by these device(s) from the start of i^{th} hyperperiod to the point of their respective first usage in it.

Combining all the above and simplifying,

$$E'_{nc}(S_p) = \alpha + \beta + \sum_{j=1}^{k}(\mathcal{I}_j^1 - B_{actual}^j) \cdot P_a^j + \sum_{i=1}^{3m} E_{sw}^i$$

$\alpha > 0$, $\beta > 0$ and $\mathcal{I}_j^1 - B_{actual}^j > 0$ $(j = 1 \dots k)$ implies, $E'_{nc}(S_p) > \sum_{i=1}^{3m} E_{sw}^i$ which is a contradiction. $\square$

Corollary 11. *A $\mathcal{Z}$-schedule where* N-tasks *are scheduled in a non-preemptive manner and in the same sequence during every hyperperiod has a device energy consumption of X units in one hyperperiod.*

Observe that the execution time of *N-tasks*, $C_i = s_i$ $(i = 1 \dots 3m)$ are such that $\frac{B}{4} < C_i < \frac{B}{2}$. This follows directly from the size constraint in the *3-Partition* problem. As such, a feasible $\mathcal{Z}$-schedule where the *N-tasks* are scheduled in a non-preemptive manner must contain m contiguous execution sequences for the *N-tasks*. Each of these execution sequences are of length B and delimited by *S-task* executions. Further, there are *exactly* 3 *N-tasks* in each of these execution sequences.

Property 6. *A non-preemptive feasible $\mathcal{Z}$-schedule for the PRT-DPM instance can exist only with exactly three* N-tasks *executed in every device idle interval of the* critical *device.*

Lemma 15. *The* PRT-DPM *instance admits a "YES" answer if and only if the corresponding* 3-Partition *instance admits a "YES" answer.*

Proof. Recall from Property 5 that a $\mathcal{Z}$-schedule has m device idle intervals, $\mathcal{I}_1 \dots \mathcal{I}_m$, each of length B.

If the *PRT-DPM* instance admits a "YES" answer, then Corollary 11 and Lemma 14 imply that this is achievable only by the special $\mathcal{Z}$-schedule with non-preemptive execution pattern for *N-tasks*. From Property 6, in this pattern, there must be exactly three *N-tasks* executed in each $\mathcal{I}_i$, $i = 1 \dots m$. Since each of these device idle intervals are exactly of length B units: $\sum_{T_j \in \mathcal{I}_i} s_j = B$, $i = 1 \dots m$. Thus, a solution to the *3-Partition* instance can be constructed by assigning each set of three *N-tasks* executed in $\mathcal{I}_i$ to $\mathcal{S}_i$, $i = 1 \dots m$, and the *3-Partition* instance admits a "YES" answer.

Conversely, if the *3-Partition* instance admits an "YES" answer, then there exists m disjoint sets $\mathcal{S}_1 \dots \mathcal{S}_m$ such that $\sum_{j \in \mathcal{S}_i} s_j = B$. Further, each $\mathcal{S}_i$ contains exactly three elements. Thus, by scheduling the tasks that correspond to the elements of $\mathcal{S}_i$ over interval $\mathcal{I}_i$, $i = 1 \dots m$, a non-preemptive feasible $\mathcal{Z}$-schedule for the PRT-DPM instance can be constructed. From Corollary 11, such a schedule consumes exactly X units of device energy in one hyperperiod and the *PRT-DPM* instance admits a "YES" answer. $\square$

Finally, note that the reduction described can be achieved in polynomial-time. Hence the theorem follows.

Appendix B: Proof of Theorem 3

First the problem is formulated as a decision problem.

DFR-FSB: Given a periodic task set $\psi = \{T_1 \dots T_n\}$ and a set of forbidden regions $\phi = \{(\Delta_1, \Pi_1) \dots (\Delta_m, \Pi_m)\}$, can all job instances of ψ meet their respective deadlines under preemptive EDF ?

DFR-FSB is shown to be co-NP-Hard in the strong sense by reduction from the *3-partition* problem which is known to be NP-Hard in the strong sense [56].

3-Partition: Given a set $\mathcal{S} = \{s_1 \dots s_{3m}\}$ of $3m$ positive integers, an integer B where $\frac{B}{4} < s_i < \frac{B}{2}$ $(i = 1, \dots, 3m)$ and $\sum_{i=1}^{3m} s_i = mB$, can $\mathcal{S}$ be partitioned into m disjoint subsets $\mathcal{S}_1, \dots, \mathcal{S}_m$ such that the sum of elements in each subset is exactly B ?

Note that the *3-Partition* instance admits a "YES" answer if and only if there are exactly three elements in $\mathcal{S}_i$, $i = 1 \dots m$, such that $\sum_{j | s_j \in \mathcal{S}_i} s_j = B$.

Given an instance of the *3-partition* construct the following instance of the *DFR-FSB* problem.

- There are $3m+1$ devices in the system $(\{D_0, D_1 \dots D_{3m}\})$. All devices are associated with forbidden region parameters. Device D_0 is associated with FR_0 with $\Delta_0 = 2B+1$ and $\Pi_0 = (m+2)(B+1)$. Other devices $D_{i \neq 0}$ are associated with FR_i having $\Delta_i = s_i$ and $\Pi_i = (m+2)(B+1)$.

- There are two periodic tasks, $\psi = \{T_1, T_2\}$. T_1 has worst-case execution time of 1 unit and period $B+1$ units. T_2 has worst-case execution time of 1 unit and period $(m+2)(B+1)$ units. T_1 uses no system device, while T_2 uses all $3m+1$ devices $\{D_0, D_1, \dots, D_{3m}\}$.

ψ is a harmonic task set with hyperperiod $H = (m + 2)(B + 1)$ and the total utilization U given by:

$$U = \frac{1}{B+1} + \frac{1}{(m+2)(B+1)} \leq 1$$

First, note that all $(m + 2)$ instances of T_1 meet their respective deadlines within the hyperperiod. The first $(m + 1)$ instances of T_1 have higher priority compared to the single instance of T_2 and they are not subject to delay by any FR interference. The last $((m+2)^{th})$ instance of T_1 has the same priority as that of T_2 and can be delayed by at most $C_2 = 1$ time unit. Even with this interference from T_2, as it is not delayed by any FR activation, the last instance of T_1 is guaranteed a timely completion.

Remark 6. *All instances of T_1 meet their respective deadlines under any FR activation pattern and preemptive EDF scheduling.*

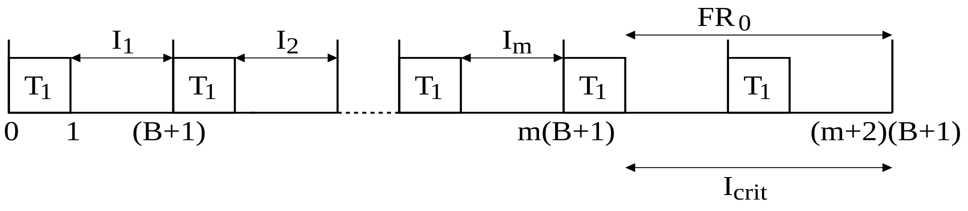


Figure B.1: EDF Schedule with worst-case FR enforcement for T_2

Now, focusing on the execution of T_2, which is affected by all $|\phi| = 3m + 1$ forbidden regions, an arguement is made that T_2 is definitely schedulable if FR_0 is *not* activated after the completion of the $(m + 1)^{th}$ instance of T_1 and until the end of hyperperiod (i.e. in the interval $I_{crit} = [(m(B + 1) + 1, (m + 2)(B + 1)]$, as shown in Figure B.1). In other words, if FR_0 is not activated in the interval I_{crit}, then there always exists a feasible EDF schedule.

To see this, assume FR_0 is activated at some other interval $[t_1, t_2]$ in the hyperperiod. In this case, observe that the activation of FR_0 will overlap with *at least* one of the first $(m + 1)$ instances of T_1, each of which has higher priority with respect to T_2. As a result

T_2, cannot be delayed by more than:

$$\Delta_0 + m \cdot C_1 + \sum_{i=1}^{3m} \Delta_i = 2B + 1 + m + mB = (m+1)(B+1) - 1$$

time units and T_2 will meet its deadline with its $C_2 = 1$ unit execution time requirement.

Remark 7. *In the worst-case FR activation pattern for T_2, FR_0 should be activated in interval I_{crit}*

If FR_0 is activated in the interval I_{crit}, then T_2 can only be executed in one the following m intervals (where the $(m+1)$ high priority instances of T_1 are not executing, as shown in Figure B.1):

$$I = \{I_1 = [1, B+1], \ldots, I_m = [(m-1)(B+1)+1, m(B+1)]\}$$

The total length of intervals in I is mB and deciding on the schedulability of T_2 depends on whether there exists a FR activation pattern for $FR_1 \ldots FR_m$ (whose FR durations sum up to mB), such that the intervals in I can be continuously and fully covered.

In the worst-case interference pattern for T_2, let I_{fr} denote the total interference of $FR_1 \ldots FR_m$ during the intervals $I = \{I_1 \ldots I_m\}$. Given that $I_{fr} \leq \sum_{i=1}^{3m} \Delta_i = mB$, the following properties can be easily verified:

Remark 8. *T_2 misses its deadline if and only if $I_{fr} = mB$.*

Remark 9. *$I_{fr} = mB$ if and only if the following three facts hold:*

(1) Each $FR_i \in (\phi - FR_0)$ is enforced once during the interval $[0, (m+2)(B+1)]$.

(2) No two enforcements of $FR_1 \ldots FR_m$ overlap.

(3) No enforcements of $FR_1 \ldots FR_m$ overlaps with the execution of T_1

It is now show that a *DFR-FSB* instance admits a "NO" answer if and only if the corresponding *3-partition* instance admits a "YES" answer, completing the proof.

Claim A: If the *DFR-FSB* instance admits a "NO" answer then the corresponding *3-partition* instance admits a "YES" answer.

If the *DFR-FSB* instance admits a "NO" answer then T_2 must have missed its deadline (Remark 6). From Remarks 7, 8 and 9 if T_2 misses its deadline then for each $I_k \in I$, there exists a set of forbidden regions $F_k \subset (\phi - FR_0)$ such that $\sum\limits_{i \in F_k} \Delta_i = B$. Further, at run-time the forbidden regions in F_k are enforced back-to-back to form a single contiguous unit of length B that prevents execution of T_2 over I_k. Note that by definition, each FR_i is enforced at run-time as a *contiguous* unit of length Δ_i (i.e. a single enforcement of an FR cannot be split into multiple parts). Also, each FR_i can interfere with the execution of T_2 at most once during the interval $[0, (m+2)(B+1)]$ (since all FRs have a period of $(m+2)(B+1)$). Thus:

$$F_1 \cap F_2 \cap \ldots \cap F_m = \emptyset$$

Further, since $I_{fr} = mB$ when T_2 misses its deadline (Remark 8):

$$F_1 \cup F_2 \cup \ldots \cup F_m = \phi - FR_0$$

Also note that $\frac{B}{4} < \Delta_i = s_i < \frac{B}{2}$. Thus, if $\sum\limits_{i \in F_k} \Delta_i = B$, then F_k contains exactly three FRs from the set $\phi - FR_0$. A solution to the *3-partition* instance can be constructed by assigning each set of three FRs in F_k to S_k $(k = 1, \ldots m)$.

Claim B: If the *3-partition* instance admits a "YES" answer then the corresponding *DFR-FSB* instance admits a "NO" answer.

If the *3-partition* instance admits a "YES" answer then there exists $\{S_1 \ldots S_m\}$ such that $\sum\limits_{i \in S_i} s_i = B$ $(i = 1, \ldots, m)$. Thus, by enforcing the FRs corresponding to elements of S_i in interval I_i back-to-back $(i = 1 \ldots m)$, one can create a total interference of $I_{fr} = mB$ thus forcing T_2 to miss its deadline.

Appendix C: Proof of Theorem 4

The theorem is proved by contradiction. Assume there is a deadline miss and the following holds:

$$\forall k_{k=1\ldots n} \sum_{i \in \Upsilon_k} \left(\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}\right) + \sum_{j=1}^{k} \frac{C_j}{P_j} \leq 1$$

Let t_d be the first time a job misses its deadline in the EDF schedule. Let t_s be the last time before t_d such that there are no pending job execution requests with arrival times before t_s and deadlines at or before t_d. t_s is well defined as no requests can arrive before $t = 0$.

At time t, where $t_s \leq t \leq t_d$, let $\mathcal{A}(t)$ denote the set of ready jobs with pending execution requirements whose arrival times are in interval $[t_s, t]$ and deadlines in interval $[t_s, t_d]$. By choice of t_s, $\mathcal{A}(t)$ is a non-empty set throughout the interval $[t_s, t_d]$. Hence, there is always a pending job with deadline no greater than t_d in this interval.

Let $t_d - t_s = X$. Let k be the largest index satisfying the condition $P_k \leq X$. At any time t, $t_s \leq t \leq t_d$, the job instances in $\mathcal{A}(t)$ belong to a subset of tasks in $\{T_1 \ldots T_k\}$ (since tasks are ordered according to periods). Υ_k denotes the set of forbidden regions affecting one or more tasks in $\{T_1 \ldots T_k\}$.

As a result, at any time t in the interval $[t_s, t_d]$ either a job in $\mathcal{A}(t)$ should be executing *or all* the jobs in $\mathcal{A}(t)$ should be prevented from execution (blocked) by one or more FRs in Υ_k. Therefore, the entire interval $[t_s, t_d]$ can be seen as a sequence of intervals where in each interval either a job in $\mathcal{A}(t)$ runs *or* all jobs in $\mathcal{A}(t)$ are blocked by one or more FRs in Υ_k.

Let α_1 denote total length of interval in $[t_s, t_d]$ where *all* jobs in $\mathcal{A}(t)$ are blocked due to FR enforcements. α_1 is bounded by the total length of FRs in Υ_k when they are activated with their minimum separation times (Π values) in $[t_s, t_d]$. Thus, $\alpha_1 \leq \sum\limits_{i \in \Upsilon_k} \lceil \frac{X}{\Pi_i} \rceil \cdot \Delta_i$.

Let α_2 denote the length of intervals in $[t_s, t_d]$ where a job in $\mathcal{A}(t)$ is executed. α_2 is bounded by the total execution time of jobs in $\mathcal{A}(t)$, $t_s \leq t \leq t_d$. Thus, $\alpha_2 \leq \sum_{j=1}^{k} \lfloor \frac{X}{P_j} \rfloor \cdot C_j$.

Since a job misses its deadline at time t_d, $\alpha_1 + \alpha_2 > X$

$$\sum_{i \in \Upsilon_k} \lceil \frac{X}{\Pi_i} \rceil \cdot \Delta_i + \sum_{j=1}^{k} \lfloor \frac{X}{P_j} \rfloor \cdot C_j > X$$

Since $\lceil Y \rceil \leq \lfloor Y \rfloor + 1$ and $\lfloor Y \rfloor \leq Y$, $\sum_{i \in \Upsilon_k} (\frac{X}{\Pi_i} + 1) \cdot \Delta_i + \sum_{j=1}^{k} \frac{X}{P_j} \cdot C_j > X$

$$\sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{X}) + \sum_{j=1}^{k} \frac{C_j}{P_j} > 1$$

By choice of k, $P_1 \leq P_2 \ldots \leq P_k \leq X$. Thus, $\sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{P_j} > 1$, giving a contradiction and proving the theorem.

Appendix D: Proof of Theorem 5

The theorem is proved by contradiction. Assume there is a deadline miss and the following holds:

$$\forall k_{k=1\dots n} \frac{b_k}{P_k} + \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{P_j} \leq 1$$

Let t_d be the first time a job misses its deadline in the EDF schedule. Let t_s be the last time before t_d such that there are no pending job execution requests with arrival times before t_s and deadlines at or before t_d. t_s is well defined as no requests can arrive before $t = 0$.

At time t, where $t_s \leq t \leq t_d$, let $\mathcal{A}(t)$ denote the set of ready jobs with pending execution requirements whose arrival times are in interval $[t_s, t]$ and deadlines in interval $[t_s, t_d]$. By choice of t_s, $\mathcal{A}(t)$ is a non-empty set throughout the interval $[t_s, t_d]$. Hence, there is always a pending job with deadline no greater than t_d in this interval.

Let $t_d - t_s = X$. Let k be the largest index satisfying the condition $P_k \leq X$. At any time t, $t_s \leq t \leq t_d$, the job instances in $\mathcal{A}(t)$ belong to a subset of tasks in $\{T_1 \dots T_k\}$ (since tasks are ordered according to periods). Υ_k denotes the set of forbidden regions affecting one or more tasks in $\{T_1 \dots T_k\}$.

As a result, at any time t in the interval $[t_s, t_d]$ either a job in $\mathcal{A}(t)$ should be executing *or all* the jobs in $\mathcal{A}(t)$ should be prevented from execution (blocked) by either FRs in Υ_k or execution of jobs not in $\mathcal{A}(t)$. Below it is first shown that there can be at most one job not in $\mathcal{A}(t)$ that can execute in the interval $[t_s, t_d]$.

Proposition 13. *There can be at most one job not in $\mathcal{A}(t)$ that can execute in the interval $[t_s, t_d]$.*

Proof. First observe that a job not in $\mathcal{A}(t)$ will only execute in the interval $[t_s, t_d]$ if it is holding a non-preemptable resource that is required by some job in $\mathcal{A}(t)$. The above claim is proved by contradiction. Assume there were more than one such jobs. Then there must be at least two such jobs J_a and J_b with one of them having preempted the other. Without loss of generality, assume J_a preempts J_b. Since J_b began execution before J_a and is blocking some J_i in $\mathcal{A}(t)$, at the time J_a preempts J_b, the system ceiling is at least $pl(J_i)$. Since J_a is able to preempt J_b, $pl(J_a)$ must be greater than the current system ceiling. Thus, $pl(J_a) > pl(J_i)$. However, by definition the relative deadline of J_i is smaller than that of J_a leading to a contradiction. □

The entire interval $[t_s, t_d]$ can be seen as a sequence of intervals where in each interval either

- A job not in $\mathcal{A}(t)$ runs.
- All jobs in $\mathcal{A}(t)$ are blocked by one or more FRs in Υ_k.
- A job in $\mathcal{A}(t)$ runs.

Let α_1 denote the total length of interval in $[t_s, t_d]$ where jobs not in $\mathcal{A}(t)$ are executed. From Proposition 13, it is know that there can be at most one job J_l not in $\mathcal{A}(t)$ that can be executed in $[t_s, t_d]$. Thus, α_1 is bounded by the longest time J_l causes resource blocking uses the shared resource(s). This in turn is bounded by b_i for every J_i in $\mathcal{A}(t)$. In particular,

$$\alpha_1 \leq b_k$$

Let α_2 denote total length of interval in $[t_s, t_d]$ where *all* jobs in $\mathcal{A}(t)$ are blocked due to FR enforcements. α_2 is bounded by the total length of FRs in Υ_k when they are activated with their minimum separation times (Π values) in $[t_s, t_d]$. Thus,

$$\alpha_2 \leq \sum_{i \in \Upsilon_k} \lceil \frac{X}{\Pi_i} \rceil \cdot \Delta_i$$

Let α_3 denote the length of intervals in $[t_s, t_d]$ where a job in $\mathcal{A}(t)$ is executed. α_3 is bounded by the total execution time of jobs in $\mathcal{A}(t)$, $t_s \leq t \leq t_d$. Thus,

$$\alpha_3 \leq \sum_{j=1}^{k} \lfloor \frac{X}{P_j} \rfloor \cdot C_j$$

Since a job misses its deadline at time t_d,

$$\alpha_1 + \alpha_2 + \alpha_3 > X$$

$$b_k + \sum_{i \in \Upsilon_k} \lceil \frac{X}{\Pi_i} \rceil \cdot \Delta_i + \sum_{j=1}^{k} \lfloor \frac{X}{P_j} \rfloor \cdot C_j > X$$

Since $\lceil Y \rceil \leq \lfloor Y \rfloor + 1$ and $\lfloor Y \rfloor \leq Y$,

$$b_k + \sum_{i \in \Upsilon_k} (\frac{X}{\Pi_i} + 1) \cdot \Delta_i + \sum_{j=1}^{k} \frac{X}{P_j} \cdot C_j > X$$

$$\frac{b_k}{X} + \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{X}) + \sum_{j=1}^{k} \frac{C_j}{P_j} > 1$$

By choice of k, $P_1 \leq P_2 \ldots \leq P_k \leq X$. Thus,

$$\frac{b_k}{P_k} + \sum_{i \in \Upsilon_k} (\frac{\Delta_i}{\Pi_i} + \frac{\Delta_i}{P_k}) + \sum_{j=1}^{k} \frac{C_j}{P_j} > 1$$

Giving a contradiction and proving the theorem.

Appendix E: Proof of Theorem 6

To prove Theorem 6, create an input sequence ψ such that for any given positive small value δ and for any online algorithm $\mathcal{A}$, $\mathcal{A}$ accrues a value no more than $E - e_{max} + \delta$ and the adversary gains a total value of E. Thus, the competitive factor of $\mathcal{A}$ will be shown to be no better than $\frac{E-e_{max}}{E}$. Recall that E and e_i for any job J_i are exact multiples of δ. This implies e_{max}, the upper bound on the size of any job, is also an exact multiple of δ.

Let $E = k_1 \cdot \delta$ and $e_{max} = k_2 \cdot \delta$. Where, k_1 and k_2 are integers such that $k_1 \geq k_2$, $k_1 \geq 1$ and $k_2 \geq 1$. In the following, feed algorithm $\mathcal{A}$ the input sequence ψ such that there exists a time t, where $\mathcal{A}$ obtains a total value of no more than $E - e_{max} + \delta$. Further, $\mathcal{A}$ will be unable to accrue any additional value after time t.

Start with time $t = 0$. The adversary introduces $(k_1 - k_2 + 1)$ *sequential jobs* with size δ (recall sequential jobs have zero laxity and are released back to back one at a time). At time $t_1 = (k_1 - k_2 + 1) \cdot \delta$, let $m \geq 0$ denote the number of jobs that are *not executed* by $\mathcal{A}$. There are the following three cases.

- Case 1: $m = 0$. In this case $\mathcal{A}$ has executed to completion a total workload of $W = (k_1 - k_2 + 1) \cdot \delta$, accruing a value of W units while also depleting W units of the intial energy budget E. At time t_1, the adversary introduces a single job with size $e_{max} = k_2 \cdot \delta$. With only $(k_2 - 1) \cdot \delta$ units of energy left, $\mathcal{A}$ cannot execute this job. On the other hand, the adversary by executing $(k_1 - k_2)$ jobs of size δ and the job with size e_{max} accrues a value of $(k_1 - k_2) \cdot \delta + e_{max}$. Thus, the competitive factor is given by:

$$\frac{(k_1 - k_2 + 1) \cdot \delta}{(k_1 - k_2) \cdot \delta + e_{max}} = \frac{E - e_{max} + \delta}{E}$$

- Case 2: $m \geq k_2$. In this case $\mathcal{A}$ has skipped execution of at least k_2 jobs. Thus, at time t_1, the value accrued by $\mathcal{A}$ is no more than $(k_1 - 2k_2 + 1) \cdot \delta$ and its remaining energy budget is no more than $(2k_2 - 1) \cdot \delta$. At t_1, the adversary introduces a single job of size e_{max}. By executing this job, $\mathcal{A}$ can increase its value to at most $(k_1 - 2k_2 + 1) \cdot \delta + e_{max}$. Similar to Case 1 above, the adversary makes a value of E by executing $(k_1 - k_2)$ jobs of size δ along with the job of size e_{max}. The competitive factor is given by:

$$\frac{(k_1 - 2k_2 + 1) \cdot \delta + e_{max}}{E} = \frac{E - e_{max} + \delta}{E}$$

- Case 3: $m < k_2$. In this case first observe that in the time interval $[0, t_1]$, $\mathcal{A}$ accrues a value of $W_1 = (k_1 - k_2 + 1 - m) \cdot \delta$. Starting from time t_1, the adversary incrementally releases *sequential jobs* of size δ until one of the following two conditions hold: (1) k_2 jobs have been released *and* $\mathcal{A}$ has executed $0 \leq m' < m$ of these jobs. (2) $\mathcal{A}$ executes m of these jobs. There are the following two sub-cases.

  - Case 3A: k_2 jobs of size δ have been released *and* $\mathcal{A}$ has executed $0 \leq m' < m$ of these jobs. Thus, in the time interval $(t_1, (k_1 + 1) \cdot \delta]$, $\mathcal{A}$ accrues a value of $W_2 = m' \cdot \delta$. The adversary can accrue a total value of E by executing $(k_1 - k_2)$ jobs released in interval $[0, t_1]$ and k_2 jobs released interval $[t_1, (k_1 + 1) \cdot \delta]$. The competitive factor is given by:

$$\frac{W_1 + W_2}{E} = \frac{(k_1 - k_2 + 1 + (m' - m)) \cdot \delta}{E}$$

    Since $m' < m$,

$$\frac{(k_1 - k_2 + 1 + (m' - m)) \cdot \delta}{E} < \frac{(k_1 - k_2 + 1) \cdot \delta}{E} = \frac{E - e_{max} + \delta}{E}$$

  - Case 3B: $\mathcal{A}$ executes m of these jobs. Let $t_2 \leq (k_1 + 1) \cdot \delta$ be the time when $\mathcal{A}$

completes executing m jobs. At time t_2, $\mathcal{A}$ has accrued a value of

$$W = W_1 + m \cdot \delta = (k_1 - k_2 + 1) \cdot \delta = E - e_{max} + \delta$$

Also, at time t_2 the remaining energy budget with $\mathcal{A}$ is $(k_2 - 1) \cdot \delta$ units. The adversary at t_2 releases a single job with size e_{max} which $\mathcal{A}$ cannot execute. The adversary by executing $(k_1 - k_2)$ jobs released in interval $[0, t_1)$ and the job with size e_{max} released at t_2 accrues a total value of E. Thus, the competitive factor is given by $\frac{W}{E} = \frac{E - e_{max} + \delta}{E}$.

From Cases 1, 2 and 3, no online algorithm can achieve a competitive factor greater than $\frac{E - e_{max} + \delta}{E}$. Since $\frac{\delta}{E}$ can be arbitrarily low, the upper bound on the achievable competitive factor is given by $\frac{E - e_{max}}{E}$.

Appendix F: Proof of Theorem 8

In order to prove Theorem 8, an input instance is provided for which the ratio of the total value accrued by any online algorithm $\mathcal{A}$ to that of a clairvoyant adversary cannot be more than $\frac{1}{(k_{max})^{\frac{e_{max}}{E}}}$.

This instance consists of a series of *periods* $P_1, P_2, \ldots, P_m \quad m \geq 1$. The exact number of periods (m), as well as the exact number of jobs released in each period, depend on the actions of the algorithm $\mathcal{A}$. The strategy will consist in showing that, **in each period**, the ratio of the value accrued by $\mathcal{A}$ to that of the adversary cannot be greater than $\frac{1}{(k_{max})^{\frac{e_{max}}{E}}}$. This, in turn, will establish the upper bound over all the periods, that is, the competitive factor of the entire input instance.

Let the remaining energy of $\mathcal{A}$ at the beginning of the period P_i be E_i. Clearly, $E_1 = E \geq e_{max}$. All the jobs released within a period are released back-to-back and with laxity zero; that is, at the deadline of a job, a new job is released. First the structure of the first period P_1 is described and then show how it can be generalized to multiple periods.

In the first period, first a job with value density k_{min} and size $X \leq e_{max}$ is released by the adversary. The adversary keeps releasing such jobs with value density k_{min} and size X back to back (i.e. the following job is released at the deadline of the previous one) until one of the following conditions occurs:

a. Either the online algorithm $\mathcal{A}$ does not accept *any* of these jobs until the total energy requirement (the total size) of the released jobs in period P_1 reaches E_1.

   In this case, the adversary stops releasing any new jobs. No more periods are introduced and this marks the end of the input instance as well. While $\mathcal{A}$ obtained a value of zero in this period, the adversary announces that it has accrued a non-zero value by executing all $\lfloor \frac{E_1}{X} \rfloor$ jobs that it has released; yielding the total value ratio zero in this (last) period.

b. Or, the online algorithm $\mathcal{A}$ accepts one of these jobs at some point before their total energy requirement does not exceed E_1.

At this point, the adversary releases another job of size X but with the value density $k \cdot k_{min}$ at the deadline of the first job accepted by $\mathcal{A}$. If $\mathcal{A}$ executes this second job as well, the adversary releases a third job of size X, and value density $k^2 \cdot k_{max}$ at the deadline of the second job and so on. This pattern continues, i.e. the adversary keeps releasing jobs where the value density of each job is equal to k times that of the previous one, until one of the following conditions is satisfied: either the remaining energy of the player becomes strictly less than e_{max}, or, $\mathcal{A}$ rejects executing a job even though its remaining energy is greater than or equal to e_{max}. Each of these cases is examined in detail below.

b1. After executing $s+1$ jobs with value densities $k_{min}, k \cdot k_{min}, \ldots, k^s \cdot k_{min}$, the remaining energy of $\mathcal{A}$ becomes strictly smaller than e_{max}.

In this case, as the last job of this period (and as the last job of the entire input instance), the adversary releases a job of size e_{max}, zero laxity and value density $k^{s+1} \cdot k_{min}$. Observe that $\mathcal{A}$ cannot execute this job due to the energy deficiency. However, the adversary announces that it has skipped all the jobs with value density k_{min}, but executed the s jobs with value densities $k \cdot k_{min}, \ldots, k^s \cdot k_{min}$ and the last one with value density $k^{s+1} \cdot k_{min}$ in this period[1].

Thus, the ratio of the values of accrued by $\mathcal{A}$ and the adversary in this last period is given by:

[1]Observe that, in this case, $E_1 = (s+1) \cdot X + Y$ where $s \geq 0$ and $0 \leq Y < e_{max}$. Recalling that $E_1 \geq e_{max}$ and $X \leq e_{max}$, one can infer that $E_1 \geq (X+Y) \geq e_{max}$; because if this was not true, there would not be a non-negative integer s for which $E_1 = (s+1) \cdot X + Y$ holds. So, one can re-write E_1 as $s \cdot X + (X+Y)$ where $(X+Y) \geq e_{max}$; that is, the adversary has sufficient energy to execute the s jobs with size X, along with the last one of size e_{max}.

$$\begin{aligned} c &= \frac{(1+k+k^2+\ldots+k^s)\cdot X \cdot k_{min}}{(k+k^2+\ldots+k^s)\cdot X\cdot k_{min} + (k^{s+1}k_{min})\cdot e_{max}} \\ &\leq \frac{(1+k+k^2+\ldots+k^s)\cdot X \cdot k_{min}}{(k+k^2+\ldots+k^s)\cdot X\cdot k_{min} + (k^{s+1}k_{min})\cdot X} \\ &\leq \frac{1}{k}. \end{aligned}$$

b2. Or, after executing $s+1$ jobs with value densities $k_{min}, k\cdot k_{min}, \ldots, k^s \cdot k_{min}$, $\mathcal{A}$ rejects executing the job with value density $k^{s+1}\cdot k_{min}$ even though its remaining energy is greater than or equal to e_{max}.

In this case, in this period $\mathcal{A}$ accrues a total value of $k_{min}\cdot X \cdot \sum_{i=0}^{s} k^i$. Then, the adversary announces that it has skipped all the jobs it has released with value density k_{min}; instead, it has executed the $(s+1)$ jobs with value densities $k\cdot k_{min}, \ldots, k^s \cdot k_{min}, k^{s+1}\cdot k_{min}$, making a total value of $k_{min}\cdot X\cdot \sum_{i=1}^{s+1} k^i$. Hence, the ratio of the total value of $\mathcal{A}$ to that of the adversary in this period is:

$$c = \frac{k_{min}\cdot X\cdot \sum_{i=0}^{s} k^i}{k_{min}\cdot X\cdot \sum_{i=1}^{s+1} k^i} = \frac{1}{k} \tag{F.1}$$

At this point, this period ends and a new period with the same job release pattern as that in the previous one is started. Observe that, up to this point, the adversary and $\mathcal{A}$ have executed exactly the same number of jobs with the same size; hence, at the beginning of the next period, their remaining energy levels are identical. Further, by the very nature of the condition that must be satisfied at the beginning of the case (b2.), this energy level is definitely greater than or equal to e_{max}.

Thus, the adversary starts a new period by releasing jobs of size X and value density k_{min} back to back, until $\mathcal{A}$ picks up one of these and the whole analysis above can be repeated to establish that the value ratio at the end of the second, third, and all subsequent periods cannot exceed $\frac{1}{k}$. Obviously, this sequence of periods will end after a finite number of steps, either when $\mathcal{A}$ does not pick up any job in the sequence of back-to-back released jobs with value density k_{min} (in case (a.)), or by the triggering condition of the case (b1.) above (as the remaining energy monotonically decreases whenever $\mathcal{A}$ executes a job). By considering the fact that the value ratio cannot exceed $\frac{1}{k}$ in any of the periods, one can show that the competitive factor is indeed bounded by $\frac{1}{k}$.

What needs to be examined now is how large k can be. Let P_j be the period during which the algorithm $\mathcal{A}$ executes the maximum number of jobs. The last job released by the adversary in this period P_j has value density k^{i+1}. The value density of this job is fixed to k_{max}. Thus, $k^{i+1} = k_{max}$.Observe that $\mathcal{A}$ executed exactly $i+1$ jobs in this period P_j.Note that the number of jobs executed by $\mathcal{A}$ in the period P_j puts a constraint on $i+1$; that is, $(i+1) \cdot X \leq E$. That is, $(i+1)$ can be, at most, equal to $\frac{E}{X}$. As a result one obtains, $k_{max} = k^{i+1} = k^{\frac{E}{X}}$; or equivalently, $k = (k_{max})^{\frac{X}{E}}$. To maximize k, choose the maximum possible value for X, which is equal to e_{max}. In that case, the competitive factor c is bounded by at most $\frac{1}{k} = \frac{1}{(k_{max})^{\frac{e_{max}}{E}}}$, completing the proof.

Bibliography

Bibliography

[1] J. Anderson, V. Bud, and U.C. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems*, 2005.

[2] B. Andersson and K. Bletsas. Sporadic Multiprocessor Scheduling with Few Preemptions. In *Proc. of the Euromicro Conference on Real-Time Systems*, 2008.

[3] B. Andersson, K. Bletsas, and S. Baruah. Scheduling Arbitrary-Deadline Sporadic Task Systems Multiprocessors. In *Proc. of the Real-Time Systems Symposium*, 2008.

[4] T.A. AlEnawy and H. Aydin. Energy-Aware Task Allocation for Rate Monotonic Scheduling. In *Proc. of Real Time and Embedded Technology and Applications Symposium,* 2005.

[5] T.A. AlEnawy and H. Aydin. Energy-Constrained Scheduling for Weakly-Hard Real-Time Systems. In *Proc. of the Real-time Systems Symposium*, 2005.

[6] T.A. AlEnawy and H. Aydin. On Energy-Constrained Real-Time Scheduling. In *Proc. of the European Conference on Real-Time Systems*, 2004.

[7] B. Andersson, S. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proc. of the Real-Time Systems Symposium, December,* 2001.

[8] N. Audsley, A. Burns, K. Tindell, M. Richardson and A. Weilings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling, In *Software Engineering Journal,* vol. 8, no. 5, pp. 284-295, 1993.

[9] H. Aydin, R. Melhem, D. Mosse and P.M. Alvarez. Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics. In *Proc. of the EuroMicro Conference on Real-Time Systems,* 2001.

[10] H. Aydin, V. Devadas, and D. Zhu. System-Level Energy Management for Periodic Real-Time Tasks. In *Proc. of the Real-Time Systems Symposium*, 2006.

[11] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-Aware Scheduling for Periodic Real-Time Tasks. *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 584-600, 2004.

[12] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *Proc. of International Parallel and Distributed Processing Symposium,* 2003.

[13] T.P. Baker. A Stack-Based Resource Allocation Policy for Realtime Processes. In *Proc. of the Real-Time Systems Symposium,* 1990.

[14] N. Bansal and K. Pruhs. Speed Scaling to Manage Temperature. In *Proc. of the Symposium on Theoretical Aspects of Computer Science*, 2005.

[15] N. Bansal, T. Kimbrel, and K.Pruhs. Dynamic Speed scaling to manage energy and temperature. In *Proc. of the Symposium on Foundations of Computer Science*, 2004.

[16] S. Baruah, L. Rosier, and R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on one Processor. In *Real Time Systems(2)*, vol. 2, no. 4, pp. 301-324, 1990.

[17] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. Online Scheduling in the presence of Overload. In *Proc. of the Symposium on Foundations of Computer Science*, 1991.

[18] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the Competitiveness of Online Real-Time Task Scheduling. In *Proc. of the Real-Time Systems Symposium*, 1991.

[19] S. Baruah. Overload Tolerance for Single-Processor Workloads. In *Proc. of the Real-Time Technology and Application Symposium*, 1998.

[20] S. Baruah and J. Haritsa. Scheduling for Overload in Real-Time Systems. *IEEE Transactions on Computers*, vol. 46, no. 9, pp. 1034-1039, 1997.

[21] S. Baruah and M. E. Hickey. Competitive Online Scheduling of Imprecise Computations. *IEEE Transactions on Computers*, vol. 47, no. 9, pp. 460-468, 1998.

[22] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, J. Duato. A Simple Power-Aware Scheduling for Multicore Systems when running Real-time Applications. In *Proc. of International Parallel and Distributed Processing Symposium,* 2008.

[23] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on VLSI Systems*, vol. 8, no. 3, pp. 299-316, 2000.

[24] E. Bini, G.C. Buttazzo and G. Lipari. Speed Modulation in Energy-Aware Real-Time Systems. In *Proc. of the Euromicro Conference on Real-Time Systems*, 2005.

[25] E. Bini, G.C. Buttazzo and G.M. Buttazzo. Rate Monotonic Analysis: The Hyperbolic Bound. In *IEEE Transactions on Computers,* vol. 52, no. 7, pp. 933-942, 2003.

[26] S. Borkar. Thousand core chips: A Technology Perspective. In *Proc. of the Design Automation Conference (DAC),* 2007.

[27] A. Borodin and R. El-Yavin *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[28] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. New strategies for Assigning Real-Time Tasks to Multiprocessor Systems. In *IEEE Transactions on Computers,* vol. 44, no. 12, pp. 1429-1442, 1995.

[29] G. Buttazzo, G. Lipari, and L. Abeni. Elastic Task Model for Adaptive Rate Control. In *Proc. of the Real-Time Systems Symposium,* 1998.

[30] G. Buttazzo and L. Abeni. Adaptive Workload Management through Elastic Scheduling. In *Journal of Real-Time Systems*, vol. 23, no. 1-2, pp. 7-24, 2002.

[31] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Second Edition, Springer, 2005.

[32] G.C. Buttazzo. Rate Monotonic vs. EDF: Judgment Day. In *Journal of Real-Time Systems,* vol. 29, no. 1, pp. 5-26, 2004.

[33] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Joseph Y-T Leung (ed). Chapman Hall/ CRC Press*, 2004.

[34] H.L. Chan, W.T. Chan, T.W. Lam, L.K. Lee, K.S. Mak, and P. Wong. Energy Efficient Online Deadline Scheduling. In *Proc. of the Symposium on Discrete Algorithms*, 2007.

[35] J.J. Chen and L. Thiele. Expected System Energy Consumption Minimization in Leakage-Aware DVS systems. In *Proc. of International Symposium on Low Power Electronics and Design (ISPLED)*, 2008.

[36] J.J. Chen and T.W. Kuo. Voltage Scaling Scheduling for Periodic Real-Time Tasks in Reward Maximization. In *Proc. of the Real-Time System Symposium*, 2005.

[37] J-J. Chen, C-Y. Yang, H-I. Lu, and T-W. Kuo. Approximation Algorithms for Multiprocessor Energy-Efficient Scheduling of Periodic Real-Time Tasks with Uncertain Task Execution Time. In *Proc.of the Real-Time and Embedded Technology and Applications Symposium,* 2008.

[38] H. Cheng and S. Goddard. Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation. In *Proc. of the International Workshop on Power-Aware Real-Time Computing*, 2005.

[39] H. Cheng and S. Goddard. EEDS-NR: An online Energy-Efficient I/O Device Scheduling Algorithm for Hard Real Time Systems with Non-Preemptive Resources.In *Proc. of the Euromicro Conference on Real Time Systems*, 2006.

[40] H. Cheng and S. Goddard. Online Energy-Aware I/O Device Scheduling for Hard Real-Time Systems. In *Proc. of Design Automation and Test in Europe*, 2006.

[41] H. Cheng and S. Goddard. SYS-EDF: A System-wide Energy-efficient Scheduling Algorithm for Hard Real Time Systems. In *the International Journal of Embedded Systems on Low Power Real-Time Embedded Computing,* vol. 4, no. 4, pp.45-56, 2006.

[42] K. Choi, R. Soma and M. Pedram. Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-Chip Access to On-Chip Computation Times. In *Proc. of Design, Automation and Test in Europe*, 2004.

[43] J.Y. Chung, J.W.S. Liu and K.J. Lin. Scheduling Periodic Jobs that allow imprecise results. In *IEEE Transactions on Computers*, vol. 19, no. 9, pp. 1156-1173, 1990.

[44] Z. Deng, J.W.S. Liu and J. Sun. A Scheme for Scheduling Hard Real-Time Applications in Open System Environment. In *Proc. of the Euromicro Workshop on Real-Time Systems*, 1997.

[45] M. Dertouzos. Control Robotics: The Procedural Control of Physical Processes. In Articial Intelligence and Control Applications, IFIP Congress, 1974.

[46] M. Dertouzos and A.K. Mok. Multiprocessor Online Scheduling for Hard Real-Time Tasks. *IEEE Transactions on Software Engineering*, vol 15, no. 12, pp. 1497-1506, 1989.

[47] J.K. Dey, J. Kurose. D. Towsley, C.M. Krishna and M. Girkar. Efficient On-line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. In *Proc. of SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* 1993.

[48] J. Dorsey *et. al.* An integrated Quad-Core Opteron Processor. In *Proc. of IEEE Intl. Solid State Circuits Conference*, 2007.

[49] T. Ebenlendr and J. Sgall. Semi Online Preemptive Scheduling: One Algorithm for All Variants. In *Proc. International Symposium on Theoretical Aspects of Computer Science*, 2009.

[50] R. Ernst and W. Ye. Embedded Program Timing Analysis based on Path Clustering and Architecture Classication. In *Proc. of the International Conference on Computer-Aided Design,* 1997.

[51] W. Feng and J.W.S. Liu. An Extended Imprecise Computation Model for Time-Constrained Speech Processing and Generation. In *Proc. of the Workshop on Real-Time Applications*, 1993.

[52] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Efficient Server Clusters. In *Workshop on Power Aware Computing Systems*, 2002.

[53] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[54] W. Feng and J.W.S. Liu. Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines. In *Proc. of the IEEE Transactions on Software Engineering,* vol. 23, no. 2, pp. 93-106, 1997.

[55] N. Fisher, J.J. Chen, S. Wang, and L. Thiele. Thermal-Aware Global Real-Time Scheduling on Multicore Systems. In *Proc. of the Real-Time Technology and Applications Symposium (RTAS)*, 2009.

[56] M.R. Garey and D.S. Jhonson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, 1979.

[57] J. Goossens, S. Baruah and S. Funk. Real-time Scheduling on Multiprocessors. In *Proc. of the International Conference on Real-Time Systems,* 2002.

[58] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. SIAM Journal on Applied Mathematics, vol. 17, no. 2. pp. 416-429, 1969.

[59] F. Gruian. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *Proc. of the 2004 International Symposium on Low Power Electronics and Design (ISLPED)*, 2001.

[60] M. Hamdaoui and P. Ramanathan. A Dynamic Priority Assignment Technique for Streams with (m, k)-firm Deadlines. In *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 1443-1451, 1995.

[61] P.J.M. Havinga and G.J.M. Smith. Design Techniques for Low-Power Systems. *Journal of Systems Architecture*, vol. 46, no. 1, 2000.

[62] S. Herbert and D. Marculescu. Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors. In *Proc. of the Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2007.

[63] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. Srivastava. Power Optimization of Variable Voltage Core-based Systems. In *Proc. of the Design Automation Conference,* 1998.

[64] I. Hong, M. Potkonjak and M. B. Srivastava. On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor. In *Proc. of the International Conference on Computer-Aided Design,* 1998.

[65] S. Irani, S. Shukla and R. Gupta. Algorithms for Power Savings. In *Proc. the Symposium on Discrete Algorithms*, 2003

[66] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proc. of the IEEE/ACM Intl. Symp. on Microarchitecture (MICRO),* 2006.

[67] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. of the International Symposium on Low Power Electronics and Design,* 1998.

[68] K. Iwama and S. Taketomi. Removable Online Knapsack Problems. In *Proc. of the International Colloquium on Automata, Languages and Programming*, 2002.

[69] K. Jeffay, D. F. Stanat, and C. U Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Proc. of the Real-Time Systems Symposium*, 1991.

[70] K. Jeffay and D.L. Stone. Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems. In *Proc. of the Real-Time Systems Symposium*, 1993.

[71] R. Jejurikar and R. Gupta. Dynamic Voltage Scaling for System-Wide Energy Minimization in Real-Time Embedded Systems. In *Proc. of the International Symposium on Low Power Electronics and Design*, 2004.

[72] B. Kalyanasundaram and K. Pruhs. Speed is as Powerful as Clairvoyance. In *Proc. of the Symposium on Foundations of Computer Science*, 1995.

[73] H. Kim, H. Hong, H-S. Kim, J-H Ahn and S. Kang. Total Energy Minimization of Real-Time Tasks in an On-Chip Multiprocessor Using Dynamic Voltage Scaling Efficiency Metric. In *IEEE Transactions of Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 11, pp. 2088-2092, 2008.

[74] W. Kim, D. Shin, H.S. Yun, J. Kim, S.L. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002.

[75] G. Koren and D. Shasha. *D*-over: An Optimal Online Scheduling Algorithm for Overloaded Real-Time Systems. In *Proc. of the Real-Time Systems Symposium*, 1992.

[76] G. Koren, D. Shasha and S. C. Huang. MOCA: A Multiprocessor Online Competitive Algorithm for Real-Time Scheduling. In *Proc. of the Real-Time Systems Symposium*, 1993.

[77] R. Kumar and G. Hinton. A Family of 45nm IA Processors. In *Proc. of the Intl. Solid-State Circuits Conference*, 2009.

[78] S. Lauzac, R. Melhem and D. Mosse. An Efcient RMS Admission Control and its Application to Multiprocessor Scheduling. In *Proc. of the International Parallel Processing Symposium,* 1998.

[79] C.H. Lee and K.G. Shin. On-Line Dynamic Voltage Scaling for Hard Real-Time Systems Using the EDF Algorithm. In *Proc. of the Real-Time Systems Symposium*, 2004.

[80] J. Lehoczky, L. Sha and J.K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proc. of the Real-Time Systems Symposium,* 1987.

[81] J. Lehoczky, L. Sha and Y. Ding. The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour. In *Proc. of the Real-Time Systems Symposium,* 1989.

[82] L-F. Leung, C-Y. Tsui and X.S. Hu. Exploiting Dynamic Workload Variation in Low Energy Preemptive Task Scheduling. In *Proc. of Design Automation and Test in Europe (DATE)*, 2005.

[83] J. Liu. Real Time Systems Prentice Hall, NJ, 2000.

[84] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In *Journal of the Association for Computing Machinery,* vol 20, no. 1, pp. 46-61, 1973.

[85] D.C. Locke, D. Vogel and T. Mesler. Building a Predictable Avionics Platform in Ada: A Case Study. In *Proc. of the Real-Time Systems Symposium,* 1991.

[86] J. Lopez, J. Diaz, M. Garcia and D. Garcia. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In *Proc. of the Euromicro Workshop on Real-Time Systems,* 2000.

[87] J. Lorch and A. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *ACM SIGMETRICS,* 2001.

[88] Y. Lu, L. Benini, and G. D. Micheli. Power Aware Operating Systems for Interactive Systems. In *IEEE Transactions on VLSI Systems,* vol. 10, no. 2, pp. 119-134, 2002.

[89] D. Luenberger.*Linear and Nonlinear Programming.* Addison-Wesley, Reading Massachusetts, 1984.

[90] H.-Y. McCreary, M. A. Broyles, M. S. Floyd, A. J. Geissler, S. P. Hartman, F. L. Rawson, T. J. Rosedahl, J. C. Rubio, M. S. Ware. EnergyScale for IBM POWER6 microprocessor-based systems. In *IBM Journal of Research and Development*, vol 21, no. 6, pp. 775-786, 2007.

[91] R. McGowen, C.A. Poirier, C. Bostak, J. Ignowski, M. Millican, W.H. Parks, and S. Naffziger. Power and Temperature Control on a 90-nm Itanium family processor. In *Journal of Solid-State Circuits,* vol. 41, no. 1, pp. 229-237, 2006.

[92] A.K. Mok. Fundamental Design Problems of Distributed Systems for Hard Real Time Environments, PhD Thesis, Laboratory for Computer Science, MIT, 1983.

[93] A.K. Mok and M.L. Dertouzos. Multiprocessor Scheduling in a Hard Real-Time Environment. In *Proc. Texas Conference Computing Systems,* 1978.

[94] L. Mosley. Power Delivery Challenges for Multicore Processors. In *Proc. of CARTS USA*, 2008.

[95] A. Naveh *et al.*. Power and Thermal Management in the Intel Core Duo Processor. *Intel Technology Journal*, Vol. 10, no. 2, May 2006.

[96] L. Niu. Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee. In *Proc. of the Conference on Embedded and Real-Time Computing Systems and Applications (RTSCA'10)*, 2010.

[97] D. Oh and T. P. Baker. Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. In *Journal of Real-Time Systems,* vol. 15, no. 2, pp. 183-192, 1998.

[98] M.A. Palis. Competitive Algorithms for Fine-Grain Real-Time Scheduling. In *Proc. of the Real-Time Systems Symposium*, 2004.

[99] M. Pedram. Power Minimization in IC Design: Principles and Applications. *ACM Transactions on Design Automation of Electronics Systems*, vol. 1, no. 1, pp. 3-56, 1996.

[100] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal Time-Critical Scheduling via Resource Augmentation. In *Proc. of the ACM Symposium on Theory of computing*, 1997.

[101] P. Pillai and K.G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the Symposium on Operating Systems Principles*, 2001.

[102] K. Pruhs, J. Sgall, and E. Torng. In *the Handbook of Scheduling, Algorithms, Models and Performance Analysis.* CRC press, FL, USA, 2004, edited by J.Y.T. Leung.

[103] A. Qadi, S. Goddard, and S. Farritor. A Dynamic Voltage Scaling Algorithm for Sporadic Tasks. In *Proc. of the Real-Time Systems Symposium*, 2003.

[104] G. Quan and X. Hu. Enhanced Fixed-Priority Scheduling with (m, k)-firm guarantee. In *Proc. of the Real-Time Systems Symposium*, 2000.

[105] X. Qi and D. Zhu. Power Management for Real-Time Embedded Systems on Block-Partitioned Multicore Platforms. In *Proc. of the Intl. Conf. on Embedded Software and Systems (ICESS),* 2008.

[106] P. Ramanathan. Overload Management in Real-Time Control Applications using (m, k)-firm guarantee. In *IEEE Transactions on Parallel and Distributed Systems,* vol. 10, no. 6, pp. 549-559, 1999.

[107] C. Rusu, R. Melhem, and D. Mosse. Maximizing Rewards for Real-Time Applications with Energy Constraints. *ACM Transactions for Embedded Computing Systems*, vol. 2, no. 4, pp. 537-559, 2003.

[108] C. Rusu, R. Melhem, and D. Mosse. Maximizing the System Value while Satisfying Time and Energy Constraints. In *Proc. of the Real-Time System Symposium*, 2002.

[109] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority Real-Time Systems. In *Proc. of the Real-Time and Embedded Technology and Applications Symposium*, 2003.

[110] E. Seo, J. Jeong, S. Park, and J. Lee. Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors. In *IEEE Transactions of Parallel Distributes Systems,* vol. 19, no. 11 pp. 1540-1552, 2008.

[111] K. Seth, A. Anantaraman, F. Mueller and E. Rotenberg. FAST: Frequency-Aware Static Timing Analysis. In *Proc. of the Real-Time Systems Symposium*, 2003.

[112] L. Sha, R. Rajkumar and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronisation. In *IEEE Transactions on Computers,* vol. 39, no. 9, pp. 1175-1185, 1990.

[113] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. of the Design Automation Conference,* 1999.

[114] A. Sinkar and N. Kim. Analyzing Potential Power Reduction with Adaptive Voltage Positioning Optimized for Multicore Processors. In *Proc. of the International Symposium on Low Power Electronics and Design* 2009.

[115] J. E. Sergent and A. Krum, Thermal Management Handbook, McGraw-Hill, 1998.

[116] D. Snowdon, S. Petters, and G. Heiser. Accurate Online Prediction of Processor and Memory Energy Usage under Voltage Scaling. In *Proc. of the International Conference On Embedded Software*, 2007.

[117] B. Sprunt, J. Lehoczky and L. Sha. Aperiodic Task Scheduling for Hard Real-Time Systems. In *Real-Time Systems Journal,* vol. 1, no. 1, pp. 27-60, 1989.

[118] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. In *Real-Time Systems Journal,* vol. 10, no. 2, pp. 179-210, 1996.

[119] V. Swaminathan and K. Chakrabarty. Energy Conscious Deterministic I/O Device Scheduling in Hard Real-Time Systems. In *Proc. of the International Conference in Computer Aided Design*, 2003.

[120] V. Swaminathan and K. Chakrabarty. Pruning-Based, Energy-Optimal Deterministic I/O Scheduling for Hard Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 1, pp. 141-167, 2005.

[121] V. Swaminathan, K. Chakrabarty and S.S. Iyengar. Dynamic I/O Power Management for Hard Real-Time Systems. In *Proc. of the International Conference on Hardware-Software Co-design and System Synthesis*, 2001.

[122] H. W. Turnbull. Theory of Equations. Oliver and Boyd, London, 1947.

[123] S. Wang and R. Bettati. Reactive Speed Control in Temperature-Constrained Real-Time Systems. In *Proc. of the EuroMicro Conference on Real-Time Systems*, 2006.

[124] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *USENIX Symposium on Operating Systems Design and Implementation*, 1994.

[125] H. Wu, B. Ravindran, and E. D. Jensen. Utility Accrual Real-Time Scheduling Under the Unimodal Arbitrary Arrival Model with Energy Bounds. In *IEEE Transactions on Computers*, vol. 56, no. 10, pp. 1358-1371, 2007.

[126] R. Xu, D. Mosse, and R. Melhem. Minimizing Expected Energy Consumption in Real-Time Systems through Dynamic Voltage Scaling. *ACM Transactions on Computer Systems*, vol. 25, no.4, 2007.

[127] R. Xu, C. Xi, R. Melhem, and D. Mosse. Practical Pace for Embedded Systems. In *Proc. of the International Conference on Embedded Software*, 2004.

[128] C-Y. Yang, J-J. Chen, T-W. Kuo, and L. Thiele. An Approximation Scheme for Energy-Efficient Scheduling of Real-Time Tasks in Heterogeneous Multiprocessor Systems. In *Proc. of the Conference on Design, Automation and Test in Europe,* 2009.

[129] C. Yang, J. Chen, and T-W. Kuo. An Approximation Algorithm for Energy-Efficient Scheduling on A Chip Multiprocessor. In *Proc. of the Conference on Design, Automation and Test in Europe,* 2005.

[130] F. Yao, A. Demers and S. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proc. of the Foundations of Computer Science,* 1995.

[131] L.-T. Yeh and R. C. Chu, Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices, ASME Press, 2002

[132] F. Zhang and Chanson. Processor Voltage Scheduling for Real-Time tasks with Non-Preemptible Sections. In *Proc. of the Real-Time Systems Symposium,* 2002.

[133] X. Zhong and C.-Z. Xu. System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation. In *Proc. of the International Conference on Computer-Aided Design*, 2006.

[134] X. Zhong and C.-Z. Xu. Frequency-Aware Energy Optimization for Real-Time Periodic and Aperiodic Tasks. In *Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems*, 2007.

[135] D. Zhu, R. Melhem and B. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems. In *IEEE Transactions on Parallel and Distributed Systems,* vol. 14, no. 7, pp. 686-700, 2003.

[136] D. Zhu and H. Aydin. Energy Management for Real-Time Embedded Systems with Reliability Requirements. In *Proc. of the International Conference on Computer-Aided Design*, 2006.

[137] D. Zhu, R. Melhem and D. Mosse. The Effects of Energy Management on Reliability in Real-Time Embedded Systems. In *Proc. of IEEE/ACM Intl. Conf. on Computer Aided Design (ICCAD),* 2004.

[138] J. Zhuo and C. Chakrabarti. System-Level Energy-Efficient Dynamic Task Scheduling. In *Proc. of the Design Automation Conference*, 2005.

[139] Advanced Configuration and Power Interface Standard, http://www.acpi.info/.

[140] IBM Power 7 Overview.
http://www.redbooks.ibm.com/redpapers/pdfs/redp4638.pdf

[141] Introduction to Intel Core Duo Processor Architecture. In *Intel Technology Journal*, vol 10, no. 2, pp. 89-97, 2006.

[142] Intel i7 Processor Specifications.
http://www.intel.com/products/processor/corei7/specifications.htm

[143] Intel i7-800 and i5-700 Processor series. Datasheet - Volume 1.
http://download.intel.com/design/processor/datashts/322164.pdf

[144] Intel Xeon Specifications.
http://www.intel.com/design/intarch/xeon/specifications_xeon.htm

Curriculum Vitae

Vinay Devadas received his B.S. degree in Computer Science and Engineering from Visvesvaraya Technological University, Bangalore, India in 2005 and his M.S. degree in Computer Science from George Mason University, Fairfax, VA in 2007.