JOINT RELIABILITY AND ENERGY MANAGEMENT FOR REAL-TIME EMBEDDED SYSTEMS

by

Baoxian Zhao
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]	Hakan Aydin, Dissertation Director
[illegible]	Songqing Chen, Committee Member
[illegible]	Fei Li, Committee Member
[illegible]	Chaowei Yang, Committee Member
Sanjeev Setia	Sanjeev Setia, Department Chair
[illegible]	Lloyd J. Griffiths, Dean, Volgenau School of Engineering
Date: July 18, 2012	Summer Semester 2012 George Mason University Fairfax, VA

Joint Reliability and Energy Management for Real-Time Embedded Systems

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

By

Baoxian Zhao
Master of Science and Engineering
Nanjing University of Aeronautics and Astronautics, 2005
Bachelor of Science and Engineering
Nanjing University of Aeronautics and Astronautics, 2002

Director: Hakan Aydin, Professor
Department of Computer Science

Summer Semester 2012
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my wonderful family. Especially to my understanding wife, Ting, who has put up with these many years of research and has given me continuous support, and to our lovely daughter, Ivy, who is the joy of our lives. I must also thank my loving parents and my terrific in-laws who have helped so much and have given me their fullest support. My special gratitude is due to my brother, my sister and their families for their loving support.

Acknowledgments

I would like to thank the following people who made this possible.

First, I would like to express my deepest sense of gratitude to my advisor, Dr. Hakan Aydin. During these years, he has patiently provided the vision, encouragement and advise necessary for me to proceed through PhD program and complete my dissertation. His wisdom, knowledge and commitment to the highest standards inspire and motivate me.

I owe my most sincere gratitude to, my first advisor at Mason, Dr. Ravi Sandhu, who gave me the opportunity to join the PhD program and gave me untiring help during my difficult moments. Also I am grateful to Dr. Daniel A. Menascé for his encouragement and his practical advice through my whole PhD study. I am also thankful to him for his great help and untiring support during my job hunting.

Special thanks to my committee, Dr. Songqing Chen, Dr. Fei Li, and Dr. Chaowei Yang for their support, guidance and helpful suggestions. Their guidance has served me well and I owe them my heartfelt appreciation. I warmly thank, our collaborator, Dr. Dakai Zhu, for his valuable advice and friendly help. His extensive discussions around my work and interesting explorations in the future directions have been very helpful for my research.

A very special thank you to one of my best friends and my previous supervisor at Mason Honor Committee, Dr. Donna Fox, for her valuable suggestion, continuous encouragement, as well as her terrific friendship.

I am very proud of having so many friends around me during my PhD study. Although I could not list all their names within a single page, I do appreciate their great support and care, which has helped me overcome setbacks and stay focused on my PhD study through these difficult years.

Finally, I wish to extend my warmest thanks to all those who have helped me with my work in the Department of Computer Science as well as Volgenau School of Engineering. The financial supports from Mason and NSF are gratefully acknowledged.

Table of Contents

List of Tables

List of Figures

Abstract

JOINT RELIABILITY AND ENERGY MANAGEMENT FOR REAL-TIME EMBEDDED SYSTEMS

Baoxian Zhao, PhD

George Mason University, 2012

Dissertation Director: Hakan Aydin

The Dynamic Voltage Scaling (DVS) technique is the basis of numerous state-of-the-art energy management schemes proposed for real-time embedded systems. However, recent research has illustrated the alarmingly negative impact of DVS on system reliability, in terms of increased vulnerability to transient faults, leading to soft errors. The main theme of this dissertation is to investigate several open problems and trade-off opportunities in energy, reliability, and timeliness dimensions.

This dissertation, first, investigates the problem of maximizing the overall reliability of real-time embedded applications while meeting the deadlines and a given *energy budget* constraint. Optimal static solutions and effective dynamic (online) solutions are developed.

Second, the dissertation proposes a new approach, called the *shared recovery (SHR)* technique, to minimize the system-level energy consumption while still mitigating the reliability loss induced by DVS. The main idea of the SHR technique is to avoid the off-line allocation of separate recovery tasks to the scaled tasks by assigning a global/shared recovery block that can be used by any task at run-time. Specifically, an array of reliability-aware energy management algorithms are presented for both independent and dependent tasks.

Third, the dissertation presents the foundations of a general reliability-oriented energy management framework, where the objective is to achieve any reliability level with minimum energy consumption and timeliness guarantees. For periodic real-time tasks, the framework is extended to address multiple reliability objectives that can be set by the designer and vary from task to task.

Finally, this dissertation considers the problem of minimizing the *expected* energy consumption for a real-time embedded application. This part of the research integrates optimally DVS and Dynamic Power Management (DPM) techniques that can put some system components to *sleep* states when they are not in use for the case where the workload is known only *probabilistically*.

Chapter 1: Introduction

Real-time embedded systems, where satisfying the *timing constraints (deadlines)* of applications at run-time is critical, became prevalent in numerous areas ranging from industrial control, telecommunication, multimedia, virtual reality, and military systems, among others. Fault tolerance and reliability have always been important concerns in the design and operation of real-time systems: *faults* must be detected and proper *recovery* operations must be undertaken before the application's deadline, whenever possible. For instance, *transient faults* that cause temporary *soft errors* at run-time are much more common than *permanent faults* that cause a complete system halt [16, 34]. Effective mechanisms to tolerate transient faults are discussed in [39, 50].

More recently, with the emergence of computing and communication devices that rely on battery power, *energy management* has become another important research venue [7, 48, 55, 72, 75]. However, research efforts that consider both reliability and energy management dimensions for real-time embedded systems are relatively few.

The *Dynamic Voltage Scaling (DVS)* technique [73] is recognized as one of most effective energy management techniques. By considering the strictly convex relationship between the supply voltage and CPU power consumption, DVS attempts to save energy by scaling down the frequency along with the supply voltage. However, at low processing frequencies, tasks take longer to complete, and a significant body of research has been recently devoted to the *energy-aware real-time systems* area where the objective is to minimize the energy consumption while still meeting the deadlines for various task/processor and scheduling models [6, 48, 55]. On the other hand, for *energy-constrained* real-time embedded systems, energy is set as a *hard* constraint [2, 42, 71]. This research line is motivated by applications where the system has to remain functional during a well-defined mission/operation time, with fixed and non-replenishable energy budget. Example applications include military, space and disaster recovery applications, as well as emerging portable medical monitoring and life support devices. In such settings, it is imperative to avoid scenarios where the real-time embedded system runs out of energy in the middle of a mission with potentially detrimental consequences [3, 42, 54].

Recent research [26, 91] has shown that using DVS can result in significant reliability degradation due to increased transient fault rates at low frequency and voltage levels [91]. Hence, there is a growing awareness about the need to apply DVS only after careful evaluation, especially for mission-critical real-time embedded applications where both high level of reliability and low energy consumption are important. In fact, a number of recent research articles [87–89] promoted the so-called *reliability-aware power management (RA-PM)* framework, where the aim is to minimize the system-wide energy consumption through DVS while *maintaining the original system reliability*. Here the original system reliability is defined as the one that is obtained when tasks are executed without voltage scaling (i.e. at the maximum frequency).

The primary objective of this dissertation is to investigate several open problems and trade-off opportunities on *energy, reliability* and *timeliness* dimensions for real-time embedded systems. In particular, the dissertation's focus is on settings where the energy consumption or target reliability is given as a hard constraint and the objective is to optimize the other metric while still meeting the timing constraints.

For instance, the problem of *maximizing reliability* subject to energy budget and deadline constraints (i.e., in energy-constrained real-time systems settings) is an important and open problem. The problem formulation needs to model and exploit carefully the impact of DVS on the application's reliability and response time.

Regarding the problem of maximizing energy savings with reliability considerations, there are a number of open problems. The existing RA-PM solutions target preserving the original system reliability, and the central mechanism behind them is essentially to schedule a *separate recovery task*, for every task whose execution frequency is scaled down through DVS at run-time [87–89]. Specifically, when a transient fault is detected at the end of task execution, a recovery takes place in the form of re-execution at the maximum frequency before the task's deadline. In existing RA-PM schemes, only after reserving CPU time (slack) for recovery tasks the remaining slack is used to determine the task-level low processing frequencies for energy management. As a result, the existing RA-PM schemes are *conservative*, in the sense that statically allocating the available slack for multiple recovery blocks of a pre-determined set of tasks reduces the prospects for energy savings by reducing opportunities for DVS, calling for more effective solutions.

Also a more comprehensive framework is highly desirable in order to achieve *any* reliability goal, which can be set by the designer to be *lower* or *higher* than the application's original reliability. This is because, some modest reliability degradation might be acceptable in energy-scarce settings. In contrast, it may be necessary to achieve very high reliability levels for safety-critical systems (e.g. electronics-hostile settings encountered in space applications). Clearly, merely maintaining task-level reliabilities would be too conservative or insufficient, respectively, in these two scenarios.

Besides DVS, another energy management technique is *Dynamic Power Management (DPM)* that puts the idle system components into *low-power (sleep)* states whenever possible. Memory and I/O devices are the common target system components for DPM. However, a major challenge for DPM is to decide, at run-time, whether the time/energy overheads associated with the state transition will be offset during the time the component remains in sleep state. While the research efforts that focus on only on DVS or DPM are numerous, solutions that propose integrating both policies under a unified framework are relatively few in real-time systems research. The growing importance of system-wide energy management clearly mandates integration of both policies. Recently, an optimal solution based on the exact characterization of the variation of the system energy with both DVS and DPM has been proposed in [21]. Despite its accuracy and novelty, that solution assumes a deterministic and worst-case workload for the real-time embedded application. However, in practice, the actual workload of an application differs from the worst-case, and can be typically known *probabilistically* [72, 75]. Hence, integrating DVS and DPM for a real-time application with probabilistic workload information is another relevant and open problem.

1.1 Problem Statement

In this dissertation, the following four main problems are addressed.

1.1.1 Maximizing Reliability for Energy-Constrained Real-Time Embedded Systems

This dissertation first investigates the problem of determining task frequency assignments to *maximize overall reliability* for energy-constrained real-time systems, subject to *energy budget* and *deadline* constraints. While existing research explored various aspects of the interplay between DVS, reliability and energy consumption from energy-aware operation point of view, to the best of my knowledge, maximizing reliability in energy-constrained settings (with hard energy constraint) has not been studied before.

1.1.2 Reliability-Aware Energy Management through Shared Recovery Technique

There is a growing literature on reliability-aware power management (RA-PM) research [49, 60, 87, 88, 90] that aims to minimize energy consumption while maintaining the original system reliability. The key observation in existing RA-PM studies is that provisioning for a separate recovery task for every task that is scaled down through DVS will automatically preserve that task's original reliability, assuming that the recovery task will be executed at the maximum CPU speed and before the deadline. However, by doing so, the available system slack for DVS and potential energy savings may significantly be reduced. Hence, *a flexible mechanism where the recovery tasks may be shared among tasks and activated upon the occurrence of transient faults may greatly improve the energy savings.* Obviously meeting the deadlines and preserving the original reliability are still stringent constraints in this *shared recovery (SHR)*-based RA-PM framework.

Moreover, in a number of applications, real-time tasks may depend on each other for input/output data relationships and thus may have precedence constraints. Thus, this dissertation further considers the problem of extending the SHR framework to task systems whose precedence constraints are represented by *directed acyclic graphs (DAGs)*.

1.1.3 Generalized Reliability-Oriented Energy Management

The existing RA-PM solutions operate under the assumption that mitigating the reliability loss due to DVS is necessary and sufficient for reliability-cognizant energy management. This dissertation aims also at developing a more general framework where the objective is to *minimize energy consumption subject to achieving arbitrary reliability levels set by the designer*. This target reliability level can be lower or higher than the system's original reliability. Moreover, the dissertion considers the problem of extending this reliability-oriented energy management framework in two directions:

1. Maximizing energy savings to achieve *arbitrary* reliability levels for *individual* periodic tasks, when employing DVS. This may prove very useful for applications with different/mixed criticality (or, importance) levels whose requirements may not be fully captured by simply preserving the original reliability levels or achieving uniform reliability levels. Such task-variant reliability objectives can neither be expressed nor achieved in existing RA-PM solutions.

2. Exploring the potential of deploying *dynamically allocated recoveries* for periodic tasks, in co-management of reliability and energy. As opposed to the current RA-PM schemes that *statically* allocate a separate recovery to each and every scaled job, the framework is based on providing every periodic task with a certain *recovery allowance* that may be used at arbitrary times during execution. With these two leverage dimensions, the dissertation seeks to formulate and tackle the general problem of *determining recovery allowance and frequency assignments to minimize energy consumption, while meeting the timing constraints and task-level reliability targets.*

1.1.4 Minimizing Expected Energy Consumption through Optimal Integration of DVS and DPM

While the DVS technique targets the CPU energy consumption, the Dynamic Power Management (DPM) technique is another mainstream approach to reduce energy in off-chip components, such as memory and I/O devices. DPM involves transitioning individual system components to *low-power (sleep)* states, when they are not in use. However, combining DPM and DVS poses non-trivial difficulties as DPM requires leaving significant CPU idle time to transition the devices, reducing opportunities for DVS. While recent research has explored this problem for *deterministic (worst-case)* workloads, this dissertation investigates the problem of determining frequency assignment and device state transition decisions for a real-time application whose workload is known only *probabilistically* to minimize the *expected energy consumption*. The solution must still make sure that the timing constraints are met even if the worst-case workload is encountered at run-time.

1.2 Contributions

For the problem of *maximizing reliability for energy-constrained real-time embedded systems*, a number of contributions are made, considering both frame-based (common deadline) and periodic task models. First, under the frame-based task model, this dissertation formulates the problem as a nonlinear optimization problem and shows how to obtain the static optimal solution. Then, it proposes online (dynamic) algorithms that detect early completions and adjust the task frequencies at runtime, to improve overall reliability. Furthermore, the dissertation extends these solutions to the periodic task model, with both static and dynamic solutions. All these solutions ensure that all timing constraints are met while the cumulative energy consumption of tasks does not exceed the given energy budget. The simulation results indicate that the suggested algorithms perform comparably to a clairvoyant optimal scheduler that knows the exact workload in advance.

The following contributions are made in the area of *reliability-aware energy management* *through shared recovery technique.*

- A new RA-PM approach called *shared recovery (SHR)* technique is developed, based on the deployment of recovery blocks that may be shared among multiple tasks to achieve further energy savings, as opposed to scheduling a separate recovery for each managed task. In the offline phase, the SHR technique allocates CPU time only for the largest recovery that may be needed by any task, essentially making available to the DVS mechanism the valuable slack that would be typically reserved for the recoveries of separate tasks in the previous RA-PM solutions. The energy savings of the new scheme are shown to approach those of the optimal (but not reliability-aware) DVS solutions. Further, SHR is formally shown to preserve the original system reliability. Also, a dynamic extension is proposed to improve energy and reliability management at run-time by reducing the size of the recovery block and re-using the slack that arises from early task completions.

- The SHR technique is also extended to dependent tasks that are represented by the directed acyclic graphs (DAGs). It is shown that given a task set represented by a DAG, employing the earliest deadline first (EDF) algorithm with effective deadlines derived through the *as late as possible (ALAP)* policy is optimal, in the sense that one can obtain a feasible schedule of tasks whenever it is possible to do so through any other algorithm. Then, a shared recovery based frequency assignment technique for these settings, called *SHR-DAG*, is developed and its optimality is demonstrated. An extension to dynamic settings, exploiting the additional slack that arises from early completion of tasks, is proposed.

For the problem of *generalized reliability-oriented energy management*, the following contributions are made:

- To achieve arbitrary system-level reliability figures for frame-based real-time embedded applications, a new technique, called the *Generalized Shared Recovery (GSHR)*, is developed. This technique determines the optimal number of recoveries to deploy as well as task-level processing frequencies to minimize the energy consumption while achieving the reliability goal and meeting the timing constraints. The potentially multiple recoveries may be still shared among tasks, improving the prospects of DVS compared to existing reliability-aware power management frameworks. The experimental evaluation points to the close-to-optimal energy savings of the new proposed technique.

- Further, a general framework to achieve arbitrary reliability objectives that may vary for each periodic task (i.e. targeting *task-level reliability objectives*) is explored. A pseudo-polynomial time feasibility test, as well as static and dynamic algorithms to determine the recovery allowance and frequency assignments, are presented. A critical building block of the solution is the use of *dynamically allocated* recoveries: It is shown that providing a relatively modest recovery allowance to a given periodic task helps to achieve surprisingly high reliability levels, as long as these allowances can be reclaimed on-demand at run-time. The experimental evaluation points to the significant gain potential of the new framework in terms of both energy and reliability figures.

Finally, the following contributions are made towards the problem of *minimizing expected energy consumption through optimal integration of DVS and DPM*: By considering a real-time application whose workload distribution is known probabilistically, algorithms to determine optimal frequency assignments and device state transition decisions are developed first on settings with a single off-chip device. Then the solutions are extended to the case where multiple devices may be used by the application at run-time. The evaluations indicate that the new technique provides clear (up to 35%) energy gains over the existing solutions that are proposed for deterministic workloads.

1.3 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 gives background and Chapter 3 presents the details of system models and assumptions. Chapter 4 addresses the reliability maximization problem for energy-constrained real-time embedded systems. In Chapter 5, the shared recovery technique is proposed to achieve further energy savings for both independent and dependent frame-based tasks, as opposed to scheduling a separate recovery for each managed task. Following that, the generalized reliability-oriented energy management for real-time embedded applications is discussed in Chapter 6. Chapter 7 addresses the solutions to minimize expected energy consumption through optimal integration of DVS and DPM for probabilistic workloads. Finally, Chapter 8 concludes the dissertation.

Chapter 2: Background

2.1 Real-time Embedded Systems

In a real time embedded system, the application that consists of a set of tasks has to satisfy the timing constraints during operation. Each real-time task is characterized by its *release time*, *deadline*, and *worst-case execution time (WCET)*. The *release time* is the time instance when the task becomes available for execution. The *worst-case execution time* of a task is the maximum execution time it requires on a given single-CPU platform. The *deadline* of a task is the time instant by which the task must finish its execution.

Based on the nature of timing constraints, a real-time system can be either *soft* or *hard*. In a hard real-time system, deadlines misses can lead to unpredictable, very serious, and even disastrous consequences. In contrast, in soft real-time systems, deadline misses can result in degraded performance or quality-of-service (QoS).

Real-time tasks can be further classified into two categories: *aperiodic* and *periodic* tasks. An *aperiodic* task is executed only once and has typically a soft deadline. On the other hand, most of the real-time tasks are *periodic*. In the periodic task model, the computation of each task is executed repeatedly at regular or semi-regular time intervals (known as *period*) in order to maintain a system function on a continuing basis. A given instance of a periodic task is sometimes called a *job*. Each job of a periodic task must finish its computation before or at the beginning of the next period. For periodic tasks, the *hyperperiod* is defined as the *least common multiple (LCM)* of all the periods.

In a sizeable number of periodic real-time applications, the tasks execute in a cyclic basis, i.e. one after another and within a *time frame* whose length is equal to the common period/deadline. These systems are called *frame-based* systems. The execution order may be chosen to satisfy the *precedence constraints*, that can be derived through input/output relationships among tasks and represented by a *directed acyclic graph (DAG)*. Otherwise, if the tasks can execute in any order, they are said to be *independent*.

A scheduling policy is used to guarantee the timing constraints of the real-time embedded application. In a *feasible* schedule, all tasks complete their executions no later than their deadlines, can execute only after they are released, and all the precedence constraints (if any) are satisfied. Depending on whether the task can be preempted or not during its execution, the scheduling policies can be classified as *preemptive* or *non-preemptive*. A scheduling policy is said to be *optimal* if and only if it always produces a feasible schedule for any task set whenever such a feasible schedule exists for that set.

2.2 Fault Tolerance in Real-Time Embedded Systems

During the execution of an application, a *fault* (leading to an *error*) may occur due to various reasons, such as hardware failure, software design errors, electromagnetic interference and cosmic ray radiations. Typically faults can be classified into two categories: *permanent* and *transient* faults [62].

Permanent faults, causing *hard errors*, are caused by defects in the silicon or metalization of processor package, such as process and manufacturing defects (called *extrinsic failures*), as well as processor wear-out over time due to specified conditions (viewed as *intrinsic failure*). Permanent faults can be typically tolerated only by deploying extra hardware (e.g. processing units).

Transient faults, leading to *soft errors* or single-event upsets (SEUs), may be caused in the execution of the application by electrical noise or external radiation, rather than processor design or manufacturing-related defects. Although transient faults can cause computation errors or data corruption in a specific execution of a task, they are not persistent: they do not fundamentally damage the microprocessor and therefore the faulty applications can be typically re-executed by using suitable fault tolerance techniques.

With the continued scaling of CMOS technologies and reduced design margins for higher performance, it is expected that, in addition to the systems that operate in electronics-hostile environments (such as those in outer space), practically all digital computing systems will be remarkably vulnerable to transient faults [39]. This dissertation focuses on *transient faults*.

In the fault tolerance area, *redundancy* is employed to mask or otherwise work around these faults, thereby preserving a certain desired level of functionality [39]. In general terms, *redundancy* is defined as the deployment of extra resources for the application. There are four main types of redundancy: *hardware*, *software*, *information* and *time* redundancy [39]. *Hardware redundancy* is implemented by incorporating extra hardware (e.g. processors) into the system to provision for a faulty component. *Software redundancy* involves having two or more independent versions of the software, and is effective against software failures. Redundant information (e.g. through extra error detection/correction bits) is used in *information redundancy* technique. *Time redundancy* is typically used to re-execute the same program on the same hardware. Since the majority of faults are transient and the chances of having the independent executions of the same task experience the same fault are very small, time redundancy is known to be effective against transient faults [39].

Three different implementation methods for time redundancy are *checkpointing*, *recovery block* and *recovery through re-execution. Checkpointing* technique inserts checkpoints during the execution of a task. Within a checkpoint, the state of a system is checked and correct states are saved to a stable storage. When faults are detected, the execution is rolled back to the latest correct checkpoint and re-computes the faulty section by exploring the temporal redundancy. With the large number of checkpoints, the time overhead caused by this technique may be prohibitive. Hence, this technique is seldom used in real-time embedded systems. The *recovery block* approach is an alternative method providing a task with one or more *backups*. Once the original version of the program fails, the system switches to the executions of its backup(s) (i.e., different versions). Notice that the execution times of the original task and its backups may be different. This method is mainly used to tolerate software errors. *Recovery through re-execution* technique is widely used to tolerate transient faults, by re-executing the original task if a fault occurs. It is assumed that transient faults are detected by using *sanity* (or *consistency*) checks at the completion of a task's execution. When faults are detected, the system restores the system state to a previous safe state and the recovery task is dispatched, in the form of re-execution [39].

2.3 Energy Management for Real-Time Embedded Systems

With the proliferation of computing and communication devices that rely on battery power, energy management has been a very active research topic in the last decade. In this regard, *Dynamic voltage scaling* (DVS) is recognized as one of most effective and fundamental techniques. By considering the strictly convex relationship between the supply voltage and CPU power consumption, DVS attempts to save energy by scaling down the frequency along with the supply voltage. In general, there is a time and energy overhead associated with the run-time voltage and frequency changes involved in DVS. For instance, the time and energy overhead in frequency/voltage transitions is in the range of 10-120 μs and 0.5-2.6 $\mu Joules$ respectively, for the state-of-the-art Mobile AMD Athlon and Intel Xscale architectures.

In existing DVS research studies, the CPU frequency/speed spectrum is considered in two different ways. In *continuous frequency* model, it is assumed that the CPU frequency (voltage) can assume any value between a lower and upper bound. However, in current commercially available DVS-enabled processors, there are only a limited number of (typically less than 10) frequency and voltage levels. This model is called the *discrete frequency* model. There are, on the other hand, well-established techniques that allow adapting frequency assignments obtained through the continuous model into the discrete model [32].

Another important energy management technique is *Dynamic Power Management* (DPM) that puts the idle system components into low-power states whenever possible. In fact, off-chip devices (such as I/O devices and the main memory) have an *active* state and at least one low-power *sleep* state. However, significant transition energy/time overheads may be involved in state transitions of the devices. In fact, a minimum idle interval length (called the *device break-even time*) is needed in DPM to justify the transition of the device to the *sleep* state. DPM reduces energy by putting the device into the low-power *sleep* state when the idle time is predicted to be no less than the device break-even time.

In real-time embedded systems, timing constraints (deadlines) should be satisfied even under worst-case execution scenarios. Therefore, the deadline misses must be avoided when applying DVS or/and DPM techniques to minimize energy consumption in real-time embedded systems.

2.4 Previous Work on DVS in Real-Time Embedded systems

In *energy-aware* real-time systems research, a major problem is to meet the application's timing constraints while minimizing energy consumption through DVS for various task/processor and scheduling models.

Early research efforts can be classified in two categories, depending on the granularity of voltage scaling at run-time:

- The *inter-task DVS schemes* focus on allotting the CPU time to multiple tasks and perform frequency scaling only at task preemption/completion points. The first inter-task algorithm is discussed in [73]. After that, many inter-task DVS solutions have been proposed [7, 48, 55] for different scheduling policies and system models.

- The *intra-task DVS schemes*, on the other hand, changs the frequency while a task is executing (in its allocated CPU time). The PACE scheme [44] is recognized as the first intra-task DVS scheme. In that work, it was shown that if the computational requirement of a task is only known probabilistically, there is no constant optimal speed for the task and the expected energy consumption is minimized by gradually increasing the speed as the task makes progress. Another two well-known intra-task algorithms are GRACE algorithm [75] and PPACE algorithm [72] that are suitable mostly for soft real-time applications.

Note that these early studies mostly focus exclusively on the energy consumed by the

CPU. That is, these schemes account for the energy consumed by CPU (i.e. frequency-dependent energy) by ignoring the energy consumed by other components such as disk, I/O and so on (i.e. frequency-independent energy). At reduced frequencies, the frequency-independent energy consumption increases although the frequency-dependent component decreases. Therefore, when minimizing system-wide energy, a trade-off between two energy components should be sought. [5] presents a solution to this problem by taking into account both the frequency-independent and frequency-dependent energy consumptions. The *energy-efficient frequency* concept [5,91], is introduced to balance the off-chip device energy and CPU energy consumed during the task's execution. The system-wide energy model is widely used in recent DVS research [81,82,84,85,87,89].

On the other hand, for *energy-constrained* real-time embedded systems, energy is set as a *hard* constraint [2, 42, 71]. This research line is motivated by applications where the system has to remain functional during a well-defined mission/operation time, with fixed and non-replenishable energy budget. Example applications include military, space and disaster recovery applications, as well as emerging portable medical monitoring and life support devices. In such settings, it is imperative to avoid scenarios where the real-time embedded system runs out of energy in the middle of a mission with potentially detrimental consequences [3, 42, 54]. The *Mars Rover* system has been often mentioned as a well-known energy-constrained system that must complete its mission-critical tasks and return to the base within a well-defined operation time [17,42,71]. In energy-constrained real-time systems area, the researchers exploited the problem of maximizing the total utility [71], reward [2, 17, 54], value [3], and throughput [76] without exceeding the available energy budget.

2.5 Previous Work on DPM in Real-Time Embedded systems

One of the critical issues when applying DPM in real-time embedded systems is to decide when to transition a device to a low-power state at run-time. Based on power management policies, DPM can be classified into two major classes [10]: *predictive* and *stochastic* schemes. The rationale in all predictive techniques is to exploit the correlation between the past history of the workload and the near future, as well as information about task arrival patterns, in order to make reliable predictions about future events. Hence, effective DPM decisions highly depend on the accuracy of the predictive techniques. The stochastic schemes formulates power optimization as a stochastic optimization problem under uncertainty, by assuming that the workloads follow some probability distributions. Targeting different task/device settings, many off-line/on-line DPM schemes are suggested in recent research [18,22,64–66].

2.6 Previous Work on Combining DVS and DPM in Real-Time Embedded systems

While the research efforts that use DVS or DPM separately are numerous, solutions that propose integrating both policies under a unified framework are relatively few [19,21,37, 58,86]. Authors in [58] apply a stochastic DPM policy by using the different DVS voltage levels as multiple active power modes. The work in [86] proposes a DVS-DPM policy that maximizes the operational lifetime of an embedded system powered by a fuel cell based hybrid power source. The frequency scaling level is chosen in [37] by investigating the trade-offs between the DVS-enabled CPU and the DPM-enabled devices. The SYS-EDF algorithm is a heuristic-based energy management scheme for periodic real-time tasks [19]. A DPM framework based on the *device forbidden regions* concept has been proposed for periodic tasks and then integrated with DVS through sophisticated rules for rate-monotonic

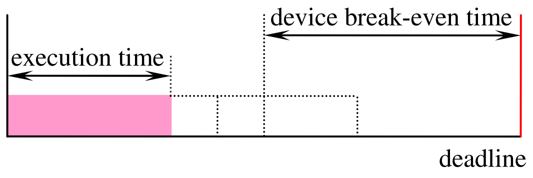


Figure 2.1: The interplay of DVS and DPM

and earliest-deadline-first scheduling policies in [22, 23].

The growing importance of the system-wide energy management clearly mandates integration of both policies. Yet most of the existing solutions, while noteworthy, adopt heuristic-based solutions with no performance guarantees. In fact, it is somewhat surprising that the full solution for a single real-time embedded application using both DVS and DPM policies was published only recently [21].

DVS and DPM solutions often work against each other. As shown in Figure 2.1, if the processing frequency is lowered through DVS to save energy, the task execution time is extended. As a result, the idle time is shortened, which prevents DPM from putting the devices into the low-power sleep states. On the other hand, the device energy can be reduced by executing the task at higher frequency to obtain enough idle time for putting the devices to the low power sleep state while this results in additional transition energy overhead and CPU energy consumption. Moreover, the application may be using multiple devices with different power and break-even time characteristics, making an optimal solution non-trivial. An exact and formal characterization of the interplay between DVS and DPM for a real-time application using multiple devices was recently obtained in [21]. In the same work, an algorithm to compute the optimal processing frequency was also derived. However, the solution in [21] is derived assuming a known deterministic/worst-case workload.

2.7 Reliability-aware Power Management in Real-Time Embedded Systems

The research community started to explore the problems at the intersection of reliability-aware design and power management in recent years. In [56], the authors studied the energy minimization problem for mapping frame-based dependent real-time tasks on DVS-enabled multi-processor systems and proposed an efficient two-step iterative approach based on genetic algorithms. Izosimov et al. [35] obtain and solve an optimization problem for mapping a set of tasks with reliability constraints, timing constraints and precedence relations to processors and for determining appropriate fault tolerance policies (re-execution and replication). This study does not explicitly model the effects of DVS on transient fault rates.

A significant number of existing research work focused on tolerating a fixed number of faults within the context of energy-aware real-time operation [46, 70, 77]. The checkpointing technique was adopted in [46] to tolerate faults at run-time while exploiting system slack through DVS to reduce energy consumption. By adopting the checkpointing technique for soft periodic real-time tasks, Zhang et al. developed an online fault tolerance algorithm in [77]. Further, they suggested a fixed priority scheme through the Rate-Monotonic algorithm (RMA) to tolerate faults in hard real-time systems. Based on the characterization of feasibility with RMA, in [70], the authors proposed an efficient online scheme to minimize energy consumption by applying DVF policies and considering the run-time behaviors of tasks and fault occurrences while still satisfying the timing constraints.

Despite the effectiveness of DVS to reduce energy consumption, it was shown that DVS has a significant and negative effect on the *system reliability*, primarily because of the significantly increased *transient fault rates* at low supply voltage and frequency levels. The *reliability* of a task is defined as the probability that the task will successfully complete either without any transient faults, or thanks to the execution of the recovery, should a fault be detected. The system reliability is defined as the product of individual task reliabilities [39]. Hence, there is a growing awareness about incorporating reliability constraints to energy management techniques. The negative impact of DVS on transient fault rates and investigation of the resulting reliability/energy consumption trade-offs gave rise to another line of research.

As a result, recently a number of research studies promoted the so-called *Reliability-Aware Power Management (RA-PM)* framework, where the aim is to minimize the system-wide energy consumption through DVS while *maintaining the original system reliability*[1]. Specifically, the RA-PM schemes reserve some CPU time (slack) for recovery tasks that may be invoked at run-time upon the detection of faults at the end of task executions. The modeling of system reliability as a function of frequency/voltage assignment and a preliminary analysis of related trade-off dimensions has been conducted by Zhu et al. in [91]. The work in [87] proposed a reliability-aware power management (RA-PM) scheme that dynamically schedules a recovery job at task dispatch time whenever DVS is applied, preserving the overall (original) reliability. The scheme is further extended to multiple tasks with a common deadline [88] and to periodic real-time tasks [89]. In all this RA-PM research, a recovery task is invoked at run-time in case that the application, whose execution frequency is scaled down through DVS, incurs a transient fault. A main feature of the existing RA-PM schemes is that the algorithms first reserve some portion of the system slack (CPU time) for potential recovery of the tasks that may be subject to transient faults when executed with voltage scaling. Only then, the remaining slack is used to determine the task-level low processing frequencies for energy management.

Recently, considering multiprocessor systems, a few RA-PM schemes have been studied in [28, 52]. Ejlali et al. studied a number of schemes that combine the information about hardware resources and temporal redundancy to save energy and to preserve system reliability [25]. By employing a feedback controller to track the overall miss ratio of tasks in soft real-time systems, Sridharan et al. [60] proposed a reliability-aware energy management algorithm to minimize the system energy consumption while still preserving the overall system reliability. Pop et al. propose a novel framework [49] where user-defined reliability goals are first transformed to the objective of tolerating a fixed number of faults

[1]The system reliability is the one that corresponds to the case where all tasks execute at the maximum frequency.

through re-execution for individual tasks, by using the reliability model from [91]. The authors developed a constraint-logic-programming (CLP) based solution to minimize energy consumption with these goals, for dependent tasks represented by directed acyclic graphs (DAGs). Dabiri et al. [20] studied the problem of assigning frequency/voltage to tasks for energy minimization subject to reliability and timing constraints. They proposed a convex programming formulation. Recently, hardware redundancy technique [24, 61] is deployed through an extra spare processor to minimize system energy consumption while still preserving original system reliability. More recently, in order to achieve more energy savings in a standby sparing scheme, Tavana et al. suggested the feedback-based energy management in [67]. In [29], Haque *et al* extended the standby technique to periodic real-time applications.

Chapter 3: System Model and Assumptions

3.1 Processor and Workload Model

This dissertation considers a single-processor system with DVS capability, where the clock frequency can vary from a minimum available frequency f_{min} to a maximum frequency f_{max}. For convenience, all the frequency (f) values are normalized with respect to f_{max}, that is, it is assumed that $f_{max} = 1.0$.

The workload consists of a real-time embedded application with a set Γ of n independent periodic frame-based tasks $T_1, T_2, \ldots, T_n$. The relative deadline of task T_i is assumed to be equal to its period. The j^{th} job of T_i, denoted as T_{ij}, arrives at time $(j-1) \cdot p_i$ and has a deadline of $j \cdot p_i$. The *hyperperiod* H of the task set is defined as the least common multiple (LCM) of all tasks' periods. In case of *frame-based* systems, the common period P coincides with the common deadline d. The total number of jobs of task T_i during the hyperperiod is represented by $k_i = \frac{H}{p_i}$. The workload of each task T_i is characterized by its *worst-case number of cycles* c_i. The execution time of task T_i under the frequency f_i is given by $\frac{c_i}{f_i}$. Under maximum frequency $f_{max} = 1.0$, the task's execution time corresponds to c_i time units, which is also called its *worst-case execution time (WCET).* The *utilization* U of the task set is given as $\frac{\sum c_i}{p_i \cdot f_{max}} = \frac{\sum c_i}{p_i}$; in other words, it corresponds to the *load* under the maximum CPU frequency. Hence, throughout the whole dissertation, I assume that the system utilization U when all tasks are executed at f_{max} never exceeds 1.0 in order to guarantee the deadlines (feasibility).

3.2 Energy Model

DVS exploits the fact that there is a linear relationship between the supply voltage and operating frequency, and the supply voltage is reduced alongside the processing frequency. This dissertation adopts the general system-level power model proposed in [91] and subsequently used in prior RA-PM research [87, 88, 90]. Hence, the total system power consumption P is given by:

$$P_{total} = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + af^m) \tag{3.1}$$

Here P_s is the static power, which includes the power to maintain basic circuits, and keep the clock running and the memory in sleep modes. It can be removed only by powering off the whole system. P_{ind} is the frequency-independent active power and P_d is the frequency-dependent active power. P_d is the power mainly consumed by the CPU, in addition to any power that depends on the frequencies. $\hbar$ represents the system states and indicates whether active powers are currently consumed in the system. Specifically, when the system is active, $\hbar = 1$; otherwise, the system is in sleep modes or turned off and $\hbar = 0$. The effective switching capacitance a and the dynamic power exponent m (which is, in general, no smaller than 2) are system-dependent constants and f is the processing frequency. Since there exists an excessive overhead associated with turning on/off a system, this dissertation assumes that the system is always active while the periodic application is running. As P_s is not manageable, I will ignore the sleep power P_s and concentrate on the frequency-independent active power P_{ind} and frequency-dependent active power P_d in my analysis.

Notice that, the frequency-independent power component P_{ind} may vary from task to task. Consequently, the overall *energy* consumption expression for a task T_i with execution cycles c_i and the frequency-independent power P_{ind_i} at the frequency f_i can be obtained as in [91]:

$$E_i(f_i) = (P_{ind_i} + af_i^m) \cdot \frac{c_i}{f_i} = P_{ind_i} \cdot \frac{c_i}{f_i} + a \cdot c_i \cdot f_i^{m-1} \tag{3.2}$$

Although DVS can reduce energy consumption because of the reduced frequency-dependent active power P_d at reduced processing frequencies, the application will take more time to complete at low frequencies. As a result, more energy may be consumed because of the prolonged device active times (due to frequency-independent active power P_{ind}). Therefore, considering the system-level power, lower frequencies may not be always best for energy savings and it is shown that there exists a minimum energy-efficient voltage/frequency pair. In [5], the energy-efficient frequency of a task T_i for the power model is given as:

$$f_{ee_i} = \sqrt[m]{\frac{P_{ind_i}}{(m-1)a}} \tag{3.3}$$

Hence, it is not energy-efficient to run any task at a frequency below f_{ee_i} even when the timing constraints allow; doing otherwise would result in higher energy consumption. In fact, if f_{ee_i} exceeds the maximum available frequency level f_{max}, then the system should not reduce its speed below f_{max} [5].

The total energy consumption of the application with n tasks over a given time interval is:

$$E_{total} = \sum_{i=1}^{n} E_i(f_i) \tag{3.4}$$

Finally, in settings where DPM is also applicable, the total energy expression (3.4) needs to be augmented by E_{tr}, which corresponds to the total energy consumption due to device state transitions.

3.3 Fault and Reliability Models

This dissertation focuses on transient faults which are reported to be much more common than permanent faults [16, 33, 34]. Traditionally, transient faults have been modeled by Poisson distributions with the average arrival rate λ [77]. However, considering the effects of voltage scaling on transient faults, the average rate λ will depend on system processing frequency and supply voltage. In the analysis and simulations, this dissertation focuses on the exponential fault rate model proposed in [91] and again used in prior RA-PM research [88, 89, 91]:

$$\lambda(f) = \lambda_0 \, 10^{\frac{q(1-f)}{1-f_{min}}} \tag{3.5}$$

where the exponent q (> 0) is a constant, indicating the sensitivity of fault rates to voltage scaling, while λ_0 is the average fault rate corresponding to the maximum frequency $f_{max} = 1$ (and supply voltage V_{max}). That is, reducing the supply voltage and frequency for energy savings results in exponentially increased fault rates. The maximum average fault rate is assumed to be $\lambda_{max} = \lambda_0 10^q$, which corresponds to the lowest frequency f_{min} (and the supply voltage V_{min}).

The *reliability* of a task is defined as the probability of completing the task successfully (i.e. without encountering errors triggered by transient faults). Assuming that the transient faults follow a Poisson distribution, the reliability of the task T_i with the worst-case number of cycles c_i at the frequency level f_i is [91]:

$$R_i(f_i) = e^{-\lambda(f_i) \cdot \frac{c_i}{f_i}} \tag{3.6}$$

where $\lambda(f_i)$ is defined as in (3.3). The system's *original* reliability is the one observed when the voltage and frequency scaling is disabled (i.e when each task executes at the maximum frequency level f_{max}) . The overall reliability of a real-time system depends on the correct execution of all tasks in the application. As a result, for n independent tasks in this application model, the system original reliability is given as $\prod_{i=1}^{n} R_i^0$, where $R_i^0 = R_i(f_{max}) = e^{-\lambda_0 c_i}$ represents the *original reliability* of T_i (also called original task-level reliability). Note that if a scheme maintains all the original task-level reliabilities, the system's original reliability will be automatically preserved.

This dissertation assumes that transient faults are detected by using *sanity* (or, *consistency* checks) at task completion points [50]. If the outcome of the test is positive, the results are committed. Otherwise, the results are not committed, and in settings where the recovery is employed, it takes place through the *re-execution* of the task *at the maximum frequency.* The overhead for fault detection can be incorporated into the worst-case execution times of tasks [90].

Chapter 4: Reliability Maximization for Energy-Constrained Real-Time Embedded Systems

4.1 Introduction

While existing research explored various aspects of the interplay between DVS, reliability and energy consumption from *energy-aware* operation point of view, to the best of my knowledge, *maximizing reliability in energy-constrained settings* (with hard energy constraint) has not been studied before. This chapter assumes energy-constrained operation settings [2, 17, 42, 54, 71] where the system's energy consumption must not exceed a given bound. By modelling the transient fault rates as a function of task voltage/frequency levels, this chapter considers and optimally solves the problem of determining task level frequency assignments to maximize overall reliability (i.e. the probability of completing the application successfully), without exceeding the given energy budget/allowance or missing the deadlines. The findings of this research are published in [81, 83].

I first address and solve the *static* version of the problem under the frame-based task model, where all tasks execute their worst-case workload (Section 4.2.1) within the same given deadline/period. Then, this chapter extends the framework to dynamic settings and propose three on-line reclaiming algorithms that detect early completions and adjust the task frequencies at run-time, with the objective of improving the application's reliability by making the best use of excess energy at run-time (Section 4.2.2). Next, these solutions are extended to the periodic task model where tasks may have different periods (Section 4.3). This chapter shows that it is always possible to find an optimal solution with maximum reliability where all the jobs of a given periodic task run at the same frequency throughout the hyperperiod. This important result, in turn, allows to re-use the static optimal solution

I developed for the frame-based systems. Next, dynamic reclaiming algorithms are provided for periodic tasks to further improve the reliability at run-time by using excess energy that arises from early completions to speed up task executions. The experimental evaluation (presented in Section 4.2.3 and Section 4.3.3) indicates that the new dynamic algorithms (for both frame-based and periodic task models) perform comparably to a clairvoyant optimal scheduler that knows the exact workload in advance. To the best of my knowledge, this research effort is the first to address and optimally solve the problem of maximizing the overall reliability for *energy-constrained* real-time embedded applications.

4.2 Reliability Maximization for Frame-based Tasks

The analysis starts with the case of frame-based tasks where tasks share a common deadline d_f and each task has its worst-case execution cycles c_i. I first provide the static solution under the assumption that all tasks will take their worst-case workload, and then develop dynamic solutions where task frequencies are adjusted at run-time to improve overall reliability when the actual workload deviates from the worst-case.

4.2.1 Static Solution

In this section, I formalize and optimally solve the problem of finding task-level frequency assignments to maximize the system reliability, with a given energy budget E_f during a period d_f.

I first consider the impact of time and energy overhead [47] due to DVS at run-time on the problem formulation. Let Δ and ψ be the maximum transition time and energy overhead between any two frequency/voltage levels, respectively. In a given frame, during the execution of task set $T_1, ..., T_n$, the system's voltage/frequency changes at most n times (at task dispatch times). As a result, the total time overhead due to voltage/frequency scaling within a frame is bounded by $n \cdot \Delta$. Similarly, the total energy dissipation due to transitions is bounded by $n \cdot \psi$. Hence, it is safe to assume that the time and energy available for task execution within a frame is given by $d = P_f - (n \cdot \Delta)$ and $E = E_f - (n \cdot \psi)$, respectively. For simplicity, in the rest of the chapter, I refer to d and E as (modified) deadline and energy budgets.

Recall that the probability of completing the task T_i without a fault (that is, its *reliability*) at the processing frequency f_i is $R_i(f_i) = e^{-\lambda(f_i) * \frac{c_i}{f_i}}$, where $\lambda(f_i)$ is given in fault model. Hence, $R_i(f_i)$ is a strictly concave and increasing function of f_i. The total (i.e. frequency-dependent and frequency-independent) energy consumption of T_i at the frequency f_i can be expressed as [87]:

$$E_i(f_i) = P_{ind_i} \frac{c_i}{f_i} + a c_i f_i^2 \tag{4.1}$$

Notice that $E_i(f_i)$ is a strictly convex function and is minimized when $f_i = f_{ee_i}$ (Section 3.2).

Let $\varphi_i(f_i) = \lambda(f_i) \cdot \frac{c_i}{f_i}$. The problem can be stated as *to find* f_i $(1 \le i \le n)$ *values so as to maximize*:

$$R = \prod_{i=1}^{n} R_i = e^{-\sum_{i=1}^{n} \varphi_i(f_i)} \tag{4.2}$$

Subject to:

$$\sum_{i=1}^{n} \frac{c_i}{f_i} \le d \tag{4.3}$$

$$\sum_{i=1}^{n} E_i(f_i) \le E \tag{4.4}$$

$$f_{min} \le f_i \le f_{max} \; (1 \le i \le n) \tag{4.5}$$

Above, the inequality (4.3) corresponds to the deadline constraint, while (4.4) encodes the hard energy constraint. The constraint set (4.5) gives the range of feasible frequency assignments.

Considering the well-known features of the exponential functions, I can re-express the objective as to *minimize* $\sum_{i=1}^{n} \varphi_i(f_i)$ subject to the constraints (4.3), (4.4) and (4.5). In the rest of the chapter, this optimization problem will be called *Energy-Constrained Reliability Management* (ECRM) problem.

Let E_{limit} be the minimum energy that must be allocated to the given task system to allow their completion before or at the deadline d. Given the task parameters, E_{limit} can be computed by the polynomial-time algorithm developed in [5]. As a by-product, the same algorithm yields also the optimal task-level frequency assignments $(fl_1, fl_2, ..., fl_n)$ when the total energy consumption is *exactly* E_{limit}. Obviously, if $E < E_{limit}$, then there is no solution to this problem, since the system would lack the minimum energy needed for timely completion. Also, let $E_{max} = \sum_{i=1}^{n} E_i(f_{max})$ be the energy consumption of the task set when all tasks run at f_{max}. As another boundary condition, when $E \geq E_{max}$, executing all tasks at the maximum frequency is the optimal solution (since $R_i(f_i)$ is monotonically increasing with f_i and the system has sufficient energy to run at f_{max} continuously). Therefore, in the remaining of the chapter, I will focus exclusively on settings where $E_{limit} \leq E < E_{max}$.

Lemma 1. *In the optimal solution to ECRM,* $\forall i \quad f_i \geq f_{ee_i}$ *where* $f_{ee_i} = (\frac{P_{ind_i}}{2a})^{\frac{1}{3}}$ *is the energy-efficient frequency for* T_i.

Proof. This follows from the observation that executing T_i at a speed lower than f_{ee_i} would result in *more energy* consumption for T_i (Section 3.2). As a result, if a task were to execute at a frequency lower than f_{ee_i} in the optimal solution, then increasing its frequency to f_{ee_i} would actually *decrease* its energy consumption (while still satisfying (4.4)) and *increase* the overall reliability considering the positive impact of higher frequencies on task reliability – giving a contradiction. $\square$

Thanks to Lemma 1, the constraint (4.5) can be re-written as $f_{low_i} \leq f_i \leq f_{max}$ $(1 \leq i \leq n)$ where f_{low_i} is $\max(f_{min}, f_{ee_i})$.

Lemma 2. *If* $E_{limit} \leq E < E_{max}$*, in the optimal solution to ECRM, the total energy consumption* $\sum_{i=1}^{n} E_i(f_i)$ *must be exactly equal to* E.

Proof. Assume that the statement is false. Since $E < E_{max}$ by assumption, there must be a speed $f_i < f_{max}$. In this case, it should be possible to increase f_i by $\varepsilon > 0$ such that $(f_i + \varepsilon) \leq f_{max}$ and $\sum_{j \neq i} E_j(f_j) + E_i(f_i + \varepsilon) \leq E$. It is clear that the deadline and energy constraints are still satisfied after this modification. Further, the overall system reliability has improved due to the execution of T_i at a speed higher than f_i. Thus, the proposed solution cannot be optimal and a contradiction is reached. $\square$

Lemma 2 allows to conclude that if $E_{limit} \leq E < E_{max}$, then the constraint (4.4) can be re-written as an equality in the form of $\sum_{i=1}^{n} E_i(f_i) = E$. Consequently, I obtain a new non-linear (convex) optimization problem ECRM', defined as: *find* f_i $(1 \leq i \leq n)$ *values so as to minimize*:

$$\sum_{i=1}^{n} \varphi_i(f_i) \tag{4.6}$$

Subject to:

$$\sum_{i=1}^{n} \frac{c_i}{f_i} \leq d \tag{4.7}$$

$$\sum_{i=1}^{n} E_i(f_i) = E \tag{4.8}$$

$$f_{low_i} \leq f_i \leq f_{max} \; (1 \leq i \leq n) \tag{4.9}$$

The problem ECRM' can be solved, for instance, by *Quasi-Newton* techniques developed for constrained non-linear optimization [9]. The technique exploits the well-known Kuhn-Tucker optimality conditions for non-linear programs in an iterative fashion by transforming the original problem to a quadratic programming problem and solving it optimally [9]. While optimal, a theoretical complication with this approach is that it is practically impossible to *express* the maximum number of iterations as a function of the number of *unknowns* which, in this case, corresponds to the number of tasks n. However, in my experience, the algorithm is rather fast: for instance on a 1 GHz CPU with 1 GB memory, the implementation was able to return the optimal solution in less than 1.8 seconds even for task sets with 1000 tasks. The reported experimental results are based on using this optimal algorithm.

However, I also developed a heuristic algorithm that provably runs in polynomial-time. This algorithm, named ECRM-LU, satisfies the deadline, energy and frequency range constraints. Further, it yields solutions that are extremely close to the optimal solution. ECRM-LU proceeds as follows. I temporarily ignore the deadline constraint (4.7) and solve the problem ECRM' only by considering the energy constraint (4.8) and frequency range constraints (4.9). Notice that, by excluding the deadline constraint, the problem is transformed to a separable convex optimization problem with n unknowns, $2n$ inequality constraints and a single equality constraint. This problem, in turn, can be solved in time $O(n^3)$ by iteratively manipulating the Kuhn-Tucker optimality conditions in a way similar to the technique illustrated in algorithm given in [5]. Now, if this solution satisfies also the deadline constraint (4.7), obviously, it is also the solution to ECRM'. Otherwise, I re-write the constraint set (4.9) as:

$$fl_i \leq f_i \leq f_{max} \;\; (1 \leq i \leq n) \tag{4.10}$$

where fl_i is the frequency assignment to task T_i in the solution where all tasks complete *at exactly* d and with energy allocation E_{limit}. Recall that $\{fl_i\}$ values can be also computed in time $O(n^3)$. By enforcing the constraint set (4.10), I make sure that the final speed assignments satisfy also the deadline constraint. Once again, this version of the problem where the deadline constraint is handled implicitly by enforcing the lower bounds on frequency assignments can be solved in time $O(n^3)$. Hence, the overall time complexity of ECRM-LU is also $O(n^3)$. The extensive simulation studies show that ECRM-LU performs very well compared to the optimal solution through the almost entire spectrum: the reliability figures yielded by ECRM-LU are close to the optimal one by a margin of 0.03%, when $\frac{E}{E_{limit}} \geq 1.02$. In a tiny portion of the interval where $1.0 \leq \frac{E}{E_{limit}} < 1.02$, I observed a difference of at most 1%.

4.2.2 Dynamic Extensions

The static solution presented in the previous section is optimal under the assumption that all tasks will present their worst-case workload (i.e. their WCCs). While provisioning for worst-case scenarios is imperative in real-time systems, in practice, many real-time tasks complete early without consuming their WCCs. In fact, numerous DVS studies published in recent past were based on detecting and reclaiming unused CPU time (i.e. the dynamic *slack*) to enhance energy savings by reducing the processing frequency at run-time [6, 7, 48]. A similar opportunity exists here: the *excess energy* that arises from early completions of tasks, can be used to *increase* the speeds of tasks at run-time, to improve the system reliability. Clearly, utmost care must be exercised to make sure that the system remain within its energy allowance (budget), before making such adjustments.

In this section, I develop on-line (dynamic) reliability-aware schemes for reclaiming the excess energy at run-time. In the following algorithms, I assume that n tasks in the real-time embedded application are executed in the order $T_1, T_2, ..., T_n$. The three dynamic algorithms that I developed are:

- BR: dynamic *basic reclaiming* algorithm. In this solution, an initial speed assignment is made by solving the static problem presented in the preceding section, assuming worst-case workload for each task. At task completion points, the excess energy that may be available (due to early completions) is effectively re-cycled within the system: a new speed (frequency) assignment is made for the *remaining* tasks by considering the remaining (updated) energy budget and time to deadline. This assignment is obtained by re-invoking my optimal solution to the problem ECRM.

- GRE: dynamic *greedy* algorithm. Although BR satisfies the energy and deadline constraints, it is pessimistic in the sense that it assumes WCCs for all tasks when re-distributing the excess energy. An alternative approach may be to allocate the excess energy *entirely* to the next task[1] T_{next} at each task completion point, relying on the

[1]Note that if the next task cannot be assigned the entire excess energy due to the maximum frequency limitations, then the following task(s) will be able to reclaim energy at the next task completion points.

fact that T_{next} is also likely to complete early and release excess energy for the use of the remaining tasks. In the mean time, the reliability of T_{next} will be significantly improved due to execution at high speed. *GRE* preserves feasibility in terms of timing and energy constraints: compared to the initial static solution obtained by ECRM, the task speeds never decrease (guaranteeing the timely completion) and only the excess energy that is obtained at run-time is re-assigned.

- *AGR*: dynamic *aggressive* algorithm. This scheme represents the most speculative solution, in the sense that it counts on *probable* early completions before execution, and makes speed assignments accordingly. The main idea is to aggressively give sufficient energy to the current task by still leaving minimum required energy for the tasks to follow, based on their WCCs. It is speculative, because under a worst-case scenario (where most of the tasks present high workload), many tasks in the chain would be forced to execute at low speeds to guarantee the completion within the energy budget, significantly lowering the overall reliability. However, in settings where the actual workload is likely to deviate from the worst-case with high probability, this strategy will (and, as I show in the performance evaluation section, *does*) pay off. Specifically, *AGR* is implemented through the following steps: First, E_{limit} (which is defined as the minimum energy needed to complete all the tasks by their deadline (Section 4.2.1) is computed. Next, a speed assignment $(fl_1, fl_2, \ldots, fl_n)$ based on WCCs of all tasks but with energy allocation equal to E_{limit} is computed. $T_2, \ldots, T_n$ are *tentatively* assigned the frequencies $(fl_2, \ldots, fl_n)$ and their energy consumption with these assignments and WCCs ($E_{reserve}$) are evaluated. Then, the entire remaining energy (i.e. $E - E_{reserve}$) is allocated to the first task T_1, allowing it to execute as fast as possible within the given constraints. At task completion points, the above steps are repeated by considering the early completions and *actual* remaining energy for remaining tasks. The fact that this solution preserves the energy and deadline constraints follows from the properties of the speed assignments that corresponds to E_{limit}, which corresponds to the minimum energy needed for a feasible solution.

4.2.3 Simulation Results and Discussion

To evaluate the performance of the new dynamic algorithms under varying workload conditions, a discrete-event simulator in C was designed. In the simulator, I implemented *basic reclaiming scheme (BR)*, *greedy reclaiming scheme (GRE)*, *aggressive scheme (AGR)*, in addition to the *static* scheme which computes the processing frequencies using the optimal solution to problem ECRM assuming the worst-case workload for each task. *Static* does not use any on-line component in the sense that no dynamic speed adjustment is performed, regardless of the actual workload. Finally, the *clairvoyant scheme (Bound)* was also implemented. That scheme knows the *actual* workload of each task in advance and computes the optimal speed assignments to maximize the reliability by solving the problem ECRM accordingly. *Bound* is not a practical scheme (since it assumes the knowledge of the actual workload in advance); however, it characterizes the upper bound on the performance of *any* static or dynamic algorithm.

In the simulations, I considered 1000 task sets each containing 8 tasks, with the frame/period length of $d = 1000$. The total *utilization (U)* of each task set under maximum frequency is varied from 0.2 to 1.0 (full load). The worst-case execution time *(WCC)* of each task (under f_{max}) is randomly generated from $0.01 \cdot U \cdot d$ to $0.9 \cdot U \cdot d$. To model the variations in the actual workload, I use the ratio $\frac{ACC}{WCC}$, which denotes the ratio of the *average-case number of cycles (ACC)* to the worst-case number of cycles (WCC). The lower this ratio, the more the actual workload deviates from the worst-case. For each task set and utilization value, $\frac{ACC}{WCC}$ is changed from 0.2 to 1.0. The actual workload ac_i of each task is generated randomly, using normal distribution. Also, the confidence intervals at 97% confidence level are reported. The results that are shown correspond to the average of all runs.

This chapter models a DVS-enabled CPU where the normalized processing frequencies can change from $f_{min} = 0.1$ to $f_{max} = 1.0$. The power figures are normalized with respect to the frequency-dependent power component P_d, which is taken as $1.6W$ at the maximum frequency (modeled after the power consumption of Intel XScale as in [1,72]). This chapter models the frequency-independent power P_{ind} following the methodology and actual external device specifications from [19]. As in [19], this chapter assumes that each task uses up to 2 devices during execution, which gives a normalized frequency-independent power range of $[0, 2]$ per task. The specific P_{ind} value for each task is determined randomly according to uniform distribution in this range.

To analyze the impact of system's energy budget E on the performance, E was varied from E_{limit} (minimum energy needed to meet the deadline, see Section 4.2.1) to E_{max} (energy consumption at f_{max}). The ratio $\frac{E}{E_{limit}}$ shown in the plots, is a measure of the available energy in the system; for example, when $\frac{E}{E_{limit}} = 1.2$ the system has 20% more energy than the minimum needed to meet the deadline. This chapter assumes that the transient faults' occurrence is determined by Poisson distribution and given in fault model, where $q = 3$ and $\lambda_0 = 10^{-6}$. $\lambda_0 = 10^{-6}$ corresponds to 100,000 FITs (failures in time, in terms of errors per billion hours of use) per megabit at f_{max}, which falls to the reasonable range of the SER rate as suggested in [30, 93]. For each task, once the actual workload ac_i and its run-time frequency f_i is determined by the underlying algorithm at run-time, its reliability is computed as $R_i(f_i) = e^{-\lambda(f_i) * \frac{ac_i}{f_i}}$. The overall reliability of the task set is then derived as $R = \prod_{i=1}^{n} R_i(f_i)$.

Fig. 4.1 shows the relationship between the probability of failure *(PoF)* and the energy budget when $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$. As expected, *PoF* (defined as $1 - \mathit{reliability}$) generally decreases with increasing energy budget ($\frac{E}{E_{limit}}$ ratio), since more energy enables the system to use higher processing speeds with improved reliability. The clairvoyant *Bound* scheme achieves a constant probability of failure, because even when $E/E_{limit} = 1$, all tasks can be executed at f_{max} thanks to a priori knowledge of actual execution times, which are, on the average, half of the worst-case in these experiments – since processing speeds beyond f_{max} are not available, giving more energy to *Bound* does not further help. It can be observed that, among the other schemes, the static scheme (which does not perform any dynamic energy reclamation) performs worst and *AGR* is the best, with very close

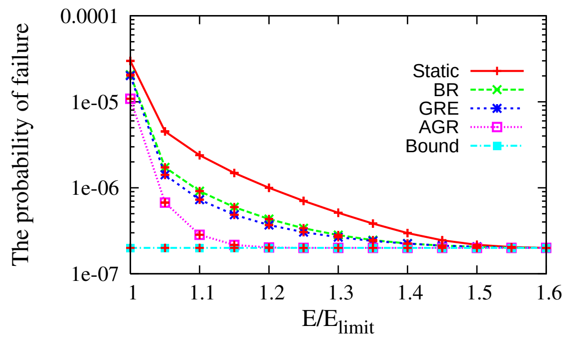


Figure 4.1: The probability of failure vs. E/E_{limit} with $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$

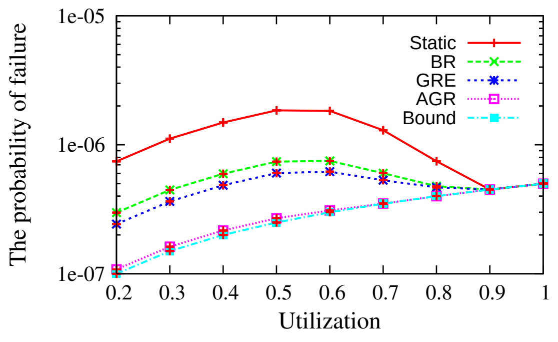


Figure 4.2: The probability of failure vs. utilization(U) with $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$

performance to *Bound* when ($\frac{E}{E_{limit}} > 1.1$). This result indicates that aggressively giving maximum energy to the tasks that will execute early typically pays off in these settings, since these tasks are also likely to generate excess energy due to early completions, which can be allocated to the later tasks. *BR* is a little worse than *GRE*; but both perform consistently better than *Static*, verifying the benefits of dynamic reclaiming. Observe that once the available energy is 40% or more compared to E_{limit}, all dynamic schemes perform almost the same.

Fig. 4.2 illustrates how the probability of failure changes as a function of task utilization U, for $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$. In general, the impact of increasing utilization is manifested in two different ways on the overall reliability. First, as the utilization increases, the workload (in terms of number of cycles) increases, translating into the increased probability of incurring transient faults, under comparable transient fault rates. Second, with increased utilization, the system *may* be forced to adopt higher frequencies to meet the deadline, depending on the available energy. These two factors tend to affect the overall reliability in reverse directions as implied by Equation 3.6. The relative ordering of the schemes remains the same as in Fig. 4.1. Two interesting trends are observed: For *Static*, *BR* and *AGR* schemes, as we increase the utilization towards the range of $0.5 - 0.6$, the probability of failure first increases. In the utilization range $[0.2 - 0.5]$, for these three schemes, the increase in frequency is modest and the *PoF* is primarily determined by the increase

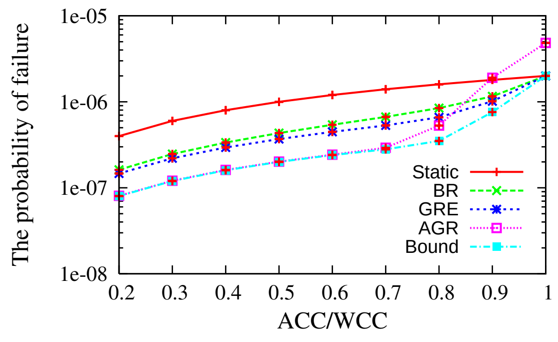


Figure 4.3: The probability of failure vs. the $\frac{ACC}{WCC}$ ratio with $U = 0.4$ and $E/E_{limit} = 1.2$

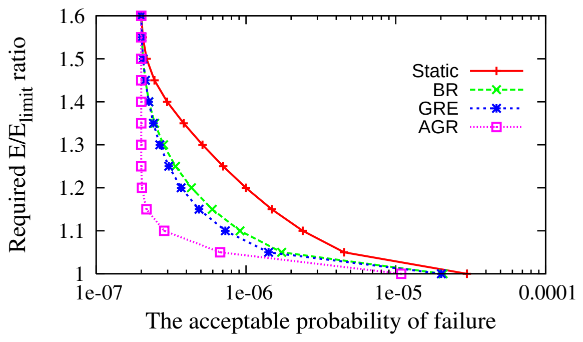


Figure 4.4: The acceptable probability of failure vs. the required E/E_{limit} ratio

in the workload. However, higher utilization values force the system to adopt significantly higher frequencies in order to meet the deadline and the positive impact of this on reducing the fault rates becomes the primary factor after a certain point. In fact, after $U = 0.8$, all tasks are executed at speeds close to f_{max} and then PoF drops sharply to a minimal value. On the other hand, the probability of failure for AGR and $Bound$ are quite low and it tends to increase monotonically (but relatively modestly) throughout the spectrum. This is because, due to low ratio of $\frac{ACC}{WCC} = 0.5$, AGR and $Bound$ can execute almost all the tasks with f_{max} while remaining within the energy budget, even starting at $U = 0.2$. Therefore, when executing all tasks with fixed speed f_{max}, the PoF for AGR and $Bound$ is mainly determined by the execution times, which increase with increasing utilization. Also, it can be noticed that when the utilization increases from 0.9 to 1.0, the PoF for all schemes increases since all the tasks can be executed with f_{max} starting from $U = 0.9$.

Fig. 4.3 shows the impact of the variability in the actual workload (i.e. the $\frac{ACC}{WCC}$ ratio) on the probability of failure, with $E/E_{limit} = 1.2$ and $U = 0.4$. In general, it can be found that the probability of failure increases with the increased ratio of $\frac{ACC}{WCC}$. This is to be expected, because with the increased ratio of $\frac{ACC}{WCC}$, at run-time, tasks execute longer and they are subject to transient faults with higher probabilities. However, observe that the dynamic schemes are able to significantly reduce the probability of failure compared to *Static* thanks to online reclaiming features, especially at low $\frac{ACC}{WCC}$ ratios. When $\frac{ACC}{WCC} = 0.9$, the probabilities of failure of *Static*, *BR* and *GRE* converge to that of *Bound*, since there are almost no early completions or excess energy at run-time. But, it is interesting to note that the probability of failure of AGR is slightly higher from $U = 0.9$. This is because, when all tasks almost take their WCC with the high utilization $U \geq 0.9$, the expected early completions typically do not occur, while the aggressive nature of AGR still forces the later tasks to execute at relatively low speeds, causing a loss in reliability.

These patterns can be also used to *establish guidelines* for system designers who need to figure out the *minimum amount of energy supply* that must be provided to the system, to achieve a certain target probability of failure. Fig. 4.4 establishes these thresholds for $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$. As can be seen from the figure, to achieve a target *PoF*, the amount of energy that must be supplied to the system is largest for *Static*, that cannot reclaim excess energy at run-time. This amount sharply decreases with the use of dynamic reclaiming algorithms, and is minimum for the best-performing algorithm AGR. For example, 20% additional energy (beyond E_{limit}) must be provided to the system to achieve a probability of failure of 10^{-6}, with *Static*. However, if dynamic schemes (e.g. *BR* or *GRE*) are available, then 7% additional energy is sufficient. The difference between the schemes becomes very important at *low* values for the target probability of failure (e.g. smaller than 10^{-6}); indicating that in safety-critical applications, the available energy budget may be a prime factor to achieve a target reliability figure.

Table 4.1: Dynamic Reclaiming Overheads

Scheme	4 Tasks	8 Tasks	16 Tasks
AGR	0.67ms	0.68ms	0.68ms
GRE	0.65ms	0.65ms	0.66ms
BR	1.6ms	2.9ms	5.5ms
BR^*	0.86ms	1.23ms	1.79ms

Run-time Characteristics: I also conducted experiments to measure the run-time overhead of dynamic reclaiming routines that require computation of excess energy and re-assignment of CPU frequencies. The results, presented in Table 4.1 for task sets of different sizes, show the extra CPU time used by *AGR, GRE* and *BR* at each invocation point. These results are obtained on a 1 GHz CPU with 1 GB of memory. *GRE* has the lowest overhead, by virtue of modifying the frequency of only one task at run-time. *AGR* has also a rather low overhead, which is different from *BR* which needs to re-solve the ECRM problem at each invocation, by considering the remaining energy. I also implemented an alternative version of *BR*, denoted by BR^*, that uses the polynomial-time algorithm ECRM-LU as the main optimization routine. The results suggests that BR^* is very effective in reducing the run-time overhead of the basic reclaiming scheme. Overall, it can be concluded that the dynamic reclaiming algorithms can be implemented with low overhead in practice.

4.2.4 Extension to dependent tasks given by DAGs

In this section, I discuss how to extend the above findings to dependent frame-based tasks. Specifically, this section considers a real-time application represented by a directed acyclic graph (DAG) $G = (V, E)$. The set of vertices (nodes) $V = \{T_1, ..., T_n\}$ represents the set of tasks with common period and deadline D_f. D_f is also called the frame-length. The set of edges $E = \{E_1, ..., E_m\}$ represents a partial order corresponding to the *precedence constraints* [14] among tasks, with the interpretation that whenever the edge $(T_i, T_j) \in E$, the task T_j cannot start to execute until T_i has been completed. I assume that, given the DAG, an execution order of tasks that satisfies the precedence constraints has been determined and that the task indices reflect the execution order in that sequence (i.e. T_i is the i^{th} task to execute in each frame). Such an execution order can be obtained, for example, through *topological sorting* algorithm in time $O(m + n)$ [14]. Once the execution order is obtained, the above solutions obtained for independent frame-based tasks can be directly applied. Fig. 4.5 presents an example task set with 6 tasks and precedence constraints. In Fig. 4.6, we show the execution order within a frame, obtained through topological sorting.

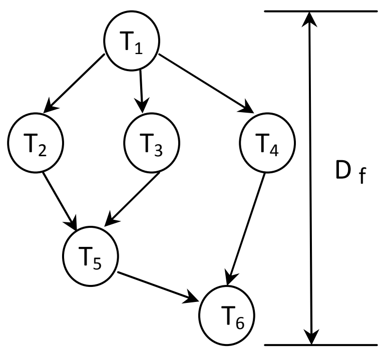


Figure 4.5: An example task set with precedence constraints

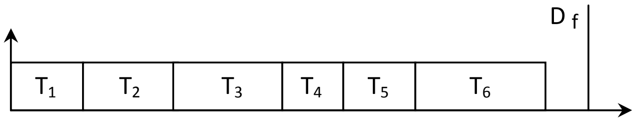


Figure 4.6: The task execution order within the frame

4.3 Reliability Maximization for Periodic Real-Time Tasks

In this section, the above solutions are extended to a more general model where tasks may have different periods. Specifically, I consider a set of periodic real-time tasks $\Gamma = \{T_1, .., T_n\}$. Under periodic task model, the effective system utilization depends on the execution frequencies of the tasks. Hence, the effective utilization of task T_i at frequency f_i as $U_i(f_i) = \frac{u_i}{f_i}$, where $u_i = \frac{c_i}{p_i}$ is the nominal utilization of the task T_i under maximum frequency. The (nominal) system utilization under maximum frequency ($\sum U_i(1.0) = \sum_{i=1}^{n} u_i$) is denoted by U here. Here, I assume preemptive Earliest-Deadline-First (EDF) scheduling, which is known to be an optimal real-time scheduling policy: with EDF, the necessary and sufficient condition for the feasibility of the task set is $U \leq 1.0$ [41]. Hence, throughout this section, it is assumed that the system utilization U when all tasks are executed at f_{max} never exceeds 100% in order to guarantee the feasibility.

4.3.1 Static Solution

For periodic real-time tasks, considering that it is sufficient and necessary to obtain an optimal solution during a hyperperiod (the least common multiple (LCM) of all the periods). As in Section 4.2.1, it is assumed that the available energy budget E during LCM has been adjusted by taking into account the total number of jobs $N = \sum_{i=1}^{n} LCM/p_i$ and transition energy overhead ψ. In the same vein, the transition time overhead can be factored in the worst-case execution time c_i of each job. Note that for periodic task sets, the overall reliability of the application is the product of *all* the jobs of all the tasks that are executed during the hyperperiod. Moreover, in general, each task instance T_{ij} (the j^{th} job of task T_i) may have its own frequency f_{ij} which could be potentially different than the frequency of other jobs of the same task T_i in the optimal solution. Considering that the total number of jobs during the hyperperiod could be exponential in the number of tasks, at first, the problem looks rather challenging since even the number of unknowns (f_{ij} values) is prohibitively large. Fortunately, the following theorem indicates that one can commit to *task-level* frequency assignments where a given task runs at constant speed f_i, without compromising optimality:

Theorem 1. *For periodic task sets, it is always possible to find an optimal solution where all the jobs of a given task T_i run at the same frequency throughout the hyperperiod.*

Proof. Consider an optimal solution with overall reliability R where the task instance (job) T_{ij} runs at frequency f_{ij}. Call this schedule S. I will show that one can transform S to another schedule S' where *i.* the feasibility is preserved, *ii.* the energy budget constraint is satisfied, *iii.* the overall reliability is no less than R, and, *iv.* all jobs of a given task T_i run at constant speed f_i. During the hyperperiod, there are $k_i = \frac{LCM}{p_i}$ jobs of task T_i. Now consider the schedule S' where T_i runs at constant speed $f_i = \frac{\sum_{j=1}^{k_i} f_{ij}}{k_i}$.

First, I show that S' is feasible with these frequency assignments. To start with, observe that $f_i \leq max\{f_{ij}\}$. If S is feasible, $max\{f_{ij}\} \leq 1.0$, and I get $f_i \leq 1.0$ implying that no task runs at a frequency higher than $f_{max} = 1.0$ in S'. Similarly, one can easily see that $f_i \geq min\{f_{ij}\} \geq f_{min}$. Moreover, since S is feasible, the total execution time during hyperperiod should not exceed the length of LCM, that is, $\sum_{i=1}^{n} \sum_{j=1}^{k_i} \frac{c_i}{f_{ij}} \leq LCM$. In other words, $\sum_{i=1}^{n} \sum_{j=1}^{k_i} \frac{c_i}{f_{ij}} \frac{1}{LCM} \leq 1$. Applying first $u_i = \frac{c_i}{p_i}$, and then $\frac{p_i}{LCM} = \frac{1}{k_i}$, it can be concluded that $\sum_{i=1}^{n} \frac{1}{k_i} \sum_{j=1}^{k_i} \frac{u_i}{f_{ij}} \leq 1$ holds.

On the other hand, the effective system utilization in S' is $U' = \sum_{i=1}^{n} \frac{u_i}{f_i}$. Due to convex nature of the function $U(f) = \frac{u}{f}$, $\frac{u_i}{f_i} \leq \frac{1}{k_i}\sum_{j=1}^{k_i} \frac{u_i}{f_{ij}}$ holds. Consequently, $U' = \sum_{i=1}^{n} \frac{u_i}{f_i} \leq \sum_{i=1}^{n} \frac{1}{k_i}\sum_{j=1}^{k_i} \frac{u_i}{f_{ij}} \leq 1$, giving $U' \leq 1$, which is necessary *and* sufficient for feasibility under EDF. As a result S' is still feasible.

Next, I show that, in S' the energy constraint is not violated. Let the energy consumption of schedules S and S' during LCM be X and X', respectively. Obviously, $X \leq E$ by assumption. Since $f_i = \frac{\sum_{j=1}^{k_i} f_{ij}}{k_i}$ and $E_i(f)$ is a strictly convex function, one can infer that the energy consumption of a given task T_i in S', namely $k_i \cdot E_i(f_i)$, does not exceed the energy consumption of the same task in S, namely $\sum_{j=1}^{k_i} E_i(f_{ij})$. Hence, I get $\sum_{i=1}^{n} k_i \cdot E_i(f_i) \leq \sum_{i=1}^{n}\sum_{j=1}^{k_i} E_i(f_{ij})$, giving $X' \leq X \leq E$ and showing that S' still satisfies the energy constraint.

Finally, along the same lines, I show that reliability R' of S' is no worse than the reliability R of the schedule S. Using the given parameters, I compute $R = \Pi_{i=1}^{n} R_i = \Pi_{i=1}^{n}\Pi_{j=1}^{k_i} R_{ij} = e^{-\sum_{i=1}^{n}\sum_{j=1}^{k_i}\varphi_i(f_{ij})}$, while the new reliability is $R' = \Pi_{i=1}^{n} R'_i = e^{-\sum_{i=1}^{n} k_i \varphi_i(f_i)}$. Once again, since φ_i is a convex function, $k_i \varphi_i(f_i) \leq \sum_{j=1}^{k_i} \varphi_i(f_{ij})$ holds due to the fact that $f_i = \frac{\sum_{j=1}^{k_i} f_{ij}}{k_i}$. As a result, in the new schedule, none of task-level reliabilities, which is the product of the reliabilities of individual jobs for a given task, decreases. This implies that the total reliability over all the tasks, R', cannot be worse than the original reliability R, completing the proof. □

Thanks to Theorem 1, the total energy consumption of T_i within *LCM*, at the optimal frequency level f_i can be expressed as:

$$E_i(f_i) = (P_{ind_i} + af_i^3)\frac{u_i}{f_i} LCM = (P_{ind_i} + af_i^3)\frac{c_i}{f_i} n_i \tag{4.11}$$

where $n_i = \frac{LCM}{p_i}$ is the number of jobs of T_i within a LCM. Therefore, the problem for the periodic tasks can be formally expressed *to find task-level frequency* f_i $(1 \leq i \leq n)$ *so as to* *maximize*:

$$R = \prod_{i=1}^{n} R_i = e^{-LCM \sum_{i=1}^{n} \frac{\varphi_i(f_i)}{p_i}} = e^{-\sum_{i=1}^{n} n_i \varphi_i(f_i)} \tag{4.12}$$

Subject to:

$$\sum_{i=1}^{n} \frac{u_i}{f_i} \leq 1 \tag{4.13}$$

$$\sum_{i=1}^{n} E_i(f_i) \leq E \tag{4.14}$$

$$f_{min} \leq f_i \leq f_{max} (1 \leq i \leq n) \tag{4.15}$$

where the inequality (4.13) encodes the feasibility constraint for EDF. Once again, by noting the monotonic features of the exponential functions, the objective can be re-expressed as to *minimize:* $\sum_{i=1}^{n} n_i \varphi_i(f_i)$ with the constraints (4.13), (4.14) and (4.15). This optimization problem is called *Periodic Energy-Constrained Reliability Management* (P-ECRM) problem.

To solve P-ECRM, similar to the solution for ECRM, I first need to find E_{limit} and E_{max}. Here, E_{limit} is re-defined as the minimum energy that must be allocated to the given periodic task set to preserve the feasibility with EDF during the hyperperiod, and E_{max} represents the energy consumption of the task set when all jobs run at f_{max}. With given task parameters, E_{limit} can be computed by the polynomial-time algorithm suggested in [5] and E_{max} is equal to $\sum_{i=1}^{n} E_i(f_{max})$. Obviously, if $E < E_{limit}$, then there is no solution to P-ECRM because E_{limit} is the minimum energy needed to guarantee the timing constraints. Likewise, when $E \geq E_{max}$, I can simply assign the maximum frequency to all the tasks without exceeding the energy budget E, to achieve the best system reliability. Therefore, to solve P-ECRM, one needs to focus mainly on the case where $E_{limit} \leq E < E_{max}$.

Let $\varphi'_i(f_i) = n_i \cdot \varphi_i(f_i)$, which is still a convex function. A close inspection of P-ECRM reveals that it has the same *mathematical form* as the ECRM problem introduced in Section 4.2.1, since task execution times and the deadline in ECRM are replaced by task utilizations and 1.0 (EDF's utilization bound). As a result, the same solution methodology suggested in Section 4.2.2 immediately applies.

4.3.2 Dynamic Solutions for Periodic Tasks

Similar to the case of frame-based tasks, the static solution is optimal only when all tasks execute their WCCs in every instance. Since early completions are rather common in actual executions, there exist again opportunities for improving overall reliability by speeding up execution at run-time by re-cycling the excess energy. In this section, three on-line (dynamic) reliability-aware schemes are suggested for reclaiming the excess energy that becomes available for periodic tasks. In all these algorithms, re-computation and re-allocation of excess energy are performed only at job completion points.

In fact, it is tempting to use the framework suggested in Section 4.2.1, by re-computing the new frequency assignments by considering *all* jobs yet-to-complete. However, unlike the case of frame-based tasks, under preemptive EDF scheduling one may have a number of *preempted* jobs in the ready queue that already started their execution in addition to jobs that will be released in the future, at reclamation points. These preempted jobs may have varying amounts of workload already completed under a prior frequency assignment. In new suggested schemes, to preserve run-time efficiency and to be able to use P-ECRM algorithm with slight modifications for task-level frequency assignment, such preempted jobs are not considered for excess energy allocation. Rather, excess energy is distributed only to the jobs that will be released in the future. In addition, Theorem 1 implies that a task-level frequency assignment gives an optimal re-distribution of excess energy to these jobs. Consequently, the proposed algorithms take into account the energy needed to allow the preempted jobs to complete their remaining workload at their current frequency assignments. The new proposed algorithms are:

- ***P-BR***: dynamic *basic reclaiming* algorithm for periodic tasks. Initially all frequency assignments are made by solving the P-ECRM problem under worst-case workload assumption. When a job completes without consuming its WCC, this excess energy is re-allocated to jobs that have not yet started their executions. The new task-level frequency assignments for these jobs can be again obtained by invoking the optimal solution to the problem P-ECRM with updated energy budget. When re-invoking the algorithm P-ECRM at run-time, one needs to only update n_i in expressions (4.11) and (4.12) for the number of future jobs of task T_i. Compared to the initial speed assignments, the new frequencies never decrease due to excess energy available on the run-time. As a result, the feasibility of the resulting schedule is guaranteed in *P-BR* scheme.

- ***P-GRE***: dynamic *greedy* algorithm for periodic tasks. *P-GRE* is an alternative approach that attempts to allocate the entire excess energy to the next task to execute (T_{next}), when a task instance completes. In other words, all future jobs of T_{next} will be given a new (higher) frequency and their reliability will be significantly improved. If T_{next} cannot use the entire excess energy due to the maximum frequency limitations, then the remaining part is allocated to another task at the next job completion point. Again, the feasibility of *P-GRE* follows from the fact that none of the tasks runs at a frequency level lower than its original frequency assigned by the static solution.

- ***P-AGR***: dynamic *aggressive* algorithm for periodic tasks. This scheme extends the speculative *AGR* scheme proposed for the frame-based tasks, by speeding up all the future jobs of the next task to be dispatched as much as possible, by reserving only a minimum amount of energy for instances of other tasks. First, the algorithm computes E_{limit} and stores the frequency assignments $(fl_1, fl_2, \ldots, fl_n)$ obtained through E_{limit}. Then, at reclamation point, if a job task T_i is about to start its execution for the first time, the frequency is re-computed for the jobs of the remaining tasks $\Gamma - T_i$ by evaluating the speed assignments with E_{limit} and their energy consumption with these assignments and WCCs ($E_{reserve}$) is evaluated. Then, the entire remaining energy (i.e. $E - E_{reserve}$) is allocated exclusively to the current and future job instances of the task T_i, allowing it to execute as fast as possible within the given constraints. This solution also preserves the energy and feasibility constraints, which can be seen from

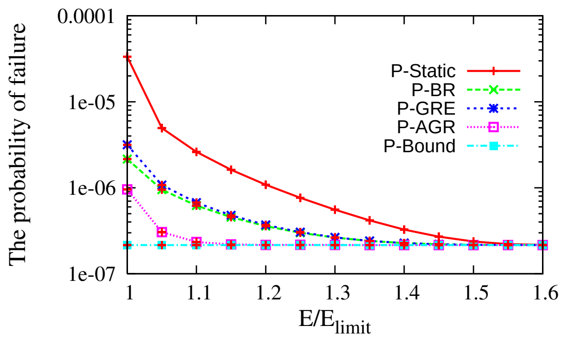


Figure 4.7: The probability of failure vs. E/E_{limit} with $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$

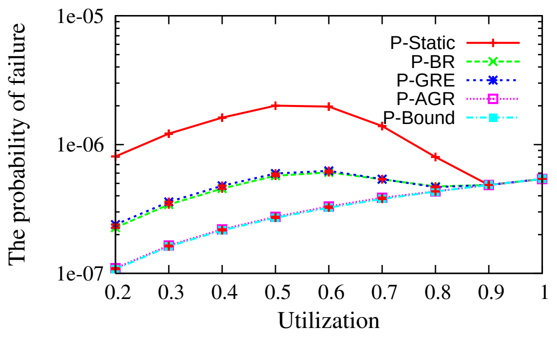


Figure 4.8: The probability of failure vs. utilization(U) with $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$

the properties of the frequency assignments that correspond to E_{limit} (the minimum energy requirement for a feasible solution).

4.3.3 Simulation results and discussion

In this section, I evaluate the performance of the new dynamic reclaiming schemes proposed for periodic tasks. In addition to the *Static Scheme (P-Static)*, I implemented in the simulator *P-BR, P-GRE*, and *P-AGR*. I also implemented the *Clairvoyant Scheme (P-Bound)* that computes an absolute bound on the optimal solution, by using the P-ECRM algorithm and the actual workload information *in advance*.

The experimental methodology followed in this section is parallel to that described in Section 4.2.3. I generated 1000 task sets each with 8 tasks. The periods of the tasks are generated randomly in the range $[72, 1080]ms$, which are comparable to task period values encountered in real applications [43]. I made sure that the least common multiple period LCM of the periods is 1080 for every task set. This allows to compare different task sets over the same operation period $LCM = 1080$ for various utilization and energy budget values. The confidence intervals at 97% confidence level are reported.

Fig. 4.7 shows how the probability of failure changes with increasing energy budget when $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$. First, it is observed the same decreasing trends for probability of failure with increasing energy budget and same relative order of reliability among these schemes as those in Fig. 4.1 obtained for frame-based tasks. Compared to Fig. 4.1, there are two main differences: First, *P-BR* provides a slightly better reliability performance compared to *P-GRE.* In the case of periodic tasks, there are typically a large number of jobs that can benefit from dynamic reclamation during the hyperperiod, and a more balanced energy allocation that considers all the tasks tends to be better than the greedy approach. Second, the performances of three dynamic schemes show further improvements with respect to the static scheme *P-Static*, even when $\frac{E}{E_{limit}} = 1$. With new dynamic schemes, the more jobs complete early, the more jobs in the future can be executed at the higher frequencies (occasionally, even at f_{max}) due to the dynamic reclaiming. In fact, even when $E = E_{limit}$, many jobs complete early when $\frac{ACC}{WCC} < 1.0$. Hence, the advantages of dynamic schemes over the static scheme *P-Static* become more pronounced with increased number of jobs that translate to increased number of reclamation/speed-up points.

Fig. 4.8 illustrates the relationship between the probability of failure and task set utilization U for $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$. Again, the main trends observed for frame-based tasks can be found in the case of periodic tasks. Specifically, increasing the utilization has potential impact on two factors that affect the reliability differently: the increase in task workloads tends to increase *PoF*; but for cases where the system has to use significantly higher frequencies, this becomes the dominant factor and *PoF* drops. The latter case can be observed in the utilization range $[0.5 - 0.9]$ for *P-Static, P-BR* and *P-GRE.* A notable difference compared to the experiments with frame-based tasks is that the performance of *P-AGR* is almost identical to that of *P-Bound.* The reason is that with large number of jobs during the hyperperiod, there typically are sufficient energy to aggressively assign the current job a frequency close to f_{max}, which approaches that assigned by *P-Bound* through the advance knowledge of actual workload. Due to the jobs' very frequent early completions (recall that $\frac{ACC}{WCC} = 0.5$ in these experiments), the speculative strategy of *P-AGR* pays off.

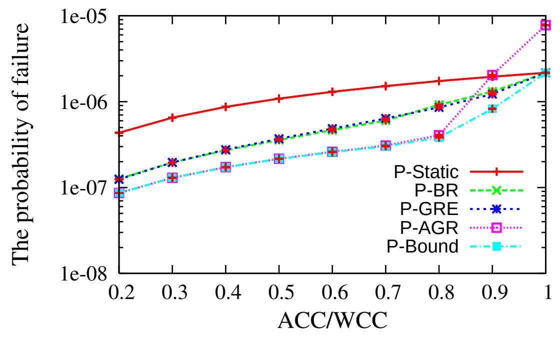


Figure 4.9: The probability of failure vs. the $\frac{ACC}{WCC}$ ratio with $U = 0.4$ and $E/E_{limit} = 1.2$

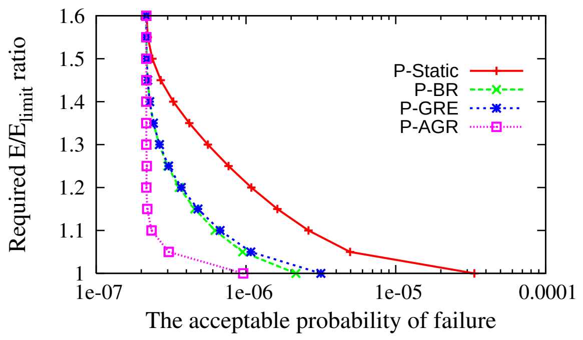


Figure 4.10: The acceptable probability of failure vs. the required E/E_{limit} ratio

Fig. 4.9 shows the impact of the variability in the actual workload (i.e. the $\frac{ACC}{WCC}$ ratio) on the probability of failure, with $E/E_{limit} = 1.2$ and $U = 0.4$. In general, similar to the results presented in Fig. 5.3 for frame-based tasks, under fixed utilization, the probability of failure is primarily determined by the effective workload, which tends to increase with larger $\frac{ACC}{WCC}$ ratios. This results in monotonically increasing PoF values. Compared to the results for frame-based tasks, the difference between static schemes and three new suggested dynamic schemes becomes bigger, which verifies that new dynamic schemes can achieve better reliability for periodic tasks since there are more available energy to be dynamically reclaimed due to early completions of numerous jobs during the hyperperiod. An interesting observation is that the almost perfect performance of *P-AGR* degrades rapidly when $\frac{ACC}{WCC} > 0.8$, implying that it should be used only when there is great confidence about high variability in the actual workload. With settings $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$, Fig. 4.10 shows the required minimum energy supply to achieve a target (acceptable) probability of failure. As one can see, if the system's acceptable probability of failure is 10^{-6}, it is necessary to allocate 20% additional energy (beyond E_{limit}) when using *Static* scheme. However, when dynamic schemes (e.g. *P-BR* or *P-GRE*) are applied, then 5% additional energy is sufficient. Furthermore, when using *P-AGR*, as long as the likelihood of early completions is high (such as the case here with $\frac{ACC}{WCC} = 0.5$), there is practically no need to allocate energy beyond E_{limit} if the target probability of failure is less than 10^{-6}. I also repeated the

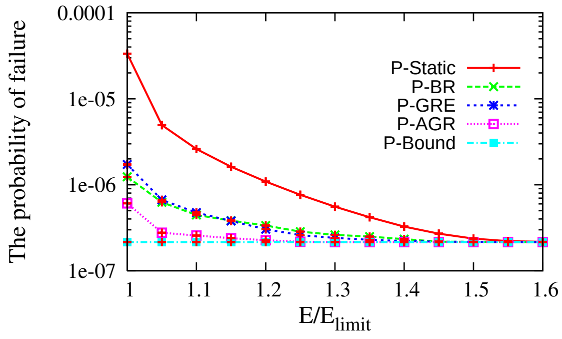


Figure 4.11: The probability of failure vs. E/E_{limit} with $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$ ($LCM = 2160$)

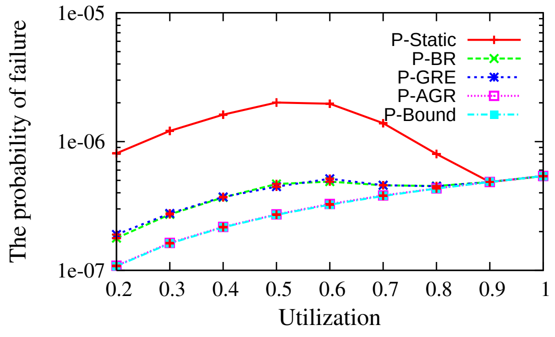


Figure 4.12: The probability of failure vs. utilization(U) with $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$ ($LCM = 2160$)

experiments for a larger LCM, namely for $LCM = 2160$. The results are quite similar to those reported above. Due to space limitations, I only report the impact of energy budget and utilization on the probability failure, in Fig. 4.11 and Fig. 4.12, respectively.

To summarize, these results indicate that new dynamic schemes can achieve even better performance compared to the static schemes in the case of periodic tasks, because there are more dynamic energy reclamation opportunities with the larger number jobs that execute within the hyperperiod.

Table 4.2: Dynamic Reclaiming Overheads

Scheme	4 Tasks	8 Tasks	16 Tasks
P-AGR	0.68ms	0.69ms	0.70ms
P-GRE	0.66ms	0.66ms	0.66ms
P-BR	1.9ms	3.4ms	6.5ms
*P-BR**	0.98ms	1.42ms	2.13ms

Run-time Characteristics: The measurements to explore the run-time overhead of dynamic reclaiming algorithms for periodic task sets, presented in Table 4.2, yielded figures similar to those obtained for frame-based tasks (Table 4.1). Again, the numbers correspond to average CPU time used by the reclamation routines at each invocation (job completion) point. *P-BR* a has a larger overhead compared to *BR*, primarily due to the need of solving P-ECRM online for large number of jobs. However, it is noted that its overhead can be reduced by employing the ECRM-LU algorithm (adapted to periodic settings) as the optimization routine. That variation is denoted as $P\text{-}BR^*$ in Table 4.2.

4.4 Chapter Summary

Recent research has identified significant and negative impact of the popular energy management technique DVS on the reliability of real-time embedded systems. This chapter considered both frame-based and periodic task models and showed how to compute frequency assignments (which translate to task-level energy allocations) to maximize the overall reliability, while satisfying a hard energy constraint. By assuming a frequency value that can vary from a lower bound to an upper bound, this chapter first presented the static optimal scheme. Then, the framework was extended with several online (dynamic) schemes that exploit early task completions and speed up task executions at runtime, to improve overall reliability. All new suggested static and dynamic solutions guarantee both timing and energy budget constraints in all execution scenarios. Moreover, the experimental results indicate that new suggested algorithms perform comparably to a clairvoyant optimal scheduler that can maximize overall reliability thanks to its prior knowledge about the exact workload.

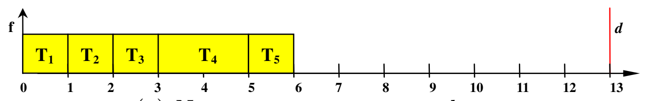
Chapter 5: Reliability-Aware Energy Management through Shared Recovery Technique

5.1 Introduction

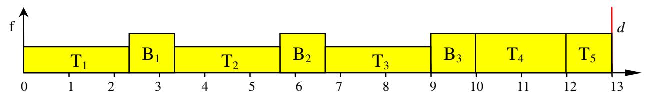
A line of research, called *reliability-aware power management (RA-PM)*, was first introduced in [87]. The central idea behind RA-PM is to schedule a *recovery task* that may be invoked at run-time, in case that the application, whose execution frequency is scaled down through DVS, incurs a transient fault [87]. Specifically, when a transient fault is detected at the end of task execution, a recovery takes place in the form of re-execution at the maximum frequency before the task deadline. Based on this principle, several schemes are proposed for frame-based [88] and general periodic [89] tasks. A main feature of the existing RA-PM schemes is that the algorithms first reserve some portion of the system slack (CPU time) for potential recovery of the tasks that may be subject to transient faults when executed with voltage scaling. Only then, the remaining slack is used to determine the task-level low processing frequencies for energy management.

This part of the dissertation is motivated by the observation that the existing RA-PM schemes are *conservative*, in the sense that statically allocating the available slack for *multiple recovery blocks of a pre-determined set of tasks reduces the prospects for energy savings* by decreasing the available slack for DVS. Instead, the novel solution is based on allocating a **shared recovery block/slack** that can be used by *any* faulty task at run-time. This, in turn, improves energy savings. The findings of this chapter are published in [80, 82].

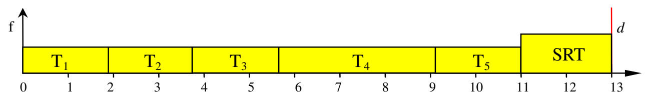
To illustrate this point, consider a frame-based real-time application that consists of five tasks and a common period/deadline of 13. The worst-case execution times of T_1, T_2, T_3 and T_5 under maximum frequency are given as 1 time unit each, while that of T_4 is 2.



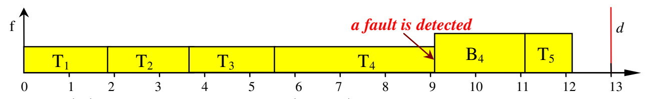
(a) No power management scheme



(b) Greedy (GRE) scheme



(c) Shared recovery (SHR) scheme without faults



(d) Shared recovery (SHR) scheme with a fault

Figure 5.1: The motivational example

As can be seen in Figure 5.1(a), the system potentially has access to $13 - 6 = 7$ units of slack when all tasks execute at the maximum frequency. If we consider one of the existing RA-PM schemes (e.g. the greedy (GRE) scheme proposed in [88]), three *separate* recovery tasks (denoted by $\{B_i\}$) will be statically allocated to T_1, T_2 and T_3, leaving 4 units of total slack for voltage scaling of these three tasks and yielding the RA-PM schedule depicted in Figure 5.1(b). Applying the power model from [88] indicates that the energy consumption of the GRE solution is $0.74 \cdot E$ where E is the energy consumed when all tasks run at the maximum frequency (no power management (NPM) case). It can be also shown that since all tasks that are scaled down have recovery tasks, each task's reliability is no worse than the one in the NPM case (Figure 5.1(a)). One may further verify that the system reliability, which is the product of the individual task reliabilities [88,89], actually improves in the GRE solution to a level of 99.99998% when using the fault rate model from [88].

On the other hand, the solution in this chapter is based on assigning a *shared recovery block* that can be used for *any* fault detected in any scaled task. That is, instead of reserving slack for separate recovery tasks in advance, the new scheme (called *shared recovery (SHR)* scheme) will only provision for any single recovery that may be needed at run-time. The amount of CPU time reserved for this recovery will be set to that of the largest recovery/re-execution time that may be needed for any scaled task. Just like the original RA-PM schemes [87, 92], this chapter assumes that the recovery, if needed, will take place at the maximum frequency. As a result, with the new SHR scheme, the system will typically have access to a larger slack for slowdown and energy savings, after the CPU time reservation for the shared recovery. Re-visiting the motivational example, it is noted that the SHR scheme will reserve a recovery slack of only 2 time units, which is the maximum needed by any recovery. This essentially gives the system an opportunity to use 5 units of slack for, and further, manage *all* the five tasks through, DVS. It can be verified that the SHR solution given in Figure 5.1(c) will result in an energy consumption of $0.51 \cdot E$, an improvement of 31% over the traditional GRE scheme.

Obviously, another important objective is still to preserve the *original system reliability*. It is clear that as long as faults do not occur, each scaled task will have access to a recovery slack that is large enough for re-execution at its dispatch time, in the new scheme. However, one can see that after a fault occurs and the shared recovery block has been executed, if all the remaining tasks are still executed at the originally-computed reduced frequency levels, their task-level reliability will not be preserved. For this reason, the SHR scheme switches to the maximum CPU frequency until (and only until) the end of the frame/period for the remaining tasks once a recovery takes place. In fact, in Section 5.2, I will formally prove that the SHR scheme preserves *the system's original reliability* by maintaining the task-level reliability.

Figure 5.1(d) shows a scenario where the shared recovery is dynamically allocated to the re-execution of T_4 at f_{max} after the detection of a transient fault in this task. As a result, T_5 is executed at the maximum frequency. Interestingly, in this example, the reliability achieved by SHR (computed as 99.99999999%) turns out to be even better than that of GRE. This is essentially due to the fact that the GRE schedule in Figure 5.1(b) could not allocate any recovery tasks for T_4 and T_5, whereas SHR improved reliability for *all* the tasks through the shared recovery. While it is true that the GRE solution can recover from some multiple-fault scenarios (affecting, for example, T_1 and T_2), an important system design principle is to optimize more common cases. In this example, covering *all* single-fault scenarios turns out to be more important than covering some multiple-fault scenarios that can occur with very low probability. The simulation results will further confirm this trend.

The proposal in this chapter also includes a *dynamic* extension to the base SHR scheme, that adjusts the size of the shared recovery and processing frequencies at run-time, by taking into account the early completions as well as large tasks that complete successfully without a fault at run-time. The extensive experimental evaluation indicates the new scheme offers not only significant energy gains over existing RA-PM schemes, but a markedly close performance to optimal (but reliability-oblivious) DVS schemes that reserve the entire slack for slow-down. Finally, I extend the solutions to dependent tasks with DAG settings.

Problem Description. In this chapter, the objective is to *minimize the system energy consumption as much as possible for multiple frame-based tasks through DVS while still preserving system original reliability and meeting the deadline constraints.* In what follows, I present the details of the new RA-PM scheme, which offers significant advantages over existing schemes. In accordance with the prior RA-PM research [87–89, 91, 92], this chapter assumes that the recovery takes the form of re-execution and is executed at the maximum frequency when a fault is detected.

5.2 The Shared Recovery Technique for Independent Frame-based tasks

In this section, the solution to independent frame-based tasks is presented. The reliability-aware power management (RA-PM) framework is based on maintaining the original reliability of the real-time embedded application even in DVS settings. The existing schemes [87–89, 92] achieve this objective by allocating, in the offline analysis phase, a separate recovery task with any task that will be executed through voltage and frequency scaling.

However, both the recovery and DVS mechanisms essentially *compete* for the available *system slack.* The novel solution is motivated by the observation that such an explicit, static and separate recovery task allocation in the offline phase severely limits the slow-down opportunities. In fact, to maintain the task-level reliabilities, it is sufficient to ensure that, at the dispatch time of any scaled task T_i, there will be sufficient time to execute a recovery in case that T_i incurs a transient fault. Moreover, the system can even reserve the CPU time only for *a shared recovery block* that may be used by *any* task if need arises: since transient fault probabilities are typically low, any CPU time reserved but not needed for the recovery can be immediately be made available to the use of the *next* scaled task. Consequently, the SHR scheme has the potential of maximizing the slack available for DVS while still maintaining the system reliability.

It is, however, important to determine the size of the CPU time reserved for the shared recovery carefully. Let us define the available system slack as $L = d - \sum_{i=1}^{n} c_i$. Observe that any task T_j where $c_j \geq L$ cannot be managed by any RA-PM solution, since there is simply no sufficient slack for possible recovery, should T_j be scaled and subsequently fail. As a result, only the tasks in $\Gamma_1 = \{T_i \,|\, c_i < L\}$ will be managed through DVS. Further, the shared recovery should be large enough to allow the re-execution of any scaled task, suggesting that $\alpha = max\{c_i \,|\, T_i \in \Gamma_1\}$ units of CPU time should be reserved. Now, the problem of determining the processing frequency assignments for the managed tasks to minimize the energy consumption can be formulated as a convex program:

$$\text{minimize} \sum_{T_j \in \Gamma_1} E_j(f_j) \tag{5.1}$$

$$\text{subject to:} \sum_{T_j \in \Gamma_1} \frac{c_j}{f_j} + \sum_{T_i \in \, (\Gamma - \Gamma_1)} c_i + \alpha \leq d \tag{5.2}$$

$$f_{min} \leq f_j \leq f_{max} (\forall j : T_j \in \Gamma_1) \tag{5.3}$$

where the inequality (5.2) encodes the deadline constraint and (5.3) gives the available speed constraints. Since $\sum(c_i)+\alpha$ in (5.2) is a constant, this problem can be seen to be an instance of the system-wide energy optimization problem that can be solved in polynomial-time (for example, in time $O(n^3)$ by the algorithm given in [5]). The offline frequency assignment phase of the SHR scheme can be found in Algorithm 1.

The online operation of SHR is relatively simple and presented in Algorithm 1 as well. The tasks are executed at the pre-computed and optimal frequency levels as long as transient faults do not occur. Once a fault has been detected, the recovery is executed at f_{max}. Then, since the shared recovery slack may be partially or fully used for the current period, the new suggested scheme simply disables voltage scaling and executes all the remaining tasks at f_{max} until the next frame (period) boundary. Not only does this have only a minimal impact on the expected energy savings (because the transient faults occur rarely), but also, as I formally show below, makes sure that no task reliability is worse than its original reliability level (when executed without voltage scaling). One can easily verify that the online overhead of SHR is minimal.

Algorithm 1 Steps of the SHR scheme

```
1: Compute the available slack $L = d - \sum_{i=1}^{n} c_i$;
2: Determine $\Gamma_1 = \{T_i | c_i < L\}$;
3: Compute the optimal frequency assignments $f_i^*$ for all tasks $T_i \in \Gamma_1$, by solving the
   non-linear optimization problem given by (5.1), (5.2) and (5.3);
4: Set $f_i^* = f_{max}$ for tasks $T_i \in (\Gamma - \Gamma_1)$.
5: **Online Phase**:
6: At every frame (period) boundary: set *flag* = *true* ;
7: At dispatch time of task $T_i$:
8: **if** *flag* = *true* **then**
9:    Set the CPU frequency to $f_i^*$
10: **else**
11:    set the CPU frequency to $f_{max}$;
12: **end if**
13: At completion time of task $T_i$ with $f_i < f_{max}$:
14:   Check whether a transient fault has occurred;
15:   When the fault is detected, execute its recovery with $f_{max}$ and set *flag* = *false*.
```

Before formally proving the reliability-preserving feature of SHR, we recall from [92] that the overall reliability of a task T_i executed at frequency $f_i < f_{max}$ is given by:

$$R_i(f_i) + (1 - R_i(f_i)) \cdot R_i(f_{max}) \tag{5.4}$$

when a recovery task of size c_i is guaranteed to execute at f_{max} in case of a transient fault incurred by T_i. The first term in (5.4) is the probability that the main scaled task will complete successfully. The second term captures the probability that the recovery executed at f_{max} will succeed when the main task fails (which can occur with probability $(1-R_i(f_i))$.

Lemma 1. *The SHR scheme preserves the system's original reliability in every period.*

Proof. I will prove the statement by showing that SHR preserves the original reliability of each task individually, in each period. Consider the first task T_1 executed by SHR. If $f_1 = f_{max}$, its reliability is exactly $R_1^0 = R_1(f_{max})$, hence no reliability degradation occurs. On the other hand if $f_1 < f_{max}$, its reliability is given by (5.4) since SHR *must have assigned a shared recovery of size at least* c_i. But the expression (5.4) is guaranteed to be greater than $R_i(f_{max}) = R_1^0$ [92], implying that T_1's reliability does not decrease.

For the second task T_2, if T_1 fails and the shared recovery slack is used, T_2 and all the remaining tasks are executed at f_{max} until the period boundary, which maintains the original reliability of the entire application during that period. Otherwise, if T_1 does not fail, the recovery slack is not used and the same argument can be repeated for T_2 to demonstrate that a large enough recovery slack will preserve its reliability in case that it is executed at a low frequency level. Continuing in this way, it can be shown that the reliability of a task set executed by SHR is never worse than its original reliability while meeting the deadline. □

5.2.1 Experimental Evaluation

To assess the performance of the new SHR technique on both energy efficiency and system reliability, a discrete-event simulator was designed in C programming language. The following schemes are considered in the evaluation:

- The **Greedy (GRE)** algorithm (from [88]): In this scheme, the tasks are selected for reliability-aware power management in greedy fashion. That is, for the first task, a separate recovery is allocated and its frequency is reduced as much as possible (up to the energy-efficient frequency f_{ee}). The process of selecting tasks for management continues in this manner until all available slack is depleted.

- The **SUEF** algorithm (from [88]): In this scheme, first, the *slack usage efficiency factors (SUEFs)* of all tasks are computed and the slack is allocated in decreasing order of SUEF values. The SUEF value for task T_i running at the frequency f is defined as $\frac{E_i^0 - E_i(f)}{s_i(f)}$, where E_i^0 and $E_i(f)$ are the energy consumption of task T_i at f_{max} and f, respectively, and $s_i(f)$ is the total amount of slack needed (including the slack reserved for recovery) [88].

- The **Static Power Management (SPM)** solution that computes the optimal system-level DVS slow-down factors to minimize the overall energy consumption [5]. Note that SPM neither reserves any slack for recovery nor considers reliability preservation. SPM is included in the comparison simply because it yields the maximum energy savings that can be achieved by any algorithm – reliability-aware or not.

Transient faults are assumed to follow Poisson distribution with an average fault rate of $\lambda_0 = 10^{-6}$ at f_{max}, which is a realistic fault rate as reported in [30, 93]. A DVS-enabled CPU is modeled where the normalized processing frequencies can change from $f_{min} = 0.1$ to $f_{max} = 1.0$. The simulation uses a cubic frequency-dependent power component P_d which is equal to unity at f_{max}. The frequency-independent power component P_{ind} for each task is normalized with respect to P_d. All results are normalized with respect to the **No Power Management (NPM)** scheme that executes all the tasks without any frequency and voltage scaling at f_{max}. Note that the reliability level achieved by NPM corresponds to the original reliability of the system (Section 3.3 in Chapter 3).

Within a task set, the worst-case number of cycles c_i for each task is randomly generated through a uniform distribution, in such a way that the worst-case execution time under maximum frequency falls in the range $[1ms, 10ms]$. Each task set contains 10 tasks.

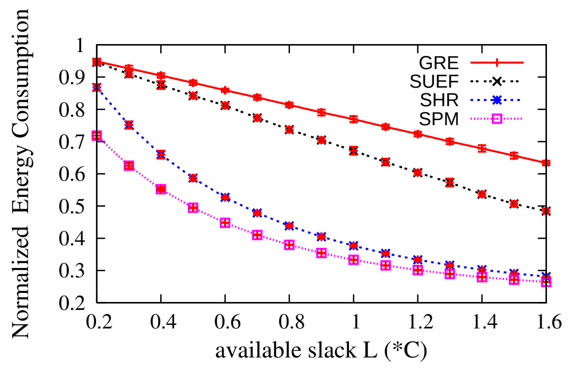


Figure 5.2: The impact of the available slack on system energy consumption

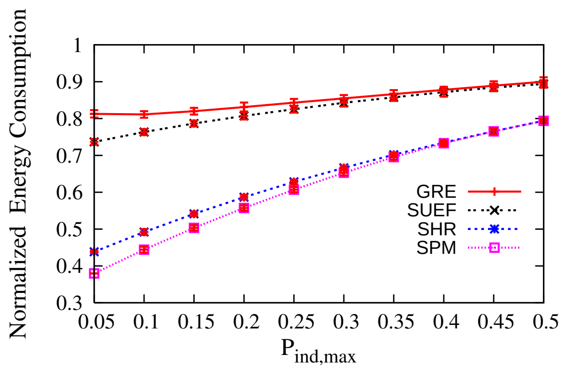
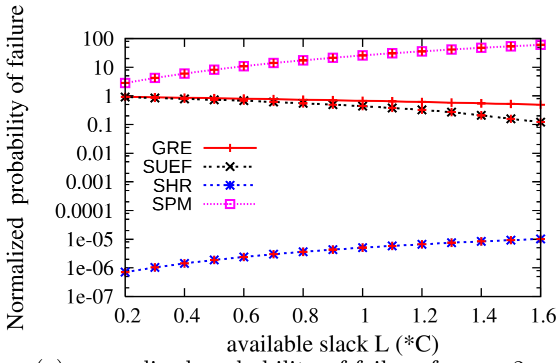


Figure 5.3: The impact of P_{ind} on the system energy consumption

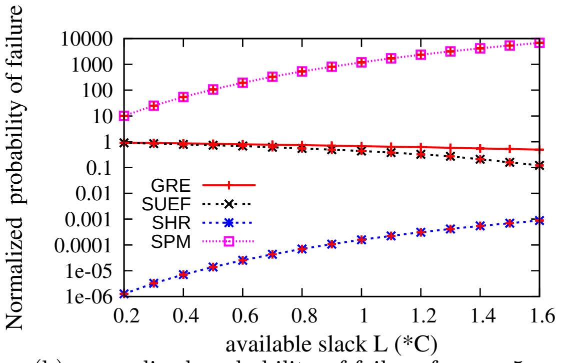
Each point in the presented plots is obtained by averaging the values obtained through the simulations. The confidence intervals at 97% confidence level are reported.

First, I evaluate the performance of the schemes from energy consumption point of view. Figure 5.2 shows the relationship between the available slack and the energy consumption. In these experiments, the exponent q in the fault rate function (Section 3.3 of Chapter 3) is set to 2 and the frequency-independent power is 0.05. Let $C = \sum c_i$. The available slack is represented by $L = d - C$. Naturally, the larger the slack, the more opportunities for dynamic voltage scaling and energy reduction; and all schemes achieve lower energy consumption with increasing L. It is observed that the SHR scheme can achieve significant (up to 35%) energy savings over the previous RA-PM algorithms, GRE and SUEF. This is because, unlike SUEF and GRE that allocate separate recovery tasks statically, SHR reserves only a minimal amount of slack as a shared recovery and is able to allocate much larger slack for energy-efficient slow-down. Moreover, notice that as the available slack gets larger (e.g. when $L \geq 0.7$), the performance of SHR becomes remarkably close to that of the SPM, which is the upper bound for any DVS-based energy management algorithm.

Figure 5.3 illustrates the effects of having different frequency-independent active power P_{ind} on the system energy consumption when the available slack $L = 0.8 \cdot C$ and $q = 2$. The minimum value of P_{ind} is set to $P_{ind,min} = 0.05$. P_{ind} for each task is generated randomly and separately in the range $[P_{ind,min}, P_{ind,max}]$. The energy consumption figures of all the schemes increase with increasing $P_{ind,max}$. This is because, as $P_{ind,max}$ increases, the frequency-independent energy consumption increases and further, the energy-efficient frequency thresholds for tasks become higher, limiting the voltage scaling opportunities (Section 3.2 in Chapter 3). Again it is observed that the SHR scheme dominates over GRE and SUEF and further, its performance is extremely close to the ideal bound obtained by SPM.



(a) normalized probability of failure for $q = 2$



(b) normalized probability of failure for $q = 5$

Figure 5.4: The impact of slack on system reliability

Next, I investigate the system reliability dimension for these schemes. For convenience, I present the *probability of failure (PoF)* (defined as 1− *reliability*) achieved by all schemes. Again, all results are normalized with respect to the *PoF* achieved by NPM. Obviously, any reliability-preserving scheme should achieve a normalized *PoF* value that does not exceed 1.0. Figure 5.4(a) and Figure 5.4(b) show the reliability performance when the fault rate exponent q is 2 and 5, respectively. P_{ind} is set to 0.05 for all tasks. Consistent with previous research, the findings indicate that SPM is not able to maintain the original reliability; further, it experiences PoF increases of several orders of magnitude. As expected, SHR, GRE and SUEF are all able to maintain the system's original reliability. Moreover, the simulation results point to a somewhat counter-intuitive result: SHR has a clear advantage also on the reliability side through most of the evaluated spectrum, despite the fact that it uses a shared recovery block for all the tasks. The main reason behind this phenomenon is that GRE and SUEF choose to allocate statically the recovery tasks to individual tasks. As a result, except when the slack is very large, only a subset of tasks can be managed by SUEF and GRE while the remaining tasks do not receive any recovery. On the other hand, SHR allocates a single recovery block that can be used by *any* task in the case of a fault. In essence, SHR provides a better protection for single-fault scenarios that are typically much more likely than some multiple-fault scenarios that SUEF and GRE provision for, at the expense of some entire tasks. With increasing slack, the *PoF* of SHR relatively increases as the system can use more aggressive voltage scaling. When $q = 5$, the fault rate is very sensitive to the voltage scaling and normalized *PoF* of SHR approaches (but never exceeds) 1.0.

5.2.2 Dynamic Extensions

While the basic SHR scheme provides a powerful mechanism to increase energy savings while maintaining the system reliability, some run-time optimizations can further improve the performance. In fact, there are two main opportunities that can be exploited at run-time:

- Initially, the SHR scheme is forced to reserve CPU time for the potential recovery of the largest task. As tasks complete within a frame, the system can dynamically reduce the amount of CPU time for shared recovery when faults are not encountered. For example, when the largest task completes successfully (without a fault), the system can further slow down the remaining tasks by using some part of the recovery block which is no longer needed. This can be repeated for other large tasks in the same frame.

- Typically, real-time embedded applications complete early, without consuming their worst-case number of cycles. In fact, many DVS frameworks [7,48,55] were proposed in the past to detect and use the *dynamic slack* that can arise from early completions to further reduce the CPU processing frequency. The same approach can be incorporated to this framework.

Thus, the **Dynamic Shared Recovery (DSHR)** technique is based on determining the initial frequency assignments as in the base SHR scheme presented in Section 5.2. At run-time, when a task completes early and/or without a fault, frequency assignments are updated by taking into account the time to deadline, the size of the recovery slack needed, and worst-case workload of the *remaining* tasks in that frame.

Experimental Evaluation

In this section, I evaluate the performance of the dynamic scheme experimentally. The experimental settings are essentially the same as those in Section 5.2.1; however, in order to model the variations in the actual workload, the ratio $\frac{WCC}{BCC}$ is used, which denotes the ratio of the worst-case number of cycles (WCC) to the best-case number of cycles (BCC). For each task set, first the worst-case number of cycles for each task is determined as before. At run-time, the actual workload (number of cycles) of each task is determined between its BCC and WCC, using a uniform probability distribution. Clearly, the higher this ratio, the more the actual workload deviates from the worst-case. In the simulations, the fault rate exponent q is set to 2.

To obtain a fair comparison, though they were originally proposed only as static algorithms in [88], the simulation extended GRE and SUEF algorithms to use dynamic reclaiming in case of early completions and not needed recovery tasks. These dynamic extensions are named as DGRE and DSUEF, respectively. Essentially, at every task completion point, GRE and SUEF are re-invoked to allow the re-adjustment of the processing frequency, just like DSHR. Finally, I implemented the scheme that invokes the static power management (SPM) algorithm to compute optimal frequency assignments using the knowledge of the *actual* workload in advance. This algorithm, named simply *Bound* in the following discussion, is not a practical scheme since it assumes the knowledge about the future workload; but its performance is used again as a natural bound on the energy consumption of any algorithm. Again, in the provided results, all results are normalized with respect to that of the NPM scheme.

First, I evaluate the energy savings of the schemes. Figure 5.5 shows how the energy consumption changes as a function of the available slack L when $\frac{WCC}{BCC} = 4$ and $P_{ind} = 0.05$.

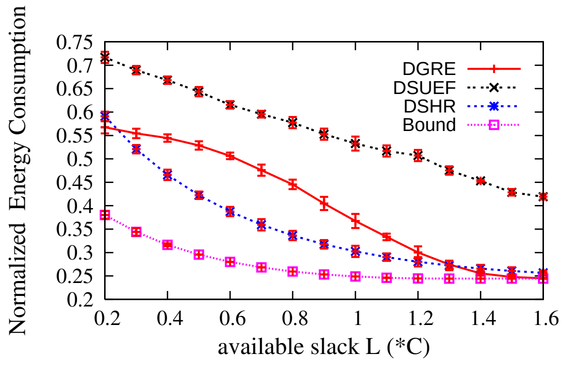


Figure 5.5: The impact of slack on system energy consumption

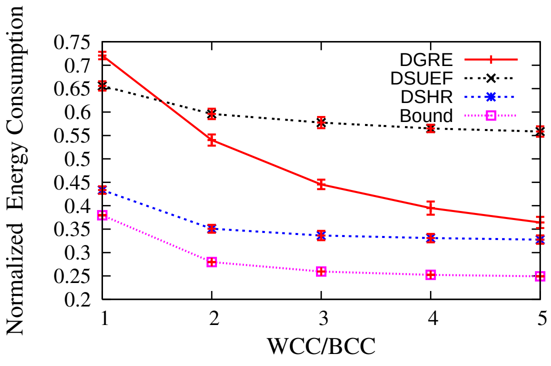


Figure 5.6: The impact of $\frac{WCC}{BCC}$ ratio on system energy consumption

As before, the more slack available, the higher the energy savings. Notice that DSHR provides again clear advantages over DGRE and DSUEF. However, at very small slack values (e.g. when $L = 0.3\ C$), the difference between the clairvoyant algorithm *Bound* and DSHR is more significant since the former is able to adopt low processing frequencies with the pre-knowledge of the actual task workloads, while the latter is forced to base its offline decisions on the worst-case workload information, as every practical real-time algorithm. An interesting observation is the greatly enhanced performance of DGRE (compared to worst-case settings). In fact, when $\frac{WCC}{BCC} = 4$ not only it outperforms SUEF but also offers a performance reasonably close to DSHR. This is because, at this high workload variability settings, many tasks complete early, and as previous DVS research suggests [7, 48], re-using the slack as early as possible typically pays off. This contrasts with DSUEF that chooses to re-allocate the slack according to slack usage efficiency factors, possibly excluding the next tasks that would immediate benefit. Note that when L is very large, DGRE, DSHR and *Bound* converge to the same level since all frequencies are naturally limited by the energy-efficient frequency level.

Figure 5.6 illustrates the impact of the variability in the actual workload by the ratio of $\frac{WCC}{BCC}$ on the energy consumption when $L = 0.8 \cdot C$ and $P_{ind} = 0.05$. Again, as expected, the system energy consumption generally decreases with decreasing actual workloads (with increasing $\frac{WCC}{BCC}$ ratios), since more dynamic slack enables the system to scale down the

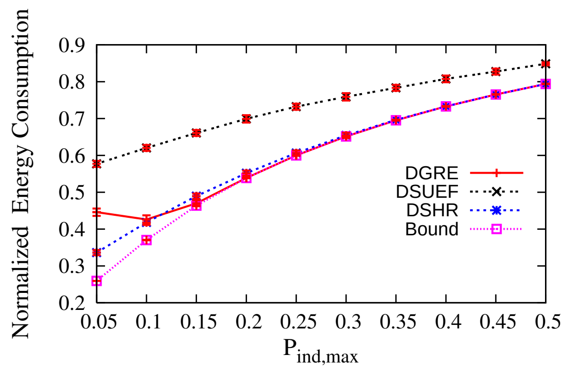


Figure 5.7: The impact of P_{ind} on system energy consumption

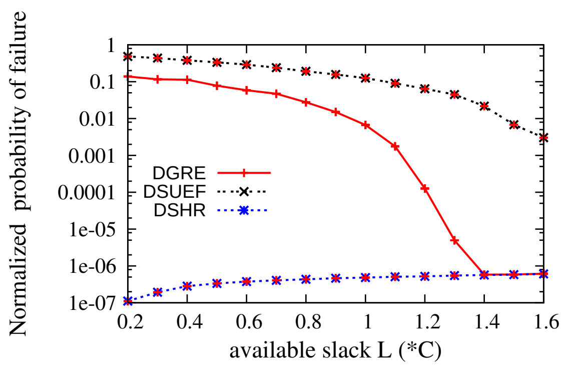


Figure 5.8: The impact of the system slack on the probability of failure

frequency and then further improve the energy savings. Among all three schemes, DSHR achieves the best energy savings, which is close to the yardstick algorithm *Bound* by at most a margin of 7%. Further, an interesting phenomenon is observed: when $\frac{WCC}{BCC} \leq 2$, DGRE's performance quickly deteriorates. This is because, when the actual workload does not exhibit high variability, DSUEF's approach of assigning slack according to slack usage efficiency factors yields typically better results, as opposed the DGRE's strategy that consists in greedily making available the slack to the next task. However, with increasing $\frac{WCC}{BCC}$, DGRE performs better and better due to higher dynamic slack resulting from tasks' earlier completions.

Figure 5.7 shows the effects of frequency-independent active power on the system energy consumptions for $L = 0.8 \cdot C$ and $\frac{WCC}{BCC} = 4$. We observe that energy consumptions increase with increasing the value of $P_{ind,max}$ and that DSHR, DGRE and *Bound* converge to the same value when $P_{ind,max} \geq 0.2$. Again, the main reason is that higher P_{ind} values result in significantly high energy-efficient frequency values that force the system to run at high frequencies regardless of the actual workload.

Next, the reliability performance of the schemes is evaluated. Figure 5.8 shows the impact of available slack on the probability of failure when $P_{ind} = 0.05$ and $\frac{WCC}{BCC} = 5$. It is observed that, while DSHR is the best, now DGRE and DSUEF greatly benefit from higher slack values. Essentially, DGRE and DSUEF can allocate more recovery tasks dynamically by recycling the slack of unused recoveries at run-time, which is reflected in PoF values. Another interesting observation is that, just like the energy dimension, DGRE appears to outperform DSUEF because of re-assigning recovery tasks in greedy fashion. Also note that when $L \geq 1.1$, once again almost all the tasks are effectively executed at the energy-efficient frequencies by DGRE and DSHR, which gives similar PoF values.

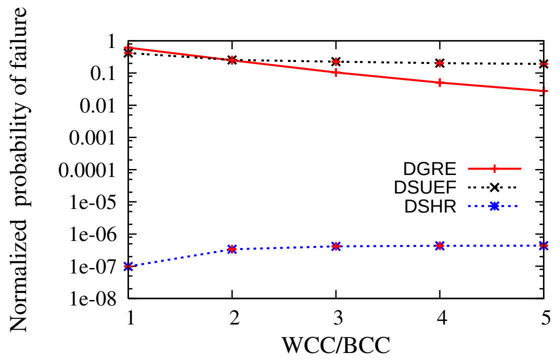


Figure 5.9: The impact of $\frac{WCC}{BCC}$ ratio on the probability of failure

Figure 5.9 illustrates the impact of the variability in the actual workload on the system reliability when $L = 0.8 \cdot C$ and all P_{ind} values are set to 0.05. In general, with fixed available slack, the probability of failure for all schemes decreases as the actual workload is reduced (i.e. as the ratio of $\frac{WCC}{BCC}$ is increased) since more slack becomes available due to early completions of tasks. Because of its greedy nature (i.e. aggressive allocation of dynamic slack to the next task), DGRE's reliability tends to degrade with small $\frac{WCC}{BCC}$ ratios. Once $\frac{WCC}{BCC} \geq 3$, DSHR can execute almost all the tasks at the energy-efficient frequencies and also almost every task can have a recovery to be executed if a fault occurs.

5.3 Extensions to Dependent Tasks

In some applications (such as feedback-loop in the control environment), real-time tasks may depend on each other for input/output data and thus have precedence constraints. Therefore, in this section, by extending the basic idea of shared-recovery technique, SHR-based

RA-PM schemes is studied for a set of frame-based *dependent* real-time tasks with *individual deadlines* and a common period where their precedence constraints are represented by *directed acyclic graphs (DAGs).*

The remainder of this section is organized as follows. Section 5.3.1 presents system models and states the assumptions, followed by a motivational example and problem description. The static and online shared-recovery based schemes for real-time tasks represented by DAGs are addressed and evaluated in Sections 5.3.2 and 5.3.3, respectively. Section 5.4 concludes the chapter.

5.3.1 Application Model

A real-time application that consists of n frame-based dependent tasks is considered. The tasks are represented by a *directed acyclic graph (DAG)*: $G = (V, E)$. Here, the set of vertices (or nodes) in G represents the set of tasks $V = \{T_1, \ldots, T_n\}$. The set of edges $E = \{E_1, \ldots, E_m\}$ represents the precedence constraints among tasks, where an edge $(T_i, T_j) \in E$ indicates that task T_j cannot start to execute until T_i finishes its execution [14, 83]. The deadline of task T_i is denoted by d_i. The common period for all tasks (i.e., the *frame* of the application) is denoted as d. That is, the application will repeat for every period of d and, within each frame, there are n dependent tasks. Note that, for any task T_i, $d_i \leq d$ $(i = 1, \ldots, n)$.

As an example, Figure 5.10 shows the DAG for an application with five dependent tasks, where each task is labeled with its WCET c_i and deadline d_i (in milliseconds). The common period (i.e., the application's frame length) is $d = 100ms$.

Problem Description: Taking the negative effects of DVS on system reliability into consideration, for frame-based dependent real-time tasks with individual deadlines and a common period that are represented by a DAG, the problem to be addressed in this chapter is to: *find the execution order of tasks and their frequency assignment to minimize system energy consumption while preserving system original reliability without violating tasks' deadline and precedence constraints.*

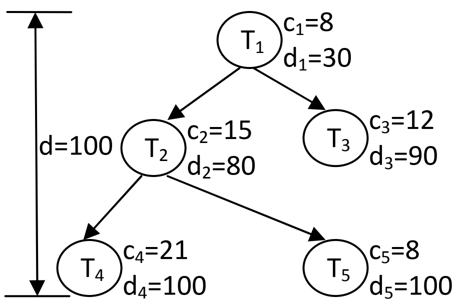


Figure 5.10: An example application with five dependent tasks.

5.3.2 Static Shared-Recovery RA-PM Scheme for Dependent Tasks

List scheduling has been the traditional scheduling approach for dependent tasks represented by DAGs [49]. However, finding a *feasible* and *optimal* schedule of dependent tasks to maximize the opportunity for energy (and reliability) management is not trivial, especially for the cases where each task has its individual deadline.

In this section, based on the *effective deadlines* of tasks (as defined below), it is first shown that the *Earliest Deadline First (EDF)* scheduling can always find a valid schedule if the tasks are feasible. Then, I propose the shared-recovery based RA-PM scheme for dependent tasks (denoted as SHR-DAG), which is proved to guarantee the system original reliability. Next, for the valid EDF schedule (i.e., execution order) of tasks, the optimal frequency assignment for SHR-DAG to minimize the *fault-free* energy consumption of the tasks is discussed, followed by the evaluations of SHR-DAG through simulations.

Feasible Schedule and EDF Scheduling

Note that, due to precedence constraints, a task may have to complete well before its deadline to ensure that all its successor tasks can finish in time. Therefore, based on *as late as possible (ALAP)* policy [40], we can define the *effective deadlines* of a task T_i as follows:

$$D_i^e = \begin{cases} d_i, & Succ(T_i) = \emptyset \\ \min\{d_i, D_j^e - c_j\}, & T_j \in Succ(T_i) \neq \emptyset \end{cases} \tag{5.5}$$

where $Succ(T_i)$ is the set of successor tasks of T_i. That is, if there exists an edge $(T_i, T_j) \in E$, we have $T_j \in Succ(T_i)$.

Theorem 1. *If the set of frame-based tasks with individual deadlines represented by a DAG are feasible, the schedule of tasks under EDF scheduling based on tasks' effective deadlines can meet all deadline and precedence constraints of tasks.*

Proof. I prove this theorem in two parts: First, from Equation 5.5, it can be seen that the effective deadline of each task is the *latest* time by which a task has to complete its execution in a valid schedule. Therefore, if the tasks under consideration are feasible and there exists a valid schedule, all tasks will complete their executions before their effective deadlines. Therefore, replacing tasks' original deadlines with their effective deadlines does not hurt the feasibility of the original problem.

Second, I show that the schedule obtained from the EDF scheduling based on tasks' effective deadlines is valid if there exists a valid schedule for the original problem. Note that, if tasks are executed under EDF based on their effective deadlines, precedence constraints among tasks are automatically guaranteed (as a task always has a smaller effective deadline than its successor tasks). Thus, based on their effective deadlines, the tasks can be treated as *independent* tasks under EDF scheduling. For independent tasks, it is well-known that the schedule under EDF scheduling is feasible if there exists a feasible schedule for the tasks [14], which concludes the proof. □

Shared-Recovery for Reliability Preservation of Dependent Tasks (SHR-DAG)

Instead of statically scheduling multiple separate recovery tasks as in the conventional RA-PM schemes [88–90], the center idea of *shared-recovery* technique is to let scaled tasks share a single recovery block and thus leaves more slack for DVS to save more energy [82]. Moreover, once the execution order of tasks is known in a valid schedule, the size of the required recovery block at any given time depends only on the current running task. Hence, different from the preliminary SHR scheme [82], where the recovery block is determined by the largest remaining tasks, the new SHR-DAG scheme varies the recovery block size aiming at more energy savings through better slack usage.

Without loss of generality, suppose that the execution order of tasks in a valid schedule is $T_1, T_2, .., T_n$. If every task T_a before task T_i (i.e., $a < i$) is executed at its scaled frequency f_a without incurring soft errors caused by transient faults, we will need to reserve a recovery block with size c_i before dispatching task T_i at a scaled frequency f_i, which has to satisfy the following time constraint:

$$\sum_{a=1}^{i} \frac{c_a}{f_a} + c_i \leq D_i^e \tag{5.6}$$

Recall that D_i^e is the effective deadline of task T_i. Moreover, in case the scaled execution of T_i encounters errors and fails, its recovery as well as all remaining tasks have to be able to finish in time at the maximum frequency $f_{max} = 1$. That is, we need to have:

$$\sum_{a=1}^{i} \frac{c_a}{f_a} + \sum_{x=i}^{k} c_x \leq D_x^e, \ \forall\ k:\ (i+1) \leq k \leq n \tag{5.7}$$

Hence, for feasibility, the scaled frequency f_i for task T_i has to satisfy $\sum_{a=1}^{i} \frac{c_a}{f_a} \leq b_i$, where $b_i = min\{D_k^e - \sum_{x=i}^{k} c_x | i \leq k \leq n\}$. Therefore, the problem can be formally expressed as: find the frequency assignments $\{f_1, ..., f_n\}$ to minimize:

$$E_{total} = \sum_{i=1}^{n} E_i(f_i) = \sum_{i=1}^{n} (P_{ind} + C_{ef} f_i^m) \frac{c_i}{f_i} \tag{5.8}$$

Subject to

$$\sum_{a=1}^{i} \frac{c_a}{f_a} \leq b_i \ (\forall\ i:\ 1 \leq i \leq n) \tag{5.9}$$

$$f_{low} \leq f_i \leq f_{max} \ (\forall\ i:\ 1 \leq i \leq n) \tag{5.10}$$

Recall that $f_{low} = \max\{f_{min}, f_{ee}\}$. Once one finds out the valid frequency assignment, the online operation of SHR-DAG is the same as that for the SHR scheme [82]: as long as there are no errors, tasks can be executed at their scaled frequencies. Once soft errors are detected at run-time, the recovery of the faulty task is immediately dispatched in the form of re-execution and the system switches to a *contingency mode* where all remaining tasks within the current frame are executed at f_{max}.

From [87], for a task T_i that is executed at a scaled frequency $f_i < f_{max}$, its overall reliability by taking into consideration of its recovery task can be given as:

$$R_i = R_i(f_i) + (1 - R_i(f_i)) \cdot R_i^0 > R_i^0 \tag{5.11}$$

where the first term on the right side is the probability of the scaled execution of the primary task completing successfully, and the second term captures the probability of the recovery executes at f_{max} successfully when the primary task fails. That is, with the help of the recovery task, task T_i's original reliability R_i^0 is preserved.

Lemma 2. *For an application with a set of frame-based dependent tasks, the system's original reliability in every frame will be preserved under the SHR-DAG scheme.*

Proof. I will prove the statement by showing that SHR-DAG preserves the original reliability of every task, in each period. Consider the first task T_1 executed by SHR-DAG. If $f_1 = f_{max}$, its reliability is exactly $R_1^0 = R_1(f_{max})$, hence no reliability degradation occurs. Otherwise, if $f_1 < f_{max}$, SHR-DAG *must have assigned a recovery block with size of* c_i and its reliability can be given by Equation (5.11), which is guaranteed to be greater than R_1^0.

For the second task T_2, if T_1 fails and the shared recovery has been used, T_2 and all remaining tasks within the current frame are executed at f_{max}, which maintains their original reliability as well. Otherwise, if T_1 does not fail, the same argument for T_1 can be repeated for T_2, whose original reliability will be preserved. Continuing in this way, it can be shown that the reliability of a task set under SHR-DAG is never worse than its original reliability while satisfying the deadline constraints. □

Optimal Frequency Assignments

In this section, for a given valid schedule with fixed execution order of tasks, I discuss how to obtain the optimal frequency assignments assuming all tasks take their WCETs. By ignoring the frequency constraints, the following problem is denoted as the *SHR-DAG$_{part}$* problem: finding the speed assignments $\{f_1, \ldots, f_n\}$ to minimize:

$$E_{total} = \sum_{j=1}^{n} E_j(f_j) = \sum_{j=1}^{n} (P_{ind} + C_{ef} \cdot f_j^m) \frac{c_j}{f_j} \tag{5.12}$$

Subject to

$$\sum_{a=1}^{j} \frac{c_a}{f_a} \leq b_j (\forall\ j :\ 1 \leq j \leq n) \tag{5.13}$$

which is the sub-problem of SHR-DAG that considers only deadline constraints. In order to solve *SHR-DAG$_{part}$* problem, Kuhn-Tucker conditions [45] can be applied and then obtain the following expressions:

$$((m-1)C_{ef} \cdot f_j^{m-2} - \frac{P_{ind}}{f_j^2})c_j - \sum_{a=j}^{n} \mu_a \frac{c_j}{f_j^2} = 0\ (\forall 1 \leq j \leq n) \tag{5.14}$$

$$\mu_j (\sum_{a=1}^{j} \frac{c_a}{f_a} - b_j) = 0\ (\forall 1 \leq j \leq n) \tag{5.15}$$

$$\mu_j \geq 0\ (\forall 1 \leq j \leq n) \tag{5.16}$$

where $\mu_1, ..., \mu_n$ are Lagrange multipliers. Expression (5.14) can be further written as:

$$f_j = (\frac{P_{ind} + \sum_{a=j}^{n} \mu_a}{(m-1)C_{ef}})^{\frac{1}{m}} (\forall 1 \leq j \leq n) \tag{5.17}$$

Based on expressions (5.15), (5.16) and (5.17), it is clearly that if $\forall\, j : 1 \leq j \leq n, \mu_j = 0$, the optimal solution is $f_j = f_{ee}$. So I focus on the case when there exists $\mu_j > 0$. Fortunately, under this case, *OPT* algorithm [73] can be adapted to optimally solve the *SHR-DAG*$_{part}$ problem.

First, in order to solve the original problem, I first consider the problem, (called SHR-DAG-L) by ignoring the upper bound constraints given by Equation (5.10). Formally, the problem SHR-DAG-L is defined as to minimize:

$$E_{total} = \sum_{j=1}^{n} E_j(f_j) \tag{5.18}$$

Subject to

$$\sum_{a=1}^{j} \frac{c_a}{f_a} \leq b_j (\forall\, j :\ 1 \leq j \leq n) \tag{5.19}$$

$$f_{low} \leq f_j (\forall\, i :\ 1 \leq j \leq n) \tag{5.20}$$

Lemma 3. *A set of speed assignments $F = \{f_1, ..., f_n\}$ is the solution to SHR-DAG-L if F is the solution to SHR-DAG$_{part}$ and satisfies constraint sets (5.20).*

Proof. The necessary and sufficient Kuhn-Tucker conditions for the problem SHR-DAG-L include the constraints (5.19), (5.20), and:

$$E_j'(f_j) - \sum_{a=j}^{n} \mu_a \frac{c_j}{f_j^2} - \overline{\mu_j} = 0\ (\forall 1 \leq j \leq n) \tag{5.21}$$

$$\mu_j (\sum_{a=1}^{j} \frac{c_a}{f_a} - b_j) = 0 (\forall 1 \leq j \leq n) \tag{5.22}$$

$$\overline{\mu_j} (f_{low} - f_j) = 0 (\forall 1 \leq j \leq n) \tag{5.23}$$

$$\mu_j \geq 0 (\forall 1 \leq j \leq n) \tag{5.24}$$

$$\overline{\mu_j} \geq 0 (\forall 1 \leq j \leq n) \tag{5.25}$$

where μ_j and $\overline{\mu_j}$ $(\forall j)$ are the Lagrange multipliers. Then it can be seen that the condition given in the lemma coincides with the case where $\overline{\mu_j} = 0$. Indeed, in that case, necessary and sufficient Kuhn-Tucker conditions to SHR-DAG-L become identical to those of SHR-DAG$_{part}$. Therefore, the above lemma holds. □

Lemma 4. *If F violates the lower bound constraints given by (5.20), then, in the solution of SHR-DAG-L, $f_i = f_{low} \forall i \in \Delta$, where $\Delta = \{i | \frac{f_{low}^2}{c_i \sum_{a=i}^{n} \mu_a} E_i'(f_{low}) \geq \frac{f_{low}^2}{c_j \sum_{a=j}^{n} \mu_a} E_j'(f_{low}) \forall j\}$*

Proof. If F violates the lower bound constraints, then due to (5.23), it can be concluded that $\exists\, \overline{\mu_j} > 0$ and $f_j = f_{low}$. I will prove that, under the condition specified in the lemma, the Lagrange multipliers $\overline{\mu_i} > 0$ for $\forall i \in \Delta$, which will imply (by (5.23)) that $f_i = f_{low}$.

Assume $\exists\, m \in \Delta$ such that $\overline{\mu_m} = 0$. Through the above discussion, it can been known that $\exists\, \overline{\mu_j} > 0$ and $f_j = f_{low}$. Using (5.21), the following can be obtained: $\frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} E_m'(f_m) = \frac{f_{low}^2}{c_j \sum_{a=j}^{n} \mu_a} E_j'(f_{low}) - \mu_j \frac{f_{low}^2}{c_j \sum_{a=j}^{n} \mu_a}$.

Then we obtain $\frac{f_{low}^2}{c_j \sum_{a=j}^{n} \mu_a} E_j'(f_{low}) > \frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} E_m'(f_m)$. Moreover, the function $\frac{f_m^2}{c_j \sum_{a=m}^{n} \mu_a} E_m'(f_m)$ is strictly increasing when increasing with f_m. At this point, since $f_m \geq f_{low}$, hence, $\frac{f_{low}^2}{c_j \sum_{a=j}^{n} \mu_a} E_j'(f_{low}) > \frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} E_m'(f_m) \geq \frac{f_{low}^2}{c_m \sum_{a=m}^{n} \mu_a} E_m'(f_{low})$, which contradicts the assumption that $m \in \Delta$. □

Lemma 5. *If the solution of SHR-DAG-L violates upper bound constraints given by (5.10) then, in the solution of SHR-DAG, $f_i = f_{max} \forall i \in \Lambda$, where $\Lambda = \{i | \frac{f_{max}^2}{c_i \sum_{a=i}^{n} \mu_a} E_i'(f_{max}) \leq \frac{f_{max}^2}{c_j \sum_{a=j}^{n} \mu_a} E_j'(f_{max}) \forall j\}$*

Proof. This proof is very similar to that of Lemma 4. First, I obtain the necessary and sufficient Kuhn-Tucker conditions for the problem SHR-DAG, which include (5.9), (5.10) and:

$$E'_j(f_j) - \sum_{a=j}^{n} \mu_a \frac{c_j}{f_j^2} - \overline{\mu_j} + \nu_j = 0 \ (\forall 1 \leq j \leq n) \tag{5.26}$$

$$\mu_j(\sum_{a=1}^{j} \frac{c_a}{f_a} - b_j) = 0(\forall 1 \leq j \leq n) \tag{5.27}$$

$$\overline{\mu_j}(f_{low} - f_j) = 0(\forall 1 \leq j \leq n) \tag{5.28}$$

$$\nu_j(f_{low} - f_j) = 0(\forall 1 \leq j \leq n) \tag{5.29}$$

$$\mu_j \geq 0(\forall 1 \leq j \leq n) \tag{5.30}$$

$$\overline{\mu_j} \geq 0(\forall 1 \leq j \leq n) \tag{5.31}$$

$$\nu_j \geq 0(\forall 1 \leq j \leq n) \tag{5.32}$$

where μ_j, ν_j and $\overline{\mu_j}$ $(\forall j)$ are the Lagrange multipliers. Comparing the Kuhn-Tucker conditions for problems SHR-DAG and SHR-DAG-L, it is noted that at least one ν_j should be definitely larger than zero, if the solution of SHR-DAG-L violates some upper bound constraints of SHR-DAG. This is because when $\nu_j = 0$ $(\forall j)$, the equations (5.30) and (5.32) vanish and the equation (5.26) is identical to the equation (5.21), which means that both sets of Kuhn-Tucker conditions (hence, the optimal solutions of two problems) coincide. But this contradicts the assumption. Similarly, if $\exists i \ \nu_i > 0$, (which should be the case if the solution of SHR-DAG-L violates the constraints of SHR-DAG), then $\overline{\mu_j}$ should be zero. Otherwise, Equations (5.28) and (5.29) would suggest that $f_i = f_{low} = f_{max}$, which is a contradiction.

Then it will be proven that, under the condition specified in the lemma, the Lagrange multipliers ν_i ($\forall i \in \Lambda$) are all non-zero, which will imply (by (5.29)) that $f_i = f_{max}$. Assume that $\exists m \in \Lambda$ such that $\nu_m = 0$ (implying $\overline{\mu_m} > 0$. For the above discussion, it is known that there must exist task T_j such that $\nu_j > 0$ (implying $\overline{\mu_j} = 0$) and $f_j = f_{max}$. Applying equation (5.26), one gets: $\frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} E'_m(f_m) - \overline{\mu_m} \frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} = \frac{f_{max}^2}{c_j \sum_{a=j}^{n} \mu_a} E'_j(f_{max}) + \nu_j \frac{f_{max}^2}{c_j \sum_{a=j}^{n} \mu_a}$; then one can conclude that $\frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} E'_m(f_m) > \frac{f_{max}^2}{c_j \sum_{a=j}^{n} \mu_a} E'_j(f_{max})$. Since $f_m \leq f_{max}$, then $\frac{f_{max}^2}{c_m \sum_{a=m}^{n} \mu_a} E'_m(f_{max}) \geq \frac{f_m^2}{c_m \sum_{a=m}^{n} \mu_a} E'_m(f_m)$. Finally one can conclude that $\frac{f_{max}^2}{c_m \sum_{a=m}^{n} \mu_a} E'_m(f_{max}) > \frac{f_{max}^2}{c_j \sum_{a=j}^{n} \mu_a} E'_j(f_{max})$ which contradicts the assumption that $m \in \Lambda$. □

Before presenting the algorithm to find the optimal frequency assignments, I first give similar definitions and terms as used in Yao's algorithm [73]:

Definition 1. *The intensity of the interval $I = [z, z']$ is defined as to be:*

$$g(I) = \frac{\sum\limits_{T_i \in X_I} c_i}{z' - z} \quad \text{where} \quad X_I = \{T_j | z \leq b_j \leq z'\} \tag{5.33}$$

which is the average frequency required to execute all the tasks whose b_is values satisfy $z \leq b_i \leq z'$ for a given interval $I = [z, z']$.

Algorithm 2 summarizes the basic steps to find the optimal frequency assignment for SHR-DAG. Here, I first compute the effective deadline and the value of b_j for each task T_j. The time complexity for this step with n tasks is $O(n^2)$. Steps from line 3 to 13 operate in a way similar to Yao's algorithm. Specifically, the new algorithm first computes the interval with maximum intensity g_{max}, and then sets the frequency of all the tasks whose b_i values sare contained in that interval to g_{max}, and repeats the procedure for the remaining intervals. However, in order to satisfy the frequency bound constraints (f_{min} and f_{max}), the frequency obtained after invoking the Yao's algorithm must be adjusted in line 5. There is are most n iterations before the task set Θ becomes empty. During each iteration, it takes at most n steps to compute the value of $g(I)$ (line 4). Therefore, the time complexity to perform from line 3 to 13 is $O(n^2)$ and the overall complexity of Algorithm 2 is $O(n^2)$.

Algorithm 2 Optimal Frequency Assignment for SHR-DAG

```
1: Under ALAP policy, compute the effective deadlines $(D_1^e, D_2^e, ..., D_n^e)$ and $b_j = min\{D_k^e - \sum_{i=j}^{k} c_i (\forall\ k :\ j \le k \le n)\}$ values for all tasks under EDF scheduling.
2: Set $z = 0$, $\Theta = \{T_1, T_2, .., T_n\}$;
3: while $\Theta$ is not empty do
4:   Identify $T_m \in \Theta$ such that $g(I = [z, z' = b_m]) = g_{max} \ge g(I' = [z, b_i]) \forall\ T_i \in \Theta$;
5:   Let $s = max\{f_{low}, min\{g_{max}, f_{max}\}\}$;
6:   if $s = f_{low}$ then
7:     set the frequency of tasks in $\Theta$ to $s$ and return;
8:   else
9:     set the frequency of tasks in $\Lambda = \{T_i | b_i \le b_m \wedge T_i \in \Theta\}$ to $s$;
10:    $\Theta = \Theta - \Lambda$;
11:    set $z = z + \sum \frac{c_i}{s} (\forall i : T_i \in \Lambda)$;
12:  end if
13: end while
```

Evaluations and Discussions

To evaluate the performance of the new SHR-DAG scheme on both energy efficiency and system reliability, a discrete-event simulator with C/C++ was designed and implemented. For fair comparison, I also extended GRE and SUEF algorithms [88] for tasks with precedence constraints. These new extensions are called GRE-DAG and SUEF-DAG, respectively. I also implemented a static power management in DAG settings, namely SPM-DAG, which focuses on minimizing energy consumption while ignoring reliability degradations without scheduling any recovery task. In fact, SPM-DAG is a special case of SHR-DAG without reserving any recovery slack. In other words, SPM-DAG can be implemented by SHR-DAG algorithm by using D_j instead of b_j in Algorithm 2. Notice that SPM-DAG can not preserve system reliability and is included here to show the maximum achievable energy savings. The complexity of these schemes is given in Table 5.1.

For the parameters in system model, $\lambda_0 = 10^{-6}$ is given for the lowest error rate, which is realistic according to historical data of transient faults [30,93]. The processing frequency of the system can vary from $f_{min} = 0.1$ to $f_{max} = 1.0$. Furthermore, it is assumed $C_{ef} = 1$

Table 5.1: The complexity of the schemes (n is the number of tasks)

Scheme	Complexity	Scheme	Complexity
GRE-DAG	$O(n)$	SUEF-DAG	$O(n \log n)$
SPM-DAG	$O(n^2)$	SHR-DAG	$O(n^2)$

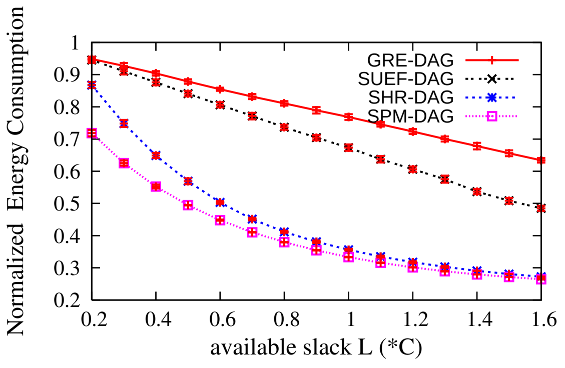


Figure 5.11: The impact of system slack on the probability of failure

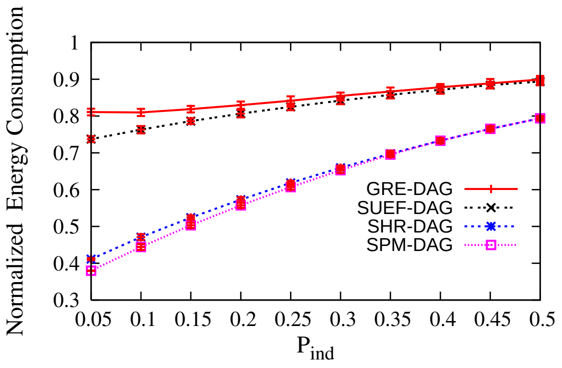


Figure 5.12: The impact of P_{ind} on system energy consumption

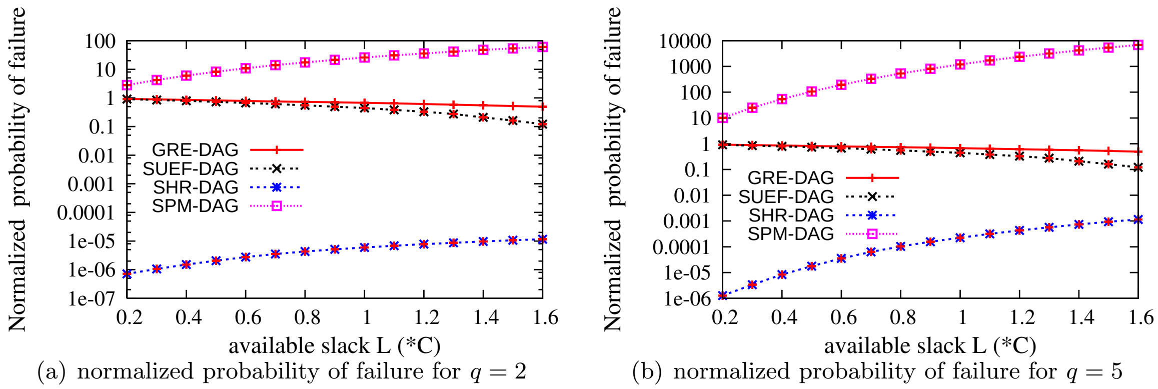
and $m = 3$. The frequency-independent power P_{ind} for each task is normalized with respect to the maximum frequency-dependent power $P_d^{max} = 1$. All results are normalized with respect to those of the *no power management (NPM)*, which executes all tasks at f_{max}.

The tasks are generated with the well-known TGFF tool [68]. Since different numbers of tasks per task set yield similar performance trends of all schemes, this chapter reports only the results for cases with 10 tasks per task set. Within a task set, the worst-case execution time c_i for each task is randomly generated by the UUniFast algorithm [11] in the range of $[10ms, 100ms]$. Each result point in the figures is obtained by averaging the values obtained by three types of graph topologies: 1) *Independent tasks*, without precedence constraints among any tasks; 2) *Chain Graph*, where the precedence constraints among tasks form a chain; and 3) *Tree Graph*. Also, for each graph topology, the simulation results are the values by averaging the values through the simulations. The confidence intervals at 97% confidence level are reported.

First, Figure 5.11 shows the impact of available slack on system energy consumption. In these experiments, the exponent q for the error rate in Equation (3.3) is set to 2 and the frequency-independent power is $P_{ind} = 0.05$. The available slack is represented by

$L = d - C$, where $C = \sum c_i$. Intuitively, the larger the slack, the more opportunities for DVS and energy savings. Therefore, all schemes can get lower energy consumption with increasing value of L. Moreover, compared to the existing RA-PM schemes (GRE-DAG and SUEF-DAG), SHR-DAG can obtain significantly more (up to 35%) energy savings. The reason is that, unlike SUEF-DAG and GRE-DAG that allocate separate recovery tasks statically, SHR-DAG reserves only a small amount of slack as a shared recovery and is able to allocate much more slack for DVS to scale down tasks' execution and save energy. Moreover, as more slack is available gets (e.g. $L \geq 0.6 \cdot C$), the performance of SHR-DAG becomes remarkably close to that of SPM-DAG, which stands for the upper bound for any DVS-based energy management algorithm.

Figure 5.12 shows the relationship between different frequency-independent active power P_{ind} and the system energy consumption when the available slack is set as $L = 0.8 \cdot C$. The minimum value of P_{ind} is set to 0.05. The energy consumption for all schemes increases with increased P_{ind}. This is because, with larger values of P_{ind}, the frequency-independent energy consumption increases and the threshold of energy-efficient frequency for tasks become higher, which limits the DVS opportunities. Again, SHR-DAG outperforms GRE-DAG and SUEF-DAG and its energy performance is extremely close to the bound obtained by SPM-DAG.



(a) normalized probability of failure for $q = 2$ (b) normalized probability of failure for $q = 5$

Figure 5.13: The impact of slack on system reliability

Next, the system reliability dimension is evaluated for these schemes. For convenience, the *probability of failure (PoF)* is presented (defined as $1-$*reliability*) achieved by all schemes.

Again, all results are normalized to that of NPM. Figures 5.13(a) and 5.13(b) show the reliability performance when the exponent q are 2 and 5, respectively. Here, P_{ind} is set to 0.05 for all tasks. Without special provision for reliability, SPM-DAG can lead to several orders of magnitude degradation for system reliability. As expected, all RA-PM schemes (SHR-DAG, GRE-DAG and SUEF-DAG) can preserve the system's original reliability.

More interestingly, the results point to a somewhat counter-intuitive phenomena: SHR-DAG has also a clear advantage on the reliability side through the evaluated spectrum, despite the fact that it uses a shared recovery block for all tasks. The main reason comes from the fact that GRE-DAG and SUEF-DAG allocate separate recovery tasks statically. As a result, except for cases where the available slack is very large, only a subset of tasks can be managed by SUEF-DAG and GRE-DAG while the remaining tasks do not receive any recovery. On the other hand, SHR-DAG allocates a single recovery block that can be used by *any* task in case it fails. In essence, SHR-DAG provides a better protection for the single-fault scenarios that are typically much more likely than multiple-fault scenarios that SUEF-DAG and GRE-DAG provision for, at the expense of losing recovery for some tasks. When more slack is available, the *PoF* of SHR-DAG increases slightly as the system can use more aggressive voltage scaling. For the cases where the error rate is very sensitive to the voltage scaling (i.e., $q = 5$), the normalized *PoF* of SHR-DAG approaches (but never exceeds) 0.001.

5.3.3 Online Extension of SHR-DAG

While the offline SHR-DAG scheme provides a powerful mechanism to save more energy while preserving system original reliability, some run-time optimizations can further improve its performance. Typically, real-time applications only consume a small fraction of their WCETs and abundant amount of dynamic slack can be available at run-time. In fact, many DVS frameworks [7,48,55] have been proposed in the past to exploit the *dynamic slack* that arises from early completions of tasks for better energy savings. The same approach can be incorporated in SHR-DAG as well.

In this section, an extension is considered to the static SHR-DAG scheme. The online SHR-DAG algorithm, namely DSHR-DAG, will dynamically reclaim excessive slack at run-time to further scale down the processing frequency of tasks for better energy savings. Specifically, DSHR-DAG operates as follows: initially, SHR-DAG is invoked to obtain the scaled frequency for the first running task. Then, at run-time, when a task completes early and/or without encountering errors, the frequency assignment for the next task will be updated by re-invoking part of SHR-DAG with the size of recovery block needed and the worst-case workload for the remaining tasks. The details of DSHR-DAG is given in Algorithm 3. At runtime, the system original reliability can be strictly guaranteed since frequency assignments are decided by iteratively invoking SHR-DAG, which can preserve system original reliability as shown in Lemma 2. When without knowing the actual execution times in advance, applying SHR-DAG among remaining tasks for each step can maximize energy savings. In other words, at each scheduling point, SHR-DAG algorithm should be invoked. Hence the complexity of DSHR-DAG at each scheduling point is $O(j^2)$, where j is the total number of remaining tasks. A further observation is that, at runtime, only the speed assignment should be decided for the ongoing task. That is, SHR-DAG algorithm can be stopped when the speed assignment for current ongoing task is obtained (i.e., just a single iteration from line 4 to line 5 in Algorithm 2 with the complexity $O(j)$). Therefore, the complexity of DSHR-DAG at each scheduling point can be improved to $O(j)$. In the following simulations, the improved version of DSHR-DAG is used.

Evaluations and Discussions

In this section, the performance DSHR-DAG is evaluated. For comparison, I also implemented DGRE-DAG and DSUEF-DAG by extending GRE-DAG and DSUEF-DAG to dynamic settings, respectively. Moreover, assuming that the actual workload of tasks is known, a *clairvoyant* scheme (denoted as Bound-DAG) computes an absolute bound for energy savings following the principle of SPM-DAG.

The simulation settings are essentially the same as those in Section 5.3.2. Moreover, to

Algorithm 3 Online Frequency Assignment for DSHR-DAG

```
1: Under ALAP policy, compute the effective deadlines $(D_1^e, D_2^e, ..., D_n^e)$ and $b_j = min\{D_k^e - \sum_{i=j}^{k} c_i (\forall\ k:\ j \le k \le n)\}$ values for all tasks under EDF scheduling.
2: for $i := 1$ to $n$ do
3:     Invoke SHR-DAG algorithm (Algorithm 2) for tasks $T_i$ to $T_n$ to get initial frequency assignments $\{f_i\}$;
4:     Execute $T_i$ at the frequency level $f_i$ and record its completion time $a_i$.
5:     if a fault is detected at the $T_i$'s completion then
6:         Schedule a recovery in form of re-execution at $f_{max}$ and update $a_i = a_i + c_i$;
7:     end if
8:     for $j := i + 1$ to $n$ do
9:         $b_j = b_j - a_i$; //update $b_j$ by considering the total elapsed time
10:    end for
11: end for
```

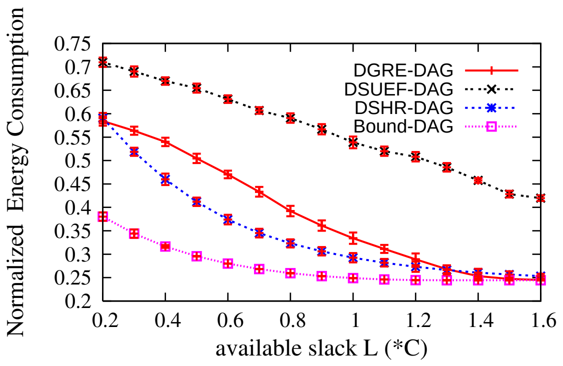


Figure 5.14: The impact of slack on system energy consumption

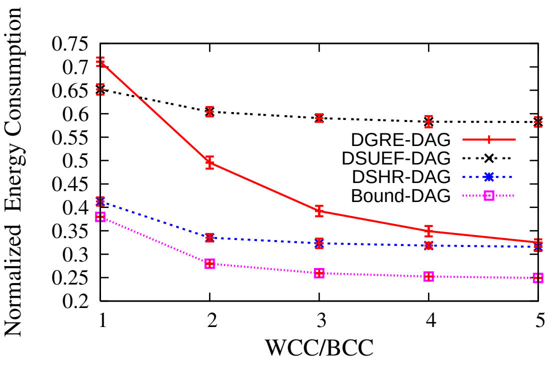


Figure 5.15: The impact of $\frac{WCC}{BCC}$ ratio on system energy consumption

model the variations of the actual workload of tasks, the ratio $\frac{WCC}{BCC}$ is used, which denotes the ratio of the worst-case execution time (WCC) to the best-case execution time (BCC). At run-time, the actual execution time of each task is determined between its BCC and WCC with the uniform probability distribution. Here, the higher ratio of $\frac{WCC}{BCC}$ indicates that the actual execution time of tasks deviates more from the worst-case and more dynamic slack can be expected at run-time. In the simulations, $q = 2$ is set and all results are normalized with respect to that of the NPM scheme.

Figure 5.14 first shows the effects of available slack L on system energy consumptions, for $\frac{WCC}{BCC} = 3$ and $P_{ind} = 0.05$. As before, when more slack is available, more energy savings can be obtained. For most cases, DSHR-DAG has clear advantages over DGRE-DAG and DSUEF-DAG. However, when the available slack is limited (e.g. $L = 0.3 \cdot C$), the difference between the clairvoyant algorithm *Bound-DAG* and DSHR-DAG is more significant since Bound-DAG can utilize lower frequencies with the knowledge of tasks' actual execution time.

An interesting observation is the greatly enhanced performance of DGRE-DAG (compared to the worst-case settings). When $\frac{WCC}{BCC} = 3$, DGRE-DAG well outperforms SUEF-DAG and its achieved energy savings are quite close to that of DSHR-DAG. The reason is that, many tasks complete much early at such high workload variation settings. As suggested by previous DVS research [7, 48], re-using the slack as early as possible typically pays off. However, DSUEF-DAG chooses to re-allocate the slack according to slack usage efficiency factors, which can possibly exclude the next tasks that would immediately benefit. When L is very large, DGRE-DAG, DSHR-DAG and *Bound-DAG* converge to almost the same level, which is determined by the energy-efficient frequency.

Figure 5.15 further illustrates how system energy consumptions changes for different variation of tasks' actual execution time workload (i.e., different ratios of $\frac{WCC}{BCC}$) with $L = 0.8 \cdot C$ and $P_{ind} = 0.05$. As expected, the system energy consumption generally decreases with decreasing actual workloads (i.e., large rations of $\frac{WCC}{BCC}$ ratios). Among the three schemes, DSHR-DAG obtain the most energy savings, which is close to the yardstick algorithm *Bound-DAG* by at most a margin of 7%.

Interestingly, when $\frac{WCC}{BCC} = 1$, DGRE-DAG's performance quickly deteriorates. It is because, when the actual workload does not exhibit high variations, DSUEF-DAG assigns slack according to tasks' slack usage efficiency factors, which typically yields better results, compared to DGRE-DAG that make all available slack to the next task.

Next, Figure 5.16 shows the probability of failure as the function of the available slack with $P_{ind} = 0.05$ and $\frac{WCC}{BCC} = 5$. Although DSHR-DAG still performs the best, DGRE-DAG and DSUEF-DAG greatly benefit from dynamic slack reclamation, especially for larger value of L. Essentially, DGRE-DAG and DSUEF-DAG can allocate more recovery tasks at run-time by recycling the slack of unused recoveries, which leads to smaller values of

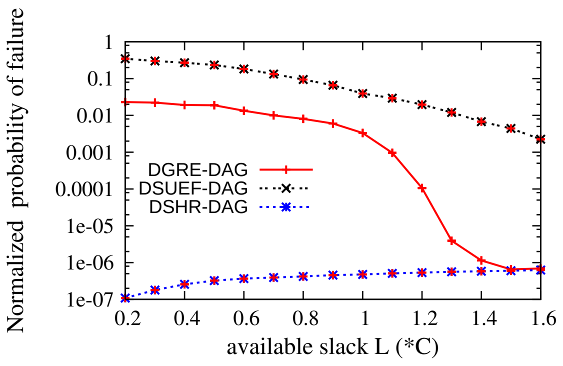


Figure 5.16: The impact of the system slack on the probability of failure

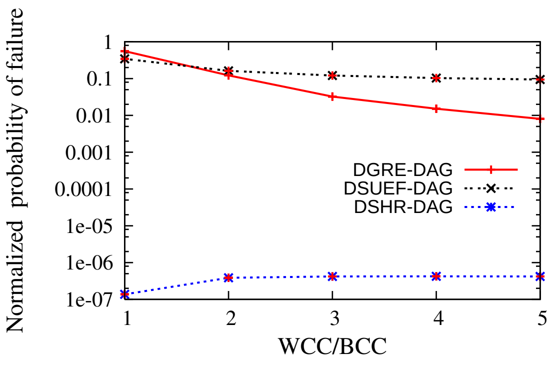


Figure 5.17: The impact of $\frac{WCC}{BCC}$ ratio on the probability of failure

PoF. Another interesting observation is that, just like the energy dimension, DGRE-DAG appears to outperform DSUEF-DAG because of re-assigning recovery tasks in a greedy fashion. Moreover, when $L \geq 1.5 \cdot C$, almost all tasks will be effectively executed at the energy-efficient frequencies by DGRE-DAG and DSHR-DAG, which gives similar PoF values.

Figure 5.17 illustrates how system reliability changes for different variations of tasks' actual workload when $L = 0.8 \cdot C$ and $P_{ind} = 0.05$. In general, the probability of failure for DGRE-DAG and DSUEF-DAG decreases as the actual workload becomes smaller (i.e. larger ratio of $\frac{WCC}{BCC}$) since more slack becomes available due to early completions of tasks. Because of the greedy nature of DGRE-DAG (i.e. aggressively allocate all dynamic slack to the next task), the achieved system reliability under DGRE-DAG tends to degrade with smaller values of $\frac{WCC}{BCC}$. When $\frac{WCC}{BCC} \geq 2$, DSHR-DAG can execute almost all tasks at the energy-efficient frequencies, which yields almost stable PoF values.

5.4 Chapter Summary

The existing RA-PM schemes normally schedule a separate recovery task for any task whose execution is scaled down, which is rather conservative as the recovery tasks may not be needed simultaneously, especially for non-preemptive execution of frame-based real-time tasks.

In this chapter, a new RA-PM scheme is proposed and evaluated, called the shared recovery (SHR) technique for both *independent* and *dependent* frame-base tasks. First, considering independent frame-based tasks, in the offline phase, SHR allocates CPU time only for the largest recovery that may be needed by any task, essentially making available to the DVS mechanism the valuable slack that would be typically reserved for the recoveries of separate tasks in the previous RA-PM solutions. The energy savings of the new scheme are shown to approach those of the optimal (but not reliability-aware) DVS solutions. Further, SHR is formally shown to preserve the original system reliability. The experimental evaluation indicates that SHR has also a clear advantage on the reliability dimension for most of the spectrum that this chapter considered. Also, this chapter proposed a dynamic extension of the scheme, called DSHR, that is able to reclaim any dynamic slack that results from early completions or not needed recovery operations. This dynamic reclamation enables the system to further improve the energy savings, effectively approaching the performance of a potentially clairvoyant algorithm.

Second, while focusing on a set of frame-based real-time tasks with precedence constraints that have individual deadlines and the common period, a *shared-recovery* based RA-PM scheme (namely SHR-DAG) was propose and evaluate. First, based on tasks' effective deadlines that are derived from precedence constraints of tasks following as late as possible (ALAP) policy, I show that EDF scheduling can always obtain a valid schedule provided that the task set is feasible. Based on the shared-recovery technique, where all scaled tasks can share a single recovery block, the SHR-DAG scheme is shown to be able to preserve system reliability. For a given valid schedule with fixed execution order of tasks, the static optimal frequency assignment of SHR-DAG is presented and proved. I further propose an online extension of SHR-DAG aiming at better utilizing dynamic slack for more energy savings. Simulation results show that, compared to the existing individual-recovery based RA-PM schemes, SHR-DAG can not only obtain more energy savings but also have a clear advantage on the reliability dimension. The energy savings under SHR-DAG are very close to the upper-bounds for both static and dynamic settings.

Chapter 6: Generalized Reliability-Oriented Energy Management

6.1 Introduction

Existing RA-PM studies share certain fundamental characteristics and limitations. First, the necessary and sufficient objective is casted as to *preserve* the task set's *original reliability*, which is defined as the reliability when all tasks run at the maximum frequency, and without any recovery task. Second, to achieve this aim a *separate* recovery task is scheduled for every task that runs at a low-frequency.

The main objective of this chapter is to lay the foundations of a more general RA-PM framework by addressing these two limitations. The motivation to address the first point is very natural: a more comprehensive framework is highly desirable in order to achieve *any* reliability goal, which can be set by the designer to be *lower* or *higher* than the application's original reliability. This is because, some modest reliability degradation might be acceptable in energy-scarce settings. In contrast, it may be necessary to achieve very high reliability levels for safety-critical systems (e.g. electronics-hostile settings encountered in space applications). Clearly, merely maintaining task-level reliabilities would be too conservative or insufficient, respectively, in these two scenarios.

The second motivation is to further *reduce energy consumption*: by scheduling a separate recovery for every scaled task, the existing RA-PM solutions inherently reduce the amount of available slack for DVS and thus the extent of energy savings. A preliminary solution to this problem was recently proposed: the technique, called *shared recovery (SHR)*, reserves only one shared recovery task that can be used by any *single* faulty task at runtime as described in Chapter 5. Hence, SHR can save more energy as larger slack is made available for slow-down through DVS. Upon the detection of a fault, the shared recovery task is invoked and the remaining tasks are forced to run at the maximum frequency until the end of the current period to preserve the system's original reliability. However, this technique is not suitable for obtaining reliability levels that are different than the original reliability. Moreover, the SHR technique is not applicable to *preemptive* execution settings, even when the target reliability is set to the original reliability. This is because, once a fault has been detected and a recovery task has been executed, switching to the maximum frequency no longer provides guarantees to preserve the original reliability of the tasks that are partially executed and await resumption in the ready queue. Consequently, a more general approach that can be applied also to periodic execution settings where preemption may be necessary is highly desirable.

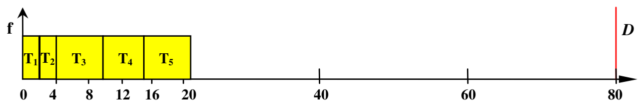
In this chapter, the *Generalized Shared Recovery (GSHR)* technique is first presented to optimally use the DVS technique in order to achieve a given reliability goal for frame-based real-time embedded applications. Then a more general framework is proposed, where the aim is to achieve arbitrary reliability levels that may vary for each periodic task.

Contributions of this chapter are published in [84, 85].

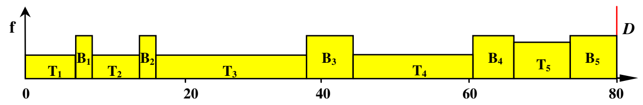
6.2 Targeting Arbitrary Reliability for Frame-based Applications

In this section, considering frame-based tasks, the *Generalized Shared Recovery (GSHR)* technique is proposed to potentially use *any* number of recoveries, and using this as a leverage, I tackle the generalized **Energy-Optimal Reliability Configuration (EORC)** problem: *determine the number of recoveries and task-level frequency assignments to minimize the system level energy consumption while satisfying given target reliability and deadline constraints.* Note that each additional (potential) recovery task, while improving the reliability, reduces the available slack for DVS and limits the opportunities for scaling down frequencies. In other words, the problem mandates an efficient technique in three-dimensional *timing, energy,* and *reliability* space to satisfy all the constraints. For this problem a fast algorithm is developed with comparable performance to that of an optimal, but computationally expensive exhaustive-search based solution.

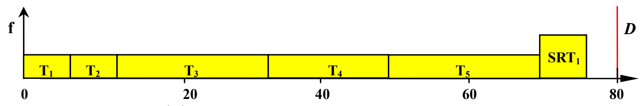
Motivational Example. To illustrate the potential gains that can be obtained by the new GSHR technique, let us consider an application with five tasks and a common period/deadline of $80\,ms$. The worst-case execution times of T_1 and T_2 are given as 2 ms each, while those of T_3 and T_5 are $6\,ms$ and that of T_4 is $5\,ms$. As can be seen in Figure 6.1(a), originally there is $80 - 21 = 59\,ms$ slack when all tasks execute at the maximum frequency $1\,GHz$ (*No Power Management* case). In that case, by using the fault rate model from [88], the original *probability of failure (PoF)* is computed as $\rho_0 = 2.1 \times 10^{-7}$. Here the PoF is defined as $1 -$ *reliability*, where the *reliability* is the probability that all tasks will complete without encountering soft errors caused by transient faults. If we consider the traditional greedy RA-PM scheme [88], five *separate* recovery tasks (denoted by $\{B_i\}$) will be statically allocated to five tasks, yielding the RA-PM schedule depicted in Figure 6.1(b). In that schedule, the first four tasks are executed at the energy-efficient frequency [36] $f_{ee} = 0.29\,GHz$ and the fifth task runs at the frequency $0.78\,GHz$. Applying the power model from [88] indicates that the energy consumption of the RA-PM solution is 7.88 and its actual PoF is 9×10^{-10}. Now, if the objective is to merely preserve the original reliability ρ_0, alternatively, we can use the GSHR technique with one shared recovery task of size $6\,ms$ (denoted by SRT_1) as shown in Figure 6.1(c). Then, all the tasks can be executed at the frequency $0.31\,GHz$. Simple calculation shows that the total energy consumption is 5.4, giving a 32% reduction with respect to RA-PM. Figure 6.1(d) presents the GSHR solution with two recoveries that yield the exact reliability obtained from traditional RA-PM scheme (Figure 6.1(b)). Thus, one can obtain the reliability achieved by RA-PM through GSHR with 15% less energy (6.68), by using the frequency $0.46\,GHz$. Notice that with two shared recoveries, the system can also potentially recover from two faults affecting the same task (unlike RA-PM that statically allocates one recovery to each task), which enables achieving the target reliability yielded by RA-PM ($PoF = 9 \times 10^{-10}$) with more energy savings.



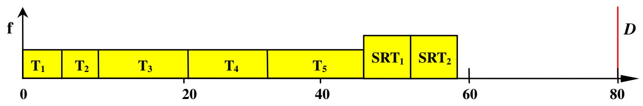
(a) No power management scheme



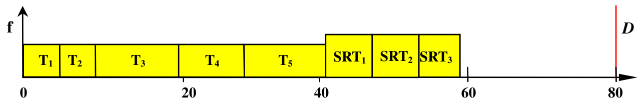
(b) The RA-PM scheme (from [88])



(c) GSHR with one recovery



(d) GSHR with two recoveries



(e) GSHR with three recoveries

Figure 6.1: The motivational example

Further, through GSHR one can achieve reliability levels even higher than that achieved by RA-PM, for example $PoF = 5 \times 10^{-10}$, by deploying three shared recoveries and still consuming less energy than RA-PM (7.47) at the uniform frequency level of $0.51\,GHz$. In all these examples, the potential gains of GSHR were illustrated in terms of energy and reliability by using a simple *uniform frequency* approach – as I demonstrate later in this section, these gains can be further improved by optimizing the frequency selection.

The remainder of this section is organized as follows. The generalized shared recovery technique (GSHR) and the energy-optimal reliability configuration problem are introduced Section 6.2.1. The new specific solution to the problem, the *IRCS* algorithm, is presented in Section 6.2.2. Section 6.2.3 presents the simulation results.

6.2.1 Generalized Shared Recovery Technique

In Chapter 5, the *shared recovery (SHR)* technique has been proposed where all tasks share one recovery, which is utilized to recover the first faulty task and the remaining tasks within one frame (period) run at f_{max} for reliability preservation. Extending this idea, for general reliability goals, a fixed number ($j \geq 0$) of recovery blocks can be employed, which will be shared by n tasks within a frame. By choosing proper task frequency assignments, a broad spectrum of reliability can be obtained. This scheme is called *Generalized Shared Recovery (GSHR)* technique. Unlike SHR, *task-level* reliability degradation is allowed in GSHR, as long as the system's target reliability is still achieved. Specifically, in GSHR, after a fault and possible recovery/ies, the remaining tasks can still run at their pre-calculated scaled frequencies. Moreover, for j shared recoveries, GSHR reserves a total CPU time equal to the sum of the first j largest tasks.

Let $R^j(T_1, .., T_n)$ denote the reliability (i.e. the probability of completing successfully the execution of tasks $T_1, .., T_n$) by using the slack of j recoveries if necessary. Further, let R_g be the reliability goal to achieve. Then, the **Energy-Optimal Reliability Configuration (EORC)** problem can be formally expressed as: find the optimal number j of recovery tasks and frequency assignments $f_1, f_2, ..., f_n$ to minimize:

$$E = \sum_{i=1}^{n} E_i(f_i) = \sum_{i=1}^{n} (P_{ind,i} \frac{c_i}{f_i} + ac_i f_i^{\alpha-1}) \tag{6.1}$$

Subject to:

$$R^j(T_1, .., T_n) \geq R_g \tag{6.2}$$

$$\sum_{i=1}^{n} \frac{c_i}{f_i} \leq D' \tag{6.3}$$

where $D' = d - S_j$, and S_j is the total slack required for j recoveries.

Intuitively, with more recoveries, the target reliability R_g can be achieved more easily. However, the deadline constraint further complicates the problem, as each additional recovery will reduce the available slack and limit the opportunities for DVS to save energy. Figure 6.2 depicts these non-trivial trade-off dimensions for the simultaneous consideration

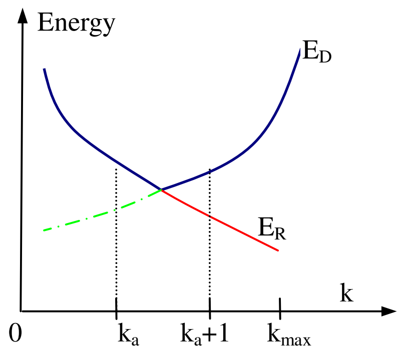


Figure 6.2: Energy-Optimal Reliability Configuration

of the reliability and deadline constraints. The curve E_D illustrates the minimum energy consumption that can be achieved when deploying k recoveries by ignoring the reliability constraint R_g. Similarly, the curve E_R gives the minimum energy consumption figure satisfying the target reliability constraint, but without taking the deadline constraint into consideration. Since the actual solution must satisfy both reliability and deadline constraints, the best energy performance that can be achieved with k recoveries corresponds to the union of the upper parts of these two plots, namely $max\{E_D(k), E_R(k)\}$. As a result, the optimal number of recoveries is either k_a or $k_a + 1$ as illustrated in the figure.

Before concluding this section, I sketch how the reliability of n tasks with j recoveries, namely $R^j(T_1, .., T_n)$, can be computed. The overall reliability when no recovery task is scheduled, is by definition the product of individual task reliabilities, i.e. $R^0(T_1, .., T_n) = \prod_{i=1}^{n} R_i(f_i)$ [88]. Similarly, considering that the recovery tasks are always invoked at f_{max}, the reliability with a single recovery is given by [88]:

$$\begin{aligned} R^1(T_1, .., T_n) = R_1(f_1)R^1(T_2, .., T_n) + \\ (1 - R_1(f_1))R_1(f_{max})R^0(T_2, .., T_n) \end{aligned} \tag{6.4}$$

In the general case, the reliability with j recoveries can be expressed recursively as:

$$R^j(T_1,..,T_n) = R_1(f_1)R^j(T_2,..,T_n) + \\ (1 - R_1(f_1))R_1(f_{max})R^{j-1}(T_2,..,T_n) \quad (6.5)$$

The first term above corresponds to the scenario where task T_1 completes successfully and all j recoveries are made available to the remaining tasks. Similarly, the second term captures the scenario where task T_1 fails and subsequently is recovered with one recovery block. The remaining tasks will have $j-1$ recoveries to complete their executions. The value of $R^j(T_1,..,T_n)$ can be quickly computed by the well-known dynamic programming technique (with linear asymptotic complexity $O(j \cdot n)$).

6.2.2 Computing Energy-Optimal Configurations: Algorithm IRCS

While an exact and fast solution may be out of reach due to the multi-dimensional interplay of *deadline, energy* and *reliability* factors, in the following, I propose an algorithm called *Incremental Reliability Configuration Search (IRCS)* and establish its good performance through experimental evaluation. The basic idea of the IRCS algorithm is to iteratively determine the best frequency assignments for different recovery task allocation options, and select the best alternative at the end. That is, for a given number of recoveries k, IRCS reduces the frequencies of individual tasks as long as the overall reliability is no smaller than the target reliability, R_g. Initially, $f_i = f_{max} = f_\ell$ is set for all tasks. Then the task is scaled down to the next (lower) frequency level, by estimating the task that would yield large energy savings with relatively low reliability degradation in every step.

Specifically, I first define $W_i^j = E_i(f_{j+1}) - E_i(f_j)$, which indicates the energy savings obtained when the task T_i's frequency is scaled down from f_{j+1} to f_j. Similarly, $Q_i^j = R_i(f_{j+1}) - R_i(f_j)$ indicates the reliability degradation following this frequency reduction. Now, I define $ERR_i^j = \frac{W_i^j}{Q_i^j}$ as the *energy-reliability ratio (ERR)* of task T_i when scaling

Algorithm 4 Incremental Reliability Configuration Search (IRCS)

```
1: For all tasks, compute $W_i^j$, $Q_i^j$, $S_i^j$ and $ERR_i^j$;
2: Sort all task execution time in decreasing order as $b_1, ..., b_n$;
3: $AC = d - \sum_{i=1}^{n} c_i$;
4: $k = max\{ j \mid \sum_{i=1}^{j} b_i \leq AC\}$ //compute the maximum number of recoveries.
5: Set $k_{opt} = 0$;
6: Set $f_i = f_\ell$ $(i = 1, ..., n)$;
7: Set $E = \sum E_i(f_i)$;
8: for $j = 0$ to $k$ do
9:     if $j > 0$ then
10:        $AC = AC - b_j$
11:    end if
12:    Assign-frequencies($\{S_i^m\}, AC$);
13:    Compute the total energy consumption $E_{total}$ under the new frequency assignments $\{f_i^*\}$
14:    if $E > E_{total}$ then
15:        Set $E = E_{total}$;
16:        Set $k_{opt} = j$;
17:        Set $f_i = f_i^*$ $(i = 1, .., n)$;
18:    end if
19: end for
20: return $k_{opt}$ and $f_1, ..., f_n$
```

down from frequency $j+1$ to the frequency j. ERR is a measure of *utility* (i.e. energy savings per unit reliability degradation) that guides this search.

The IRCS algorithm (Algorithm 4) iterates over possible number of recoveries (Lines 8-19), selecting the best frequency assignment along the way. For this purpose, Algorithm 5 *Assign-Frequencies* is invoked to choose the task with the largest ERR value for the subsequent slow-down decision(s). *Assign-Frequencies* continues to reduce the task frequencies iteratively by this principle as long as the target reliability R_g is still satisfied. Note that, the input to Algorithm 5 is an array of task-level frequency-dependent slack usage factors $\{S_i^m\}$ and available total slack for DVS (AC) after reserving the CPU time for the given number recoveries. $S_i^j = \frac{c_i}{f_j} - \frac{c_i}{f_{j+1}}$ denotes the CPU time needed to scale task T_i from the frequency f_{j+1} to f_j.

Algorithm 5 *Assign-frequencies*($\{S_i^m\}, AC$)

```
1: For all tasks, set frequency $f_i^* = f_\ell$ and $z_i = \ell - 1$;
2: Set $X = AC$;
3: while $X > 0$ and there exists $i$ such that $S_i^{z_i} \leq X$ do
4:     In the set $\alpha = \{T_i | S_i^{z_i} \leq X\}$, find task $T_y$ with maximum $ERR_y^{z_y}$ value;
5:     $f_y^* = f_{z_y}$;
6:     Compute the new system reliability $R_{new}$ by Eq. (6.5);
7:     if $R_{new} < R_g$ then
8:         Set $f_y^* = f_{z_y+1}$ and break
9:     else
10:        Set $X = X - S_y^{z_y}$;
11:        Set $z_y = z_y - 1$;
12:    end if
13: end while
14: return $f_1^*, ..., f_n^*$
```

6.2.3 Performance Evaluation

To evaluate the performance of the new proposed solution, a discrete-event simulator was implemented in C programming language. In the simulator, the following four schemes were implemented:

- The *uniform frequency* **(UF)** scheme, which chooses the best *constant* frequency level for all tasks. UF simply evaluates all discrete frequency levels (whose number is typically a small integer) in terms of energy consumption and meeting the reliability goal, and selects the best solution.
- The new *incremental reliability configuration search* **(IRCS)** scheme which tries to obtain best energy savings by scaling down the tasks step by step based on their energy-reliability ratio (ERR) values.
- The greedy *RA-PM* **(RAPM)** scheme [88] which aims to minimize the energy consumption and still maintain the system's original reliability. Notice that RAPM is designed to reach a reliability level no less than the system's original reliability, and cannot be configured to yield *arbitrary* reliability figures.
- The *Optimal* **(OPT)** scheme, which obtains the energy optimal reliability configuration by performing an exhaustive search in all possible frequency assignments for all

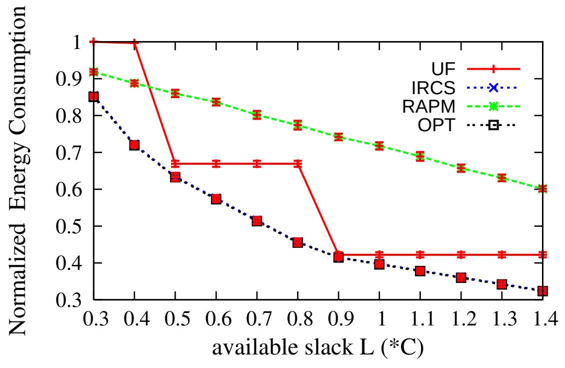


Figure 6.3: The impact of slack on energy when $R_g = 1 - \rho_o$

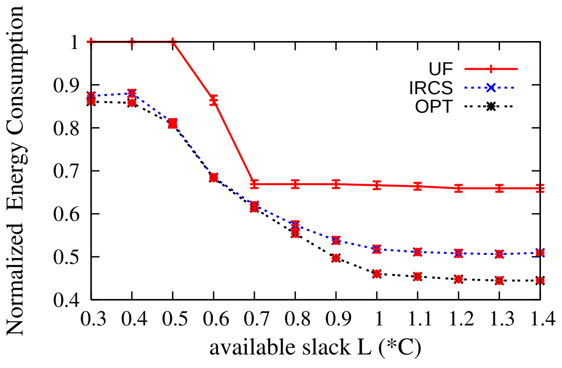


Figure 6.4: The impact of slack on energy when $R_g = 1 - \frac{\rho_o}{1000}$

the tasks. Even with small number of frequency levels, the number of distinct task frequency assignments grows exponentially, and this solution is not scalable. While not practical, OPT is included in the comparison as a yardstick algorithm.

In the simulations, transient faults are assumed to follow Poisson distribution with an average fault rate of $\lambda_0 = 10^{-6}$ at f_{max}, which is a realistic fault rate as reported in [93]. Moreover, the fault rate exponent q is set to 2. The six discrete frequency levels that I assume are modeled after Intel Xscale processor [72]. A cubic frequency-dependent power component P_d is used, which is equal to unity at $f_{max} = 1.0$. The frequency-independent power component P_{ind} for each task is normalized with respect to P_d and is generated according to the uniform distribution in the range of $[0, 2]$.

For each task set (that contains 10 tasks), the worst-case execution time c_i for each task is randomly generated through a uniform distribution, such that the worst-case execution time under maximum frequency falls in the range of $[1ms, 10ms]$. All energy consumption results are normalized with respect to the *no power management (NPM)* scheme that executes all tasks without any frequency and voltage scaling at f_{max}. The original probability of failure PoF without any energy management (i.e. $1 - R_0 = 1 - \prod R_i(f_{max})$) is denoted by ρ_o. Each point in the presented figures is obtained by averaging the results obtained through simulations.The confidence intervals at 97% confidence level are reported.

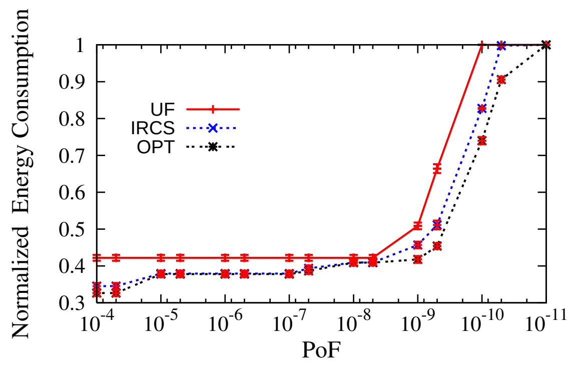


Figure 6.5: The impact of PoF on system energy consumption

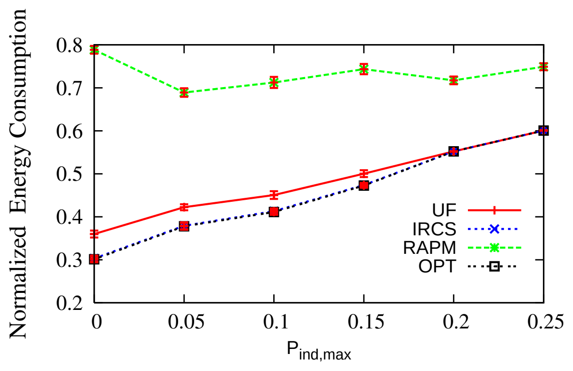


Figure 6.6: The impact of P_{ind} on system energy consumption

First, I analyze the impact of available slack on the energy consumption, as shown in Figures 6.3 and 6.4. The frequency-independent power is set to $P_{ind} = 0.05$ and the available slack is represented by $L = D - C$, where $C = \sum c_i$. In Figure 6.3, the target reliability is set to original system reliability $R_g = 1 - \rho_o$. The energy savings of IRCS is extremely close to that of OPT and represents an improvement of up to 30% compared to RAPM. Notice that even UF is able to outperform RAPM, when using GSHR for recovery task assignments. The larger slack, the more more opportunities for both DVS and recovery assignment; and all schemes achieve more energy savings with increasing L. At large L values, IRCS and UF can easily use low frequencies by deploying suitable number of shared recoveries. However, RAPM's options are more limited, as it is forced to allocate separate recovery tasks to different tasks.

Figure 6.4 repeats the evaluation for the case when a more strict reliability goal of $R_g = 1 - \frac{\rho_o}{1000}$ is imposed. In other words, in this case the system is supposed to be configured to reach a probability of failure level 1000 times smaller than the original case. RAPM simply cannot reach this very high reliability level, and thus is excluded from comparison. While similar trends are observed here, note that the energy consumptions of all schemes increase due to high-frequency executions, mandated by a more stringent reliability goal. The difference between IRCS and OPT becomes more visible in this scenario, however it does not exceed 10%.

Figure 6.5 further illustrates the impact of the reliability target R_g on system energy consumption, with the available slack $L = 1.1 \cdot C$ and $P_{ind} = 0.05$. Here, the target PoF varied from 10^{-4} to 10^{-11}. As expected, the energy consumptions of all schemes get higher with increasing reliability objective (i.e decreasing PoF). This is due to the need for scheduling more recovery tasks to meet the target reliability, resulting in less available slack for DVS. When $PoF < 10^{-9}$, the energy savings of IRCS is very close to that of the ideal OPT scheme. After that, OPT starts to perform slightly better, but as mentioned before, at very high computational cost. When $PoF = 10^{-11}$, all schemes are forced to run at the maximum frequency f_{max}, and also deploy recoveries, to achieve this high reliability level.

Finally, I investigate the effects of different frequency-independent active power P_{ind} on the system energy savings when $L = 1.1 \cdot C$ and the target reliability is the system original reliability $R_0 = 1 - \rho_0$. P_{ind} value for each task T_i is randomly generated from $[P_{ind,min}, P_{ind,max}]$ where $P_{ind,min}$ is set to 0. Figure 6.6 shows that energy consumptions of all schemes increase with increasing $P_{ind,max}$. This is because, as $P_{ind,max}$ increases, the frequency-independent energy consumption increases and further, the energy-efficient frequency thresholds for tasks become higher, which limits the DVS opportunities. Consequently, when $P_{ind,max} \geq 0.2$, the energy consumptions of the IRCS and UF coincide with that of ideal OPT scheme. After this threshold, by executing all tasks at the energy-efficient frequency levels with the suitable numbers of shared recoveries, the original reliability can be preserved.

6.3 Energy Management under General Task-Level Reliability Constraints

In this section, I propose a more general framework where the aim is to achieve arbitrary reliability levels that may vary for each periodic task. The main objectives of this research effort are twofold:

1. To lay the foundations of a more comprehensive framework to achieve *arbitrary* reliability levels for *individual* periodic tasks, when employing DVS. This may prove very useful for applications with different/mixed criticality (or, importance) levels whose requirements may not be fully captured by simply preserving the original reliability levels. For instance, some critical tasks may require very high reliability levels – in fact, reliability levels that require the use of recoveries even when not using DVS may be sought. Conversely, for some other tasks, a modest reliability reduction may be acceptable in exchange for high energy savings. *Such task-variant reliability objectives can neither be expressed nor achieved in existing RA-PM solutions.*

2. To investigate and exploit the potential of deploying *dynamically allocated recoveries* for periodic tasks, in co-management of reliability and energy. As opposed to the current RA-PM schemes that *statically* allocate a *separate* recovery to each and every scaled job, the new framework is based on providing every periodic task with a certain *recovery allowance* for the execution. The recoveries can be used by any number of the jobs of the task under consideration during the hyperperiod, as long as the recovery allowance is not exceeded. The analysis and results indicate that such a dynamic recovery allocation strategy is highly effective, in the sense that: *i.* even with rather small number of recovery allowances, a surprisingly high reliability levels can be obtained, and, *ii.* energy savings can be significantly improved due to the less conservative and task-dependent recovery reservation. Eventually, these two leverage dimensions help us to formulate and tackle the general problem of *determining recovery allowance and frequency assignments to minimize energy consumption, while meeting the timing constraints and task-level reliability targets.*

The remaining of this section is organized as follows. After presenting the assumptions and basic definitions in Section 6.3.1, I compare existing recovery strategies and illustrate the principles, as well as the potential of, dynamic allocation of recoveries through concrete examples in Section 6.3.2. In Section 6.3.3, I discuss the main dimensions of the general problem, which are subsequently addressed in Section 6.3.4 (the feasibility problem) and Section 6.3.5 (the frequency and recovery allowance assignment problem). Section 6.3.5

presents two new proposed schemes as well as their dynamic extensions, and includes their detailed experimental comparison.

6.3.1 Assumptions and Definitions

This section assumes that the DVS-enabled processor has ℓ discrete speeds levels, $s_{min} = s_1 < s_2 \ldots < s_\ell = s_{max}$. Here, $s_{min} = s_1$ stands for the minimum available speed of the processor. Moreover, for simplicity, speed levels are normalized with respect to the maximum speed s_{max}, where $s_{max} = s_\ell = 1.0$. Note that, in modern processors, ℓ is typically a small number not exceeding 10.

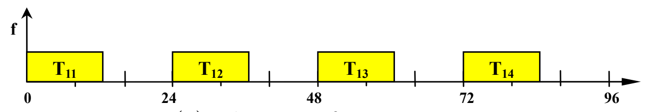
Assuming preemptive Earliest-Deadline-First (EDF) scheduling, the necessary and sufficient condition for feasibility under the maximum speed s_{max} is $U \leq 1.0$ [41]. The execution frequency (speed) of task T_i is denoted by f_i. Clearly, the execution frequency of task T_i, f_i, can assume only one of the discrete speed levels in $\{s_1, \ldots, s_\ell\}$. It is assumed that the task may take up to $\frac{c_i}{f_i}$ time units when executed at frequency f_i.

The *original* reliability of a single job of T_i, denoted by R_i^0, is the one that corresponds to the case where the job runs at the maximum processing frequency. That is, $R_i^0 = R_i(1.0)$.

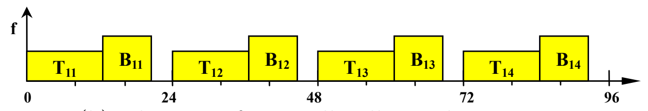
Definition 1. *The* task-level reliability *of task T_i, denoted by Φ_i, is the probability of completing all k_i instances of T_i successfully during a hyperperiod H.*

Definition 2. *The system's overall reliability, denoted by Φ_{sys}, is defined as completing all jobs successfully during a hyperperiod H, and is given by $\Phi_{sys} = \prod_{i=1}^{n} \Phi_i$.*

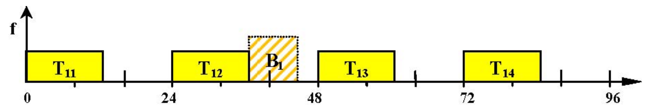
It can be seen that task-level and system-level reliabilities will, among other factors, highly depend on the running frequency assignment to tasks. The task-level and system-level *original* reliabilities are defined as the ones that result from running all jobs at the maximum speed during a hyperperiod [90], which are denoted by Φ_i^0 and Φ_{sys}^0, respectively. Finally, the *target* reliability of task T_i is denoted by Φ_i^t.



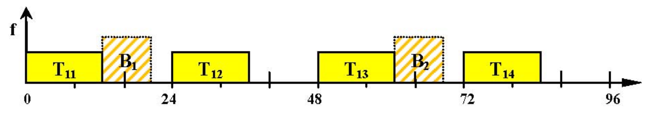
(a) The case of no recovery



(b) The case of statically allocated recoveries



(c) The case of dynamically allocated single recovery



(d) The case of dynamically allocated dual recoveries

Figure 6.7: Impact of recovery strategies for a task running at the scaled frequency $f = 0.6$ over $k_i = 4$ consecutive instances

6.3.2 Recovery Strategies for Periodic Execution Model

To recover from the soft errors triggered by the transient faults, we can exploit backward recovery technique and improve task/system's reliability. In this section, I compare the impact of different recovery allocation strategies on the reliability of periodic tasks. Consider the case where the task T_i runs at the frequency f_i during hyperperiod H. The exact expression of its task-level reliability Φ_i will depend on the *number* and *distribution* of recovery tasks.

- **Case 1: No recoveries.** In this case, no provisions are made to recover from potential transient faults that can affect individual jobs. As a result, the entire execution will be successful if and only if there are no errors induced by transient faults during the hyperperiod. Ona can obtain $\Phi_i = (R_i(f_i))^{k_i}$, and further, $\Phi_{sys} = \prod_{i=1}^{n}(R_i(f_i))^{k_i}$, which corresponds to the probability of completing all job instances without incurring any transient faults during the hyperperiod.

  This approach has the clear drawback of reducing the reliability by great margins. As a concrete example, consider the task T_1 with worst-case execution time $c_1 = 8ms$ and the period $p_1 = 24ms$. Assume the hyperperiod includes $k_i = 4$ job instances of T_1, namely, T_{11}, T_{12}, T_{13} and T_{14}. The transient fault model uses the parameters from [90] with $\lambda_0 = 10^{-6}$, $q = 3$ and $s_{min} = 0.1$. When all the jobs run at the maximum frequency (1.0), the overall probability of failure (*PoF*, defined as *1 - Reliability*), is evaluated as 8×10^{-9}. If one scales down all these jobs to a low frequency $f = 0.6$ as shown in Figure 6.7(a), the new *PoF* of the task is found as 1.15×10^{-5}. Observe how the lack of provision for recoveries results in a reliability degradation by more than four orders of magnitude even for a single task, during the hyperperiod.

- **Case 2: Statically allocated recoveries.** Now, consider the case where the recovery jobs are assigned *statically* to a subset of jobs of the periodic task, while the remaining jobs run without relying on a recovery.

  Specifically, if a recovery job is assigned statically to one of the instances of T_i, then the recovery is executed at the maximum frequency if a fault is detected at the completion time of that specific instance. As a result, the probability of successfully completing that single instance (in other words, its new reliability) is [87]:

  $$R''(f_i) = R_i(f_i) + R'_i(f_i)$$

  where $R'_i(f_i) = (1 - R_i(f_i))R_i(1.0)$. Above, the first component $R_i(f_i)$ corresponds to the probability of completing the job without any transient fault, while the second component $R'(f_i)$ indicates the probability of having a transient fault, which is later successfuly recovered from by re-executing at the maximum normalized frequency 1.0. Since $R_i(1.0) = R_i^0$ by definition, $R''_i(f_i)$ is known to be no less than the job's original reliability R_i^0 [87].

  For T_i's task-level reliability, if b_i instances have *statically* allocated recoveries and

$k_i - b_i$ instances run without any recovery provision, one gets:

$$\Phi_i = (R_i''(f_i))^{b_i} \times R_i(f_i)^{k_i - b_i}$$

A special case warrants further elaboration: All jobs of T_i are scaled and $b_i = k_i$. This corresponds to the traditional RA-PM solutions [90] and yields:

$$\Phi_i = \Phi_{RAPM,i} = (R_i''(f_i))^{k_i}$$

Obviously, Φ_i is maximized in this approach and the *scaled* task's reliability is guaranteed to be better than its original reliability Φ_i^0.

Returning to the running example, in order to maintain the system original reliability, the existing RA-PM scheme [90] will statically schedule a recovery job B_{1j} for each individual job instance T_{1j} during the hyperperiod as shown in Figure 6.7(b), and the new PoF is found as 9.19×10^{-13} which is now *better* than the task's original reliability by approximately four orders of magnitude.

However, allocating a separate recovery to every job instance of a scaled task requires significant amount of static slack and affects the energy savings opportunities of *other tasks* negatively. Hence, a significant number of tasks may remain *un-managed* (i.e., may have to run at the maximum frequency without any recovery) [90]. Also note that RA-PM is in general unable to target a specific reliability level which may be higher or lower than the task's original reliability.

- **Case 3: Dynamically allocated recoveries.** Another possibility, which is the focus of the research effort in this section, is to provide each task with a certain *recovery allowance* for execution.

  Specifically, provisions will be made through a static analysis to provide up to $a_i \leq k_i$ recoveries to task T_i *anywhere in the hyperperiod*, without associating a given recovery with a specific task instance in advance. The net result is that, at runtime, the task will be able to use these dynamically allocated recoveries for any a_i *arbitrary* instances, effectively covering $\sum_{j=1}^{a_i} \binom{k_i}{j}$ distinct fault scenarios for task T_i.

To illustrate these points, consider task T_i running with a single ($a_i = 1$) dynamic recovery over k_i instances during the hyperperiod H. The execution will be successful if:

– No task instance encounters a fault, or,

– The j^{th} instance encounters a fault, the single dynamic recovery is successfully executed and the job instances except T_{ij} complete without encountering a transient fault, for every $j = 1, \ldots, k_i$.

This effectively gives a reliability figure of

$$\Phi_i = R_i(f_i)^{k_i} + k_i R_i'(f_i) R_i(f_i)^{k_i - 1}$$

The reader should observe how even a single recovery allowance effectively provisions for $k_i + 1$ different execution scenarios. Moreover, due to typically low transient fault rates, these typically cover the scenarios with *maximum probability of occurrence.* In other words, the probability of having fault scenarios with increasing number of faults affecting multiple instances of the *same* task, while not exactly zero, will quickly drop to very small numbers.

Increasing the level of dynamic recovery allowance to $a_i = 2$ would provision for cases where faults affect any two arbitrary instances, covering an additional $\frac{k_i(k_i-1)}{2}$ fault scenarios. Therefore, one can be obtain:

$$\Phi_i = R_i(f_i)^{k_i} + k_i R_i'(f_i) R_i(f_i)^{k_i - 1}$$

$$+\frac{k_i(k_i-1)}{2}(R'_i(f_i))^2 R_i(f_i)^{k_i-2}$$

In general, for a task T_i running at speed f_i and with a_i recovery allowances for k_i job instances during H, the task-level reliability $\Phi_i = \Phi_i(f_i, a_i, k_i)$ is found as:

$$R_i(f_i)^{k_i} + \sum_{j=1}^{a_i} \binom{k_i}{j} (R'_i(f_i))^j R_i(f_i)^{k_i-j} \tag{6.6}$$

In this running example, if a single recovery allowance is assigned that can be used by any of the four scaled down job instances (Figure 6.7(c)) the new *PoF* is evaluated as 6×10^{-11}. In the figure, the dashed lines around the recovery task B_1 indicate that it is not statically associated by any specific job, but can be dynamically scheduled whenever a transient fault is detected in any job. Clearly, with the new *PoF* in this example, one can still preserve, and in fact improve by two orders of magnitude, the task's original reliability. Further, by dynamically scheduling $a_1 = 2$ recoveries for these job instances, the *PoF* reaches a level of 9.20×10^{-13}, which is extremely close to that obtained by the RA-PM technique [90]. This simple example illustrates that dynamic allocation of recoveries can also achieve very high reliability levels (comparable to RA-PM) by reserving smaller number of recovery slots.

I now elaborate further on how the reliability levels of a single periodic task T_i change with the recovery allowance a_i, frequency f_i, and the number of jobs k_i within the hyperperiod H.

Consider a periodic task T_i, with $c_i = 20ms$, $p_i = 100ms$ and $k_i = 100$ instances within a hyperperiod. I further assume that $\lambda_0 = 10^{-6}$ and the fault-sensitivity exponent q may vary from 3 to 5 [91]. Figures 6.8 and 6.9 show the relationship between the recovery allowance a_i and the achieved *PoF* is reported for the cases of $q = 5$ and $q = 3$, respectively. Here, *PoF* values normalized with respect to the one that corresponds to the task's original

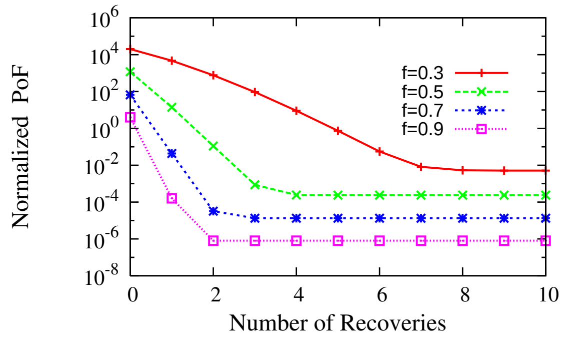


Figure 6.8: $q = 5$: Impact of number of recoveries on PoF for different frequencies

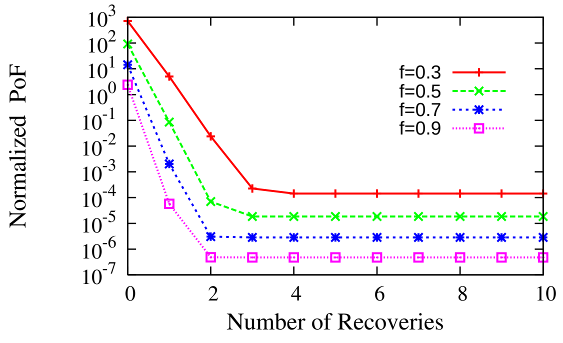


Figure 6.9: $q = 3$: Impact of number of recoveries on PoF for different frequencies

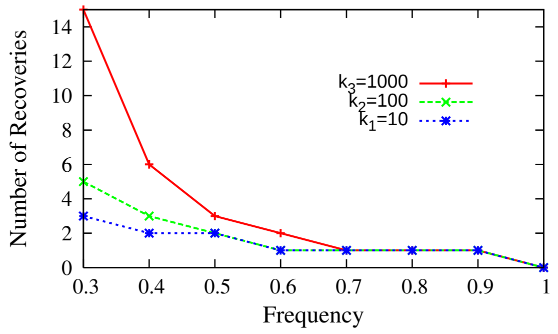


Figure 6.10: Impact of frequency on the number of recoveries needed to maintain Φ^0

reliability (i.e., $1 - \Phi_i^0$). As can be seen, for a given frequency level, increasing the number of recoveries significantly improves the reliability (reduces the PoF). However, after the recovery allowance reaches a certain number for a given frequency, the extra reliability gain becomes extremely small, in fact negligible – because the likelihood of new fault scenarios that can be covered by larger recovery allowances approaches quickly zero, thanks to the very nature of *dynamic allocation* strategy. Also, as expected, low frequency levels typically need more recoveries to achieve a given PoF. Reducing the fault-sensitivity exponent q from 5 (Figure 6.8) to 3 (Figure 6.9) makes the transient faults less likely, and the PoF values further drop with the same recovery allowance.

Figure 6.10 shows the number of recoveries needed to maintain the task-level original reliability as a function of the frequency f, for various k_i values (the number of jobs within the hyperperiod) and $q = 5$. In general, the recovery allowance needed to maintain Φ_i^0 drops with increasing frequency and decreasing k_i (a measure of the length of the hyperperiod). An interesting observation from Fig. 6.10 is that with the dynamic allocation of only a small number of recovery allowances, the task's original reliability can be maintained. In general, the same observation holds for arbitrary task-level reliability targets, underlining the potential of the new technique compared to the traditional RA-PM.

6.3.3 Dimensions of the Problem

The dynamically allocated recovery framework gives a powerful tool to enhance task-level reliabilities of periodic tasks with minimum recovery allowance. However, the ultimate design problem I aim to address is the following: **Given a set of periodic tasks with task-level reliability objectives $\{{\Phi_i}^t | \ i = 1, \ldots n\}$, how to choose frequency $\{f_i\}$ and recovery allowance $\{a_i\}$ assignments to minimize energy consumption?** Consideration of this problem mandates, in the first place, addressing the **feasibility** dimension to decide if all the timing constraints can be met with a given set of $\{f_i\}\{a_i\}$ assignments. This chapter will consider this problem in Section 6.3.4. Then this chapter will move on to the problem of **determining optimal $\{f_i\}\{a_i\}$ values** to minimize energy consumption while satisfying reliability and timing constraints at the same time (Section 6.3.5).

Notice that if the input to a specific problem instance is only the overall system target reliability Φ^t_{sys}, then the task-level reliability objectives $\{\Phi^t_i\}$ should be derived as the first step. In this case, a reasonable approach is to perform *uniform reliability scaling*, in such a way that the ratio $\frac{1-\Phi^t_i}{1-\Phi^0_i}$ is set to a common value Q across all tasks $T_1, \ldots, T_n$. Intuitively, this makes sure that the *PoF* improvement (or, degradation) uniformly applies to all the tasks, compared to the original *PoF* levels $\{1 - \Phi^0_i, \ i = 1, \ldots, n\}$.

6.3.4 Feasibility Conditions

Before addressing the problem of choosing optimal recovery allowance and frequency assignments, the feasibility problem should be considered. Specifically, if a certain set of frequency

$\{f_i\}$ and recovery $\{a_i\}$ assignments are considered for energy and reliability objectives, one must make sure that the job deadlines will be indeed met in *all* execution scenarios within the hyperperiod, where up to a_i instances of task T_i can incur transient faults and the corresponding dynamic recoveries are executed, for $i = 1, ..., n$.

Consider the execution of the task set Γ during the hyperperiod H. During actual execution, the task instance T_{ij} may complete successfully (i.e. without incurring a transient fault), or may encounter a transient fault. The occurrence/absence of transient faults in task instances may be shown by a binary string

$$X = x_{1,1}x_{1,2}\dots x_{1,k_1}x_{2,1}\dots x_{2,k_2}\dots x_{n,1}\dots x_{n,k_n}$$

where $x_{i,j} = 1$ if T_{ij} encounters a fault, and $x_{i,j} = 0$ otherwise. Note that each distinct binary string $\in \{0,1\}^{\sum k_i}$ corresponds to a distinct *execution scenario*, distinguished by a specific fault distribution over task instances during the hyperperiod.

Given a set of integers $z_1, \dots, z_n$, a *fault pattern* [4] corresponds to the set of execution scenarios where exactly z_i distinct instances of T_i encounter transient faults during the hyperperiod H. If $z_i \leq a_i$ ($\forall i\ 1 \leq i \leq n$), this pattern is called an *$\{a_i\}$-fault pattern.* For instance, assume $a_1 = 2, a_2 = 3$ and $a_3 = 1$ for three tasks T_1, T_2 and T_3. Any execution scenario where at most 2, 3 and 1 instances of T_1, T_2 and T_3 fail, respectively, constitutes a different fault pattern with respect to $\{a_1 = 2, a_2 = 3, a_3 = 1\}$ allowance. If there are 3 instances of T_1 within the hyperperiod one needs to consider all single and two-instance combinations for potential fault occurrences.

Obviously, the feasibility must be preserved for all distinct $\{a_i\}$-fault patterns for a given recovery allowance set $\{a_i\}$. At first, from a computational point of view, this appears as a prohibitively expensive task, because there are $\prod_{i=1}^{n} \sum_{j=1}^{a_i} \binom{k_i}{j}$ distinct $\{a_i\}$-fault patterns for a given recovery allowance set $\{a_i\}$, suggesting a potentially exponential number of cases to analyze.

Let $Z_{\{a_1,\dots,a_n\}}$ be the set of all $\{a_i\}$-fault patterns corresponding to a given recovery allowance assignment $\{a_i\}$. More formally, given a recovery allowance assignment $\{a_i\} = \{a_1, \ldots, a_n\}$, the fault pattern set $Z_{\{a_1,\ldots,a_n\}}$ corresponds to the set of execution scenarios where at most a_i instances of task T_i encounter faults $\ i = 1, \ldots, n$:

$$Z_{\{a_1,\ldots,a_n\}} = \{X \in \{0,1\}^{\sum k_i} | \sum_{j=1}^{k_i} x_{i,j} \leq a_i \ i = 1, \ldots, n\}$$

Also, given a recovery allowance assignment $\{a_i\} = \{a_1, \ldots, a_n\}$, consider the execution scenario where the first a_i instances of all tasks encounter faults, namely the execution scenario given by:

$$X_0 = \{1\}^{a_1}\{0\}^{k_1-a_1}\{1\}^{a_2}\{0\}^{k_2-a_2} \ldots \{1\}^{a_n}\{0\}^{k_n-a_n}$$

I will call X_0 the *deeply-red* execution scenario corresponding to the recovery allowance assigment $\{a_1, \ldots, a_n\}$, after the terminology used by Koren and Sasha in their seminal *periodic scheduling under skip model* paper [38].

Before presenting the main theorem, I introduce a couple of new variables and recall some main results from preemptive real-time scheduling theory:

- $\phi(t_1, t_2)$: the subset of task instances that are released at or after t_1 and having deadlines less than or equal to t_2. Formally: $\phi(t_1, t_2) = \{T_{ij} | t_1 \leq r_{ij} \leq d_{ij} \leq t_2\}$ where r_{ij} and d_{ij} are the release time and deadline of the task instance T_{ij}, respectively.

- $\eta_i(t_1, t_2)$: the number of task instances of task T_i whose release times and deadlines are fully contained in the interval $[t_1, t_2]$.

- $h^X(\phi(t_1, t_2))$: the total computational demand of task instances in $\phi(t_1, t_2)$ under the execution scenario X. Note that the execution requirement of $T_{ij} \ \in \phi(t_1, t_2)$ in X should be considered as $\frac{c_i}{f_i}$ if $x_{ij} = 0$ (the case of fault-free completion) and as $\frac{c_i}{f_i} + c_i$ if $x_{ij} = 1$ (the case of faulty execution followed by the recovery at the maximum speed).

Well-known results from the *processor-demand analysis* [4, 15, 38] imply the following:

Lemma 6. *All job instances meet their deadlines during the hyperperiod in the execution scenario X if and only if*

$$h^X(\phi(t_1,t_2)) \leq t_2 - t_1 \quad \forall t_1, t_2 \;\; 0 \leq t_1 \leq t_2 \leq H$$

Now, let us define:

$$C(\phi(t_1,t_2)) = \sum_{i=1}^{n} \eta_i(t_1,t_2) \cdot \frac{c_i}{f_i} \tag{6.7}$$

$C(\phi(t_1,t_2))$ represents the least upper bound on the total computational demand of the task instances in interval $[t_1,t_2]$ (excluding the potential recovery operations) and it has the same value for all execution scenarios.

Similarly, let $w^X(\phi(t_1,t_2))$ show the recovery overhead, induced by transient faults, for task instances in $\phi(t_1,t_2)$ under the execution scenario X.

Observe that:

$$h^X(\phi(t_1,t_2)) = C(\phi(t_1,t_2)) + w^X(\phi(t_1,t_2)) \tag{6.8}$$

Now, assume that in the execution scenario X, exactly $z_i \leq a_i$ instances of T_i encounter faults. Then:

$$w^X(\phi(t_1,t_2)) \leq \sum_{i=1}^{n} min(z_i, \eta_i(t_1,t_2)) \cdot c_i \quad z_i \leq a_i \, \forall i \tag{6.9}$$

This is based on the observation that in the execution scenario X with z_i faults for T_i, the total number of recoveries needed for T_i cannot exceed z_i or the total number of task instances of T_i in the given interval during the hyperperiod H.

The following theorem underlines that assessing feasibility under a single (and well-defined) worst-case execution scenario is necessary and sufficient.

Theorem 2. *For a fixed frequency and recovery allowance assignment $\{a_1, \ldots, a_n\}$, the* *periodic task set remains feasible in every fault pattern* $\in Z_{\{a_1,\ldots,a_n\}}$*, if and only if all the deadlines can be met when the first* a_i *instances of every task* T_i $(i = 1 \ldots n)$*, encounter transient faults during the same execution.*

Proof. Proving the *only-if* part is relatively easy: the execution scenario where the first a_i instances of each task T_i fails is a valid execution scenario in the set of $\{a_i\}$-fault patterns (in fact, corresponding to the deeply red execution scenario X_0), and all deadlines should be met under that execution scenario.

Now I focus on the *if* part. Given a recovery allowance assignment $\{a_i\} = \{a_1, \ldots, a_n\}$, consider an execution scenario $X \in Z_{\{a_1,\ldots,a_n\}}$. Also, let X_0 be the deeply red execution scenario corresponding to that assignment $\{a_i\} = \{a_1, \ldots, a_n\}$.

Lemma 6 implies that the proof will be complete if we can show that:

$$h^X(\phi(t_1, t_1 + D)) \leq h^{X_0}(\phi(0, D)) \tag{6.10}$$

for all execution scenarios $X \in Z_{\{a_1,\ldots,a_n\}}$ and $0 \leq t_1, D \leq H$. The strategy to demonstrate the inequality (6.10) will be to use the relationship given in (6.8). Specifically, I will prove that:

$$C(\phi(t_1, t_1 + D)) \leq C(\phi(0, D)) \tag{6.11}$$

and

$$w^X(\phi(t_1, t_1 + D)) \leq w^{X_0}(\phi(0, D)) \tag{6.12}$$

implying (6.10). To start with, observe that:

$$\eta_i(t_1, t_1 + D) \leq n_i(0, D) = \lfloor \frac{D}{p_i} \rfloor \tag{6.13}$$

Multiplying both sides by $\frac{c_i}{f_i}$ and getting the summation over all tasks, it can be obtained:

$$\sum_{i=1}^{n} \eta_i(t_1, t_1 + D) \cdot \frac{c_i}{f_i} \leq \sum_{i=1}^{n} n_i(0, D) \cdot \frac{c_i}{f_i} \tag{6.14}$$

proving (6.11) (after substitution in (6.7)).

Proving (6.12) is slightly more involved. First, note that:

$$w^{X_0}(\phi(0, D)) = \sum_{i=1}^{n} min(a_i, \eta_i(0, D)) \cdot c_i \tag{6.15}$$

Notice that since the number and distribution of faults affecting T_i's instances is precisely known in X_0, (6.15) is written as an equality.

Considering (6.9) and (6.15), the inequality (6.12) will be shown (completing the proof) if it can be proven that:

$$\begin{gathered} min(z_i, \eta_i(t_1, t_1 + D)) \leq min(a_i, \eta_i(0, D)) \\ z_i \leq a_i \;\; i = 1 \dots n \end{gathered}$$

Since $z_i \leq a_i$, it is sufficient to show:

$$min(a_i, \eta_i(t_1, t_1 + D)) \leq min(a_i, \eta_i(0, D)) \;\; i = 1 \dots n \tag{6.16}$$

I distinguish two cases:

Case 1. $\eta_i(0, D)) \geq a_i$. In this case, $min(a_i, \eta_i(0, D)) = a_i$ and (6.16) is established, since $min(a_i, \eta_i(t_1, t_1 + D)) \leq a_i$.

Case 2. $\eta_i(0, D)) < a_i$. In this case, $min(a_i, \eta_i(0, D)) = \eta_i(0, D)) < a_i$. From (6.13), $\eta_i(t_1, t_1 + D) \leq n_i(0, D)$. This gives: $\eta_i(t_1, t_1 + D) < a_i$ and $min(a_i, \eta_i(t_1, t_1 + D)) = \eta_i(t_1, t_1 + D)$. This in turn leads to:

$$min(a_i, \eta_i(t_1, t_1 + D)) = \eta_i(t_1, t_1 + D) \leq$$
$$min(a_i, \eta_i(0, D)) = \eta_i(0, D) \tag{6.17}$$

completing the proof.

□

Theorem 2 implies that for a given recovery allowance assignment $\{a_i\} = \{a_1, \ldots, a_n\}$, checking the feasibility under the deeply red execution scenario X_0 is necessary and sufficient. In turn, checking the feasibility under the deeply red execution scenario X_0 involves checking the condition

$$h^{X_0}(\phi(0, D)) \leq D \quad \forall D \ \ 0 \leq D \leq H \tag{6.18}$$

Notice that:

$$h^{X_0}(\phi(0, D)) = C(\phi(0, D)) + w^{X_0}(\phi(0, D))$$

Further:

$$C(\phi(0, D)) = \sum_{i=1}^{n} n_i(0, D) \cdot \frac{c_i}{f_i}$$

and,

$$w^{X_0}(\phi(0, D)) = \sum_{i=1}^{n} min(a_i, \eta_i(0, D)) \cdot c_i$$

$\eta_i(0, D) = \lfloor \frac{D}{p_i} \rfloor$ and moreover, it is easy to see that the value of $h^{X_0}(\phi(0, D))$ can change only at D values that correspond to period boundaries. So the condition (6.18) can be effectively checked only at the period boundaries during the hyperperiod. This leads to an overall feasibility test of complexity $O((\frac{H}{p_{min}}) \cdot n)$ where p_{min} is the smallest period among all $\{p_1, \ldots, p_n\}$.

It is worthwhile to note that this result is along the lines of existing real-time scheduling theory which suggests that, when the workload of instances of given periodic task may change (for example, by intentionally *skipping* certain instances), the worst-case occurs with the so-called *deeply red* pattern – a scenario where all tasks present their maximum demand as early as possible and delay the *skips* as much as possible [15, 38, 53]. Here essentially extends those results to fault-sensitive settings and has the same pseudo-polynomial time complexity.

6.3.5 Recovery Allowance and Frequency Assignments

Static Phase

In this section, I address the main design problem: to determine frequency and recovery allowance assignments to minimize the total energy, while meeting task-level reliability and time constraints. In general, the non-trivial interplay of *reliability, timing* and *energy* dimensions makes the problem rather challenging. A given task's energy consumption is, in general, likely to decrease with decreasing frequency; but this may also violate the task's reliability constraint. Moreover, reducing a task's frequency and/or increasing its recovery allowance may also affect slow-down opportunities for other tasks.

Note that task-level target reliability objectives impose hard constraints on the frequency and recovery allowance assignment. In particular, if task T_i runs at discrete speed level $f_i = s_j$ $(1 \leq j \leq \ell)$, then Equation (6.6) suggests *the existence of a minimum recovery allowance value* a_i, to guarantee the reliability objective Φ_i^t. Consequently, one can construct an $n \times \ell$ table (*Minimum Recovery Table (MRT)*), where $MRT_{i,j} = min\{a_i | \Phi_i(f_i = s_j, a_i, k_i) \geq \Phi_i^t\}$. Intuitively the entry (i, j) of the table gives the minimum number of dynamic recoveries that must be assigned to task T_i when it runs at speed s_j to achieve Φ_i^t. It is clear that if $f_i = s_j$, the number of recoveries should be exactly $MRT_{i,j}$: a smaller value would violate the reliability constraint and a larger one would unnecessarily narrow the feasibility space of other tasks (see Theorem 2).

Based on these observations, two schemes are proposed. The first, *Lock-Step Frequency Scaling Algorithm (LSF)*, initially sets all task frequencies to $f_i = s_{max} = s_\ell$ and $a_i = MRT_{i,\ell}$. Then it iteratively attempts to reduce the frequency level of each task by one level, as long as the reliability and feasibility constraints are satisfied. Specifically, in a given iteration of *LSF*, a specific quantity $\delta_{i,j}$ is evaluated for each task. When the current frequency of task T_i is reduced from $f_i = s_{j+1}$ to s_j, we have:

$$\delta_{i,j} = \frac{k_i(E_i(s_{j+1}) - E_i(s_j))}{R_i(s_{j+1})^{k_i} - R_i(s_j)^{k_i}}$$

above $k_i(E_i(s_{j+1}) - E_i(s_j))$ indicates the energy savings obtained by total k_i job instances of T_i within the hyperperiod, when the task f_i is scaled down from s_{j+1} to s_j. Similarly, $R_i(s_{j+1})^{k_i} - R_i(s_j)^{k_i}$ is the reliability degradation that accompanies the same tentative speed reduction. Informally, $\delta_{i,j}$ is a measure of *utility* (i.e. energy savings per unit reliability degradation) corresponding to one-level speed scaling, guiding the algorithm's operation.

In a given iteration of *LFS*, a task T_i is said to be *eligible* for frequency scaling from s_{j+1} to s_j, if the task set remains feasible with $f_i = s_j$ and $a_i = MRT_{i,j}$ assignments (assuming other tasks' assignments remain the same). Let G_h be the set of *eligible* tasks in iteration h. *LFS* selects the task $T_i \in G_h$ for scaling down by one level as the one with maximum $\delta_{i,j}$ value; and stops when either G_h is empty or the energy-efficient frequency values for all tasks have been reached. As a result, *LFS* has at most $(\ell - 1)n$ iterations and each iteration performs at most n feasibility checks. The complexity of *LFS* is therefore $O(\ell \cdot n^3(\frac{H}{P_{min}}))$.

I also experimented with a faster algorithm called the *Dual Speed (Dual)* Algorithm, that avoids checking the potential utility of scaling down every task, for every frequency level. Specifically, that algorithm first identifies the lowest *common* speed s_c for all tasks such that the feasibility is preserved with $\{f_i = s_c\}$ and $\{a_i = MRT_{i,c}\}$ assignments. These can be done in at most $log(\ell)$ steps by performing a binary search on the number of available ℓ discrete frequencies. Then it attempts to reduce the speed of some tasks to s_{c-1} while preserving feasibility, essentially using at most two distinct speed levels for the entire task set. The algorithm is based on the intuition that a uniform speed assignment is typically beneficial for real-time task sets, due to the convexity of dynamic power consumption. The dual-speed algorithm has the complexity $O((\log \ell + n) \cdot n(\frac{H}{P_{min}}))$.

Performance Evaluations: A discrete-event simulator was implemented in C programming language to evaluate the performance of the new suggested schemes. In the simulations, in addition to *LFS* and *Dual* schemes, I implemented two additional schemes:

- The periodic *RA-PM* scheme [90], whose objective is to preserve the system's original reliability. I implemented the *largest-utilization-first (LUF)* and *smallest-utilization-first (SUF)* variations to determine the managed tasks [90]. The results were very similar, and I include only the results for *LUF* below.
- The *Static Power Management (SPM)* scheme [5], which computes optimal slow-down factors to minimize energy without considering reliability objectives. It is included in the comparison to assess the potential energy cost of provisioning for reliability.

A cubic frequency-dependent power component P_d is assumed and it is set to unity at the maximum processor frequency. The frequency-independent power component P_{ind} for each task is normalized with respect to P_d and I set $P_{ind} = 0.05$ in the simulations. The ten discrete frequency levels that I assume are modeled after Intel Xscale processor [72]. Also, Poisson distribution is used to simulate transient faults with an average fault rate of $\lambda_0 = 10^{-6}$ at the maximum frequency, which is a realistic fault rate as reported in [93]. In the simulations, the fault rate exponent q is set to 3.

Each point in the presented figures is obtained by averaging the results obtained through the simulations and the confidence intervals at 97% confidence level are reported. The periods (p_i) of each task set are uniformly generated and fall in the range of $[10ms, 1080ms]$. The total utilization (U) of per task set is varied from 0.2 to 1.0 (full load) in steps of

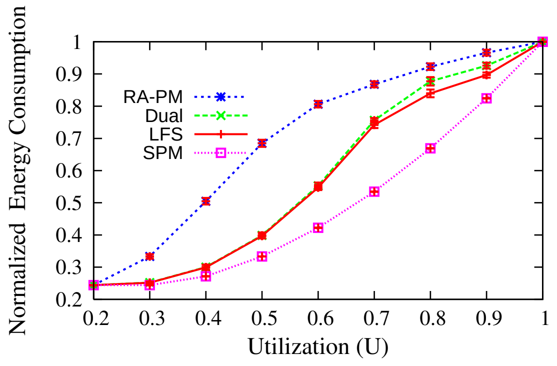


Figure 6.11: Impact of the utilization on the energy consumption with 10 tasks per set

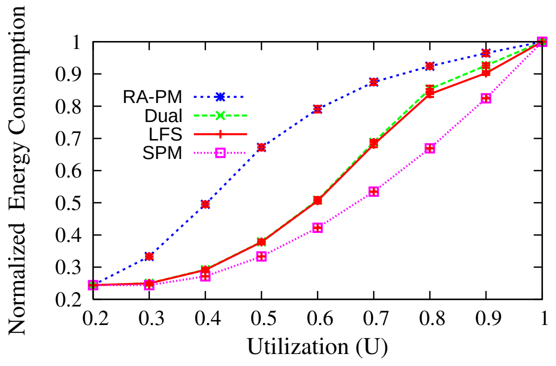


Figure 6.12: Impact of the utilization on the energy consumption with 20 tasks per set

0.1. The *UUniFast* algorithm [11] is used to generate the individual task utilizations (u_i). All energy consumption results are normalized with respect to the *no power management (NPM)* scheme that executes all tasks without any frequency and voltage scaling (i.e. at s_{max}).

First, I evaluate the impact of the utilization on the energy consumptions by setting $\Phi_i^t = \Phi_i^0$ (original reliability) for all tasks. Figure 6.11 shows the evaluation results with 10 tasks per task set. As expected, the energy consumption of all schemes increases with increasing U, due to the need for higher frequencies to preserve feasibility. The performance of *SPM* corresponds to maximum energy savings that can be obtained, *even when ignoring reliability objectives*. Compared to *RA-PM*, *Dual* and *LFS* have significant energy savings up to 50%. This is due to the dynamic recovery allocation strategy that can achieve reliability objectives with small number of recoveries, while leaving more slack for slowdown. In fact, when $U \leq 0.5$, the performance of the new schemes approaches to that of SPM, because the use of small number of recoveries can be compensated by small increases in frequency levels in that region. It is also worthwhile to note that *LFS*, despite its more complex search algorithm, has only marginal gains over *Dual* and only at high utilization values, when the objective is to maintain original reliabilities. Figure 6.12 presents the same analysis this time for 20-task sets and it is observed that the trends remain very similar. It is interesting to note that *Dual* approaches further *LFS* with increasing number of tasks as the algorithm has more chances to identify tasks for scaling down to s_{c-1} without violating feasibility.

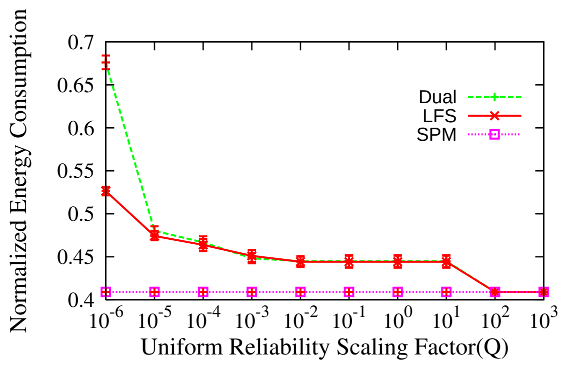


Figure 6.13: Impact of uniform reliability scaling factor on system energy consumption

Figure 6.13 shows the impact of the task-level target reliabilities (Φ_i^t) on the energy consumptions when $U = 0.5$ for 20-task sets. Here, the x axis is the *uniform reliability scaling factor* Q, defined in Section 6.3.3. Specifically, a value of $Q = 1$ corresponds to targeting original task-level reliabilities; a smaller (larger) scaling factor Q corresponds to targeting smaller (larger) *PoF* figures. The *SPM* algorithm is reliability-ignorant, but its energy consumption figure is included as a comparison. Several observations are in order: when $Q \geq 10^2$, the new suggested algorithms' performance converges to that of *SPM* because they also choose to execute tasks without (or with very small number of) recoveries since the target reliability requirements are loose. However, with decreasing Q, the algorithms are forced to schedule additional recoveries and this causes some modest increase in energy consumption. It is interesting to note that in the region $[10^{-4}, 10^{-1}]$ the energy consumption of the schemes increases only marginally. This is due to the fact that the recoveries can be assigned only in integer units; and once a new recovery is added to the application, typically overall reliability improves by great margins. This is further enhanced by the existence of discrete speed levels; making *very fine-grained* reliability control somewhat difficult. However, for very high reliability requirements (e.g. $Q \leq 10^{-5}$) the algorithms are forced to provision for additional recoveries with a corresponding energy cost. In fact, one can notice that the comprehensive search mechanism of *LFS* starts to pay off when the required *PoF* levels are very low.

Dynamic extensions

The static algorithms *LFS* and *Dual* are designed to meet the problem's timing and reliability constraints under worst-case workload assumptions. However, it is well-known that, often the tasks' actual workloads during execution deviate from the worst-case, and significant amount of dynamic slack can be expected. In fact, exploiting the dynamic slack by reducing the processor speed as appropriate is a common strategy in DVS frameworks [7,48].

While similar opportunities exist in this settings, the dynamic reclaiming becomes more challenging due to hard reliability constraints. Specifically, reclaiming slack at run-time by dynamically reducing the speed, even if it guarantees the timing constraints, *can violate the task's given target reliability objectives.*

Therefore, I present a conservative but safe technique to perform dynamic slack reclaiming at runtime while still preserving the task-level reliability guarantees. Specifically, the DRA algorithm [7] is used to keep track of the dynamic slack, and evaluate the *earliness* of each task at dispatch time. However, a given job's speed is actually reduced only when the amount of safely reclaimable slack (earliness) is large enough to schedule also a new dynamically scheduled recovery – this limits the extent of achievable dynamic slow-down, but preserves the reliability constraint. Yet, when a job completes successfully (i.e. without a fault), the algorithm can re-use the slack of the new recovery to safely slow-down another job. Based on these principles, I extended *LFS* and *Dual Speed* schemes to dynamic settings, obtaining *D-LFS* and *D-Dual* algorithms, respectively.

Performance Evaluation: I implemented *D-LFS* and *D-Dual* algorithms in the simulator. I also implemented the *Dynamic RA-PM (D-RA-PM)* scheme, which uses the *wrapper task technique* [90], to perform slack reclaiming at runtime. Finally, I implemented a *Reliability-Ignorant-Reclaiming scheme (RIR)* which uses the entire dynamic slack for slow-down at run-time, without considering the potential reliability degradation. *RIR* is essentially used as a yardstick algorithm to assess the energy savings of the new solutions.

The same evaluation settings are used as in the evaluation part of static algorithms.

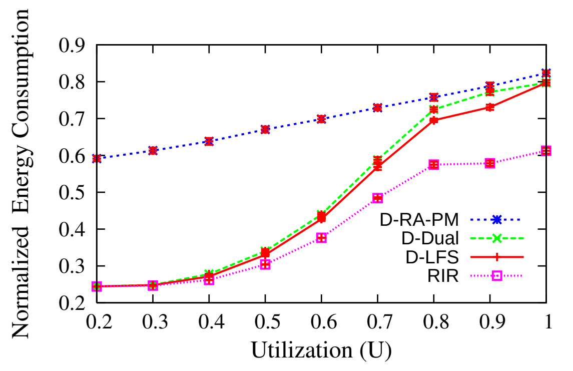


Figure 6.14: Impact of utilization *PoF* on system energy consumption

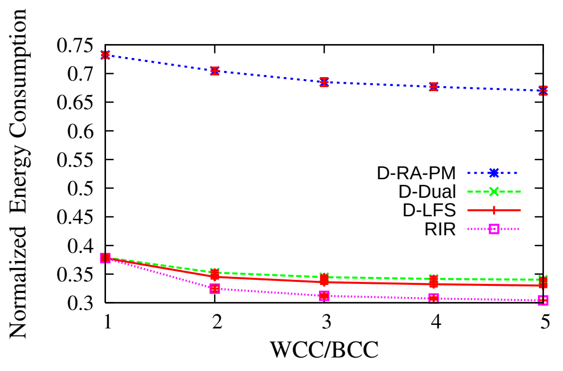


Figure 6.15: Impact of actual workloads on system energy consumption

To model the variations in the actual workload, $\frac{WCC}{BCC}$ (varying from 1 to 5) is used, which represents the ratio of worst-case execution time to the best- case execution time. The higher this ratio, the more the actual workload deviates from the worst case. I randomly generate the actual execution time of each job, by using normal distribution with the mean $\frac{WCC+BCC}{2}$ and standard derivation $\frac{WCC-BCC}{12}$ [7]. This guarantees that 99.7% of generated execution times fall in the range $[BCC, WCC]$ and values outside this range are not considered. For each points in the figures 1000 task sets (each with 20 tasks) are used; each execution is repeated 1000 times during the hyperperiod. All energy consumption results are again normalized with respect to NPM.

Figure 6.14 illustrates the energy consumptions as a function of total utilization (U), with $\frac{WCC}{BCC} = 5$ and $\Phi_i^t = \Phi_i^0$ for all tasks. The relative order of schemes remains the same with respect to Figure 6.11. However, now *D-RA-PM*'s performance relatively deteriorates compared to the static version. This is because *D-RA-PM* is based on the *wrapper-task* technique which uses the maximum frequency as the nominal speeds of all jobs to preserve reliability when dynamically reducing the speed. However, the new suggested schemes use nominal frequency assignments from the static algorithms, which are relatively lower. Also it is noticed that new suggested dynamic schemes no longer converge to *RIR* at high U values due to the need for reserving additional recovery to maintain reliability guarantees – *RIR*, by ignoring reliability, can reclaim full dynamic slack. Another observation is that

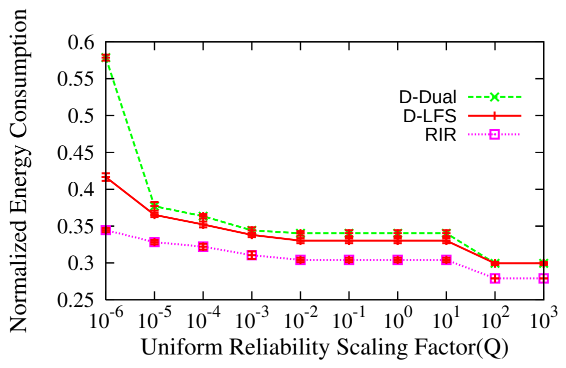


Figure 6.16: Impact of uniform reliability scaling factor on system energy consumption

with increasing utilizations, the difference of energy performances between *D-LFS* and *D-Dual* becomes more emphasized since the latter is restricted to the use of two speed levels during dynamic reclaiming.

Figure 6.15 shows the impact of variability in the actual workload (i.e. the $\frac{WCC}{BCC}$) on the energy performance, when $U = 0.6$ and $\Phi_i^t = \Phi_i^0$. In general, it is found that the energy consumption decreases with increasing dynamic slack (higher $\frac{WCC}{BCC}$ ratio). The *D-Dual*'s limitations due to the use of two speeds restrict its ability to exploit fully the increasing dynamic slack. Finally, the hard requirement for maintaining the reliability objectives results in reduced ability for reclaiming slack, compared to *RIR*.

In Figure 6.16, it shows the relationship between the task-level target reliabilities (Φ_i^t) and the energy consumptions, for $U = 0.5$ and $\frac{WCC}{BCC} = 5$. Again, patterns are similar to those observed in the case of static schemes (Figure 6.13). A notable difference is that the shortcomings of *D-Dual* are slightly more emphasized in dynamic settings.

6.4 Chapter Summary

This chapter first addressed the *energy-optimal reliability configuration (EORC)* problem for real-time embedded applications with the goal of minimizing system energy consumption while satisfying an arbitrary target reliability and deadline constraints. To this aim, this chapter developed the *Generalized Shared Recovery (GSHR)* technique through which a small number of recovery tasks are shared by all the tasks. The experimental evaluation indicates that for a broad range of reliability objective goals, the new proposed solution IRCS delivers close-to-optimal energy savings.

Further, in this chapter, a framework is presented to provide arbitrary task-level reliability guarantees to periodic real-time tasks, while controlling energy consumption through DVS. As opposed to existing reliability-aware power management (RA-PM) frameworks, where the aim is to preserve the original reliability and the recovery jobs are statically allocated to scaled jobs, this chapter introduced the concept of *recovery allowances* that can be reclaimed anywhere during the hyperperiod. This flexibility helps to improve reliability significantly, with minimum in-advance reservation for potential recoveries. This chapter presented static and dynamic algorithms that are shown to reduce energy consumption while maintaining the feasibility and reliability objectives.

Chapter 7: Minimizing Expected Energy Consumption through Optimal Integration of DVS and DPM

7.1 Introduction

In addition to DVS, Dynamic Power Management (DPM) is another well-known technique, particularly effective for off-chip system components (e.g. main memory and I/O devices). Minimizing the total system energy on systems where both DVS and DPM are available is a challenging problem. An exact and formal characterization of the interplay between DVS and DPM for a real-time application using multiple devices was recently obtained in [21]. In the same work, an algorithm to compute the optimal processing frequency was also derived. However, the solution in [21] is derived assuming a known, worst-case workload. In other words, the solution is optimal only for *deterministic* workloads. As it is shown later in this Section, if the workload exhibits probabilistic behavior, minimizing *expected* energy is more important and the algorithm in [21] (called *DET* throughout this work) becomes sub-optimal, even overly pessimistic. Although the task's actual workload cannot be predicted in advance with full accuracy, its probability distribution function can be obtained or estimated through *profiling* [44, 72, 75, 78]. This primary goal in this chapter is to determine the optimal processing frequency to minimize the expected energy at the *system-level*, when the cumulative probability distribution of the application's workload is known (Section 3.1). I consider both DVS and DPM features and formally characterize the expected energy. The new technique is illustrated by first focusing on an application using a single-device and show how the optimal frequency can be derived in linear-time. Then, we extend the solution to the case of multiple devices. The new solutions also ensure that the timing constraint of the application is met in every execution scenario. The findings from this research are published in [79].

7.2 System Models

7.2.1 Application Model

A periodic real-time application is considered, that must complete its execution within the relative deadline d, which is equal to its period (P). The application's workload, characterized by the *number of cycles*, is assumed to be known only probabilistically: it can assume values between a lower bound and an upper bound that are given by *best-case execution cycles (BCC)* and *worst-case execution cycles (WCC)*, respectively. The cumulative distribution function for the application's workload is:

$$F(x) = p(X \leq x) \tag{7.1}$$

where X is the random variable for the application's CPU cycle demand and $p(X \leq x)$ represents the probability that the application will not require more than x cycles within a given frame. To approximate the cumulative distribution function $F(x)$, the effective *histogram* technique [72, 75] is used. Specifically, it is assumed that the application's workload function is partitioned into n *cycle groups*: $(b_0, b_1], (b_1, b_2], ...,$ $(b_{n-1}, b_n]$, where $b_0 = BCC$, $b_n = WCC$ and $b_i < b_{i+1}$. As shown in Figure 7.1, the probability of executing the application with no more than x cycles is estimated conservatively as $F(b_{i+1})$, when x is in the interval $(b_i, b_{i+1}]$. Notice that $F(b_n) = 1$. Through profiling, the probability distributions for the cycle groups can be obtained [72,75]. Also, the *size* of the cycle group i is defined as $gc_i = b_i - b_{i-1}$ for $i > 0$ and $gc_i = b_i$ for $i = 0$.

7.2.2 Device Model

In this section, it is assumed that the real-time embedded application uses a set of m devices $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, .., \mathcal{D}_m\}$ during its execution. Each device $\mathcal{D}_i$ can be either in *active* or *sleep (low-power)* state, and is defined with the following parameters:

- P_a^i: The device power consumption in *active* state

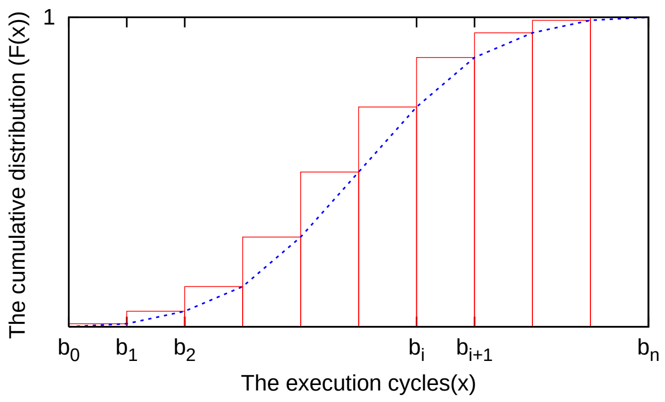


Figure 7.1: Histogram-based estimation of the application's probabilistic workload

- P_s^i: The device power consumption in *sleep* state
- E_{as}^i (E_{sa}^i): The energy overhead associated with active-to-sleep (sleep-to-active) state transition
- T_{as}^i (T_{sa}^i): The time overhead associated with active-to-sleep (sleep-to-active) state transition

Due to the periodic execution pattern, each active-to-sleep transition for a device will be eventually followed by a sleep-to-active transition. As a result, to conveniently represent the overheads, $E_{tr}^i = E_{as}^i + E_{sa}^i$ is defined as the total device transition energy overhead and $T_{tr}^i = T_{as}^i + T_{sa}^i$ as the total transition delay. Following recent literature [18, 21, 22, 64–66], this chapter assumes *inter-task device scheduling*. In inter-task device scheduling, all devices used by the real-time application must be in *active* state while it executes. In fact, the absence of knowledge about the exact time instants when the application will need a specific device and non-trivial state transition costs easily justifies the inter-task device scheduling paradigm [18, 21, 22, 65].

However, the devices can be put to *sleep* state when the application finishes its execution within a frame (until the beginning of the next frame). Considering the non-trivial energy and time overheads associated with state transitions, the device *break-even time* B_i is defined to represent the lower bound on the idle interval length so that putting the device to *sleep* state can be justified at run-time. In addition, no idle interval can be smaller than T^i_{tr}; hence, B_i is given as [18, 19, 21, 22]:

$$B_i = \max(T^i_{tr}, \frac{E^i_{tr} - T^i_{tr} P^i_s}{P^i_a - P^i_s})$$

7.2.3 Energy Model

When the DPM technique is employed for energy management, one must consider the additional energy overhead for device transitions. Following previous DPM work [18, 19, 21, 22], this chapter assumes that shutting down the entire system within a frame is not an option (hence, static power is not manageable) and concentrate on the dynamic energy consumption E_d. E_d is a function of several factors, including the CPU frequency f and power characteristics and states of individual devices (such as main memory and I/O modules). For simplicity, this chapter assumes that all device active powers are given in excess of the device sleep powers, as in [19, 21, 22].

If the application executes c cycles at the frequency f within a given frame, it will complete its execution within $t = \frac{c}{f}$ time units. Within that frame, the interval $[0, \frac{c}{f}]$ is called the *execution period*. Similarly, the interval $[\frac{c}{f}, d]$ is called the *slack period*. During the slack period, the CPU is idle and the devices used by the application can be potentially put to *sleep* states by incurring transition energy overheads. However, such a transition for a device $\mathcal{D}_i$ is energy-efficient if and only if $d - B_i > \frac{c}{f}$. For example, in Figure 7.2, the device $\mathcal{D}_1$ can be put to sleep state during the slack period, while $\mathcal{D}_2$ is forced to remain in active mode.

Let $\mathcal{D}_{\mathcal{A}}$ denote the subset of devices that are forced to remain in *active* state during the slack period, due to short idle interval lengths. On the other hand, the devices in $\mathcal{D} - \mathcal{D}_{\mathcal{A}}$ can be transitioned to *sleep* state during the slack period to save energy. The dynamic energy consumption $E_d(f)$ within a frame can be formally expressed as [21]:

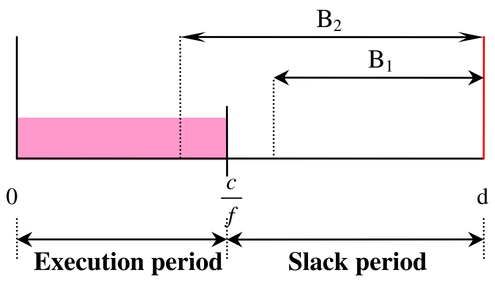


Figure 7.2: Execution and slack periods of an application

$$E_d(f) = E_e(f) + E_s(f) + E_t(f) \tag{7.2}$$

where

$$E_e(f) = (af^3 + \sum_{i=1}^{m} P_a^i)\frac{c}{f} \tag{7.3}$$

is the sum of the energy consumed by the CPU ($af^3 \cdot \frac{c}{f}$) and the devices ($\sum_{i=1}^{m} P_a^i \frac{c}{f}$) during the execution period;

$$E_s(f) = \sum_{i|\mathcal{D}_i \in \mathcal{D}_{\mathcal{A}}} P_a^i(d - \frac{c}{f}) \tag{7.4}$$

is the energy consumed by the devices that remain in active state during the slack period, and,

$$E_t(f) = \sum_{i|\mathcal{D}_i \in (\mathcal{D} - \mathcal{D}_{\mathcal{A}})} E_{tr}^i \tag{7.5}$$

is the transition energy incurred by the devices in $\mathcal{D} - \mathcal{D}_{\mathcal{A}}$, which need to be activated at the beginning of the next frame.

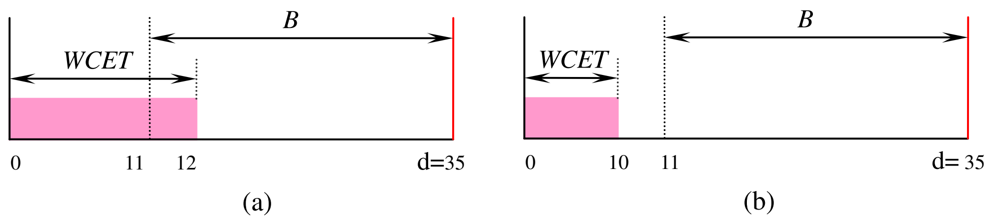


Figure 7.3: An application with $WCET = 12$ (a) and with $WCET = 10$ (b), using the IBM Microdrive device

7.3 System-level Energy: The Stochastic Case

7.3.1 Motivational Example

By using a system-level energy model very similar to that given in Section 3.2 of Chapter 3, the work in [21] derived a precise characterization of the interplay between DVS and DPM, and then showed how to obtain the optimal frequency to minimize system-level energy. A fundamental characteristic of that solution is that it assumes a deterministic workload equal to WCC. While provisioning for worst-case scenarios is important to guarantee the timing constraints, in practice, many real-time embedded applications complete early without consuming their worst-case workloads. Further, although the actual number of execution cycles cannot be known in advance exactly, its probability distribution can be obtained through profiling [72,75]. This, in turn, provides new and significant opportunities to minimize the *expected* system-level energy consumption, while meeting the application's deadline, as the following example illustrates.

Let us consider an application with deadline $d = 35ms$ running on a system with maximum frequency $1GHz$, using a single IBM Microdrive device ($P_a = 1.3$ *Watt*, $E_{tr} = 12$ *Joules* and $B = 24ms$, as specified in [18,21]). To start with, the analysis in [21] considers only the WCET information as the basis; hence, if $d - B < WCET$, the *DET* algorithm [21] would assume that the device will need to be kept in *active* mode during the slack period. For example, if $WCC = 12 \times 10^6$, *DET* would choose the optimal frequency as $f = 0.34Ghz$ (effectively planning to complete the application just at the deadline in the worst-case) since $WCET = 12ms > d - B = 11ms$ – as seen in Figure 7.3.a. Now assume that the application's workload is known probabilistically, and that it changes between $BCC = 2\times10^6$ and $WCC = 12\times10^6$ according to the normal distribution with the mean $\frac{BCC+WCC}{2}$ and the standard derivation $\frac{WCC-BCC}{12}$. While the *DET* solution indeed minimizes energy for the worst-case workload, one can compute that the *expected* energy consumption with $f = 0.34Ghz$ is $E_{DET} = 47.71$ *Joules*. On the other hand, by executing the application at $f = 0.76GHz$, one can obtain an expected energy consumption of $E = 29.70$ *Joules*, a savings of 35% over that of *DET*, while still meeting the deadline. In fact, improvements due to a probabilistic analysis do also exist when $WCET \leq d - B$. For instance, if WCC is changed as 10×10^6, as showed in Figure 7.3.b, then it is found out that DET will yield the frequency $f = 0.91Ghz$ with the corresponding expected energy $E_{DET} = 27.79$ *Joules*. But if $f = 0.69Ghz$ is used, the expected energy becomes $E = 26.45$ *Joules*, leading to approximately 5% additional energy savings. As this example suggests, minimizing the expected energy consumption while exploiting the subtle interaction between DVS and DPM warrants a full analysis.

7.3.2 Deriving Expected Energy Function

Given a processing frequency f and a cumulative distribution function for the workload, the expected energy consumption E_m within a frame can be written as the sum of three components: the expected energy consumption in the execution period $\overline{E}_e(f)$, that in the slack period $\overline{E}_s(f)$, and the expected transition energy overhead $\overline{E}_t(f)$:

$$E_m(f) = \overline{E}_e(f) + \overline{E}_s(f) + \overline{E}_t(f) \tag{7.6}$$

Let us elaborate on each of these components separately. **The execution period energy** $\overline{E}_e(f)$: During its execution, the application will execute a given cycle x in the range $[BCC, WCC]$ with a certain probability. In fact, this probability is equal to $(1 - F(x))$, where $F(x)$ is the cumulative distribution function defined in Equation (1) [72, 75]. Consequently, the expected value of overall (CPU + device) energy consumption during the execution period can be written as:

$$\begin{aligned}\overline{E}_e(f) = &\ \textstyle\sum_{x=1}^{WCC}((1-F(x))af^3\frac{1}{f}+\sum_{i=1}^{m}P_a^i(1-F(x))\frac{1}{f}) \\ = &\ \textstyle\sum_{x=1}^{WCC}(af^3+\sum_{i=1}^{m}P_a^i)(1-F(x))\frac{1}{f} \qquad (7.7)\end{aligned}$$

Further, by applying the histogram-based estimation technique, $\overline{E}_e(f)$ can be formally re-written as:

$$\overline{E}_e(f)=\sum_{j=0}^{n-1}(af^3+\sum_{i=1}^{m}P_a^i)\frac{gc_j}{f}(1-F(b_j)) \qquad (7.8)$$

The slack period energy $\overline{E}_s(f)$: The energy consumption during the slack period is due to devices that are forced to remain in *active* state when their completion time within the frame does not allow an energy-efficient transition. Specifically, when the application completes at time $t > d - B_i$ for a given device $\mathcal{D}_i$, that device will consume a total energy of $P_a^i(d-t)$ during the slack period by staying in *active* mode. Completion at each of such time instants occurs with a certain probability; hence, one can obtain:

$$\overline{E}_s(f)=\sum_{i=1}^{m}\sum_{Z=d-B_i}^{d}p(t=Z)P_a^i(d-t) \qquad (7.9)$$

where $p(t=Z)$ is the probability that the application will complete exactly at time Z.

The transition energy $\overline{E}_t(f)$: When the application completes at time $t < d-B_i$ the device $\mathcal{D}_i$ can and should be transitioned to *sleep* state during the slack period. However, each such transition will result in an energy overhead of E_{tr}^i. If t is the completion time, the expected value of E_{tr}^i is $p(t \le d - B_i)\ E_{tr}^i$. Observe that:

$$p(t \le d-B_i)=p(\frac{X}{f}\le d-B_i)=p(X\le f(d-B_i))$$

where X is the random variable for the cycle demand of the application and f is the processing frequency. Recalling that $p(X \leq x) = F(x)$, it can obtain:

$$\overline{E}_t(f) = \sum_{i=1}^{m} F((d - B_i)f) \cdot E_{tr}^i \tag{7.10}$$

At this point, I are ready to develop the new solution for the problem of minimizing the expected overall energy $E_m(f)$ while considering the probabilistic distribution of execution cycles while satisfying the deadline constraint.

7.4 Single-Device Model

In this section, I consider the case where the application uses only $m = 1$ device during its execution and derive the optimal frequency to minimize the expected energy by exploiting the interaction between DPM and DVS. This also allows us to lay the technical background for the more general case that will be addressed in Section 7.5. For simplicity, I use the notations $P_a^1 = P_a$, $B_1 = B$ and $E_{tr}^1 = E_{tr}$ throughout the section. Using the findings from Section 7.3.2, the single-device problem can be formally expressed as to *minimize*:

$$\begin{aligned} E(f) = \ & \textstyle\sum_{j=0}^{n-1} (af^3 + P_a) \frac{gc_j}{f} (1 - F(b_j)) \\ & +F((d - B)f)E_{tr} \\ & +\textstyle\sum_{Z=d-B}^{d} P(x = Z) P_a(d - x) \end{aligned} \tag{7.11}$$

Subject to:

$$\frac{WCC}{f} \leq d \tag{7.12}$$

$$f_{min} \leq f \leq f_{max} \tag{7.13}$$

Above, (7.12) encodes the deadline constraint while (7.13) indicates the feasible frequency ranges supported by the system. A non-trivial difficulty with the above optimization problem is that the unknown f appears as a parameter in the cumulative distribution function which may be of arbitrary form, making a closed form solution unlikely. However, notice that $F((d-B)f)$ can have only one of the $n+1$ distinct values that correspond to $F(b_i)$ $(0 \leq i \leq n)$ in the optimal solution. This property suggests an iterative approach: the original problem will be divided into to $n+1$ sub-problems by letting $F((d-B)f) = F(b_0), ..., F(b_n)$ successively. Each sub-problem will be attacked by assuming that $F((d-B)f) = F(b_i)$ and the expected energy consumption corresponding to that sub-problem will be recorded. Finally, the frequency that leads to the minimal energy consumption in any of the sub-problems will be selected as the global optimal.

In the following, I focus on the solution of these subproblems. Notice that when $F((d-B)f) = F(b_i)$ $(0 \leq i \leq n)$, the expression (7.10) can be readily re-written as a function of $F(b_i)$. Further, the properties of the histogram-based approach and simple algebraic manipulation show that, when $F((d-B)f) = F(b_i)$, (7.9) is equivalent to:

$$\sum_{j=i}^{n-1}(F(b_{j+1}) - F(b_j))P_a(d - \frac{b_{j+1}}{f})$$

Hence, the i^{th} sub-problem can be formally defined as to *minimize*:

$$\begin{aligned} E^i(f) &= \textstyle\sum_{j=0}^{n-1}(af^3 + P_a)\frac{gc_j}{f}(1 - F(b_j)) + F(b_i)E_{tr} \\ &\quad + \textstyle\sum_{j=i}^{n-1}(F(b_{j+1}) - F(b_j))P_a(d - \frac{b_{j+1}}{f}) \end{aligned} \tag{7.14}$$

Subject to:

$$\frac{WCC}{f} \leq d \tag{7.15}$$

$$\frac{b_{i-1}}{f} < d - B \leq \frac{b_i}{f} \tag{7.16}$$

$$f_{min} \leq f \leq f_{max} \tag{7.17}$$

where[1] the new additional constraint (7.16) is the sufficient and necessary condition to enforce $F((d{-}B)f) = F(b_i)$. Let $f_{low,i} = max(f_{min}, \frac{WCC}{d}, \frac{b_{i-1}}{d-B}{+}\epsilon)$ and $f_{up,i} = min(f_{max}, \frac{b_i}{d-B})$. Notice that $f_{low,i}$ corresponds to the lower bound[2] on the feasible frequency range for the problem while $f_{up,i}$ is the upper bound to the frequency range. It is obvious that if $f_{up,i} < f_{low,i}$, the feasible region for this subproblem is empty. On the other hand, the case of $f_{up,i} \geq f_{low,i}$ can be solved precisely. Observe that $E^i(f)$ is a strictly convex function. Therefore, the frequency f_i that minimizes $E^i(f)$ without considering constraints (7.15), (7.16) and (7.17) can be found by setting its derivative to zero:

$$f_i = (\frac{P_a(\sum_{j=0}^{n-1} gc_j(1 - F(b_j)) - \sum_{j=i}^{n-1}(F(b_{j+1}) - F(b_j))b_{j+1})}{2a \sum_{j=0}^{n-1} gc_j(1 - F(b_j))})^{\frac{1}{3}} \tag{7.18}$$

The convex nature of $E^i(f)$ justifies the following two basic properties for any $\Delta > 0$:

PROPERTY 1. $\forall f, f > f_i, E^i(f_i) \leq E^i(f) \leq E^i(f + \Delta)$

PROPERTY 2. $\forall f, f < f_i, E^i(f_i) \leq E^i(f) \leq E^i(f - \Delta)$

Based on these properties, one can obtain the following:

Theorem 1. *The optimal frequency for the i^{th} subproblem is equal to $f_i^* = max\{f_{low,i}, min\{f_{up,i}, f_i\}\}$, whenever $f_{up,i} \geq f_{low,i}$.*

Once evaluating the optimal solutions to all $n + 1$ subproblems (each requiring $O(n)$

[1]For mathematical convenience, I define $\sum_{j=n}^{n-1} x = 0$ for any value of x and assume $\frac{b_{i-1}}{f} < d - B$ automatically holds when $i = 0$.

[2]In the definition of $f_{low,i}$, ϵ is a small number arbitrarily close to 0, which is introduced to enforce the constraint $\frac{b_{i-1}}{f} < d - B_i$ with the strict inequality requirement.

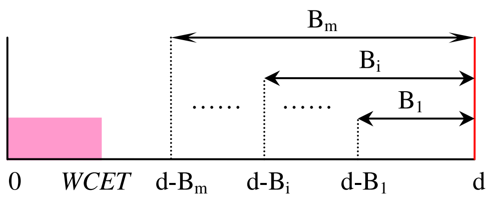


Figure 7.4: An application using m devices and break-even times

time), one can easily find the global optimal (in time $O(n^2)$). However, the procedure can be further speeded up by observing that once f_i is computed, f_{i+1} can be easily evaluated, since $f_{i+1} = f_i - \frac{(F(b_{i+2})-F(b_{i+1}))b_{i+2}}{2a\sum_{j=0}^{n-1} gc_j(1-F(b_j))}$. The denominator in this expression is indeed a constant (independent of i) and can be computed just once when evaluating f_0. Thus, one can easily design an algorithm that computes the f_i values *bottom-up*, by keeping track of the best candidate seen so far across the $n+1$ iterations. This optimization will help reduce the complexity to just $O(n \log n)$, where the dominant term will be due to the process of sorting the b_i and $F(b_i)$ values in increasing order. If they are already sorted in the input, the overall complexity is just $O(n)$.

7.5 Multiple-Device Model

In this section, the general problem is addressed, where the application uses m devices $\mathcal{D}_1, \ldots, \mathcal{D}_m$ (with corresponding break-even times $B_1, \ldots, B_m$). This chapter assumes the device indices reflect the ordering of the break-even times, as shown in Figure 7.4. With multiple devices, the problem gains a new dimension. By adjusting the processing frequency f, the application's execution time within a frame can be controlled through DVS; but this has a direct impact on the applicability of DPM. Specifically, when the application completes at time t such that $(d - B_i) < t < (d - B_{i-1})$, the devices $\mathcal{D}_1, \ldots, \mathcal{D}_{i-1}$ can be transitioned to the *sleep* state during the slack period. However, the devices $\mathcal{D}_i, \ldots, \mathcal{D}_m$ should remain in *active* state throughout the frame. Obviously, the probabilistic behavior of the workload adds another non-trivial complexity layer to the problem.

The problem can be formally stated as to minimize the expected energy consumption

$E_m(f)$ given by (7.6), where the execution period, slack period, and transition energy figures are given by (7.8), (7.9), and (7.10), respectively. Again, the optimal frequency f should satisfy the deadline constraint $\frac{WCC}{f} \leq d$ and the feasible frequency range constraint $f_{min} \leq f \leq f_{max}$.

Similar to the case of single-device, the general problem can be seen to give rise to multiple sub-problems by assuming that the product $F((d - B_i)f)$ is equal to a specific $F(b_j)$ for each device $\mathcal{D}_i$. Specifically, in the optimal solution, $F((d - B_i)\, f)$ $(1 \leq i \leq m)$ can be equal to $F(b_{a_i})$ $(0 \leq a_i \leq n)$. In other words, a_i will be equal to the *index* of the cycle group (given in the histogram) that the application is assumed to be executing at time $t = d - B_i$, in the subproblem that corresponds to each tentative optimal solution. Since the cycle group j (from b_j to b_{j+1}) should be executed *before* the cycle group $j + 1$ (from b_{j+1} to b_{j+2}), one can infer that $a_i \geq a_{i+1}$ in a subproblem. Therefore, each sub-problem will be defined by a unique ordered sequence of $a_1, \ldots, a_m$.

Following a reasoning similar to the case of the single-device, one can obtain the corresponding formulation of the subproblem for a given sequence $a_1, \ldots, a_m$ as to *minimize*:

$$\begin{aligned} E_{a_1,..,a_m}(f) = \;& \textstyle\sum_{i=1}^{m} \sum_{j=a_i}^{n-1} (F(b_{j+1}) - F(b_j)) P_a^i (d - \frac{b_{j+1}}{f}) \\ & + \textstyle\sum_{j=0}^{n-1} (af^3 + \sum_{i=1}^{m} P_a^i) \frac{gc_j}{f} (1 - F(b_j)) \\ & + \textstyle\sum_{i=1}^{m} F(b_{a_i}) E_{tr}^i \end{aligned} \tag{7.19}$$

Subject to:

$$\frac{WCC}{f} \leq d \tag{7.20}$$

$$\frac{b_{a_i - 1}}{f} < d - B_i \leq \frac{b_{a_i}}{f} \quad (1 \leq i \leq m) \tag{7.21}$$

$$f_{min} \leq f \leq f_{max} \tag{7.22}$$

where $E_{a_1,..,a_m}(f)$ is the expected energy consumption with the specific values of $a_1,..,a_m$, and the new additional constraints (7.21) are the sufficient and necessary conditions to enforce $F((d-B_i)f) = F(b_{a_i})$. For a given sub-problem, we can define $f'_{low} = max(f_{min}, \frac{WCC}{d}, \frac{b_{a_i-1}}{d-B_i} + \epsilon(\forall i))$ and $f'_{up} = min(f_{max}, \frac{b_{a_i}}{d-B_i}(\forall i))$. By following steps similar to those in Section 7.4, we can conclude that if $f'_{up} < f'_{low}$, the corresponding subproblem does not have a solution. For the case where $f'_{up} \geq f'_{low}$, the solution f'_a is given by:

$$f'_a = (\frac{\alpha - \gamma}{\beta})^{\frac{1}{3}} \tag{7.23}$$

where $\alpha = (\sum_{j=0}^{n-1} \sum_{i=1}^{m} P_a^i gc_j(1 - F(b_j))$, $\beta = 2a \sum_{j=0}^{n-1} gc_j(1 - F(b_j))$, and,
$\gamma = \sum_{i=1}^{m} \sum_{j=a_i}^{n-1} P_a^i(F(b_{j+1}) - F(b_j))b_{j+1})$.

Theorem 2. *The subproblem of minimizing $E_{a_1,..,a_m}(f)$ admits an optimal solution $f_a^* = max\{f'_{low}, min\{f'_{up}, f'_a\}\}$, whenever $f'_{up} \geq f'_{low}$.*

As a quick inspection of the formula for f'_a reveals, it takes only $O(mn)$ steps to solve each sub-problem. However, the apparent computational difficulty comes from the large number of subproblems. In fact, the total number of sub-problems is given by the number of multisets of cardinality m, with elements taken from a finite set of cardinality n, which is equal to $\frac{(n+m-1)!}{m!(n-1)!}$. This gives $O(n^m)$ potential subproblems, which is prohibitively large.

Nevertheless, it is possible to develop a faster solution by observing that *most* of the subproblems have indeed empty feasible regions and that it is necessary and sufficient to consider only at most $m(n+1)+3$ subproblems, each of which is *uniquely defined by a separate combination of the cycle group index j and device index i.* The full details of this faster algorithm of overall complexity $O(mn \log mn)$ with the accompanying proofs are presented in the Appendix A.

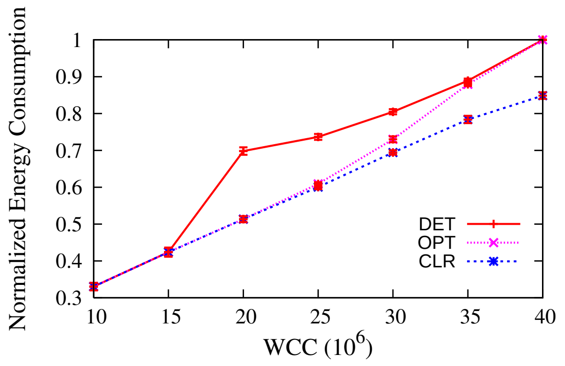


Figure 7.5: Normalized expected energy as a function of WCC $(P = 44ms)$

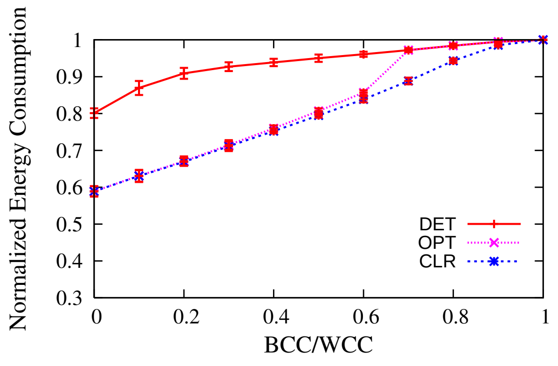


Figure 7.6: Normalized expected energy as a function of $\frac{BCC}{WCC}$ $(P = 44ms$ and $WCC = 20 \times 10^6)$

7.6 Experimental Evaluation

To evaluate the performance gains yielded by the new solution, I constructed a discrete-event simulator in C. In the simulator, I implemented the following three schemes:

- The optimal scheme, denoted by OPT and developed in this chapter, which minimizes the *expected* system energy based on the application's probabilistic workload information, by integrating DVS and DPM in an optimal way.

- The scheme DET (proposed in [21]), which minimizes the system energy consumption again by considering both DVS and DPM features, but by assuming a deterministic workload (equal to WCC).

- The *clairvoyant scheme (CLR)*, that computes the optimal frequency by using the knowledge about the actual workload (number of cycles) of the application in advance. While it is not a practical scheme (since no algorithm can know the exact workload in advance), CLR is included in the comparison to assess the extent at which the new algorithm's performance approaches absolute ideal bounds by exploiting the probabilistic information.

To be consistent with the experimental settings in [21], I performed the experiments by using the actual device specifications from [18] and the CPU power consumption is modeled after the Intel XScale specifications [72]. The application uses three devices during its execution: IBM Microdrive ($B = 24ms$) , Realtek Ethernet Chip ($B = 20ms$) and Simple Tech Flash Card ($B = 4ms$). The frame length P varied from $40ms$ to $100ms$; the application's execution cycles are generated using normal distribution with the mean $\frac{(BCC+WCC)}{2}$ and standard derivation $\frac{(WCC-BCC)}{12}$. This guarantees that 99.7% of the cycles fall in the range $[BCC, WCC]$ and cycles values outside this range are not considered. The $[BCC, WCC]$ range is divided into $n = 100$ cycle groups of equal size. The maximum CPU frequency is assumed to be $1GHz$. Also, the confidence intervals at 97% confidence level are reported.

I first evaluate the effect of the worst-case execution time on the expected energy consumption with the frame length (period) $= 44ms$ as in [21], by changing WCC from 10×10^6 to 40×10^6 cycles with $BCC = 0$ (Figure 7.5). Notice that $WCC = 40 \times 10^6$ corresponds to an almost fully utilized system. In Figure 7.5, all energy consumptions are normalized to that of OPT when $WCC = 40 \times 10^6$. As we can see, the performance of the new solution OPT is very close to that of the clairvoyant scheme CLR, especially when $WCC \leq 30 \times 10^6$. When $WCC \leq 15 \times 10^6$, DET can achieve the same performance as CLR and OPT. The reason is that, in the case of very low workload, the application can be executed with the energy-efficient frequency (f_{ee}) [21] while still meeting the deadline and leaving enough idle time to turn off all the devices. However, when WCC increases beyond 20×10^6, DET chooses the frequency ($f = U$) to minimize the system energy under the WCC case, while OPT can still use f_{ee} to achieve better expected energy savings by considering the probabilistic workload information. But once $WCC > 25 \times 10^6$, OPT has to also increase the frequency to enforce longer idle intervals for device state transitions. In fact, when $WCC = 35 \times 10^6$, considering the interaction of DVS and DPM, OPT is forced to use the frequency $f = U$ just like DET. These results suggest that OPT is able to achieve performance levels that are practically indistinguishable from the clairvoyant algorithm, except when the worst-case workload is very high.

Figure 7.6 shows the relative performance of three schemes as a function of the actual workload variability. Specifically, I consider an application with period= $44ms$, $WCC = 20 \times 10^6$, and vary the ratio $\frac{BCC}{WCC}$. Clearly, the lower this ratio, the more the actual workload deviates from the worst-case. The energy values are normalized with respect to OPT when $BCC = WCC$. It is observed that the performance of OPT is almost identical to that of CLR, when $\frac{BCC}{WCC} \leq 0.6$, exhibiting performance gains of around 35% over DET. However, when the ratio exceeds 0.7, OPT is forced to use the same processing frequency as that of DET. This is primarily due to the fact that large BCC values do not leave much opportunities to optimistically increase the frequency to create long device idle intervals. In fact, when $BCC = WCC$, all three schemes achieve exactly the same performance.

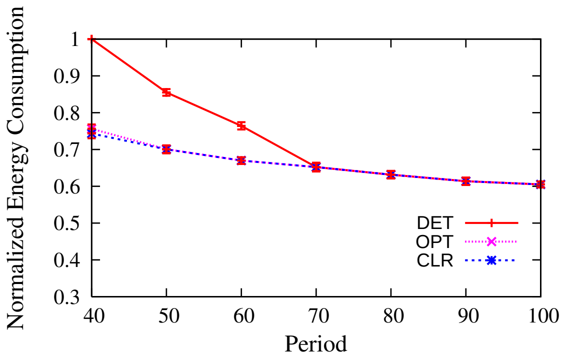


Figure 7.7: Normalized Expected Energy as a function of the application's period ($U = 0.5$)

In Figure 7.7, I study the impact of varying the application period on the energy consumption by setting $U = \frac{WCC}{P} = 0.5$. The energy values are normalized with respect to that of DET when $P = 40ms$. As expected, the energy consumption decreases as the period is increased, and increasing slack amounts enable more device state transitions. The performance of OPT is in fact indistinguishable from that of the clairvoyant scheme CLR. OPT provides energy savings of up to 35% over DET, though the savings tend to decrease with increasing period. This is because, increasing the period while keeping the device break-even times constant enables also DET to put the devices to sleep states, even when planning according to worst-case workload scenarios.

7.7 Chapter Summary

This chapter considered the problem of optimally integrating DVS and DPM policies for real-time embedded applications characterized by probabilistic workload profiles and presented algorithms to minimize the expected system-wide energy. The new solution is based on the precise characterization of the expected energy components, using DVS and DPM properties. First, the problem was solved for an application using a single device and then it was generalized to multiple devices. By observing the special characteristics of the optimal solution for the multiple-device case, a faster algorithm is suggested, which significantly reduces the search space. The experimental evaluation shows that the new algorithm can achieve significant energy savings compared to the previous algorithm proposed in [21] for deterministic workloads and performs comparably even to a clairvoyant optimal scheduler that knows the exact workload in advance.

Chapter 8: Conclusions

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

8.1 Summary of the Dissertation's Contributions

Energy management for real-time embedded systems has been a very active research area in recent past. In particular, the DVS technique that consists in adjusting the supply voltage and processor frequency (speed) has been widely employed to reduce the CPU power. However, recent research also pointed to the significantly increased transient fault rates when applying DVS. The main theme of my dissertation is to analyze and exploit the trade-off dimensions that are involved in joint energy and reliability management of real-time embedded systems.

8.1.1 Maximizing Reliability for Energy-Constrained Real-Time Embedded Systems

- Motivated by applications where the system has to remain functional during a well-defined mission/operation time, with fixed and non-replenishable energy budget, I investigated the maximization of the system reliability for *energy-constrained settings* in hard real-time systems.

- In the dissertation, I considered a real-time application consisting of multiple frame-based tasks and showed how to compute energy allocations (which translate to frequency assignments) to maximize overall reliability, while considering a hard energy constraint. Both static optimal and on-line (dynamic) schemes are developed. The simulation results indicate that the suggested algorithms perform comparably to a clairvoyant optimal scheduler that knows the exact workload in advance.

- The dissertation extended these solutions to general periodic tasks as well. Specifically, by relying on the preemptive Earliest-Deadline-First (EDF) policy, an array of algorithms were proposed, exploiting the dynamic slack in on-line and proactive fashion.

8.1.2 Reliability-Aware Energy Management through Shared Recovery Technique

- Prior RA-PM solutions adopted a conservative mechanism, in the sense that they were based on the principle of statically allocating the available slack for multiple recovery blocks of a pre-determined set of tasks. This in turn hinders the prospects for energy savings by decreasing the available slack for DVS. The dissertation developed a novel solution, which is based on allocating *a shared recovery block/slack (SHR)* that can be used by any faulty task at run-time. This, in turn, improves energy savings. Whenever the recovery task is executed after the detection of a fault in a task, the remaining tasks are executed without any voltage scaling until the end of frame, preserving their reliability by default. It is shown, through simulations, that the SHR technique yields energy savings comparable to those of optimal (but not reliability-aware) DVS solutions

- The dissertation developed also dynamic extensions to the SHR algorithm, enabling additional energy savings through run-time slow-down, whenever tasks complete without a fault (by reducing the size of the reserved recovery tasks) or early (by reclaiming dynamic slack). As a final contribution, the dissertation showed how to extend the framework to task sets represented by directed acyclic graphs (DAGs). In particular, it showed that obtaining the effective deadlines by applying the *as late as possible (ALAP)* policy and then scheduling the tasks in increasing order of effective deadlines preserves the optimality. Further, the online shared-recovery based RA-PM scheme that exploits dynamic slack at run-time is also studied to further improve energy efficiency and system reliability. The proposed schemes are evaluated through extensive simulations.

8.1.3 Generalized Reliability-Oriented Energy Management

- In an attempt to overcome the limitations of the existing RA-PM studies that aim primarily at preserving the system's original reliability, the dissertation proposed the *Generalized Shared Recovery (GSHR)* technique to optimally use the DVS technique in order to achieve a given reliability goal for frame-based real-time embedded applications. The suggested technique determines the optimal number of recoveries to deploy as well as task-level processing frequencies to minimize the energy consumption while achieving the reliability goal and meeting the timing constraints. The recoveries may be shared among tasks, improving the prospects of DVS compared to existing reliability-aware power management frameworks. The experimental evaluation points to the close-to-optimal energy savings of the proposed technique.

- For general periodic tasks, the dissertation proposed a more comprehensive framework to achieve arbitrary reliability levels for individual tasks, when employing DVS. To achieve this task-level reliability objectives, the framework relies on the novel concept of dynamically allocated recoveries. Each task is provided with a certain recovery allowance that can be reclaimed anywhere during the hyperperiod when transient faults are detected. A pseudo-polynomial feasibility test is proposed, along with static and dynamic algorithms to determine the recovery allowance and frequency assignments to minimize energy while satisfying timing and reliability constraints. The simulation results suggest that task-level reliability objectives can be typically achieved while obtaining significant energy savings,

8.1.4 Minimizing Expected Energy Consumption through Optimal Integration of DVS and DPM

- This dissertation addressed the problem of minimizing expected energy consumption of a real-time application whose workload is known only probabilistically. The real-time application is assumed to have access to both DVS and Dynamic Power Management (DPM) techniques. With DPM, the off-chip system components (such as main memory and I/O units) are put to sleep states when they are idle, but an extra transition energy overhead is incurred. First, the problem is solved for an application using a single device. Then the solution is generalized to multiple devices. Specifically, the provided algorithm determines the frequency assignment and device transition decisions to minimize the overall expected energy. The experimental evaluation shows that the proposed algorithm can achieve significant energy savings compared to the previous algorithm proposed in [21] for deterministic workloads and performs comparably to a clairvoyant optimal scheduler that knows the exact workload in advance.

8.2 Future Work

8.2.1 Joint Temperature and Power Management for Real-Time Tasks

Recent research has shown that executing applications at high frequencies can cause overheating problem for processors, as the energy consumed by the processors is converted into heat. Also, the resulting exponential increase in heat density reduces the *lifetime reliability* and increases manufacturing costs. In [74], it is predicted that over 50% of the electronic failures are caused by the increased temperature. The cost of deployed cooling systems increases rapidly with increased heat dissipation [59]. Hence, temperature management has become an important design goal in real-time systems. With the objective of minimizing peak temperature, several temperature-aware power management schemes have been suggested for hard real-time applications [27, 69]. In existing works related to temperature-aware power management [8, 27, 57, 69], the Fourier's Law of heat conduction is used to model the power as a function of temperature and the RC thermal model is used to calculate the thermal parameters.

The first interesting open problem is *energy minimization subject to temperature constraints* under DVS settings for periodic tasks. Specifically, given a real-time scheduling policy, one needs to 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. The second problem is the study of *lifetime reliability* in hard real-time systems. The *permanent (hardware) failures*, such as wear-outs and fatigue failures, are typically due to thermal cycling phenomenon. As a result, increased temperatures and scaling technologies have also an adverse impact on long-term processor *lifetime reliability* [13, 62, 63]. However, to the best of my knowledge, there is no widely-accepted and clear model expressing the relationship between lifetime reliability and the temperature in DVS settings. Hence, another interesting problem is to focus on permanent failures, mainly caused by thermal cycling, to further explore their intrinsic relationship.

8.2.2 Energy-Efficient Real-Time Scheduling on CMPs

The chip multiprocessors (CMPs) that offer multiple processing cores on a single chip have become prevalent in the computing landscape. 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 [12]. Independent and effective management of such clusters (or, the so-called *voltage islands*) would be the ultimate objective in these next-generation systems [12, 31, 51]. One open problem is to investigate *system-level energy management for CMPs* in settings with multiple voltage islands each with *Global Voltage Variable Frequency* feature. There are several non-trivial aspects to this problem, such as, determining the voltage and frequency levels for processing cores within a voltage island, static partitioning of the workload at both the voltage island and core levels, and investigating the benefits of run-time task migration and/or core activation/de-activation. Another open problem is to investigate the reliability-aware energy management for CMPs.

Appendix A: Fast Algorithm for the Multiple-Device Model

In this appendix, I present the details of the new fast solution to the expected energy minimization problem for multiple devices. First, note that for each sub-problem that targets minimizing the expected energy consumption $E_h(f) = E_{a_1,..,a_m}(f)$ corresponding to a specific a_1,a_2,..,a_i,..a_m combination, the frequency range constraints (7.20)-(7.22) can be captured in a single constraint of the form:

$$x_h \leq f \leq y_h \tag{A.1}$$

where $x_h = max(f_{min}, U = \frac{wcc}{d}, \frac{b_{a_i-1}}{d-B_i} + \epsilon(\forall i))$, and $y_h = min(f_{max}, \frac{b_{a_i}}{d-B_i}(\forall i))$. Observe that the sub-problem has a non-empty feasible region if and only if $x_h \leq y_h$. Consider constructing a sequence S, which contains the following $(n+1)m$ points: $\frac{b_0}{d-B_1}$, $\frac{b_1}{d-B_1}$,... $\frac{b_n}{d-B_1}$,... , $\frac{b_0}{d-B_m}$, $\frac{b_1}{d-B_m}$,... $\frac{b_n}{d-B_m}$. This sequence can be sorted in time $O(mn \log mn)$ in non-decreasing order. Observe that these points correspond to possible frequency ranges defined through the constraints (7.21) over all possible sub-problems. Then, the sequence S is updated by eliminating the points that correspond to the infeasible frequency ranges (i.e. those below $max(f_{min}, U)$ or above f_{max}), by removing the duplicate points, and finally by inserting $max(f_{min}, U)$ and f_{max} in proper position. After these operations, the points $z_0, z_1, \ldots$ in sequence S are in total order and their cardinality is at most $m\,(n+1)+2$. The following holds:

Lemma 1. *For each sub-problem with non-empty feasible region, x_h and y_h are obtained from two consecutive points z_i and z_{i+1} in S, when $x_h < y_h$.*

Proof. Based on its definition, x_h can be expressed as $x_h = z_i$ if $z_i = max\{f_{min}, U\}$, otherwise, $x_h = z_i + \epsilon$. Further, $y_h = z_j \,\exists\, j$. Assume that the lemma does not hold. In that case, j must be greater than $i+1$.

In this case, there exists a point $z \in S$ such that $z_i < z < z_j$. Hence, $x_h < z < y_h$ should also hold. Since $z \in S$, it can be expressed as $\frac{b_j}{d-B_k}$ for some existing j and k. Since $x_h = max(f_{min}, \frac{wcc}{d} = U, \frac{b_{a_i-1}}{d-B_i} + \epsilon(\forall i)) < z$, and then $\frac{b_{a_k-1}}{d-B_k} < \frac{b_j}{d-B_k}$ holds and one can conclude that $b_{a_k-1} < b_j$. Because the value of b_i $(\forall i)$ is increasing with increasing index i, then $j > a_k - 1$ holds. Similarly, since $z < y_h = min(\ f_{max}, \frac{b_{a_i}}{d-B_i}(\forall i))$, then $\frac{b_j}{d-B_k} < \frac{b_{a_k}}{d-B_k}$ follows and I obtain $j < a_k$. But this implies $a_k - 1 < j < a_k$, contradicting with the fact that j is an integer. A contradiction is reached, showing the validity of the lemma. $\square$

Lemma 1 implies that the frequency range $[x_h, y_h]$ is of the following form whenever $x_h < y_h$:

$$x_h = \begin{cases} z_k, & \text{if } z_k = max\{f_{min}, U\} \\ z_k + \epsilon, & \text{otherwise.} \end{cases} \tag{A.2}$$

$$y_h = z_{k+1} \tag{A.3}$$

where z_k is a point in the ordered sequence S.

Lemma 2. *Each given frequency range* $[x_h, y_h]$ $x_h \le y_h$ *uniquely determines an original subproblem that minimizes* $E_{a_1,..,a_m}(f)$ *where* $a_i = j + 1$ $(1 \le i \le m)$ *and* j *satisfies both* $b_j < (d - B_i)z_k$ *and* $b_{j+1} \ge (d - B_i)z_{k+1}$.

Proof. To prove this lemma, I will show that, given $x_h < y_h$, one can uniquely compute a_i $(i = 1, .., m)$.

Based on the construction of S, it is clear that there must exist w $(0 \le w < n)$ such that $\frac{b_w}{d-B_i} < z_k$ (i.e. $b_w \le (d - B_i)\ x_h$). In order to determine $a_i = j + 1$, one must find suitable j satisfying $b_j < (d - B_i)\ x_h$ and $(d - B_i)z_{k+1} \le b_{j+1}$ at the same time. I claim that $j = max\{w | \frac{b_w}{d-B_i} < z_k\}$ is the unique solution. In fact, if $j < max\{w | \frac{b_w}{d-B_i} < z_k\}$, then $\frac{b_{j+1}}{d-B_i} \le max\{w | \frac{b_w}{d-B_i} < z_k\} < z_k < z_{k+1}$, which is not possible. Similarly, if $j >$

$max\{w|\frac{b_w}{d-B_i} < z_k\}$, $\frac{b_j}{d-B_i} > z_k$, which contradicts the assumption. Hence, j should be exactly $max\{w|\frac{b_w}{d-B_i} < z_k\}$. It is obvious that j exists and uniquely defined; consequently, $a_i = j+1$ is uniquely determined. □

From the above discussion, it is seen that even in the worst case, one only needs to consider $m(n+1)+1$ possible frequency ranges, which is constructed by consecutive points in S. Further, as Lemma 2 shows, each of these ranges uniquely determines a sub-problem with non-empty feasible region. As a result, this enables to eliminate a large number of unnecessary subproblems.

Now I give a general solution for the lth problem $(1 \leq l \leq m(n+1)+1)$ with the frequency range $x_l < y_l$, where x_l and y_l are defined as in (A.2) and (A.3). The lth problem can be formally expressed as to *minimize*:

$$\begin{aligned} E_{m,l}(f) = \; & \textstyle\sum_{i=1}^{m}\sum_{j=a_{l,i}}^{n-1}(F(b_{j+1})-F(b_j))P_a^i(d-\frac{b_{j+1}}{f}) \\ & +\textstyle\sum_{j=0}^{n-1}(af^3+\sum_{i=1}^{m}P_a^i)\frac{gc_j}{f}(1-F(b_j)) \\ & +\textstyle\sum_{i=1}^{m}F(b_{a_{l,i}})E_{tr}^{a_{l,i}} \end{aligned} \tag{A.4}$$

Subject to:

$$x_l \leq f \leq y_l \tag{A.5}$$

where $E_{m,l}(f)$ is the expected energy with m devices for the lth problem and each $a_{l,i}$ is determined by the frequency range $[x_l, y_l]$ (as stated in Lemma 2).

Using a method similar to that in Section 7.4 and 7.5, one can get:

Theorem 3. *If there exists a feasible solution to the subproblem of minimizing $E_{m,l}(f)$, that feasible solution is equal to $f^*_{m,l} = max\{x_l, min\{y_l, f_{m,l}\}\}$, where $f_{m,l} = (\frac{\alpha-\sigma}{\beta})^{\frac{1}{3}}$,*

$\alpha = \sum_{j=0}^{n-1}\sum_{i=1}^{m} P_a^i gc_j(1-F(b_j))$,

$\beta = 2a\sum_{j=0}^{n-1} gc_j(1-F(b_j))$, and,

$\sigma = \sum_{i=1}^{m} \sum_{j=a_{l,i}}^{n-1} P_a^i(F(b_{j+1}) - F(b_j))b_{j+1}$.

Also observe that the relationship between $f_{m,l}$ and $f_{m,l+1}$ is similar to that between f_i and f_{i+1} in Section 7.4. Therefore, if the constant values of α and β are stored, then based on the result of $f_{m,l}$, one needs at most m steps to compute $f_{m,l+1}$ (since $a_{l+1,i}$ is increased by at most 1 compared to $a_{l,i}$). Hence, starting at $f_{m,1}$, one needs at most $O(mn)$ steps to directly compute all l subproblems. Note that, for completeness, one also needs to compute the expected energy numbers for two cases where $x_l = y_l$ and compare against the best energy figure obtained through the above process. In fact, this is the case only for two boundary values $max(f_{min}, \frac{wcc}{d} = U)$ and f_{max}. The complexity of the entire procedure is $O(mn)$. However, one needs to construct the set S and order the points therein, implying an overall complexity of $O(mn\log(mn))$.

Bibliography

Bibliography

[1] Mobile pentium iii processor-m datasheet. Order Number: 298340-002,Oct 2001.

[2] T. AlEnawy and H. Aydin. On energy-constrained real-time scheduling. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, 2004.

[3] T. AlEnawy and H. Aydin. Energy-constrained scheduling for weakly-hard real-time systems. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[4] H. Aydin. Exact fault-sensitive feasibility analysis of real-time tasks. *IEEE Transactions on Computers*, 56(10):1372–1386, 2007.

[5] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2006.

[6] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and aggressive power-aware scheduling techniques for real-time systems. In *Proc, of the IEEE Real-Time Systems Symposium (RTSS)*, 2001.

[7] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(10):584–600, 2004.

[8] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *Proc. of the Symposium on Theoretical Aspects of Com- puter Science (STACS)*, 2005.

[9] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. Nonlinear programming: Theory and algorithms (third edition). *A John Wiley and Sons, INC.*, pages 576–585, 2005.

[10] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. on VLSI Systems*, 8(3):299–316.

[11] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154.

[12] S. Borkar. Thousand core chips: A technology perspective. In *Proc. of the IEEE/ACM Annual Design Automation Conference (DAC)*, 2007.

[13] D. Brooks, R. P. Dick, R. Joseph, and L. Shang. Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro*, 27(3):49–62, 2007.

[14] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[15] M. Caccamo and G. C. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 1997.

[16] X. Castillo, S. Mconnel, and D. Siewiorek. Derivation and caliberation of a transient error reliability model. *IEEE Trans. on Computers*, 31(7):658–671, 1982.

[17] J. Chen and T. Kuo. Voltage-scaling scheduling for periodic real-time tasks in reward maximization. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[18] H. Cheng and S. Goddard. Online energy-aware i/o device scheduling for hard real-time systems. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2006.

[19] H. Cheng and S. Goddard. Sys-edf: A system-wide energy efficient scheduling algorithm for hard real-time systems. *International Journal of Embedded Systems on Low Power Real-time Embedded Computing*, 4(4):141–151, 2007.

[20] F. Dabiri, N. Amini, M. Rofouei, and M. Sarrafzadeh. Reliability-aware optimization for dvs-enabled real-time embedded systems. In *Proc. of the IEEE International Symposium on Quality of Electronic Design (ISQED)*, 2008.

[21] V. Devadas and H. Aydin. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *Proc. of the IEEE International Conference on Embedded Software (EMSOFT)*, 2008.

[22] V. Devadas and H. Aydin. Real-time dynamic power management through device forbidden regions. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2008.

[23] V. Devadas and H. Aydin. Dfr-edf: A unified energy management framework for real-time systems. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[24] A. Ejlali, B. M. Al-Hashimi, and P. Eles. A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems. In *Proc. of the IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis (CODES)*, 2009.

[25] A. Ejlali, M. T. Schmitz, B. M. Al-Hashimi, S. G. Miremadi, and P. Rosinger. Energy efficient seu-tolerance in dvs-enabled real-time systems through information redundancy. In *Proc.of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2005.

[26] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[27] N. Fisher, J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2009.

[28] Y. Guo, D. Zhu, and H. Aydin. Reliability-aware power management for parallel real-time applications with precedence constraints. In *Proc. Of the IEEE International Green Computing Conference (IGCC)*, 2011.

[29] M. A. Haque, H. Aydin, and D. Zhu. Energy-aware standby-sparing technique for periodic real-time applications. In *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2011.

[30] P. Hazucha and C. Svensson. Impact of cmos technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. on Nuclear Science*, 47(6):2586–2594, 2000.

[31] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proc.of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[32] T. Ishihara and H. Yasuur. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 1998.

[33] R. Iyer and D. Rossetti. A measurement-based model for workload dependence of cpu errors. *IEEE Trans. on Computers*, 33(6):518–528, 1984.

[34] R. Iyer, D. Rossetti, and M. Hsueh. Measurement and modeling of computer reliability as affected by system activity. *ACM Trans. on Computer Systems*, 4(3):214–237, 1986.

[35] V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2005.

[36] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proc. of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[37] M. Kim and S. Ha. Hybrid run-time power management technique for realtime embedded system with voltage scalable processor. In *Proc. of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2001.

[38] G. Koren and D. Shasha. Skip-over: algorithms and complexity for overloaded systems that allow skips. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 1995.

[39] I. Koren and C. M. Krishna. *Fault-Tolerant Systems.* Morgan Kaufmann, San Francisco, CA, USA, 2007.

[40] S. Kung, H. Whitehouse, and T. Kailath. *VLSI and Modern Signal Processing.* Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1985.

[41] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[42] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Proc. of the ACM/IEEE Annual Design Automation Conference (DAC)*, 2001.

[43] C. D. Locke, D. R. Vogel, and T. J. Mesler. Building a predictable avionics platform in ada: A case study. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 1991.

[44] J. Lorch and A. Smith. Improving dynamic voltage scaling algorithms with pace. In *Proc. of ACM SIGMETRICS*, 2001.

[45] D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading Massachusetts, 1984.

[46] R. Melhem, D. Mossé, and E. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Transactions on Computers*, 53(2):217–231, 2004.

[47] B. Mochocki, X. S. Hu, and G. Quan. Transition-overhead-aware voltage scheduling for fixed-priority real-time systems. *ACM Transactions on Design Automation of Electronic Systems*, 12(2):1–26, 2007.

[48] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for lowpower embedded operating systems. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[49] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proc. of the IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis (CODES)*, 2007.

[50] D. K. Pradhan. *Fault-tolerant computer system design.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[51] X. Qi and D. Zhu. Power management for real-time embedded systems on block-partitioned multicore plat- forms. In *Proc. of IEEE International Conference on Embedded Software and Systems (ICESS)*, 2008.

[52] X. Qi, D. Zhu, and H. Aydin. Global reliability-aware power management for multiprocessor real-time systems. In *Proc. of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.

[53] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)- firm guarantee. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2000.

[54] C. Rusu, R. Melhem, and D. Moss. Maximizing rewards for real-time applications with energy constraints. *ACM Transactions on Embedded Computing Systems*, 2(4):537–559, 2003.

[55] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed priority rt-systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2003.

[56] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2002.

[57] J. E. Sergent and A. Krum. *Thermal Management Handbook*. McGraw-Hill, 1998.

[58] T. Simuinic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic voltage scaling and power management for portable systems. In *Proc. of the IEEE/ACM Design Automation Conference (DAC)*, 2001.

[59] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proc. of the ACM/IEEE Annual International Symposium on Computer Architecture(ISCA)*, 2003.

[60] R. Sridharan, N. Gupta, and R. Mahapatra. Feedback-controlled reliability-aware power management for real-time embedded systems. In *Proc. of the ACM/IEEE Annual Design Automation Conference (DAC)*, 2008.

[61] R. Sridharan and R. Mahapatra. Reliability aware power management for dual-processor real-time embedded systems. In *Proc. of the IEEE/ACM Annual Conference on Design Automation (DAC)*, 2010.

[62] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The case for lifetime reliability-aware microprocessors. In *Proc. of the ACM International Symposium on Computer Architecture (ISCA)*, 2004.

[63] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2004.

[64] V. Swaminathan and K. Chakrabarty. Energy-conscious, deterministic i/o device scheduling in hard real-time systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(7):847–858.

[65] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems. *ACM Trans. on Embedded Computation Systems*, 4(1):141–167, 2005.

[66] V. Swaminathan, K. Chakrabarty, and S. S. Iyengar. Dynamic i/o power management for hard real-time systems. In *Proc. of the IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis (CODES)*, 2001.

[67] M. K. Tavana, M. Salehi, and A. Ejlali. Feedback-based energy management in a standby- sparing scheme for hard real-time systems. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2011.

[68] TGFF. http://ziyang.eecs.umich.edu/ dickrp/tgff/.

[69] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Proc. of the EuroMicro Conference on Real-Time Systems (ECRTS)*, 2006.

[70] T. Wei, P. Mishra, K. Wu, and H. Liang. Online task-scheduling for fault-tolerant low-energy real-time systems. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2006.

[71] H. Wu, B. Ravindran, and E. Jensen. Utility accrual real-time scheduling under the unimodal arbitrary arrival model with energy bounds. *IEEE Transactions on Computers*, 56(10):1358–1371, 2007.

[72] R. Xu, D. Mosse, and R. Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. for Embedded Computing Systems*, 25(4):9:1–9:40, 2007.

[73] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1995.

[74] L. Yeh and R. C. Chu. *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices.* ASME Press, 2002.

[75] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[76] F. Zhang and S. Chanson. Throughput and value maximization in wireless packet scheduling under energy and time constraints. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[77] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proc. of the IEEE/ACM Design, Automation and Test in Europe (DATE)*, 2003.

[78] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. Stan. Optimal procrastinating voltage scheduling for hard real-time systems. In *Proc. of the IEEE/ACM Annual Design Automation Conference (DAC)*, 2005.

[79] B. Zhao and H. Aydin. Minimizing expected energy consumption through optimal integration of dvs and dpm. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2009.

[80] B. Zhao, H. Aydin, and D. Zhu. Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. *ACM Transactions on Design Automation of Electronic Systems (TODAES) (Under Review).*

[81] B. Zhao, H. Aydin, and D. Zhu. Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems. In *Proc. of the IEEE International Conference of Computer Design (ICCD)*, 2008.

[82] B. Zhao, H. Aydin, and D. Zhu. Enhanced reliability-aware power management through shared recovery technique. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2009.

[83] B. Zhao, H. Aydin, and D. Zhu. On maximizing reliability of real-time embedded applications under hard energy constraint. *IEEE Transactions on Industrial Informatics*, 6(3):316–328, 2010.

[84] B. Zhao, H. Aydin, and D. Zhu. Generalized reliability-oriented energy management for real-time embedded applications. In *Proc. of the IEEE/ACM Annual Design Automation Conference (DAC)*, 2011.

[85] B. Zhao, H. Aydin, and D. Zhu. Energy management under general task-level reliability constraints. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012.

[86] J. Zhou, C. Chakrabarti, and N. Chang. Energy management of dvs-dpm enabled embedded systems powered by fuel cellbattery hybrid source. In *Proc.of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[87] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[88] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2006.

[89] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2007.

[90] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. *IEEE Trans. Computers*, 58(10):1382–1397, 2009.

[91] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2004.

[92] D. Zhu, R. Melhem, D. Mossé, and E. Elnozahy. Analysis of an energy efficient optimistic tmr scheme. In *Proc. of the IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2004.

[93] J. F. Ziegler. Trends in electronic reliability: Effects of terrestrial cosmic rays. available at http://www.srim.org/SER/SERTrends.htm, 2004.

Curriculum Vitae

Baoxian Zhao got his Bachelor degree and Master degree of Computer Science and Engineering from Nanjing University of Aeronautics and Astronautics, Nanjing, China in 2002, and 2005, respectively.