Flexible and Adaptive Real-Time Task Scheduling in Cyber-Physical Control Systems

Xiaotian Dai

PhD

University of York

Computer Science

January 2019

Abstract

In a Cyber-Physical Control System (CPCS), there is often a hybrid of hard real-time tasks which have stringent timing requirements and soft real-time tasks that are computationally intensive. The task scheduling of such systems is challenging and requires flexible schemes that can meet the timing requirements without being over-conservative.

Fixed-priority scheduling (FPS) is a scheduling policy that has been widely used in industry. However, as an open-loop scheduler, FPS has low system dynamics and no feedback from historic operation. As the working conditions of a CPCS will change due to both internal and external factors, an improved scheduling scheme is required which can adapt to changes without a costly system redesign.

In recent years, there is a large research interest in the co-design of control and scheduling systems that explicitly considers task scheduling during the design of a controller. Many of these works reveal the possibility of adapting control periods at run-time in order to accommodate varying resource requirements and to optimise CPU utilization. It is also shown that control quality can be traded off for resource usages.

In this thesis, an adaptive real-time scheduling framework for CPCS is presented. The adaptive scheduler has a hierarchical structure and it is built on top of a traditional FPS scheduler. The idea of dynamic worst-case execution time is introduced and its cause and methods to identify the existence of a trend are discussed. An adaptation method that uses monitored statistical information to update control task periods is then introduced. Finally, this method is extended by proposing a dual-period model that can switch between multiple operational modes at run-time. The proposed framework can be potentially extended in many aspects and some of these are discussed in the future work. All proposals of this thesis are supported by extensive analysis and evaluations.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to give the main thanks to my supervisor Professor Alan Burns for all his commitments and efforts. I could not have completed my PhD without his guidance, insights and extensive knowledge. I want to give my appreciation to all the academic staffs in the Department of Computer Science, including Neil Audsley (as my internal), Iain Bate, Ian Gray, Leandro Indrusiak and Robert Davis for their valuable advice. I would also like to thank my colleges in the Real-Time Systems Group at the University of York, to name a few, Benjamin Lesage, David Griffin, Frank Soboczenski, Hao Xu, Haitao Mei, James Harbin, Shuai Zhao, Xinwei Fang and Zhe Jiang. Thank you for being supportive and created such a productive and friendly research environment in this group.

Finally, to my wife Yingyi Kuang, whom I married during this PhD. Thank you for your love and tolerance. Life will not be a wonderful journey without your companionship, and I wish this journey will not have an end.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. Certain parts of the material presented within this thesis have appeared in published and submitted papers, journals and reports. Specifically, these are:

- Xiaotian Dai. The Role of Flexible Models in Adaptive Real-Time Scheduling, Qualifying Dissertation Report, Department of Computer Science, University of York, 2015.
- Xiaotian Dai and Alan Burns. Predicting Worst-Case Execution Time Trends in Long-Lived Real-Time Systems, 22nd International Conference on Reliable Software Technologies - Ada-Europe, 2017.
- Xiaotian Dai and Alan Burns. Period Adaptation of Real-Time Control Tasks with Fixed-Priority Scheduling, submitted to Special Issue on Advances in Reliable Software Technologies: An Ada Perspective, Journal of Systems Architecture (JSA), 2019.
- Xiaotian Dai, Wanli Chang, Shuai Zhao and Alan Burns. A Dual-Mode Strategy for Performance-Maximization and Resource-Efficient CPS Design, submitted to International Conference on Embedded Software (EmSoft), 2019.

This work has not previously been presented for an award at this, or any other university. All sources are acknowledged as references.

Chapter 1

Introduction

Real-Time Systems (RTS) are computer systems that have stringent timing requirements in addition to functional correctness [94] [35]. A timing requirement is often defined as a temporal specification, in which results should be produced within a specified deadline deterministically or probabilistically [120]. Typical real-time systems include real-time control, communication, data processing, power, avionic and automotive systems. These applications often involve a multiprogramming environment in which multiple tasks have to compete for limited computational resources.

Scheduling in Real-Time Systems is a research framework that includes scheduling algorithms, resource sharing protocols and analytical methods (such as feasibility or schedulability tests), to verify that real-time systems will meet their deadlines [40]. Scheduling can refer to task scheduling, network packet scheduling or resource scheduling. This thesis focuses on task scheduling, and real-time task scheduling and real-time scheduling will be used interchangeably, if not explicitly stated otherwise.

Historically, real-time tasks have been statically scheduled by cyclic executives. During the last few decades, real-time scheduling has been progressed from this static approach to dynamic priority-based scheduling methods. The seminal work of Liu and Layland in 1973 build the foundation of priority-based scheduling [93], including Rate Monotonic Fixed-Priority Scheduling (FPS-RM), and Earliest Deadline First (EDF). The use of dynamic scheduling has significantly improved resources usage when the task workload is not fully known.

With the advance of ubiquitous computing and communication, new applications have emerged, which have more complex functional requirements and increased performance demands than traditional embedded and real-time systems. Some example systems include wireless sensor network, robotic systems, intelligent transportation systems and Industry 4.0, which in their nature, are Cyber-Physical Systems (CPS). Generally speaking, a CPS is an engineering system that requires tight integration of computing, communication and control to achieve stability, performance, reliability, robustness and efficiency in dealing with physical systems in many application domains [112]. CPS often involve feedback loops where physical processes affect computations and vice verse [88]. Many CPS need to be deployed in the field for long periods of time, and often with limited or no human supervision, which are referred to as long-lived CPS in this work.

Compared with traditional applications, long-lived CPS are often more vulnerable to system changes due to the evolution of functional requirements, degradation of hardware and lack of maintenance. Although dynamic scheduling can improve system flexibility, the effectiveness of scheduling is still reliant on the correctness of the task model. However in long-lived CPS applications, system characteristics as well as system models will change and evolve over time. In addition, these systems are often working in an open environment, in which the computational workload cannot be fully modelled and predicted in advance. In this case, a scheduling analysis cannot be properly applied or cannot be applied without conservative assumptions, e.g., a large conservative worst-case execution time (WCET).

To solve these issues, flexibility and domain specific knowledge can be applied to enhance traditional scheduling methods for long-lived applications. A potential way of achieving this is to support adaptive behaviour using techniques e.g., machine learning and cloud computing, in combination with traditional schedulers. The use of adaptation will ease the process of making modifications to system software and improve the resilience against timing failures.

The vision is that this research will ultimately lead to an adaptive scheduling system, that can be fully integrated into the design, implementation, verification and testing of the next-generation cyber-physical systems. In the reminder of this chapter, scheduling issues in cyber-physical systems and a general framework of adaptive scheduling are introduced. The proposition of the thesis is then presented. The chapter completes with an outline of this thesis.

1.1 Scheduling Issues in Cyber-Physical Systems

Cyber-physical systems are emerging systems that often have close integration of physical systems and computer systems. As physical processes are involved, a CPS often has different structures and characteristics, which makes its design and implementation different from current engineering systems. There are many challenges identified in CPS that could affect scheduling of real-time tasks. To list a few:

Hardware Degradation: the hardware components of a CPS may vary with time. Hardware in general is subjected to ageing and degradation over system operation time. This could result in increased amount of computation and access time, and lead to more frequent fault handling due to temporary or permanent failure of hardware components.

Software Evolution: the software aspect of CPS is often subject to change. The functional requirement could change even after the system is deployed. In an adaptive system, the software could learn from its past executions and improve and evolve its internal structure, which often leads to a higher demand in computation.

Context Sensitive: unlike traditional real-time and embedded systems, a CPS has to respond to its environment, which is often dynamic. To interact with the environment, the system will have dependencies on its external inputs and states, which makes the system sensitive to spatial and temporal context.

Open Environment: many CPS are exposed to an open environment, which indicates the computation and communication have to deal with changing scenarios. The system may have to reconfigure its structures and parameters and reallocate its resources.

All of these issues and challenges motivate us to design a scheduling system for cyber-physical systems that is more flexible and adaptive than conventional FPS-RM and EDF scheduling.

1.2 A Framework of Adaptive Scheduling Systems

An adaptive system is a flexible system that is able to adjust its behaviour in response to its perception of the environment and the system itself [53]. Adaptive systems in general can be extensively found in natural systems, e.g., brain, nervous and immune system, atmosphere system, ecosystem, economics and even human societies.

In the context of computer systems, adaptation or self-adaptation is a well-known technique to handle growing complexities of software systems in a dynamic and unpredictable environment. The MAPE-K framework [85] is a widely accepted adaptive and autonomic computing framework that is proposed by IBM. MAPE-K stands for Monitor-Analyse-Plan-Execute over a shared Knowledge. A diagrammatic representation of the framework is shown in Figure 1.1.

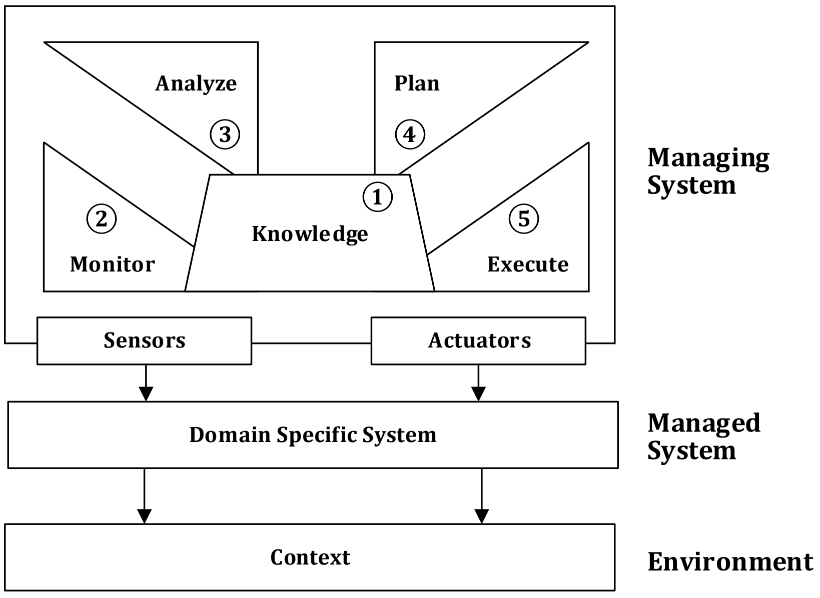


Figure 1.1: The MAPE-K structure for adaptive systems: M - Monitor; A - Analyze; P - Plan; E - Execute; K - Knowledge.

As shown in the diagram, the *Monitor* component (marked with (2) in Figure 1.1) acquires information from the managed system and the environment. The obtained information is used to update the *Knowledge* (1). Based on the knowledge, the *Analyze* component (3) will determine the need for adaptation of the managed system. A sequence of adaptation actions are planned by the

Plan module (4) and sent to be executed by the *Execute* component (5).

The use of feedback mechanism is the key in the adaptation process. The closed-loop of monitoring, analysis, plan and control ensures the system would react to the environment and condition changes accordingly. The system could also 'learn' from its historical behaviour and adapt itself to be more robust and optimised.

In this thesis, the problem of applying the MAPE-K framework into real-time scheduling context is considered. The scheduling system of a CPS should be able to adjust its behaviour at run-time in response to the perception of the system itself and its environment. To fit real-time task scheduling into the MAPE-K framework, the components of MAPE-K are mapped into their correspondences in the context of scheduling, and adjust the structure to accommodate for a CPS. The overall structure is shown in Figure 1.2, which will be referred as the Adaptive Task Scheduling Framework, or ATAS.

The ATAS framework is formed of two sub-systems: 1) the target system and 2) the adaptation manager. As in a CPS, the target system in most cases only has limited resources and computational power, it is preferable to use a distributed structure in which the computation intensive work is offloaded to a cloud computer. Multiple target systems can share one such more powerful cloud computer, and these two sub-systems are connected through network infrastructures, i.e., the Internet or a local network, which are readily available in CPS.

To make it more specific, the target system will conduct monitoring and executions, which can be integrated as callable functions or services into the real-time operating system. The modelling, analysis and planning modules are implemented on the adaptation manager in the cloud, which will have access to target systems and improve their performance through continuous monitoring and reconfiguration. The use of cloud reduces the overhead of local embedded computers, and makes this approach novel and feasible in the context of CPS. Further details of the framework will be provided in later chapters of this thesis.

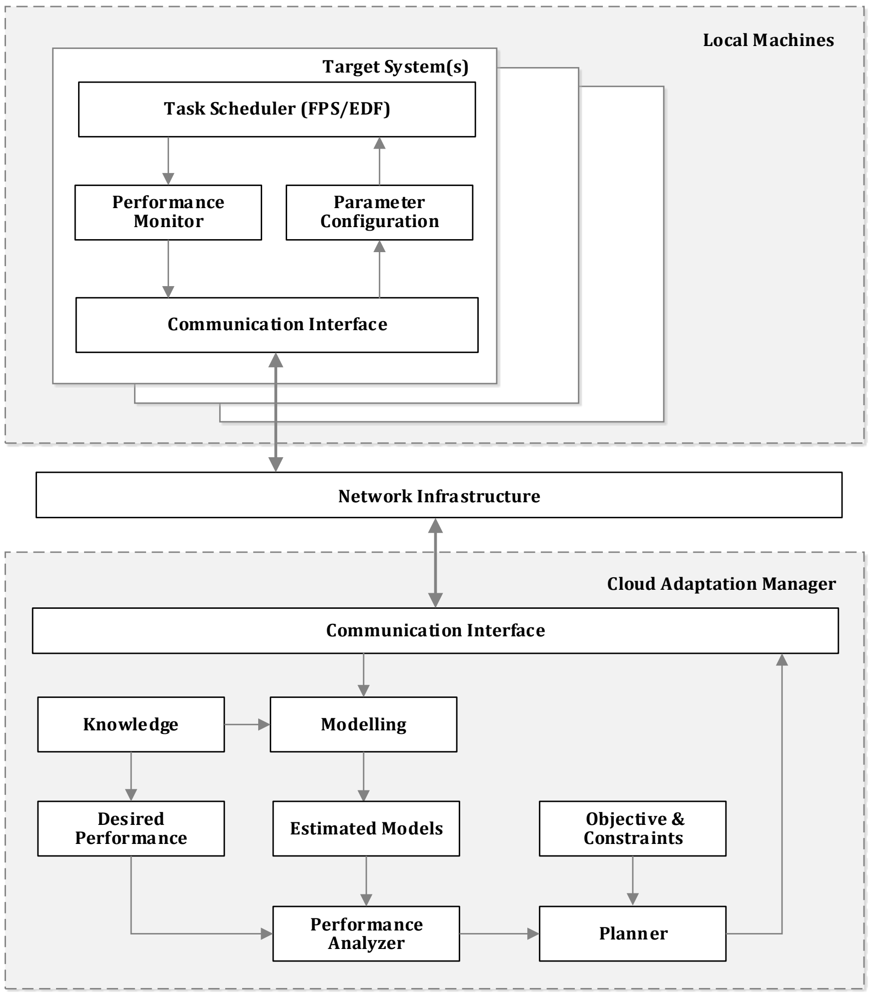


Figure 1.2: The structure and components of the proposed Adaptive Task Scheduling framework (ATAS)

1.3 Thesis Proposition

The research focus of this thesis is to support adaptive real-time scheduling in cyber-physical control systems. The central proposition of this thesis is:

Flexible and adaptive task scheduling can improve the schedulability of long-lived cyber-physical systems. This can be achieved by using novel flexible models for making design trade-offs, utilising statistical learning techniques for supervision and analysis, and using cloud computing facilities for adaptively managing resource reclaiming.

The main contribution of this thesis is a framework that enables two opposing dynamic behaviours of long-lived CPS to be addressed and balanced. First, the tendency for task execution times to increase; the second, which can be used to compensate for this deterioration, is for control programs to be exhibited conservative performance.

Such control programs can be modified to run less often thereby releasing CPU bandwidth that can be made available to tasks that are predicted, in the future, to require more than their statically determined worst-case execution times. The overall result of applying this framework is more resilient CPS.

1.4 Thesis Organisation

In this chapter, a general introduction of this thesis is given. The scheduling issues in CPS are discussed and an adaptive scheduling framework, ATAS, is proposed. The remaining contents of this thesis is organised as follows:

Chapter 2: Flexible and Feedback Methods in Real-Time Scheduling. This chapter introduces the background of this work. A general overview of real-time scheduling is introduced. A review is given on flexible scheduling methods including elastic task scheduling and feedback control scheduling; other novel scheduling methods such as probabilistic and statistical scheduling are also discussed.

Chapter 3: Trend Analysis of Dynamic Worst-Case Execution Times. In this chapter, the idea of dynamic worst-case execution times (dWCET) is discussed. The causation, consequence and monitoring methods of dWCET are discussed. The potential methods that could detect and predict trends in dWCETs are then explored, followed by a comparison experiment on four representative approaches.

Chapter 4: Period Adaptation of Real-Time Control Tasks. In this chapter, an adaptation method is demonstrated, in which a control system can adapt to workload changes by using a flexible task model and run-time monitoring. A cloud server is involved in processing, analysis and making adaptation decisions. A demonstrated example is given by using simulations.

Chapter 5: Dual-Period Task Model. This chapter introduces a novel task scheduling model in which each control task is assigned two flexible task periods. The proposed model can switch between two operational modes depending on the system state at run-time. It is identified that the dual-period model can significantly reduce CPU usage with limited impact on the control performance.

Chapter 6: Conclusions and Future Work. In this last chapter, a conclusion of this thesis is given. The contributions are summarised and re-emphasised. The thesis hypothesis is also revisited, and some of the future works are explored and discussed for each contribution.

Chapter 2

Flexible and Feedback Methods in Real-Time Scheduling

The idea of Adaptive Task Scheduling Framework (ATAS) is motivated by several parallel research. In this chapter, some of these will be studied in order to describe their advances over traditional scheduling methods, as well as identifying the issues in current practice in order to understand the contributions of this thesis.

To start with, a brief look of prevailing real-time scheduling methods and their limitations for scheduling CPS will be given, e.g., in dealing with overloads and uncertainties. Several online scheduling methods that involve runtime monitoring and actuating are then introduced as solutions which can improve system flexibility and resilience. Finally, the idea of statistical learning and making long-term improvements is proposed as a research interest that can ultimately achieve adaptive scheduling systems that have optimised resource allocation and performance.

2.1 Overview of Real-Time Scheduling

Real-time systems often require interactions with physical processes, and there could be a catastrophic consequence if one or more tasks in the system fail to complete within their deadlines. Scheduling these time-critical tasks requires methods that are analysable and verifiable, and needs a systematic approach that is significantly different from scheduling in general operating systems.

The main concern in the design of real-time systems is to meet system functional requirements under given resource and temporal constraints. This is in contrast with general purpose computing systems in which achieving high average performance is the main objective. To satisfy the timing constraints, real-time systems should be designed and validated in a way which could guarantee that the tasks will always meet their deadlines at run-time. In general, this is achieved by *real-time scheduling*, which includes theory, algorithms and methodologies for scheduling tasks and resources. This section will introduce the basis of real-time scheduling and discuss some of the most outstanding work that have been done in the last few decades.

2.1.1 Background and Terminology

Firstly some of the fundamentals of real-time systems are examined. A full list of commonly used symbols is given in Table 2.1.

A real-time application normally consists of several sub-programs, or *tasks*. Each task τ_i has a specific functionality and is invoked either by an internal/external event or a timer interrupt. The total number of tasks in a task set is defined as n. A *job* $j_{i,k}$ is a single release of a task, and is the basic unit that can be scheduled by a task scheduler. The *period* T_i of a task is defined as the inter-arrival time between two consecutive releases of jobs belonging to that task. The *release time* r_i of a task is the time instant at which the task becomes available to be executed. For a periodic task, the k^{th} release time can be worked out as $r_{i,k} = (k-1)T_i + \phi_0$, where ϕ_0 is the *phase*, i.e. the release time of the first job.

Tasks in a real-time system normally have temporal constraints on their completion time after their releases, which is known as (relative) *deadline* D_i. A deadline is said to be *hard* if missing the deadline will jeopardise the system and may lead to disastrous consequence. On the contrary, a deadline is *soft* if missing one will only degrade the performance of the system, but will not cause damages to the environment or to the operator. In addition to hard and soft, there is also a *firm* deadline, which is defined as a deadline that can be missed but the utility of the result will be unusable once after the deadline.

The amount of time a task takes to execute without any interruption is defined as *execution time*. Due to execution path divergences and hardware architecture, the execution times of a task could be different in each run. The

Table 2.1: Table of Notations

Symbol	**Description**
Γ	a task set
τ_i	a task indexed by i
n	number of tasks
$j_{i,k}$	k^{th} released job of task i
$r_{i,k}$	k^{th} release time of task i
ϕ_0	task release phase
P_i	priority of task i
C_i	worst-case execution time of task i
T_i	period of task i
D_i	deadline of task i
R_i	response time of task i
B_i	blocking time of task i
I_i	interference time of task i
U_i	utilization of task i
U_{ub}	upper bound on total utilization
U_d	desired utilization
Q_s	budget of an execution time server
T_s	period of an execution time server

upper bound on task execution time is known as the *worst-case execution time* (WCET), denoted by C_i. In real-time scheduling theory, the WCET is important to derive the *response time* R_i, i.e., the length of time that a task takes to finish after its release, which is normally larger than the computation time due to interrupts and preemptions.

In order to produce an analysable and predictable system, a few assumptions about the task model in the system being analysed are often required. One of the most commonly used task models is the *simple periodic task model*, which has the following properties:

- All tasks are periodic with a fixed period;
- All tasks have known WCETs;

- All tasks have deadline less or equal to the period;
- All tasks are executed on a uniprocessor, i.e., CPU with a single core;
- There are no execution precedence or data dependency between tasks;
- No implementation-introduced overheads are considered, e.g., context-switching.

The simple periodic task model is a simplified but yet reasonable description of a wide range of applications. Later in this thesis, it will be shown how some of these assumptions can be relaxed.

2.1.2 Scheduling Periodic Tasks

Modern real-time systems use *multiprogramming* to improve the utilization of a processor. In such an environment, multiple tasks are executing concurrently but give the impression that they were executed on a dedicated processor. Computation resources are shared by different tasks according to a *scheduling algorithm*, which will determine the orders in which a task can use the processor at a certain time instant.

Historically, tasks in a real-time system are scheduled offline using static cyclic executives [120]. Basically, the cyclic executive is a table of procedure calls, which records the sequence of the tasks being called. The complete table is called the *major cycle*, which consists of a number of *minor cycles.* The length of the major cycle is equal to the least common multiple of the periods (hyperperiod) among all the tasks, while the length of each minor cycle is fixed and is determined by the characteristic of the task set. For the cyclic executive approach, it is assumed that all the task parameters, including periods, computation times and future release times are known a priori when determining the scheduling table. However, this limits the flexibility of the system as even a small change in the characteristic of the task set will require the scheduling table to be recalculated.

In 1973, Liu and Layland proposed two dynamic scheduling algorithms: *Rate-Monotonic Fixed-Priority Scheduling* (RM-FPS, or RMS) and *Earliest Deadline First* (EDF) [93]. RMS uses a static priority assignment policy based on task periods namely the rate-monotonic policy, while EDF uses dynamic priorities according to the absolute task deadlines. RMS and EDF have proven to be optimal scheduling methods for uniprocessor systems, in which tasks are independent and can be preempted at any time. The advantage of dynamic scheduling over cyclic executives is that it makes scheduling decisions at run-time, so the exact release and execution times of the tasks are not necessarily needed.

For systems scheduled with RMS, there exists a utilization bound below which the task set in the system will always be schedulable. The least upper utilization bound of RMS is derived in [93] as $U_{ub} = n(2^{1/n} - 1)$. Thus the task set is schedulable if:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{2.1}$$

For example, when there are only two tasks, $U_{ub} = 2(2^{1/2} - 1) = 0.83$ and when n approaches ∞, $U_{ub} \to 0.693$. It is notable that this is a sufficient but not necessary condition. Task sets with total utilization higher than this bound could still be schedulable, e.g., schedulability of tasks with harmonic periods is guaranteed up to 100% utilization.

The utilization-based test for EDF is also derived in the same work [93]. It is indicated that the utilization bound of simple periodic tasks with $D_i = T_i$ is equal to 1 for all task sets:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.2}$$

Unlike the utilization test in FPS, which is only sufficient, this criterion is exact (both sufficient and necessary). For more complicated task sets, i.e., tasks with $D_i < T_i$, the feasibility of EDF can be obtained by *Processor Demand Analysis* (PDA). The feasibility of a task set is guaranteed if the process demand $g(\cdot)$ between the interval $[t_1, t_2]$ is less or equal to the available time, which is:

$$\forall \{t_1, t_2 \,|\, t_2 \geq t_1\} \quad g(t_1, t_2) \leq (t_2 - t_1) \tag{2.3}$$

Although a higher utilization can be achieved by EDF, it behaves badly when the CPU overloads. Unlike FPS, where only low priority tasks will suffer from overloads, EDF has a less predictable behaviour that is known as a ‘domino effect’ [40], i.e., a cascade of deadline misses. Another drawback of EDF is that it has higher overheads than FPS, which also limits its use in real-world applications.

2.1.3 Scheduling Sporadic and Aperiodic Tasks

Many real-time tasks have an arrival pattern that is not periodic. These tasks are often triggered by external events and may have stochastic arrival times. Depending on whether a minimal arrival interval is properly defined, these tasks can be categorised as *sporadic* (minimal inter-arrival time is defined) or *aperiodic* tasks.

One way to handle non-periodic tasks is to run them as background activities. In this case, these tasks are giving priorities lower than periodic tasks. As a consequence they can only use the spare capacity of the processor and cannot steal resources from periodic tasks [35]. The drawback of this method is that the average throughput and response time could be unsatisfactory with background execution.

In order to handle sporadic and aperiodic tasks in a predictable way with Quality-of-Service (QoS) [64] [14] requirements, a mechanism namely *Execution-Time Server* is introduced, which is a reservation-based scheduling method [120]. An Execution-Time Server is a straightforward solution for handling sporadic and aperiodic tasks. It is a dedicated task that runs periodically, which behaves just like a normal periodic task. One example of execution-time servers is the *polling server*, which is a periodic task with a fixed period T_s and a fixed execution capacity Q_s [35]. The polling server is released periodically at run time and at each release, the server checks if there are any non-periodic tasks in the ready queue. If there is a ready task, the server will execute the task with its assigned capacity. Once the budget is exhausted, the polling server is suspended and has to wait for the next release. The polling server is simple and effective. However, the capacity will be wasted if there is no task available when the server releases.

To solve the limitation of the polling server, many solutions are proposed, e.g., *Priority Exchange*, *Deferrable Server*, and *Sporadic Server* [35] [120]. These methods use principles quite similar to the polling server, but they can reduce wasted capacity by preserving it if there are no tasks available. These three algorithms are differentiated in the way of preserving and replenishing capacity, and they use different feasibility analysis to determine the maximum capacity of the server. Since unused capacity is retained if there are no aperiodic tasks, these servers are known as *bandwidth preserving servers.* The equivalent bandwidth that a server can use is equal to Q_s/T_s. Since Periodic and Sporadic Severs behave the same as a periodic task, the feasibility of a task set involving servers can be analysed by treating the servers as a period task (T_i, C_i), with $T_i = T_s$ and $C_i = Q_s$. For Deferrable Server, it is equivalent to a periodic task with release jitter [120].

Similar server techniques are used for systems scheduled with EDF, which is an extension to the fixed-priority server. One of these servers is the *Constant Bandwidth Server* (CBS) [4] [3] [1]. A CBS has an instantaneous capacity c_s, a period T_s, a maximum budget Q_S and a relative deadline d_s. Once the budget is decreased to 0, the server budget is recharged to the maximum value Q_s and the server deadline is postponed by a period T_s. An important property of CBS is that during any duration L, the demand of the server will not exceed $Q_s/T_s * L$. Thus a priori guarantee can be made, even when the actual requests may sometimes exceed the expected load [3]. The *CApacity SHaring* (CASH) and the *Greedy Reclamation of Unused Bandwidth* (GRUB) algorithm are also proposed to reclaim resources that are incorrectly assigned [120].

2.1.4 Response Time Analysis

For fixed-priority scheduling, the utilization bound for the feasibility test is both conceptually and computationally simple. However, it requires $D_i = T_i$ and P_i assigned according to rate-monotonic. Also, it is sufficient but not necessary, which in some cases is pessimistic. For example, for a task set with harmonic periods, the upper utilization bound can reach 100% [137] [36].

An exact feasibility test for fixed-priority scheduling, that derives the worst-case response time of a task, is fixed-priority *response time analysis* (RTA) [17]. The standard RTA equation is a recursive equation that is formed of two parts: the worst-case execution time, plus an interference time resulting from tasks with higher priorities [35]:

$$\begin{aligned} R_i &= C_i + I_i \\ &= C_i + \sum_{j \in hp(i)} \left\lceil \frac{Ri}{T_j} \right\rceil C_j \end{aligned} \tag{2.4}$$

in which R_i is the response time, I_i is the interference time, $hp(i)$ is the set of all tasks that have priorities higher than i, and $\lceil \cdot \rceil$ is the ceiling function that gives the smallest integer greater than the given fractional number. The response time equation can be solved by forming a recurrence relationship [16]:

$$\omega_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\omega_i^n}{T_j} \right\rceil C_j \tag{2.5}$$

The intermediate variable ω_i^n is monotonically non-decreasing. A solution is found when $\omega_i^{n+1} = \omega_i^n$. If ω_i^n excesses the period of the task, then the task will not meet its deadline. Compared with the utilization test, the RTA can be further used for tasks with arbitrary priority ordering [120] and tasks that have deadlines less than periods [108].

2.1.5 Summary

The cyclic executive packs all the tasks into 'minor cycles' in a way that will enable all task deadlines to be met. The scheduling is done offline so there is negligible overhead at run-time. The feasibility is guaranteed by constructing the scheduling table while considering the deadlines as constraints that have to be satisfied. Although cyclic executive is effective and safe, it becomes increasingly difficult to pack the cycles as the number of tasks grows in the system.

Dynamic scheduling algorithms, e.g., FPS and EDF, overcome the drawbacks of the cyclic executive by making scheduling decisions at run-time. FPS uses static priority assignment according to the periods (rate-monotonic) or deadlines (deadline-monotonic). It is proven that RMS is optimal, in the sense that if a task set with $D_i = T_i$ can be scheduled by any other fixed priority-based approach, it can also be scheduled by RMS. The optimality of EDF is proven in a similar way in the context of deadline-driven scheduling.

The concept of server is introduced as the way to incorporating non-periodic tasks into the framework of periodic scheduling. A server is characterised as (Q_s, T_s) and is scheduled as a periodic task. Different server schemes differ in their policies of replenishing the capacity and reclaiming the unused bandwidth. Guarantees can still be made if the actual load exceeds the expectation.

The scheduling algorithms introduced in this section are sometimes known as the 'plain algorithms'. However, to meet the requirement of certain applications, modifications and extensions have to be made. Some of the extended variations will be discussed in the rest of this chapter.

2.2 Research Challenges and Trends in Real-Time Scheduling for CPS

Cyber-Physical Systems [136] often have complex functional requirements and are exposed to open environments with interactions to physical elements and other systems. There are two aspects that were identified as challenges to apply traditional scheduling algorithms:

1) Execution times can be temporally or spatially dependent, and variations of system workload can be large;
2) The worst-case boundary on execution times are hard to predict and can be imprecise and inaccurate, or can only be accurate and valid for a short period of time [135].

The timing requirements imposed on real-time tasks are typically guaranteed with offline analysis by considering the worst-case scenario. For instance, the response time analysis of a uniprocessor fixed-priority scheduling uses the worst-case execution time and the critical instance to determine the maximum time a task will take to complete [35]. A guarantee is made if the analysed response time is equal to or less than the task deadline. In order to properly inference the satisfaction of the timing requirements, the WCET should be derived precisely. However, an accurate estimation of the WCET is hardware-dependant and it is difficult to derive.

In practice, there are two approaches to analyse WCET [135]: static and measurement-based analysis. Static analysis derives the WCET from the source code and an abstract processor model. It generates all possible control-flow paths from the source code, and finds the worst-case path based on the flow and processor behaviour. Measurement-based approaches, in contrast, measure the actual end-to-end execution times, and estimate the WCET based on these measurements. Static analysis is likely to overestimate the WCET, while measurement-based method may underestimate the WCET if the actual WCET path is not observed, or the worst-case state of the hardware was not experienced.

The analysis of the WCET is non-trivial since it is based on both the program control flow and hardware architecture features [135]. The initial state, input data and control-flow path that lead to the WCET is hard to derive. In addition, due to the advanced features of modern computer architecture, e.g., multi-level caches, pipelines, speculative features and out-of-order execution, the analysis of such an upper bound is becoming much more difficult. If the WCET is underestimated, the resultant scheduling system will be optimistic, in which case the system will occasionally suffer from transient overloads and is likely to have unpredictable behaviours. On the other hand, if a system is designed with too pessimistic estimations, the system resources will be wasted and effectively the functionalities that a system can implement have to be reduced. However, due to high correlation and coupling with other system-level elements and the environment, deriving the WCET of a task in CPS could be non-trivial. Hence a precise WCET should not be expected.

For most systems, more resources means higher hardware cost, larger physical size and more power consumption. It will therefore be too expensive to satisfy the worst-case, especially when some of the tasks have a large WCET but only a small average execution time. In this case, the system has to make less conservative assumptions of the workload, and as a consequence overloads will occasionally happen, which will lead to performance degradation. The term 'overload' refers to the condition when the total demand requested by the application tasks in the system is larger than the capacity that the hardware platform can support.

It is known that only the lower-priority tasks scheduled with FPS will suffer from overloads, leaving the higher-priority tasks unaffected. However, since priority is not always equivalent to importance, the system may be jeopardised if some critical but low-priority tasks fail. For EDF, a transient overload that makes a task to miss its deadline, has a chance to cause a cascade of deadline misses. Thus overloads have to be handled properly.

In general, there are two approaches to handle overloads [120]:

1) **Prevent temporal failures**: validate the system with worst-case analysis. Make sure the resources are sufficient enough to support even the worst-case. However, if the assumptions of the worst-case are violated (transient overload or resource failure), the guarantees may become invalid. Server technique can also be used to create temporal isolation between tasks.

2) **Tolerant timing faults**: assume overloads will happen and handle timing violations when they occur at run-time; Approaches include imprecise computation [95], QoS adjustment [14], elastic scheduling [40] [37] [41] [42], etc.

The reservation-based scheduling methods such as the Priority Exchange Server and the Constant Bandwidth Server that are already discussed in Section 2.1.3 can relieve some of the uncertainties by making pessimistic reservations, but the overall performance of a reservation-based server is still largely depend on a correct and fair allocation of resources [120]. CPSs are in nature more dynamic, and conservative reservations are not efficient. Since the variation in task execution time can be large, both optimistic and pessimistic allocations will decrease system performance or even lead to run-time failures. The CPU bandwidth should be dynamically allocated to each task. This requires scheduling decisions to be made online, not only for handling overloads, but also for an increased overall resource utilization.

To address these mentioned issues, many efforts have been made in recent years to use online mechanisms, that can improve scheduling for tasks with high level of uncertainties in systems including CPS. The term ‘online’ refers to the process of run-time system monitoring, resource planning and reallocation. To provide predictable performance and improved system utilization even under uncertain or unknown task parameters, these methods rely on task models with a higher level of flexibility compared to traditional task models, in which some of the task parameters are defined in a feasible range (or a collection of feasible values) instead of a fixed single value.

The *Elastic Scheduling* is one of these methods which is an online adaptive task management policy [37]. In an elastic task model, the period T_i of a task τ_i is defined in a feasible range. The utility of the task is a function of its period which decreases as the period increases. A system under elastic scheduling is analogous to a physical spring system: the periods of tasks can be adjusted to adapt to the current workload just as a chain of springs can adapt their lengths in reaction to external forces.

The *imprecise computation model* [95], which is one type of anytime algorithms, is another approach to make good use of spare processor capacity and to handle overload conditions at run-time. In such a model, a task is split into a mandatory subtask which must be done, and an optional subtask which is only executed if there is enough spare capacity.

The notion of *Quality-of-Service* (QoS) was initially proposed as a metric in soft real-time systems for measuring performance. A task under different QoS levels has a corresponding execution time, period and a contribution (e.g., value or reward) to the system. A system can make run-time trade-offs between QoS and the requested resource demands, which is achieved by searching the optimal solution of selecting QoS levels that will maximise the total system value under the current load condition.

All of the aforementioned methods have assumptions that some of the task parameters are known a priori. When it comes to open and dynamic environments, the parameters of the tasks are not known until they arrive. *Feedback Scheduling* is one method proposed to achieve desired performance in a dynamic environment. Feedback scheduling uses measurements observed from the system and feeds that information to the scheduler to make adaptations. The advantage of using feedback is that it can deal with systems in which tasks have highly dynamic execution times and inter-arrival patterns. There are also some work using control-theoretic approaches, named *Feedback Control Scheduling* (FCS) [46] [97] [100], which takes the benefit of the modelling and verification methods from Control Theory. FCS is a more systematic method for applying feedback. It provides a way for systems to adapt themselves in the presence of noise and uncertainties while still provide a predictable performance. For example, a method of using feedback control with an admission controller under EDF (FCS-EDF) is discussed in [98].

Stochastic analysis is another method that deals with tasks with arbitrary execution times and arrival rates [120]. Statistical approaches are used to observe the execution times of tasks and build a probabilistic density function for each task. A probabilistic analysis is then given to indicate the confidence of the schedulability of the task set.

The rest of this review will give details of the aforementioned algorithms and discusses their advantages and limitations. Most of these algorithms can also be seen as adaptation methods that can be used to make online trade-offs between system utilization and timing predictability.

2.3 Flexible Scheduling Algorithms

The actual task execution times at run-time have variations and even the observed maximum execution time could still be much less than the WCET. Hence the use of WCET in the design stage will lead to very inefficient usage of resources at run-time, i.e., a very low utilization in general. Since the basic element of a CPS is an embedded computer, which often has cost and size constraints, the feasible resources in the system may not be sufficient to support all the extreme scenarios. This means there is a potential chance that the system will occasionally be overloaded.

To solve the conflicts between performance and timing guarantees, many new paradigms of task scheduling are proposed, which will collectively be referred to as flexible scheduling [57]. Flexible scheduling in this review refers to a category of scheduling schemes that use a task model with a certain level of flexibility. Unlike traditional algorithms where task parameters are fixed, flexible scheduling permits some properties of a task to be defined in a feasible range or some discrete values. In some cases, additional artificial parameters (e.g., value, QoS level) are added to the task model to facilitate the selection of parameters. The benefit of using a flexible model is this makes it possible for the scheduler to adapt task parameters to the current workload in order to achieve a higher resource utilization, while still ensuring a certain level of timing guarantee. In this section, flexible scheduling algorithms in the literature will be reviewed and their performance will be discussed.

2.3.1 Elastic Scheduling

Elastic scheduling is proposed as a methodology for adapting workload by varying the rates of a subset of periodic tasks [40] [37] [41] [42]. Although there exists similar rate modulation techniques [22] [119], the elastic scheduling method is more systematic and general. In this framework, each periodic task has a range of feasible periods and is treated as analogous to a spring with an elastic coefficient and length constraints. The rate adaptation algorithm is executed periodically to enlarge or compress task periods according to the current estimated execution times. The possibility of varying rates increases the flexibility of the system. The advantage is twofold: a) in overload conditions, this reduces the number of deadline misses; b) whereas in normal conditions the algorithm increases the CPU utilization and improves QoS.

Elastic Task Model

The tasks in the elastic scheduling framework are similar to flexible springs. Each task is given a range of feasible periods $[T_{i_0}, T_{i_{max}}]$ and a non-negative elastic coefficient E_i. It is also assumed that the upper bound on execution time of each task is known and is denoted as C_i^{ub}. Thus, an elastic task is described as:

$$\tau_i \equiv (C_i^{ub}, T_{i_0}, T_{i_{max}}, E_i)$$

where T_{i_0} is the minimum feasible period and is considered as the nominal period. The maximum period of task τ_i that can be accepted is defined as $T_{i_{max}}$. The actual period T_i is constrained to be in the range $[T_{i_0}, T_{i_{max}}]$. The elastic coefficient E_i is equivalent to the rigidity coefficient of a spring, which is used to determine the relative variability of a task period. For tasks which have a fixed activation rate during load reconfiguration, E_i should be set to 0. While for other tasks whose rate could have some elasticity, E_i can be set inversely proportional to the importance of task [41].

In this framework, a period reconfiguration can be achieved by stretching and compressing task periods for all tasks with changeable periods. A feasible solution exists if the total utilization of the new schedule is less or equal to the required utilization bound. For the case when EDF scheduling is used, all tasks can be guaranteed with their minimum periods if $\sum(C_i^{ub}/T_{i_0}) \leq 1$. Otherwise the elastic algorithm is used to calculate the new assignment of task periods T_i, which satisfies $\sum(C_i/T_i) = U_d$. Here C_i is the worst-case execution time and U_d is the desired utilization ($U_d \leq 1$).

The elastic task model is reasonable for a range of applications. For example, in a control system, the sampling and actuating rate of the controller is not always rigid. A higher execution rate is useful to obtain more precise estimations of current states and to make faster reactions to external stimuli. On the other hand, decreasing a controller's frequency is likely to have a consequence of decreased control performance, but the amount of resources needed is also reduced.

Task Compression and Decompression

The compression of elastic tasks is equivalent to compression of a linear spring system. In this comparison, the utilization factor $U_i = C_i/T_i$ of a task is equivalent to the length x_i of a spring. A set of n tasks with total utilization $U_p = \sum_{i=1}^{n} U_i$, is treated as a sequence of n springs with length $L = \sum_{i=1}^{n} x_i$.

Figure 2.1 shows the situation when a linear spring is compressed from L_0 to L' by applying a force F. Define the set Γ_v of variable springs and the set Γ_f to include all the springs that cannot be compressed any more. The new length of each spring x_i can be derived by [41]:

$$\forall S_i \in \Gamma_v \quad x_i = x_{i_0} - (L_{v_0} - L_d + L_f)\frac{K_v}{k_i} \tag{2.6}$$

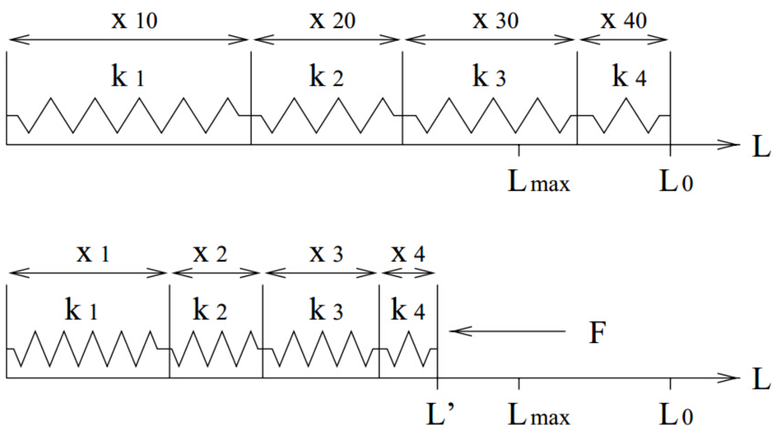


Figure 2.1: A linear spring system compressed by a force F

where

$$L_{v_0} = \sum_{S_i \in \Gamma_v} x_{i_0} \tag{2.7}$$

$$L_f = \sum_{S_i \in \Gamma_f} x_{i_{min}} \tag{2.8}$$

$$K_v = \frac{1}{\sum_{S_i \in \Gamma_v} \frac{1}{k_i}} \tag{2.9}$$

To apply Equations (2.6)(2.7)(2.8)(2.9) to elastic tasks, substitute all length parameters with utilization factors, and replace rigidity coefficients K_i and K_v with elastic coefficients E_i and E_v. Let Γ_f be the subset with all fixed tasks that already use the maximum periods, and Γ_v be the subset of all variable tasks that can still be compressed. The new compression equation of each task to achieving an overall desired utilization U_d is derived as:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_0} - (U_{v_0} - U_d + U_f)\frac{E_i}{E_v} \tag{2.10}$$

where

$$U_{v_0} = \sum_{\tau_i \in \Gamma_v} U_{i_0} \tag{2.11}$$

$$U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{min}} \tag{2.12}$$

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i \tag{2.13}$$

During utilization compression, task periods are increased. It is noticeable that if the solution of U_i is less than the lower utilization bound U_{min}, then the period of that task should be equal to the maximum period T_{max}, and the task will be removed from Γ_v and put into Γ_f. If a feasible solution exists, the period T_i of task τ_i is then derived as C_i/U_i. It was shown in [41] that when period constraints exist, the compression algorithm has a worst-case complexity of $O(n^2)$ to find the solution of U_i.

Decompression can be performed once the overload is over, and all compressed tasks can return toward their nominal periods. The new task utilization can be expanded (task periods are shortened) according to the task elastic coefficient, and a similar algorithm is used to recalculate utilization allocations.

Online Adaptation

In previous sections, it is assumed that the WCETs of all tasks are known a priori. The case when the task computation times are unknown is discussed in [37]. In this work the actual execution time is monitored and an estimated value is used as a feedback for achieving load adaptation. As illustrated in Figure 2.2, the actual execution time is monitored by the kernel, and the mean execution time over a sampling window $\hat{c_i}$ and the maximum execution time $\hat{C_i}$ is updated each period. The estimated execution time Q_i is calculated as:

$$Q_i = \hat{c_i} + k(\hat{C_i} - \hat{c_i}) \tag{2.14}$$

where $k_i \in [0, 1]$ is the guarantee factor. A larger value of k_i will decrease the weight of the mean execution time. When $k_i = 1$, the estimated execution time is equal to the WCET. As k_i decrease, the actual system utilization will increase, at the cost of more deadline misses. Once Q_i is obtained, the utilization factor can be calculated as $\hat{U_i} = Q_i/T_i$, and the actual load is estimated as:

$$U_a = \sum_{i=1}^{n} \hat{U_i} \tag{2.15}$$

The objective of the feedback loop is to make the actual utilization U_a as close as possible to the desired utilization U_d. This U_d is set close to (but less than) 1.0, to permit variations in task execution. The advantage of using an online estimation is that no prior information about execution times is needed. However, since the establishment of Q_i may take several task periods, there is

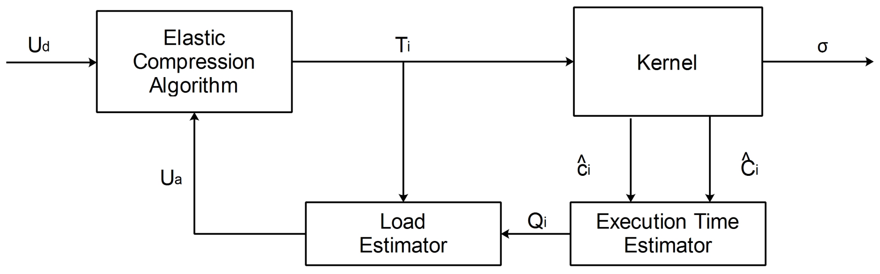


Figure 2.2: Elastic scheduling with execution time estimation

a chance that some deadline misses may occur during the start-up time. This can be improved by providing c_i to the kernel as the initial value, which will speed up the convergence time of Q_i to a stable value [37].

Other Extensions

An extension that deals with resource constraints is introduced in [42], and the use of damping coefficient to achieve smooth adaptation can be found in [38]. In Chantem's work [51], the elastic model is applied for tasks with $D < T$ and is generalised to adjust deadlines instead of task periods. An example of using the elastic model for transmitting network packets over FTT-Ethernet, which is a real-time master/multi-slave transmission protocol, is given by Pedreiras [110].

2.3.2 Quality-of-Service Adaptation

Services in real-time systems often have performance requirements that can be interpreted as quality-of-service (QoS). For example, QoS in a multimedia application may refer to frame rate per second or frame resolution; the QoS level in real-time communication may specify the permitted delay and bandwidth requirement. By defining different acceptable QoS levels that the service can provide, it is possible to adjust the QoS level of a service according to the availability of current resources. Graceful degradation can be achieved if one or more services can decrease their QoS as in the case of unanticipated overloads and failures. System utility can thus be maximised by reallocating resources at run-time and a balance can be made between predictability and performance.

In [14], a QoS-negotiation model is proposed to ensure graceful degradation when prior assumptions are violated and overload occurs. The proposed

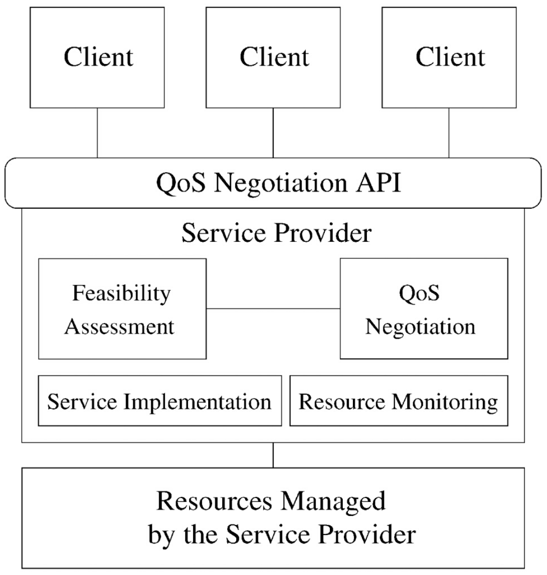


Figure 2.3: QoS Negotiation Service Provider

mechanism, as shown in Figure 2.3, permits clients to request a spectrum of acceptable QoS levels and provide utility associated with each level when a request is sent to the service provider. It is declared that the QoS negotiation model extends the typical real-time services in two ways: a) QoS degradation is an alternative to denying service, which could potentially reduce the number of service denies and improves the overall system utility; b) Compared with other approaches to achieving graceful degradation, QoS negotiation uses more generic application-specific knowledge to adapt service qualities.

In this QoS-negotiation model, the client requesting service expresses a set of negotiation options and the penalty of rejecting the request from the service provider. Each option is associated with a QoS level and a reward value. Different QoS levels represent distinct configuration of parameters, of which the semantics only need to be known by the client and the provider. The reward represents the application-perceived utility that can be gained from serving the client with the corresponding QoS level, which is a measurement of benefits that can be contributed by accepting the request. The rejection penalty defines the amount of penalty when a client's request is rejected. Once the request is accepted, the rejection penalty is considered no further.

The feasibility assessment module is responsible for checking if the current QoS configuration can be maintained by the available resources, which is monitored by the resource monitoring module. The QoS negotiation module uses a local optimization heuristic to select a proper QoS level of each client, in a way that could maximise system utility [14]. When a new task arrives, the heuristic will recompute the set of QoS levels to maximise the sum of clients' rewards. The acceptance of the new task will be only guaranteed if the penalty of rejection is larger than the resultant degradation of all other local clients.

2.3.3 Value-based Scheduling

Value-based scheduling [39] has drawn wide attention from the real-time community. In this scheme, the concept of *value* is introduced to increase scheduling flexibility, especially under overload conditions. The value parameter represents the relative benefit that can be contributed to the system when a task finishes its computation. It is useful to support the scheduler in making run-time decisions when resources are scarce. The value can be either a fixed value, or a function. It should be noted that 'value' is an approximated utility and is an artificial description of the importance derived from the application-specific domain. Thus a correct assignment of values is critical to implement this method.

The value-based scheme takes the consideration that not all tasks/services have the same utility. When an overload occurs, instead of dropping tasks with lower priorities, or serving tasks in a first-come-first-serve manner, it is more desirable to drop some less valued tasks to achieve a graceful degradation. In this way, value can be seen as the preference between different services/groups of services.

Generally there are two approaches to schedule tasks with value: a) *High Value First* (HVF): tasks are scheduled in the preference of value v_i; b) *High Value Density First* (HVDF): priorities of tasks are decided based on the value density v_i/c_i. A comparison between different value-based schemes with EDF and their mixture is studied in [39]. It is observed that when there is no guarantee mechanism, HVDF is the most effective priority assignment policy in overload conditions. Tres et. al. [132] developed a scheduling algorithm DMB (*Dynamic Misses Based*) which assigns priorities based on the original task value V_i and the percentage of timing faults MD_i:

$$P_i = V_i \times (k_1 + k_2 \times MD_i) \tag{2.16}$$

where P_i is the priority, and k_1, k_2 are weighting constants. When V_i is static and there is no timing faults, DMB behaviours exactly the same as HVDF. A comprehensive survey of value-based scheduling is given in [34], where algorithms using constant and variable values are discussed and compared.

The actual assignment of value is a critical procedure in developing value-based scheduling algorithms, yet it remains a less addressed issue in the literature. In the work of Burns et al. [34], the *representation* and *construction* problem of value is discussed. The representation problem is to answer the question that if a specific form of value (or value function) is theoretically sound to support the decision procedure. It is required that the *axioms* (conditions that are fulfilled by binary preference relations) be 'testable' or 'empirically verifiable'. It is suggested that the construction of values that specify the preference between service can be built by making pair-wise comparisons, in which users are asked to make their judgements between every pair of services alternatives. The quality of the pair-wise comparison experiment is strongly based on the way the experiment itself is represented to the decision maker. Although the inconsistency of preferences can be checked by tools, there is no guarantee that the result of such an experiment is an accurate description of the real utility of the services. Overall, the assignment of value inevitably depends on intuition and judgement, thus tool support should be used to ensure a fair and consistent assignment.

2.3.4 Imprecise Computation

Imprecise computation [95] [91] is another approach to avoid timing faults and achieve graceful degradation. In this model, every time-critical task τ_i can be logically decomposed into two sub-parts: 1) a *mandatory* sub-task M_i and 2) an *optional* sub-task O_i. The mandatory task is the sub-part that has to be completed before the deadline to obtain an acceptable result; while the optional tasks is the one that can improve the precision of result or the QoS of services if finished. The result is said to be 'imprecise' if the task terminates after the mandatory sub-task is finished. An imprecise computation can be *monotone* if the quality of the result increases as it executes longer. Examples that support monotone computation include statistical estimation and prediction, and heuristic searching for optimal solution, etc.

The key to scheduling imprecise tasks is to ensure all mandatory sub-tasks have bounded processing time requirements and have sufficient computational resources to complete their execution before the deadlines. Whenever there is unused processing capacity, optional sub-tasks can be scheduled and executed to improve the quality of the result. Scheduling problems in supporting imprecise computation is discussed in [95]. Some performance metrics that can be used for comparing different imprecise algorithms are listed below:

- Minimizing the total error, the maximum error or the average error;
- Minimization of the number of dropped optional sub-tasks;
- Minimizing the number of tasks that complete or terminate after their deadlines;
- Minimize average response time.

A solution to scheduling imprecise tasks with minimised total error is also introduced in the same paper [95], which uses a modified version of EDF. A programming language that supports implementation of imprecise computation, *flex*, is discussed in [91]. The support of imprecise model in real-time operating system that uses an *Imprecise Computation Environment* (ICE) is given in [83].

2.3.5 Summary

In this section, we first introduced a few methods that can be used to handle timing faults in dynamic real-time systems. The elastic scheduling method is first introduced as an effective way of overcoming overloads and improving resource utilization. The algorithm is derived from a mechanical spring system and has a complexity of $O(n^2)$ when utilization constraints are imposed.

The QoS method permits services to have different levels of quality that are provided to clients. Each QoS level is associated with different computation time and contribution to the system. In some sense this method is similar to the multiple versions approach. The scheduler can choose the quality level to accommodate current available resources, while still achieving an overall acceptable performance when there are not sufficient resources. Value-based scheduling is then discussed, which adds an additional parameter, task value, to indicate the preference between service alternatives. It can be inferred that the assignment of value is the key to achieving fair scheduling between tasks. Thus if the value is assigned in an inconsistent way, it is likely that an undesired decision will be made, which is not in accordance with the actual utility of services.

Finally, the imprecise computation model that enables part execution of a task is introduced and discussed. The general idea of the methods discussed here is to improve the system utility under insufficient resources by adding flexibility to the task model. By allowing a task to have different computation times, service levels or periods, it becomes possible for the scheduler to make optimal decisions and trade-offs, and to avoid rejecting tasks if the minimal requirement of the task can be accommodated by reducing the computation quality of itself/other tasks.

A pitfall in common with all these algorithms lies in that the performance of these methods is largely dependent on the way the flexible parameters are specified. For example, in elastic scheduling, the determination of E_i will largely change the way periods are changed; and in the QoS negotiation model, the relative value of contribution between service alternatives will have a great influence on the order of task's degradation. Thus sound engineering theory and empirical results that are application-specific should be used to obtain a reasonable performance if flexible scheduling is used. In the next section, we will examine a novel method that uses ideas from control theory in scheduling real-time tasks.

2.4 Feedback Scheduling

Feedback scheduling [46] [97] [100] [98] is another category of scheduling methods that is designed for handling systems with high dynamic workload with large uncertainties. Unlike execution time servers that pre-allocate resources, or flexible scheduling methods in which a reconfiguration is triggered by overload events, feedback scheduling is based on the *feedback regulation* that actively manipulate task parameters, in order to achieve desired performance even in the presence of uncertainties.

There are many real-time computer systems in which the execution times of tasks are data-dependent, e.g., web-server, database and video decoding. Also there are cases where the executions of tasks are inherently non-deterministic. An example is the heuristic for finding optimised path for an autonomous vehicle. The computation time of the heuristic is dependent not only on the destination which could possibly be changed any time by the mission planner, but also on the environmental conditions that the vehicle currently senses. Feedback can be used to adjust the system during run-time to overcome these uncertainties that cannot be predicted before execution. This is achieved by *observers* that sense the current status, *controllers* that make a control decision, and *software actuators* that manipulate parameters in the system. In this section, the construction of feedback loops and some example applications of feedback scheduling systems will be discussed.

2.4.1 Feedback Control and Its Application to Computer Systems

The general principle of the feedback mechanism is to continuously monitor system state(s), compare with desired reference(s) and make corresponding control actions that could decrease the error, i.e., the difference between the current and the desired state [13]. Since the mid-1950s, feedback control has been widely used in computer-controlled systems [23]. The plants being controlled are normally physical and mechanical systems, such as wind turbines, water control valves, mobile robots and aircraft control systems. For the case of feedback scheduling, the controlled plant is the task scheduling system.

For general purpose computer systems, the most commonly monitored state is the average performance. This is often achieved by a 'best-effort' scheduler and no guarantees can be made on performance and service quality. However, there are complex computer applications in which explicit quality-of-service is required in delivering their services. In these applications, feedback control can be applied to achieve desired performance, even in the presence of uncertainties.

Figure 2.4 shows the basic structure of a feedback-control computer system. The *Performance Sensor* is a monitor that senses the current status of internal variables of interest, such as CPU utilization, network bandwidth or system workload. The variables that are being monitored are normally identical to the parameters that need to be controlled. The measured value is then compared with the desired value to generate an 'error' signal, which is used as the input to the *Performance Controller*. The role of the controller is to calculate the amount of adjustments that needs to be made, based on the error and a specific control algorithm. Finally, the *Software Actuator* will accept the control signal and manipulate scheduling variables, e.g., queue size, task parameters, scheduling policies or CPU frequency/voltage, that can eliminate the error.

There are two categories of control problems: *regulation* and *reference*

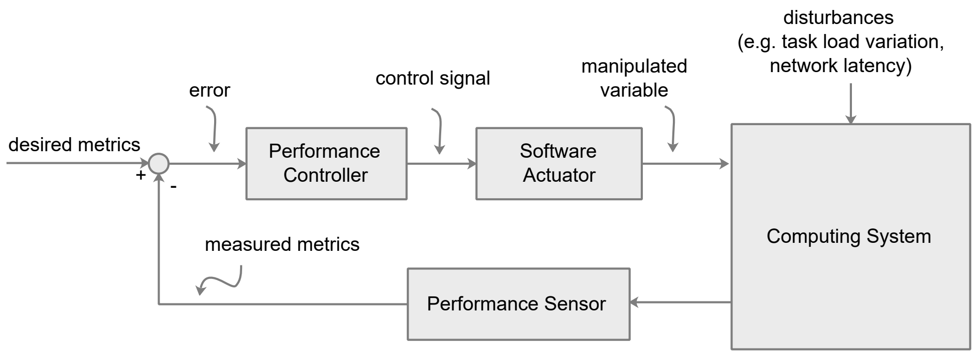


Figure 2.4: Structure of a feedback-control computer system and its components

tracking. The regulation problem focuses on maintaining the controlled variable to a specific value, i.e. the desired metric in Figure 2.4 is fixed. For example, for a wireless router, it may be required that the bandwidth it provides to each client should be maintained to a fixed fraction of the total bandwidth. If any connection exceeds this limitation, the feedback controller will regulate its bandwidth to its permitted value. While for the reference tracking problem in which the desired value is varying with time, the controller is designed to follow the reference as fast as possible. Most of feedback scheduling systems are regulation controllers, in which the desired metric is calculated through an optimal distribution of resources.

General principles and applications of feedback in computing systems can be found in the book of Hellerstein [79]. Information on feedback control theory, including modelling, controller design and stability issues, can be found in the textbook by Dorf and Bishop [62]. Some of these issues are beyond the scope of this review and hence will not be discussed here.

2.4.2 Research in Feedback Scheduling

Feedback scheduling is a 'closed-loop' method, which is in contrast to FPS and EDF that are 'open loop'. There are many research efforts in the literature where feedback is applied to real-time scheduling systems. The first one is proposed by Lu and Stankovic in 1999, which extended the traditional EDF by integrating a PID controller [98]. The proposed scheduling method, *Feedback Control EDF* (FC-EDF), monitors the deadline miss ratio and controls the admission of tasks and the service levels of admitted tasks. Stankovic et al. [127]

proposed a two-level feedback scheduling framework for distributed systems. This framework consists of two feedback controllers: a distributed (global) and a local feedback controller, and these two controllers work hierarchically.

Steere et al. [128] described a feedback scheduling method for real-rate applications, which allocates CPU proportions and rates to tasks according to the estimations of their current progress. The progress metrics of a real-rate task are expressed by the current fill-level of the input/output queue divided by the queue size. The adjustment of the CPU allocation is then calculated by a PID controller based on the estimated progress.

A control-based middleware framework is presented in [90]. The framework uses control theory to model the dynamics of the QoS adaptation and is deployed in a distributed visual tracking application. A similar middleware that supports QoS-control, the ControlWare, was proposed by Zhang in 2002 [142]. The middleware offers a control theoretic paradigm for achieving absolute/relative guarantees, and the solution of the prioritisation problem in distributed environments.

Approaches that exploit *Dynamic Voltage/Frequency Scaling* (DVFS) are discussed in [145] [101]. In Lu's work [101], a formal feedback-control DVS is developed and evaluated in a simulated multimedia system which decodes and plays MPEG video. The objective of the controller is to maintain a stable throughput, while holding the CPU at its minimum possible operating frequency to save battery energy. In this case, the CPU operating frequency/-voltage is scaled based on the average frame delay over a certain window size. In Zhu and Mueller's work [145], the integration of a DVS scheduler and a feedback controller has been achieved within EDF scheduling, targeting hard real-time systems with dynamic workloads.

Lu, Stankovic and Son etal. summarised a systematic way to design feedback control scheduling (FCS) systems that satisfies transient and steady-state performance specifications [99]. In their work, design issues such as dynamic modelling and performance evaluation are addressed, and categories of real-time applications are identified.

Other works that applied feedback in real-time systems include *End-to-end Utilization CONtrol* (EUCON) [100], web server delay control [97], feedback-feedforward scheduling that supports period adjustments in control applications [46], and feedback admission control [63]. To better understand the internals of a feedback scheduling system, two of the aforementioned works will be considered in detail: FC-EDF [98] and feedback-feedforward scheduling [46]. These topics are selected because they are well presented the scenarios in which feedback control can be applied. FC-EDF extends traditional EDF with a PID controller. The deadline miss ratio is selected as the controlled variable. In the work of feedback-feedforward scheduling, a detailed design methodology is represented. This work shows how application-specific information (control cost in this case) can be used in controller design. The topics mentioned here can be easily extended to cover more general cases and be adopted for new applications.

2.4.3 The FC-EDF Scheme

In Lu's work, a PID controller is integrated with an EDF scheduler based on the feedback control framework. The proposed architecture, which is known as *feedback control EDF* (FC-EDF), consists of a PID controller, a service level controller, an EDF scheduler and an admission controller (as shown in Figure 2.5) [98]. The system uses *deadline miss ratio* (DMR) as the controlled variable, which represents the fraction of tasks missed their deadlines among all admitted tasks. The feedback loop works as follows:

1. The DMR is monitored periodically;

2. The DMR is compared with the set point to generate an error;

3. The controller computes the required amount of control based on the error and applies control law;

4. The actuator changes the value of the manipulated variables (in this case is the service levels and the admissions of tasks) to control the system.

The objective of the controller is to achieve a high utilization and system throughput. To satisfy this objective, a small (but non-zero) DMR for admitted tasks is chosen as the set point. The reason for not choosing a zero DMR is that a system with no deadline misses is likely to ignore the secondary performance index [98], i.e., a high system utilization. A system with DMR equals to 0% could still have an extremely low utilization, which will be treated as a normal state by the controller if the reference point is set to be zero. In contrast, a set point of a non-zero DMR will always force the controller to slightly overload the system in order to achieve a high utilization. There is hardly any possibility that a system can achieve 100% utilization with no deadline misses all the time, especially in an unpredictable environment.

At this point, it can be seen that a trade-off between deadline miss ratio and utilization is unavoidable. To increase the resource usage, sacrifices have to be made and hard guarantees can no longer be held. For this reason, the method introduced here is only suitable for scheduling tasks without hard timing requirements.

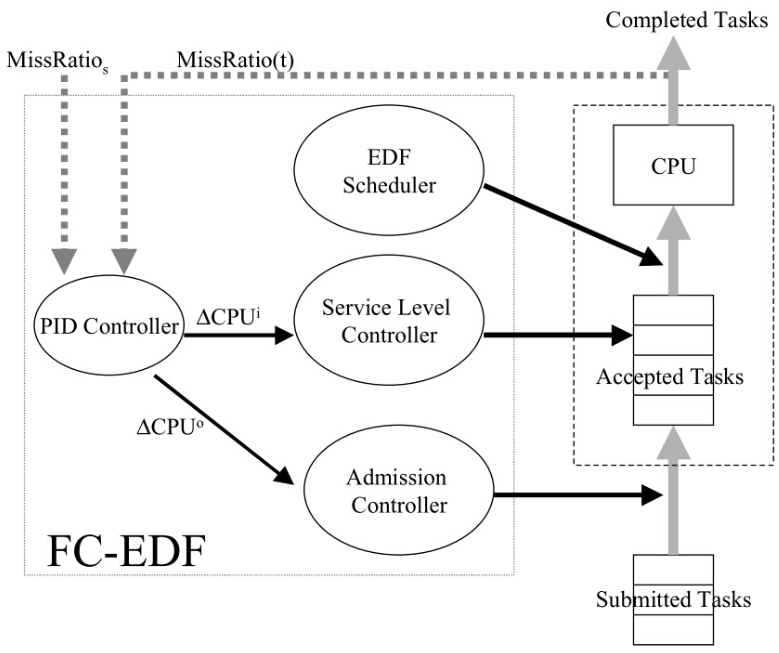


Figure 2.5: An overview structure of FC-EDF

Task Model

The task model used in FC-EDF is similar to the imprecise computation model. In this framework, each task is described as a tuple: $\tau_i \equiv (I_i, \mathrm{ET_i}, \mathrm{VAL_i}, \mathrm{S_i}, \mathrm{D_i})$. Each task is implemented with one or more logical versions I_i, which can be expressed in different forms, e.g., multiple version, sieve function or milestone method [95]. Each version is called a service level and is associated with different nominal execution times $\mathrm{ET_i} = \{\mathrm{ET_{i1}}, \mathrm{ET_{i2}}, \ldots \mathrm{ET_{ik}}\}$ and values $\mathrm{VAL_i} = \{\mathrm{VAL_{i1}}, \mathrm{VAL_{i2}}, \ldots \mathrm{VAL_{ik}}\}$. The execution time is described in the form of requested utilization, e.g., $\mathrm{ET_{ik}} = 0.02$ means the kth version of task τ_i requires 0.02% of the CPU time. Finally, S_i represents the starting time and D_i is the soft deadline of task τ_i.

PID Control

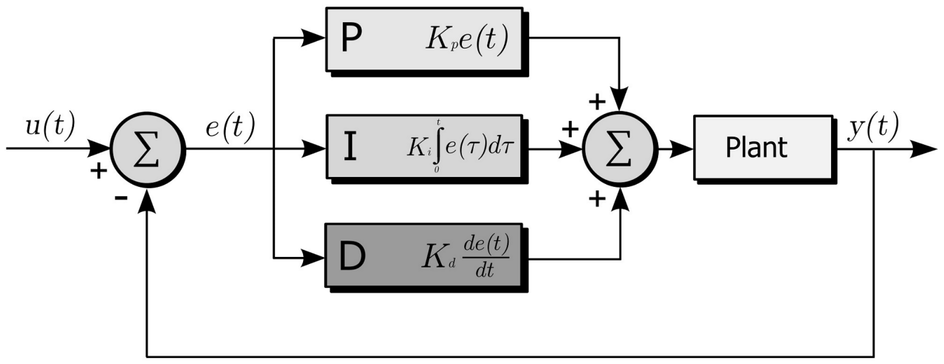


Figure 2.6: Block diagram that shows the basic structure of a feedback loop with a PID Controller and a plant being controlled.

The Proportional-Integral-Differential (PID) controller is widely used in control systems. A graphical representation of PID controller is shown in Figure 2.6, and its mathematical form in continuous time is written as [62]:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt}e(t) \tag{2.17}$$

which can also be transformed into frequency domain (s-domain):

$$U(s) = K_p E(s) + K_i \frac{1}{s} E(s) + K_d s E(s) \tag{2.18}$$

where e(t) or E(s) is the error; u(t) or U(s) is the controller output; K_p, K_i and K_d are constant coefficients, and should be tuned for a specific application in order to get a stable and satisfactory performance.

In FC-EDF, since the sampling and actuating are taken place periodically in discrete times, Equation (2.17) is discretized into the following formula:

$$\Delta\mathrm{CPU(kT)} = \mathrm{K_p e(kT)} + \mathrm{K_i} \sum_{\mathrm{kT \in IW}} \mathrm{e(kT)} + \mathrm{K_d} \frac{\mathrm{e(kT) - e(kT - DW)}}{\mathrm{DW}} \tag{2.19}$$

where T is the sampling period; ΔCPU is the changes in total requested CPU utilization, which is the manipulated variable; $e(kT)$ is the error, which is defined as the desired deadline miss ratio DMR_s minus the measured DMR at $t = kT$, i.e., $e(kT) = DMR_s - DMR(kT)$; the parameters IW and DW are the integration and derivative window, respectively.

Equivalently, what this controller does is to map the DMR into the changes of requested utilization, in a way that will drive the DMR back to its desired set point. If $\Delta CPU(kT) < 0$, the requested utilization should be decreased and vice versa. Once the change in requested utilization is calculated, the PID controller will call the Service Level Controller (SLC) and the Admission Controller (AC) to change the utilization to $CPU((k-1)T) + \Delta CPU(kT)$. The SLC will try to accommodate the desired utilization by adjusting the service levels of accepted tasks, and returning ΔCPU^0, the portion of ΔCPU that cannot be accommodated. The ΔCPU^0 is then used by the AC to limit the capacity that can be used by admitting new tasks. If ΔCPU^0 becomes negative, no new tasks will be admitted until the system load has been reduced.

FC-EDF is an early work on exploring the use of feedback control in real-time scheduling systems. It uses a feedback PID controller to adjust the deadline miss ratio of admitted tasks. An analytical model of the scheduling is also introduced to facilitate the tuning of controller parameters K_p, K_i and K_d. The authors showed the method can effectively adapt to the changes in system workload and maintain satisfactory performance, while the original EDF performs poorly in handling the same workload [98].

2.4.4 The FBFW-EDF Scheme

Cervin et. al. showed a feedback-feedforward scheduling architecture in their work [46]. The scheduler in this framework uses feedback from execution times as well as feedforward from mode changes to adjust the rates of tasks. The objective is to optimise the *Quality-of-Control* (QoC), which is defined in terms of a set of cost functions that describe the quality of a real-time controller.

From scheduling perspective, controllers are generally views as tasks with fixed periods, hard deadlines and known WCETs by the real-time community. In some situations it is possible to design a controller with different modes. Thus the controller can switch between modes with different execution times and sampling periods, according to the current workload in order to optimise QoC. The WCET assumption can also be relaxed by using an estimation based on actual execution time measurements. The relaxation of control task parameters provides the opportunity of applying scheduling algorithms in which a feedback regulator can be used.

However, online parameter adjustment of control tasks have an influence on the controller performance, which has to be taken into consideration. In the following sections, the modelling, cost function formulation, and finally the construction of feedback-feedforward loop will be discussed.

Modelling of the Controlled Plant

To formally apply control theory and to analysis control performance, a mathematical representation of the process being controlled (or plant) is essential. Each physical process can be described by a set of linear differential equations that reveals the relationship between control inputs and system outputs (or measured signals). A standard state-space representation [13] of a controlled plant is:

$$\begin{cases} \dfrac{dx(t)}{dt} = Ax(t) + Bu(t) + w(t) \\ y(t) = Cx(t) + v(t) \end{cases} \tag{2.20}$$

where $x(t)$ is the vector of system state variables; $u(t)$ is the control input; $y(t)$ is the system output that can be measured; $w(t)$ and $v(t)$ are white noises assumed to have a Gaussian distribution with a zero mean; A, B and C are constant matrices that describe the dynamics of the plant. The objective of the controller is to keep the system states close to the desired values. As the controller is implemented on a computer in discrete time, the sampling of $y(t)$ is performed periodically through an analog-to-digital converter (ADC), and the control input $u(t)$ is held by a zero-order hold via a digital-to-analog converter (DAC).

Formulation of the Cost Function

The performance of a controller can be evaluated by setting time, steady-state error and overshoot, etc [62]. To numerically evaluate the performance, they used a quadratic criterion [46], which is formulated as:

$$J = \lim_{T\to\infty} \frac{1}{T} E\Big\{ \int_0^T \big(x^T(t)Q_1x(t) + u^T(t)Q_2u(t)\big)dt \Big\} \tag{2.21}$$

From this equation, it can be seen the cost J is related to plant state $x(t)$ and control input signal $u(t)$, where $E\{\cdot\}$ represents expectation and matrices Q_1 and Q_2 are constant weights. A higher value of J indicates a larger control derivation and control efforts. Hence by attempting to optimise the cost, it actually means to minimise the weighted sum of $x(t)$ and $u(t)$. To achieve the optimization objective, a *Linear-Quadratic-Gaussian* (LQG) controller is applied in the feedback loop, which is explicitly designed to minimise the above quadratic cost function Equation (2.21).

The period of the control task, h, has an influence on $x(t)$, $u(t)$ and $y(t)$, which makes J a function of h. Overall, the feedback scheduler should select the sampling period h_i for each control task τ_i, which will optimise the summed overall control performance:

$$\min_h J(h) = \sum_{i=1}^{n} J_i(h_i) \tag{2.22}$$

subject to the utilization constraint:

$$\sum_{i=1}^{n} C_i/h_i \leq U_{sp} \tag{2.23}$$

where C_i is the average computation time of control task τ_i and U_{sp} is the desired processor utilization set point. Evaluating a cost function involves a large amount of computation and it is time consuming. To solve the cost function at run-time, a quadratic approximation is used to simplify the computation [46]:

$$J_i(h_i) = \alpha_i + \beta_i {h_i}^2 \tag{2.24}$$

$J(h)$ can also be approximated by a set of linear functions, if the change in h_i is sufficiently small:

$$J_i(h_i) = \alpha_i + \gamma_i h_i \tag{2.25}$$

An explicit solution can be obtained by applying Kuhn-Tucker conditions. It is further shown in [46] that a simple rescaling of the nominal frequency $f_{i_0}(= 1/h_{i_0})$ is optimal with respect to J, if the nominal frequency is chosen proportional to:

a. $(\beta_i/C_i)^{1/3}$, for a quadratic approximation, or

b. $(\gamma_i/C_i)^{1/2}$, for a linear cost approximation.

this implies that the ratio between optimal sampling periods is constant regardless of the available resources. Once the nominal sampling periods are chosen wisely, optimal feedback scheduling can be performed through a simple period rescaling.

The Feedback-Feedforward Scheduling Architecture

A feedback-feedforward scheduling structure is developed as shown in Figure 2.7, which supports optimising control quality under mode changes. The control tasks in this framework are assumed to have multiple modes with different execution times. The feedback scheduler receives feedforward information from control tasks when a mode change occurs. The mode change here means change of task sets due to system reconfiguration. In this way, the scheduler can respond to mode changes actively and adapts scheduling parameters of control tasks according to current modes. The processor utilization is fed back to the scheduler, and compared with the desired utilization U_{sp}. The scheduler will attempt to keep the deviation between the measured utilization U and U_{sp} as small as possible, which is done by rescaling the sampling periods $\{h_1, h_2, ..., h_n\}$, as discussed in the previous section.

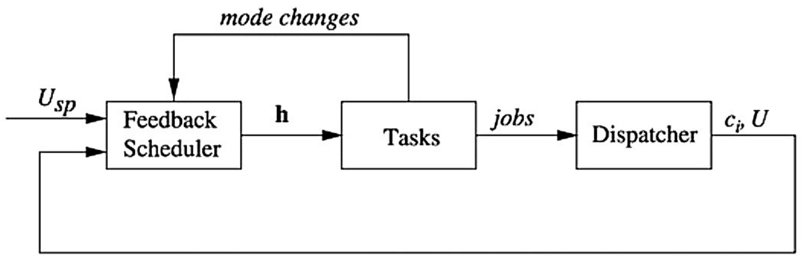


Figure 2.7: Block diagram of the feedback-feedforward structure

The set point is selected based on the scheduling policy of the dispatcher and the robustness of the controllers against deadline misses. Increasing U_{sp} will lead to a higher processor utilization and control performance, but may introduce occasionally temporal overruns. However, no analytic approach for selecting the set point is described by the authors. Hence ad-hoc trials should be applied for obtaining a reasonable U_{sp}. The execution time is estimated from the actual measurements c_i with a first-order filtering [46]:

$$\hat{C}_i(k) = \lambda \hat{C}_i(k-1) + (1-\lambda)c_i \tag{2.26}$$

where $\hat{C}_i(k)$ is the estimated execution time, λ is a forgetting factor and $\lambda \in [0, 1]$. Finally, the optimised new task periods are calculated by simple rescaling:

$$h_i = h_{i0}\frac{U_{sp}}{\hat{U}_0} \tag{2.27}$$

where h_{i0} is the nominal period and $\hat{U}_0$ is equal to $\sum_{i=1}^{n} \hat{C}_i/h_{i0}$.

The feedback-feedforward architecture is designed to support QoC optimization of control tasks with multiple versions and adaptable sampling rates. The feedback mechanism relaxes the requirement on known WCETs and improves processor utilization by periodically monitoring and adjusting workloads. The feedforward part is complementary to feedback and it provides mode changes information to the scheduler for rapid adaptation.

It is mentioned in early section that a simple rescaling is sufficient for optimising QoC with cost functions that can be approximated by quadratic and linear functions. The type of controller that is used in the feedback scheduler is not explicitly stated. However, linear rescaling of periods based on current workload is equivalent to proportional control (P control) in some sense. The current workload is calculated from execution time estimations, which are filtered by a first order filter.

The adaptation algorithm of this method has a complexity of $O(n)$, which involves $2n$ divisions, n multiplications and n sums where n is the number of control tasks. The computational efficiency of the method makes it possible to use it online.

One aspect that is not addressed by the paper is the control system stability issue. Switching between different control modes and online changing of control periods can cause potential system instability. This issue is noticed by control systems researchers, and is being studied under the framework of Hybrid Systems [59].

2.4.5 Exploring Feedback Scheduling Algorithms

In feedback scheduling, the scheduler is operated in closed-loop and adjusts task parameters periodically, thus it is robust against workload uncertainties. In addition to the two methods that were already discussed [98] [46], there are many other parallel researches on the topic of feedback scheduling structures [74] [90] [97] [100]. Table 2.2 shows a comparison between some of the works in which feedback scheduling is applied. Specifically, each method is analysed according to the controlled variable, the manipulated variable and the control method that is applied.

It can be seen that *processor utilization* is the most commonly measured variable. As one of the most important performance metrics used by modern operating systems, CPU utilization reflects a generalised estimation of the cur-

Table 2.2: Feedback Scheduling Comparison

Method	**Controlled Variable**	**Manipulated Variable**	**Control Method**
FC-EDF [98]	deadline miss ratio	service level and admission control	PID
FBFW-EDF [46]	utilization and QoC	task rates	rescaling (P control)
AC_jAC_t [74]	utilization, job/task slacks	admission threshold	MPC
control-based middleware [90]	utilization and bandwidth	task reconfiguration	PID & Fuzzy Logic
web-server delay control [97]	connection delay	process budget	PI
EUCON [100]	utilization	rate adjustment	MPC

rent workload. Processor utilization can be either averaged over a small time window or can be calculated through periods and execution time estimations. The set point of $U(t)$ is normally chosen to be a value that can slightly overload the system to achieve the best outcomes. As the nature of it, processor utilization does not reflect the satisfaction of timing.

Another measurable index that is also widely used in feedback schedulers is *deadline miss ratio* (DMR). DMR describes the fraction of deadline misses over all committed task releases. It is observed in [98] that a higher processor utilization will lead to an increased number of deadline misses. In order to apply DMR in the feedback loop, the desired set point should be non-zero, which makes this metric incompatible with systems with hard real-time requirements.

Controlled variables can also be variables of interest that are application-specific. For example, network bandwidth is used in [90], where a scheduler for distributed vision tracking system is studied. In [97], absolute and relative connection delay is chosen as the controlled variable to assess the service quality of a web-server application.

The choosing of manipulated variable can be made once the controlled variable is identified. The general principle for selecting the manipulated variable is that the variable that is being manipulated should be able to directly or indirectly affect the controlled variable. For example, if the utilization is the variable that needs to be controlled, then adjusting task release rates would be a valid manipulation. The input-output relationship, known as system dynamics that maps the controlled and manipulated variables, can be obtained by using an analytical method as in [98] or system identification as in [74].

The control method specifies the way of calculating the manipulated variable based on the measured controlled variable. In general, the PID controller [98] [90] introduced in Section 2.4.3 and its variations, e.g., PI controller [97], are the most used controllers in feedback scheduling. It has many advantages: it is robust against uncertainties and disturbances, and it is computational efficient. One of the drawbacks of PID is that it can only handle SISO (single-input-single-output) systems. For MIMO (multiple-input-multiple-output) systems with couplings between inputs and outputs, MIMO controllers such as MPC (Model Predictive Control) [74] [100] are more preferable than using multiple SISO controllers .

Feedback naturally relies on feedback information (measured variables).

The sampling process tends to smooth the measurements by, e.g., a first-order filter or by mean filtering over a certain sampling window. Hence some of the system dynamics are lost during sampling, and an average rather than an exact performance evaluation is applied. In the next section, statistical methods that generalise system behaviour from a wider range of sampling data will be discussed, which could utilise more information from system measurements.

2.5 Statistical and Probabilistic Methods in Scheduling

In the previous section, it is shown the implementation of feedback control in scheduling that are constructed in order to delivery robust performance against execution uncertainties. In this section, modelling methods that explicitly consider these uncertainties using either statistical or probabilistic analysis methods will be explored.

Many attempts were made to apply statistical analysis with probabilistic guarantees [65] [60] [71] [15] [65]. Most of these works assume the system to be either a firm or a *probabilistic hard* [31] real-time system. The notion of probabilistic hard real-time system is introduced by Bernat in 2002 [25]. Probabilistic hard real-time system is a hard real-time system in which a small amount of deadline misses are acceptable, if the probability of such misses are below a predefined threshold, e.g., 10^{-6}. The argument is that although a deterministic timing guarantee is needed for most hard real-time tasks, the requirement of timing reliability does not necessarily need to be higher than the requirement on hardware. The probabilistic specification of timing is used to overcome pessimism from overestimated WCETs, by using probabilistic WCET analysis [25] [24] [33].

In addition to probabilistic modelling, another way of modelling stochastic behaviour in task execution is to model the scheduling system as a queue. Real-Time Queueing Theory [89] is an extension of the classic queueing theory, which adds customised timing requirements to the queueing models. The analysis is not limited to a specific scheduling algorithm and it can also be extended to analyse real-time network schedules. However, as criticised by [60] there are many limitations where this method can be applied, e.g., the system is assumed to be working under heavy-traffic conditions, and the arrival and computation times are identical for all the tasks.

In the framework of probabilistic hard real-time systems, tasks can also be modelled with probabilistic execution times and inter-arrival times. Abeni and Buttazzo presented a stochastic analysis of a reservation-based system in 2001 [2] , which enables the computation of the probability of deadline misses. In their approach, a task is described by a pair of probability distribution function (PDF): $\tau_i = (U_i(c), V_i(t))$, where $U_i(c)$ is the probability of task execution time and $V_i(t)$ is the distribution of task's inter-arrival time. The notion of *probabilistic deadline* is introduced to quantify the task QoS, which is defined as a deadline that has to be fulfilled with a probability:

$$X_i(\delta) = P\{f_{i,j} \leq r_{i,j} + \delta\} < 1 \tag{2.28}$$

The system can then be modelled as a queue where stochastic analysis can take place. If the queue is stable, the probabilistic deadlines can be computed by solving an eigenvector. The result could be used as a guideline to decide the reservation parameters (T^s, Q^s) by comparing with the system specification.

Progress has also been made which extends the classic response time analysis to fit tasks with statistical parameters. Tia et al. [129] proposed an analysis method, the *Probabilistic Time Demand Analysis* (PTDA). In their work, they have shown that traditional scheduling algorithms and schedulability analysis methods can be easily modified to accommodate semi-periodic tasks, i.e. periodic tasks with widely variable execution times. The PTDA extends the Time Demand Analysis by substituting the fixed execution times with convolution of probabilistic functions. The result is the probabilistic function of the response times, based on which a system engineer can make trade-offs between more computational resources (e.g., number of processors) or lower degree of confidence that all individual task requests will meet their deadlines.

Gardner et al. [71] extends the PTDA to cover systems where tasks can have deadlines larger than periods ($D > T$). Their method, the *Stochastic Time Demand Analysis* (STDA), computes a lower probability bound that jobs in each task will meet their deadlines. This method gives a quick way to determine if the probability of missed deadlines is acceptable, giving other design goals such as processor utilization or cost. However, it is restricted to fixed priority assignments.

In 2002, Diaz et al. [60] proposed a more generalised framework to deal with both fixed-priority and dynamic-priority scheduling. In their approach, knowledge of the execution time probabilistic functions required by each job is assumed to be known *a priori.* Given that if the average utilization is less than 1, the long-term statistical behaviour can be described by a Markov chain which models a *P-level backlog* process [60]. Equations have been derived for calculating the backlog at any time, as well as task response time probabilistic functions. The statistical information can be combined with deadlines to obtain the probability of deadline misses of a specific task.

Overall, the advantages of introducing statistical analysis in real-time systems is twofold:

(a) If a probabilistic distribution, rather than the worst-case of task execution time is given, a trade-off can be made to balance the confidence level of deadline misses with available system resources;

(b) Statistical information is extracted from actual measurements, which means a priori knowledge of task execution is not essential, and the constructed model is more realistic compared with analytical methods.

It is thus possible to construct a more effective scheduler if statistic information could be incorporated within the framework of feedback scheduling.

2.6 Flexible Scheduling in Cyber-Physical Control Systems

The scheduling of tasks in Cyber-Physical Control Systems (CPCS) has become an interest of both real-time and control researchers in the last few decades. CPCS refers to the control aspect of Cyber-Physical Systems. The initial motivation came from a strong demand for *control and scheduling co-design* [9] [7], that is to take scheduling effects into consideration during the design of a control system.

In recent years, CPS are becoming increasingly complex in terms of both hardware architecture and functional requirements. On the other hand, in most of the cases the implementation platforms are using commercial off-the-shelf (COTS) hardware with cost and size constraints, which will result in temporal non-determinism and limited execution resources [48]. Within a control node, a task suffers interferences from other tasks through preemption and blocking when waiting for shared resources, e.g., CPU and network. Also for a CPS system, there are non-control related tasks which need to share resources with control tasks. Hence it is important that the computational resources are allocated properly and in an optimised way.

To achieve a real-time control system under execution resources constraints, a control/scheduling co-design approach is promising, in which flexible and adaptive scheduling policies are implemented.

2.6.1 Control-Scheduling Co-design

The idea of cooperatively designing a control system with real-time scheduling is introduced by Årzén and Cervin et.al. in [7]. Co-design is a way to overcome the limitation of independently designing control and scheduling. It is shown in this work that designing a control system separately without considering the underlying real-time facilities could result in an unexpected control performance. The authors discussed many methods for better scheduling control tasks, including flexible task models, timing compensation and feedback scheduling.

In [9], the authors further introduced scheduling issues that reduce control performance to motivate control-scheduling co-design, e.g., sampling jitter, input-output delay, interferences from higher priority tasks and non-determinism from general purpose hardware and off-the-shelf operating systems.

2.6.2 Scheduling Models for CPCS

A control task model that supports a range of periods to allow trade-offs between control performance and limited computing resources is described in [119]. In their work, a monotonic, convex and quadratic relationship is found between control frequency and control performance under a quadratic cost function.

Task decomposition [72] is another way to improved schedulability. In [44], a control task is split into two separate sub-tasks: output calculation task and state update task. It is shown that by scheduling a control task as two sub-tasks, the computational delay can often be reduced significantly, and the control performance can be improved whilst maintaining schedulability.

In [6], Årzén introduced an event-based model as an alternative to the traditional time-triggered paradigm. The event-driven method comes from the nature of the system being controlled. This method combines event-based and time-based mechanism and it is efficient in terms of resource usage.

Another commonly used model for scheduling control tasks is (m, k)-firm scheduling [76]. Instead of executing every job instance of the task, the scheduler only needs to schedule m or more instances out of any k consecutive releases. This is possible due to the rationale that occasional misses of the output update can be tolerated by most control applications. In [113], an (m, k)-firm model is used as a less stringent guarantee than the hard deadline requirement. This work shows how (m, k)-firm models can be used to achieve graceful degradation when the task set in a control system is overloaded.

2.6.3 Feedback Scheduling for CPCS

Feedback scheduling for general computing systems was discussed in Section 2.4. In the context of CPCS, there are multiple methods for feedback scheduling of control tasks [123]:

1. Increasing the control interval h_i, that is equivalent to increasing the task period T_i in scheduling;
2. Reducing the execution time C_i, e.g., by using an anytime control algorithm, or a simplified version of the controller;
3. Using the Dynamic Voltage and Frequency Scaling (DVFS) capabilities of modern computing chips. DVFS can dynamically adapt the CPU computing speed and reducing energy cost.

Among these three methods, varying control intervals is the most common way of manipulating CPU utilization, and it is used by many online feedback schedulers [70] [103] [66]. The use of feedback scheduling in real-time control systems can be categorised as [96]:

1. Resource aware feedback scheduling (RA-FS)
2. Resource and control aware feedback scheduling (RCA-FS)

In [66], a control task period allocation problem is solved by forming a recursive algorithm to compute new values for sampling frequency at each step, by approximating control cost as a quadratic function against frequency. This is categorised as RA-FS, since only utilization is considered in the optimization process.

RCA-FS considers both kernel load and current dynamics of the controlled plant [70] [103]. For RCA-FS, either the current state, or a finite horizon of predicted states is used in the optimization process, in additional to desired utilization. The intuitive idea behind RCA-FS is to give more computation capacity to control tasks of those plants experiencing severe transients.

2.7 Summary

In this chapter, flexible scheduling and feedback scheduling are reviewed as approaches that can handle overloads and execution time uncertainties. These uncertainties, either caused by defects in the task model or introduced by hardware speculative features, will degrade the performance of traditional scheduling methods. Flexible methods, e.g., elastic model and imprecise computation, increase the flexibility of the system by adding degrees of freedom to the original task model through variable parameters. Similarly, feedback scheduling constructs a closed-loop by taking measurements from the target system and applies feedback control strategies. A hierarchical scheduler can be used to monitor the current workload and apply task configuration adjustments. Finally, the motivation of control-scheduling co-design is examined, as well as using flexible and feedback scheduling methods in the context of Cyber-Physical Control Systems. The contributions of this thesis are based on these ideas and methods. In the next few chapters, details of these contributions will be introduced and discussed in detail.

Chapter 3

Trend Analysis of Dynamic Worst-Case Execution Times

Priority-based scheduling deals with execution time uncertainties by the dynamic scheduling of tasks at run-time, while task timing is guaranteed by a prior knowledge of WCETs. Hence WCET is an important abstraction of the tasks, and its validity and correctness is important. For current practices, it is often assumed that the WCET is a static value during the system lifetime. This assumption is used through the whole process of real-time system design and timing verification.

Many CPSs, including Industry 4.0, Internet-of-Things (IoT), autonomous driving and intelligent robots, have rich functional features and sophisticated interactions between computational and physical processes. The WCETs of these applications may not stay constant all the time during its long time of existence. For example, as data accumulates and hardware ages, the WCET can gradually increase over time.

The simplest solution is to use a conservative WCET in the first place, which produces a pessimistic but schedulable system. An alternative and potential better solution is to accept that WCET is subjected to change, and embrace this fact by using less conservative WCETs initially with dedicated mechanisms that could alleviate the consequence if WCET increases, which is the paradigm applied in this work.

The novelty of the idea proposed by this thesis is the dynamic perspective of WCET, or dWCET. dWCET extents the static perspective and allows WCET to vary with time, which could reduce pessimisms and improve resource utilization. By explicitly monitoring and modelling of WCETs, the scheduler can produce a more effective solution that fits not only current but also future resource requirements. Another benefit of dWCET is for handling the case when the original WCET is incorrectly modelled. As systems designed using dWCET often have capabilities to reallocate resources, a constantly overrunning of task execution can be accommodated by, e.g., increasing CPU speed, migrating tasks between cores, or terminating or reducing the frequency of less important tasks.

In this chapter, the motivation of dWCET is first introduced, followed by a range of models that could be used for dWCET modelling. The issue of trend analysis is then proposed as a problem of concern, which will be studied through a comparison of state-of-the-art methods.

3.1 The Dynamic Perspective of WCET

In Cyber-Physical Systems, the execution time that a program requires is affected by both hardware and software components in that system. For a system that is designed for operating over a long time of period, e.g., 10 years or more, the conditions of hardware and software are subjected to change, which will reflect on WCETs.

As a motivational example, considering a decision support system that produces results based on historical data in a real-time database. As the items in the database tables increase, the execution time needed for a query also increases linearly or logarithmically. Another example is for a CPU with thermal management that can change CPU voltage and frequency. The ageing of the CPU itself, the degradation of the cooling fan and the accumulation of dust that blocks the air flow will make the CPU easier to be overheated, thus the thermal management unit will reduce clock frequency that will propagate and influence execution times.

The static and dynamic perspectives will lead to two different design paradigms. For the static perspective, resources are conservatively reserved and the schedulability is verified according to the estimated WCET either from static analysis or measurement-based methods. While for the dynamic perspective, as the potential variation of WCET is explicitly considered, the resulted system will have more flexibilities with the ability to reallocate computational resources at run-time. For emerging CPS applications such as Industry 4.0, Internet-of-Things, autonomous robots and driver-less vehicles, the system will benefit from using this dynamic view of WCET to improve timing robustness and allow more functionalities to be integrated on hardware platforms with scarce resources.

In Table 3.1, different situations of WCET modelling are identified. The scheduling behaviours under static and dynamic WCET assumptions are also discussed and compared. From the table, it can be seen that in most cases more effective scheduling strategies can be deployed by migrating from a static assumption to a dynamic perspective of WCET.

It is further suggested that if a CPS satisfies one or more of the following criteria, then dWCET is worth applying:

- The system deals with accumulated data;
- The system needs to survive a long living time to provide continuous services;
- The system needs to be deployed in a harsh environment where maintenance is lacking and accessibility is limited;
- The system is designed to allow multiple levels of quality-of-service;
- The system has self-adaptive and learning behaviours that will improve its software components or change its structure over time.

It is also realised that not every CPS would benefit from this perspective shift. Here are some typical cases in which using dWCET will not be beneficial:

- The system has a strict requirement on service delivery, that requires a consistent level of quality over its life time. As adaptive behaviours often lead to variations in quality of performance, this cannot be guaranteed;
- The system only has a short working period. If the system only needs to work for a short time, the effects of changes in WCET can be negligible, and the system will likely be survived;
- The system has small scale, low complexity, and more than sufficient resources, in which case using a simpler scheme, e.g., execution time servers, is already sufficient and more preferable.

Table 3.1: Comparison of Static and Dynamic WCET Perspectives

Condition Statement	**Static WCET**	**Dynamic WCET**
C_i is perfectly modelled.	System works with satisfactory performance.	System works. However some resources used for monitoring WCET and planning schedules are wasted.
C_i is initially overestimated.	System works with utilization lower than expected.	System works, and the over reserved resources will be used by other tasks.
C_i is initially underestimated.	System will occasionally be overloaded if at run-time the execution time exceeds the expected C_i.	System works but some tasks will have to degrade their quality-of-service.
C_i grows as system runs.	System will stick with the initial WCET and will eventually become unschedulable.	As soon as the increment in C_i is identified, some resources have to be released to accommodate for additional requirements.
C_i decreases as system runs.	System will stick with the initial conservative WCET. As system runs, more idle resources will remain unused.	The reduced resource usage from τ_i can be used by other tasks and services.
C_i varies according to a pattern.	Using a WCET estimation that could bound all variations.	As long as the pattern can be modelled, a corresponding resource allocation strategy can be worked out.

However, it is still suggested the necessity to relax the constant WCET assumption, as: *a)* it is becoming more difficult to obtain a precise upper bound of the execution-times for modern commercial-off-the-shelf (COTS) CPUs [136] [135], and *b)* the increasing system complexity and level of interactions between hardware and software in Cyber-Physical Systems is forcing new design paradigms that will be more robust and resilient.

3.2 Modelling of dWCET

For design and analysis of traditional real-time systems, the WCET of a task is modelled as a constant and static value, often denoted as C_i. Existing works that extend this model include:

1) Run-time estimation of WCET [37] [82]: an estimated WCET $\hat{C_i}$ is obtained based on run-time observations of the maximum execution time;

2) Parametric WCET [20] [43]: WCET is expressed as a formula in which WCET is parametric in some of a program's variables, for example, $PWCET = 30n + m \cdot log(m) + 10$, where n and m are input variables;

3) Probabilistic WCET [25] [24] [33]: WCETs are modelled as probability distribution functions. This modelling method can be used to derive probabilities of missing a deadline, that can match the same order of magnitude as other dependability estimates [24].

However these methods are not sufficient to model dWCET as there is no description of the changing in WCET. For a long-lived CPS, many internal and external factors could affect task execution times. During the initial phase of this research, a number of factors are identified that will have influence upon the initial task execution times. These contributors can be categorised into software and hardware aspects, which are given in the following list:

1. Software considerations

    - Data accumulation. For data-dependent tasks, as the amount of data naturally grows with time, more processing time is needed. For example, many searching algorithms and heuristics have a time complexity of $\mathcal{O}(log(n))$ or $\mathcal{O}(nlog(n))$, in which n is the size of data.

- File system ageing [124]. An aged file system would behave worse than when the file system is new due to change of file layout and file fragmentation [140].
- Growing resource requirement due to self-improving behaviour or increased specification requirements.
- Increased complexity of software structure. For self-adaptive systems [117], the software components will be updated depending on the adaptation policy.
- Software upgrades. During the long lifetime of a CPS, the software is subjected to patches and updates that will change the timing behaviour of the code.
- Software ageing, which refers to an empirically observed phenomenon that as the run time of the system increases, its failure rate also increases [75].

2. Hardware considerations

   - CPU. Effects such as negative-bias temperature instability (NBTI) [50], electromigration (EM) and time-dependent dielectric breakdown (TDDB) [5] will reduce the efficiency of a CPU and the maximum clock speed the CPU is capable of will decrease over time.
   - Network-on-Chip (NoC). For large scale multi-core and many-core systems, a real-time mesh NoC [102] is often used to connect on-chip cores. If one or more of the routing nodes failed due to ageing, the end-to-end response time of messages between cores is likely to increase.
   - Battery. For battery powered embedded systems such as energy-aware Wireless Sensor Networks (WSNs) [138], the long term wear and tear of the battery will trigger power management strategies such as slowing down the CPU frequency.
   - Hard drive. The ageing of the spin motor, the mechanism of the positioning arm and the magnetic platters in the hard drive will result in reduced reading and writing speed.
   - Thermal effects. Modern CPU has thermal management which will change its frequency and voltage at run-time based on CPU activities. As the ageing of the CPU itself and the decreasing in

cooling efficiency over time, the CPU will be more likely to overheat and trigger circuitry that slows down the clock.

- Other hardware or infrastructure degradation. Fault rates and fault recovery time will likely be increased as a consequence of degraded (or faulty) hardware.

Most of these effects will make the task execution times become longer, which will degrade the schedulability even if the system is initially verified to be schedulable. Also the influence of these effects could be minimal in a short period of time, but if being examined in a larger time-scale, e.g., days, weeks or months, the impact on task execution times would be observable. As a first attempt, it is natural to model dWCET directly as a combination of all these effect variables:

$$dWCET = f(g(\mathbb{S}_h), h(\mathbb{S}_s), i(\mathbb{S}_h, \mathbb{S}_s), \mathbb{S}_e) \tag{3.1}$$

in which $\mathbb{S}_h$ represents all hardware states, $\mathbb{S}_s$ includes all software states, and $\mathbb{S}_e$ is the state of the environment. However, it is soon realised that it is too difficult to analyse how these internal and external states will propagate to influence the WCET. Also it is impractical to find a model that has the capability and richness to include every aspect. Even if such a model existed, the usability of the model is limited, as many influence factors are not easily monitored and quantified at run-time, e.g., degree of hardware degradation.

In this work, an alternative modelling method that is known as *data-driven modelling* is applied, which is to build an 'indirect model' based on historically measured WCETs. The model is defined in an iterative form, in which the current dWCET is based on a number of previous states:

$$dWCET_n = f(dWCET_{n-1}, dWCET_{n-2}, \dots, dWCET_{n-k}) \tag{3.2}$$

This model is more practical to use in real applications. Although it is not exact, this model can still enhance the static WCET perspective. Systems are often designed with a limited tolerance of worst-case execution times. To design a long-lived and reliable system, it is important to observe the variation of dWCET and predict if the WCET assumption will be violated.

Exploring worst-case execution time could also benefit task scheduling. A scheduler should not be 'short-sighted'. If a scheduler can predict future execution behaviours, it would be possible for it to allocate resources more optimally, and to reduce the number of unnecessary reallocation/redistribution

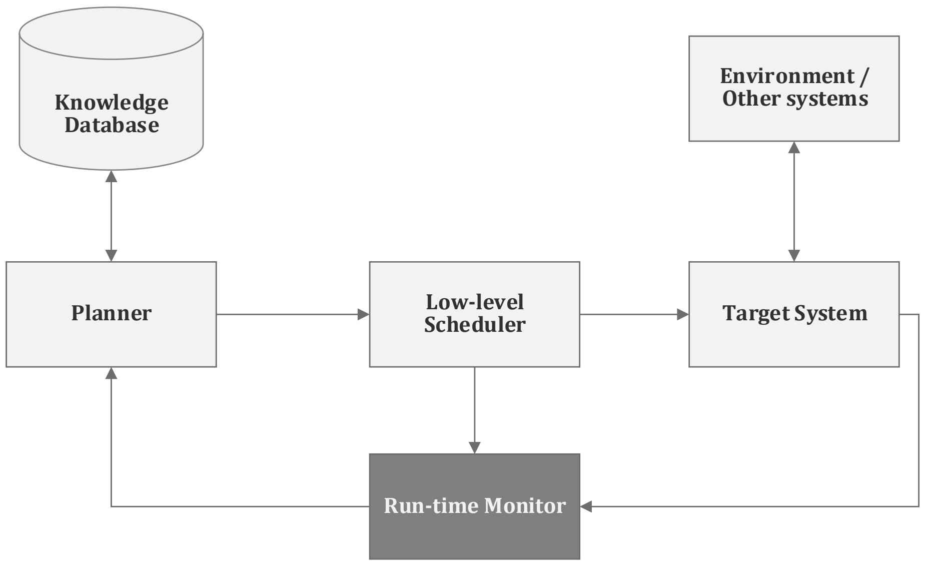


Figure 3.1: A simplified structure of the ATAS framework. The run-time monitor is in the feedback loop which consists execution time measurement module and performance indices evaluator.

actions. It is interesting to see how adaptive control, as well as flexible scheduling methods, e.g., feedback scheduling (see Chapter 2), could be applied in an integrated framework.

3.3 Execution Time Monitoring

To model dWCET, there should be methods that can measure task execution times when the system is running. The execution time monitoring process is part of the 'run-time monitor' module in the Adaptive Task Scheduling Framework (ATAS) framework (illustrated as the grey block in Figure 3.1).

Measuring and monitoring the CPU execution times of a task often needs support from either the operating system kernel or the run-time library. There are many operating systems, middleware and libraries that have readily available support for measuring task execution times. To give a few examples:

1. **Operating system support**. The real-time extension of POSIX [84] supports execution-time clocks and timers that allow an application to monitor the execution time consumption of tasks, and to set execution time limits [109] [77].

2. **Real-time programming language support**. The Ada programming language [35] has support for POSIX standard Execution-Time Clocks which can be used to measure and monitor task execution times [105] [78].

3. **Run-time libraries support**. The Simulink Real-Time Library [56] supports task execution time (TET) which measures how long it takes the kernel to run for one base-rate time step. The TET monitor in the MATLAB session is available for all Simulink Real-Time target objects.

More generally, task execution times can simply be measured using a high precision timer that counts the CPU time spent for executing each task. The timer starts when the task begins to execute, and ends when the task finishes execution. If the task is preempted by higher priority tasks, the timer will pause until the task retrieves the CPU. This can be implemented in the context switch handler that is provided by most real-time operating systems.

3.4 Trend Analysis

In the previous sections, the motivation of dWCET is introduced. The modelling and monitoring methods of dWCET are then discussed. The next chapters will discuss how the proposed dWCET model and execution time monitoring can be used to improve system schedulability.

In the following sections, a specific problem within the ATAS framework is explored: identification of trends in task worst-case execution times. A trend in this context refers to a systematic upward or downward tendency in observed execution time sequences with regard to time. From historical execution time traces, potential patterns can be inferred to explore any regularity within the observations. Here are some motivations and benefits to identify execution time trends in a cyber-physical system:

1. To understand the characteristics and influential variables of worst-case execution times;

2. To make future predictions of execution time based on the identified trend model;

3. To use the information of dWCET for enhanced feedback scheduling;

4. To make a system aware of potential timing failures before these actually happened.

Trend identification has a long history in prediction of time-series data. It has been widely used in environmental, business and financial predictions, in which streamed data is obtained at an approximately equal time interval. The objective of trend identification is to derive a model of the underlying process based on historical measurements, and to use this model to analyse and understand the process, as well as make future predictions. In the context of identifying trends in execution times, similar techniques could possibly be applied. However there is a need to understand which identification method can be used to accommodate the characteristic of execution times. It is also questionable which model is the best fit for describing long-term trends.

Since real-time systems have a high concern about worst-case execution times, it is natural for a system to observe these worst-case scenarios in a short period (known as watermarks). The focus is on increasing trends rather than decreasing ones, since these are the potential risks that could make a system fail. Systems are often designed with a tolerance of worst-case execution times. If there is any evidence indicating the WCET assumption will be violated in the near future, it is important to detect these faults before they actually cause the system to break, by e.g. eliminating the risk, or achieve a graceful degradation if possible.

Exploring execution time trends could also benefit task scheduling. A scheduler of a CPS should not be ‘short-sighted’. If the scheduler is able to predict future execution behaviours, it is possible for it to allocate resources optimally, as well as avoiding any potential timing failures. It is also interesting to see how feedback techniques as well as dynamic scheduling methods could be applied within an integrated framework.

The rest of this chapter is organised as follows: a general review of trend identification methods is introduced first. Notations and symbols used in this work then follow. A comparison experiment was made to compare different trend identification methods. Finally, the experimental result is analysed and conclusions are made.

3.5 A Survey of Trend Analysis Methods

The question of the presence of a trend in a time-series originally arose in business, economical and environmental problems. For these real-world applications, the variable of interest is either measured or calculated at an approximate constant rate. The resultant time sequence, which is referred to as time-series data, can be analysed statistically to test the existence of any trend. In this section, commonly used trend detection and identification methods will be reviewed. Here a comprehensive, rather than an exhaustive, exploration of some representative methods is given. It should also be noticed that many methods have assumptions on the data set, hence only a subset of them can be used in this case.

The awareness of a trend in data is important for understanding the impact of new policies and changes. For example in environmental studies, the trend of extreme precipitations was studied in [121], to understand its impact on rainfed agriculture in Ethiopia. Another example is presented in [130], in which the detection of trends in stratospheric ozone is studied. Many descriptive and model-based approaches have been used to detect trends. These range from correlation analysis, time-series modelling, regression analysis and non-parametric statistical methods [80].

Conventionally, statistical time series models are used for representing time series data, e.g., Autoregressive (AR), Moving Average (MA) and Autoregressive Integrated Moving Average (ARIMA) models. Box and Jenkins [29] proposed a set of effective strategies for building seasonal ARIMA models. A current extension in practise is Generalized Autoregressive Conditional Heteroskedasticity (GARCH), which is largely used in stock and financial data predictions.

An important non-parametric (i.e., distribution free) statistical test is Kendall's tau, which is widely used as a test method for trends [143] [87]. The Mann-Kendall test is often used to evaluate statistical significance. The Seasonal Kendall is an extension to the original Kendall test, in which seasonality is also considered [67]. In the work of Sen [118], a slope estimator based on Kendall's tau, known as the Theil-Sen estimator is designed, which is a non-parametric estimator that takes the median of all possible slopes of pairwise observations. This estimator is claimed to be statistically robust and unbiased. The use of Kendall's test and Theil-Sen estimator in extreme precipitation can be found in [87]. Another statistical test for trend detection is the

Spearman's Partial Rank Correlation (SPRC) [104]. It is similar to Kendall's tau as it measures the relationship between two variables, but differs in the interpretation of the correlation result.

In Visser and Molenaar's work [133], a structural time-series model is proposed which has a stochastic or a deterministic trend, and regression coefficients. The stochastic trend is described as an ARIMA process, and the overall trend-regression model is estimated by a Kalman Filter (KF). However, the KF does not consider time correlation of data. Thus it is a challenge to make long term prediction in the presence of uncertainty.

Regression analysis is a class of model-based approaches for estimating the relationship between dependent and independent variables. Linear models with a trend and a seasonal component are often applied in prediction and forecasting of time series data, where the parameters are often estimated with an ordinary least square (OLS) estimator. However for the OLS estimator, residuals of the time series are required to follow a normal distribution [118], which is not always valid. Reinsel and Tiao [115] used linear regression models to estimate trend with a correlated noise that is modelled by an autoregressive process. In their model, additional explanatory variables are used in the analysis to improve the prediction precision. Linear regression is applied by Tiao in the detection of trends in stratospheric ozone data using time series models with autoregressive noise [130].

Predicting trends is also of great interest in modelling and explaining the variation in rare and extreme events. These changes can be assessed either by a trend in frequency or magnitude of extreme events. Detecting long-term trends in the frequency of extreme events is studied in [69]. In this study, Frei and Schär modelled the counts of extreme events based on a binomial distribution, and used logistic regression to estimate trends. Several methods of detecting the change of intensity in the extreme values are reviewed in [125]. A common way to model extreme events is to use generalised extreme value (GEV) distributions [87] [143], which was first introduced by Fisher and Tippett in their study in 1928 [68]. The extreme value distribution is generally applied on block maxima, e.g., annual or monthly maxima of a time series.

One drawback of using block maxima is that only one data point in each block is used in analysis. Alternative data pre-processing approaches include the Peak-over-threshold (POT) and r-largest methods, which use relatively more data points to train a model or fit a distribution. The POT is used in [121] to study extreme precipitation in Ethiopia. In their study, the location parameter of the EV distribution is represented by a monthly constant and a yearly trend. A similar model is also applied in [125], in which the parameters of extreme distribution are estimated by maximum likelihood that are considered separately for each month.

Zhang [144] compared several methods for detecting significant linear trends in the magnitude of extreme values. The Mann-Kendall, generalised linear regression (GLS), and the r-largest methods were studied, which is followed by a Monte Carlo simulation for comparison. Zhang stated that Kendall outperforms OLS when the sample size is large, and generalised linear regression which incorporates generalised extreme value distributions has stronger power of detection compared with the other two methods.

Neural Networks have also been actively researched for trend detection, which have been widely used for time series modelling and forecasting [141] [111] [81] [19] [139]. Theoretically, Neural Networks can precisely model any linear or nonlinear relationship in the data, without making specific assumptions of the underlying model. This data-driven method is powerful in forecasting as a Neural Network is a universal function approximator, and it can learn patterns adaptively through mining the data [141]. However, few practical guidelines exist for building a time series Neural Network model, in terms of the number of input nodes and hidden layers, etc. To overcome the problem, automatic modelling methods are studied in [19] [139]. Zhang [141] stated that it is important to preprocess the data before it is fed into the Neural Network, since a time series with trend and seasonality is considered to be a nonstationary process, which can introduce errors on forecasting. Overall, the use of Neural Networks to explore trend and extreme events is still not well studied.

Support Vector Regression (SVR) is another data driven machine learning method. It belongs to non-parametric regression class and is firmly grounded in the background of statistical learning theory. It is extremely flexible because few assumptions are imposed upon the mean function of the distribution, and it is capable of revealing non-linear relationships between variables. However, non-parametric techniques are relatively more computationally intensive. A description of SVR and its mathematical details is given in [126]. SVR is a rapidly developing field of research in Machine Learning, and it has potential as a method for time series prediction.

To the author's best knowledge, there are few studies on trend analysis of worst-case execution times, in the context of real-time scheduling. Here it is assumed that the time series being analysed is formed of maximum execution times that are observed in a fixed duration, e.g., one hour, or one day. In addition, only long-term rather than transient trend patterns of the time series is considered. A comparison study follows to decide which method is the best fit for analysing the time series of interest. The impact of data pre-processing is also studied.

3.6 Methods for Comparison

In the previous section, several methods were introduced that can be used for the purpose of trend identification. In this study, the focus is on detecting trends in worst-case execution time observations, and the objective is to figure out which methods could be potentially used for this purpose. By doing preliminary experiments to evaluate each aforementioned method, four potential statistical methods are selected to compare, which can be further categorised into parametric and non-parametric statistics:

- *Ordinary Linear Regression* [OLR] (parametric)
- *Kendall's tau* and *Theil-Sen Estimator* [TSE] (non-parametric)
- *Support Vector Regression* [SVR] (non-parametric)
- *Extreme Value Distribution* [EVD] (parametric)

Note the difference between parametric and non-parametric methods is whether a distribution is explicitly or implicitly assumed in the process of modelling. As the type of data set this work focused on is less studied in the literature, the experiment is implemented more in an exploratory way. A comparison study is conducted between the listed methods, as well as different data pre-processing approaches for selecting the training data set. Overall, the research questions this work wanted to address are:

- Would those methods that are normally applied in environmental and financial studies also perform well for predicting WCET time series?
- Is there strong evidence that one or more methods achieve significantly better performance than others?

- Which is the best data selection method (raw, block maxima or r-largest) for a particular application?

To answer these questions, an experiment is designed in which these four methods are applied on a data set for comparison.

3.6.1 The Data Set

Synthetic data is used in this study to simplify the comparison, which is generated by a multi-state Markov process that is shown in Figure 3.2. In the figure, each node represents a system mode that has a distinct execution time due to different execution paths and system conditions. The edges represent the probability of state transition from the starting mode to the pointed mode.

The model used for generating the baseline data is abstracted from an application which has four major execution paths according to its operating states. It is assumed that a deterministic trend, if it exists, is only in the worst-case execution path. It is notable that the trend may also exist in less critical paths, but as the execution time of the path increases, that path will eventually overwhelm and become the worst-case path. It should also be pointed out that there are different types of trend: (i) linear deterministic trend (LDT), (ii) linear stochastic trend (LST), (iii) non-linear deterministic trend (NDT), and (iv) non-linear stochastic trend (NST). This work focuses on the first type, LDT, as other types can be decomposed and approximated by a set of linear trends.

The reason for using a Markov model instead of random sampling is that the former can simulate the time correlation between successive observations, and has characteristics that are closer to real data. In a realistic computer program, it is likely that the next state is dependant on the previous system state, which follows a certain execution chain; it is also intuitive that a program tends to stay in the same state for a while until a trigger event occurs. It is notable that this model is not a precise description of the actual behaviour considering all execution scenarios, but is a reasonable approximation of the behaviour exhibited by a general computer program with branch and condition structures. As the objective is not for precise modelling of execution time, but is to explore the patterns behind execution times that are varying as the system runs. Hence not every factor that would affect WCET is considered in the generation process.

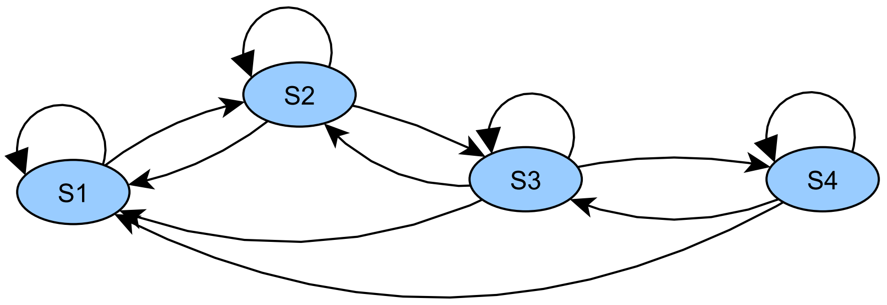


Figure 3.2: An example of a multi-state Markov process that is used to generate the experimental data sets.

In order to simulate a process where the worst-case path exhibits trend behaviour, an artificial trend is injected into the worst-case execution times. In this experiment, only time-series with a linear deterministic trend is considered. It is notable that in reality many trends are non-linear and/or stochastic, but can be decomposed and approximated by a set of linear trends in a small window. It is assumed that the execution time is only correlated with the operation time, and no other explanatory variables are used in generating the data. A linear model with a trend coefficient ω is applied, and it is assumed the data is corrupted by white noise to simulate the influence of the run-time environment, i.e., cache misses, branch predictions and resources waiting. The execution time generation model used in this experiment is represented as:

$$y(t) = C_{m(t)} + \omega_{m(t)} \times t + \varepsilon(t) \quad \{t | t \geq 0, t \in R\}; m(t) \in [1, N] \tag{3.3}$$

in which t is the time starting from 0; N is the number of states; $m(t)$ is the index of the current state sampled at time t; C_m is the initial worse-case execution time of state S_m; ω_m is the trend coefficient of state S_m where $\omega_{\{1,2,\ldots,N-1\}} = 0$ and $\omega_N = k$; the term $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ represents a Gaussian distributed white noise that is caused by unexplainable disturbances. The execution time is then sampled at discrete sampling intervals. The sampled execution time is represented as:

$$y(n) = C_{m(n)} + \phi_{m(n)} n + \varepsilon(n) \tag{3.4}$$

in which $\phi_{m(n)} = \omega_{m(n)} \times T$. During the time interval $t \in [nT, (n+1)T]$, the state s_m will transit l times according to the following state transition

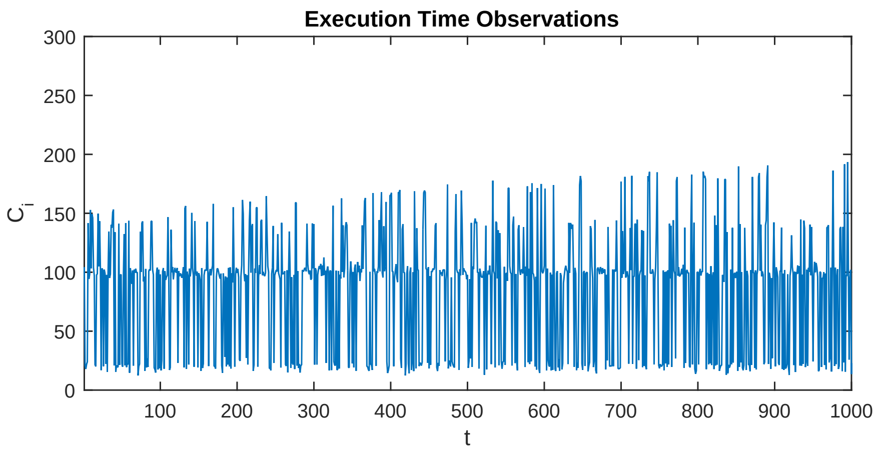


Figure 3.3: An example of generated time-series with 1% increasing trend

equation:

$$\begin{cases} s_m\,|_{t=0} = S_1 \\ s_m\,|_{t=n} = a_{ij} \times s_m\,|_{t=n-1}, \quad n = 1, 2, 3, ..., k \end{cases} \tag{3.5}$$

in which a_{ij} is the i, j element of the state transition matrix A, which is defined as the transition probability from state s_i to s_j at each step. An instance of a time series generated from this execution model with $n = 4$, $\phi = 0.05$, $l = 10$, and $var(\varepsilon) = 10$ is shown in Figure 3.3.

3.7 Compared Methods

As mentioned earlier, the trend identification methods can be categorised as parametric or non-parametric. Overall, there are four distinct methods to compare, namely: *Ordinary Linear Regression* (OLR), *Theil-Sen Estimator* (TSE), *Support Vector Regression* (SVR) and *Extreme Value Distribution* (EVD).

The objective of a prediction is to estimate the influence of a trend in the future, i.e., predicting a potential failure point where the execution time will eventually exceed a safe upper bound due to the existence of a trend. In order to evaluate the prediction precision, the *Hypothetical Failure Point* (HFP) is defined as the theoretically time point after which the system will fail the system's temporal requirements. The *Estimated Failure Point* (EFP) is also defined as the estimated HFP that is predicted by trend prediction algorithms.

For the rest of this section, more details of each mentioned trend identification method will be introduced. A short description of each method is given, as well as assumptions that are implicitly imposed.

3.7.1 Ordinary Linear Regression

Linear regression is a widely used approach for trend identification. The parameters of the model, which are in the form $y_t = b_1 x_t + b_0$ can be estimated by minimising the error sum of squares (SSE) of the residuals:

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i))^2 \tag{3.6}$$

A direct solution of Equation (3.6) can be calculated from the following Normal equation:

$$[b_0 \quad b_1] = (x^T x)^{-1} x^T y \tag{3.7}$$

OLR has an assumption that the residuals are normally distributed, while in this case it is not likely to be the case. However this work still make this assumption for the data set. As a result of violating this assumption, the quality of the analysis might be degraded.

3.7.2 Kendall's tau

The Kendall's tau test is a non-parametric statistic test for revealing the correlation between two variables. The Theil-Sen estimator is based on Kendall's tau, which minimises the tau coefficient of the residuals. The slope of the trend is estimated by the following steps:

1. For each distinct i and j, calculate $W_{ij} = (y_j - y_i)(t_j - t_i)$ if $i < j$;
2. Rank all W_{ij};
3. The estimated slope of the trend, ϕ, is selected as the median of all ranked W_{ij}:

$$\phi = \begin{cases} W_{(N-1)/2+1} & \text{(if N is odd)} \\ (W_{N/2} + W_{N/2+1})/2 & \text{(if N is even)} \end{cases} \tag{3.8}$$

Kendall's tau is robust against anomalies, i.e., large outliers will not make the result biased. The statistic significance of the trend can be evaluated by the Mann-Kendall test.

3.7.3 Support Vector Regression

The idea of Support Vector Regression (SVR) comes after the Support Vector Machine (SVM). SVM is an active research field in the machine learning community. The basic idea of SVM is to create a hyperplane that could divide different data in a higher dimension to achieve two-class data classification [28]. Unlike SVM, SVR uses only one class label and it is used to solve regression problems. The objective of SVR is to find a hyperplane in which the deviations of data from the hyperplane is minimized. This is equivalent to the maximum margin problem encountered in SVM problem. Suppose the training data is given in $\{(x_1, y_1), (x_2, y_2), ..., (x_l, y_l)\} \in \chi \times \mathbb{R}$. In ε-SV regression, the objective is to find a function $f(x)$ that has at most ε deviation from the actual data points, as well as the best flatness (to overcome over-fitting). The function $f(x)$ has the form:

$$f(x) = < \omega, x > +b \quad \text{with} \quad \omega \in X, b \in \mathbb{R} \tag{3.9}$$

where ω is weight, b is offset. The formulation of the optimization problem to solve ω and b is described below, by considering slack variables ξ_i and ${\xi_i}^*$:

$$\text{minimise} \quad \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{l}(\xi_i + {\xi_i}^*) \tag{3.10}$$

$$\text{subject to} \quad \begin{cases} y_i - < \omega, x_i > -b \leq \varepsilon + \xi_i \\ < \omega, x_i > +b - y_i \leq \varepsilon + {\xi_i}^* \\ \xi_i, {\xi_i}^* \geq 0 \end{cases} \tag{3.11}$$

Equation (3.10) can be solved by constructing a Lagrange function, which leads to a dual optimization problem. The mathematical details can be found in [126]. The problem of selecting parameters C and ϵ is described in [54].

3.7.4 Extreme Value Distribution

The idea of using extreme value distribution to identify a trend is to detect the shift of distribution parameters that are fitted with data that is from different periods of time. The Generalised Extreme Value (GEV) distribution is represented as:

$$F(z; \mu, \sigma, \xi) = \exp\{-[1 - \xi((z - \mu)/\sigma)]^{1/\xi}\} \tag{3.12}$$

where ξ is the shape parameter; μ is the location parameter and σ represents the scale parameter. For trend estimation based on extreme value distribution, both location and scale parameter can be considered to express a trend. The following model, which involves covariates by considering the location parameter μ, is used:

$$\mu = \mu_0 + \alpha t \tag{3.13}$$

where α is the trend coefficient, and μ_0 is the initial location. In this case, the parameters of the distribution are estimated by maximising a likelihood function, which is defined as:

$$\begin{aligned} L = \prod_{i=1}^{n} \sigma^{-1} \mathrm{exp}\Big[-(1-\xi\frac{y_i-\mu}{\sigma})^{1/\xi}\Big] \\ \times(1-\xi\frac{y_i-\mu}{\sigma})^{(1/\xi)+1} \end{aligned} \tag{3.14}$$

In order to estimate the three parameters of the extreme distribution μ, σ and ξ, the likelihood function of equation (3.14) needs to be maximised, or equivalently to minimise its negative logarithm form $-log(L)$. The trend that is expressed in the form of equation (3.13) can be estimated by considering two (or more) distributions that are fitted with data points that are measured at different periods.

3.8 Incorporating Data Selection

Since data selection is also important in statistic inference, this work also want to explore the effect of data pre-processing. A method can input all observations (raw data) into the training process, or alternatively refine them according to some data processing rules before they are processed. Data selection is an important process as it removes the data which is less relevant to inferencing the trend, especially for this case where the number of extreme execution times are much less than non-extreme execution times. Applying data selection also largely reduces the computational resources required by the inference process, as the size of the input data is also reduced. Hence in theory, an improved result is expected to be achieved when data selection is applied. In this study, the following methods will be considered:

- *raw*: all execution time observations are used in the analysis.

- *block maxima*: data is blocked by a group of n points, and only the maximum value in each block is used for analysis.

- *r-largest*: data is blocked by a group of n points, in which the r largest data points are used in the analysis. When $r = 1$ this is identical to the block maxima, so it is confined that $r >= 2$.

As both trend identification and the influence of data selection process need to be explored, a full combination of trend detection and data-selection methods is made, which is given in Table 3.2.

Table 3.2: Combinations of Trend Identification Methods

	raw	block maxima	r-largest
TSE	tse-raw	tse-max	tse-r
OLR	olr-raw	olr-max	olr-r
SVR	svr-raw	svr-max	svr-r
EVD	evd-raw	evd-max	evd-r

3.9 Evaluation

To make comparisons, the aforementioned trend identification algorithms are implemented in MATLAB©R2015a. Two categories of data set are generated (with trend and trend-free), and each data set consists of multiple samples that are generated by the model described earlier. In the following sections, symbols that are used in this experiment will be introdcued first, followed by experiment setup and evaluation metrics.

Note a single experiment, with one algorithm and one data set, will give rise to a large number of predictions – as the system moves from start-up to the failure point (end of the data set). Some of these predictions may be good, others not. Hence the set of predictions need to be analysed together to give an overall estimate of the quality of the algorithm in that experiment. It is assumed that the controlled system can take corrective action if the failure point, H, is identified within a relative deadline, D. But taking action too early is not useful so there is a maximum reaction-time R defined.

3.9.1 Symbols and Notations

A graphical representation of important terms and notations is given in Figure 3.4. The symbols and notations used in this experiment are listed below:

- t: the current (discrete) time; it is assumed there are no observations between two successive time points t_{n-1} and t_n.
- C_{ub}: the upper bound of task execution time. During run-time if the execution time C_m exceeds this bound, i.e., $C_m > C_{ub}$, a system failure will occur.
- k: the actual deterministic trend of a data set that is ejected while generating the data. $\hat{k}(t)$ is used to represent the trend that is estimated by trend identification methods at time t.
- H: *Hypothetical Failure Point* (HFP), which is defined as the expected time point of failure. If $k > 0$, H can be directly estimated by $H = (C_{ub} - C_{m_0})/k$. For data sets that have $k <= 0$, it is made $H = \infty$.
- R: R in this context is the reaction time which is defined as the earliest time that a system can make actions before the failure happens. If an action is made earlier than the permitted region, i.e., in $(0, H - R)$, a false positive is given.
- D: D in this context is the deadline before which any control action should have been made. If any action is made in the interval $[H - R, H - D]$, this estimator is said to behaviour correctly and it is a true positive. Otherwise, a false negative is given.
- $P(t)$: is a prediction of H made at time t; it is made $P(t) \to \infty$ if no trend or a negative trend is found. In practice it is made $P(t) = t + B$ if $P(t) \geq t + B$, where B is a boundary. This boundary indicates the failure is too far away and does not need to be concerned now.
- S: satisfactory region deviated from H, which is used to evaluate the goodness of $P(t)$. If $H - S \leq P(t) \leq H + S$, the estimation is said to be satisfactory. In the figure, this region is bounded by $-S$ and $+S$.

During run-time a system will only make a control action if the estimated failure point will be reached soon. Specifically, an action is taken if $P(t) <$

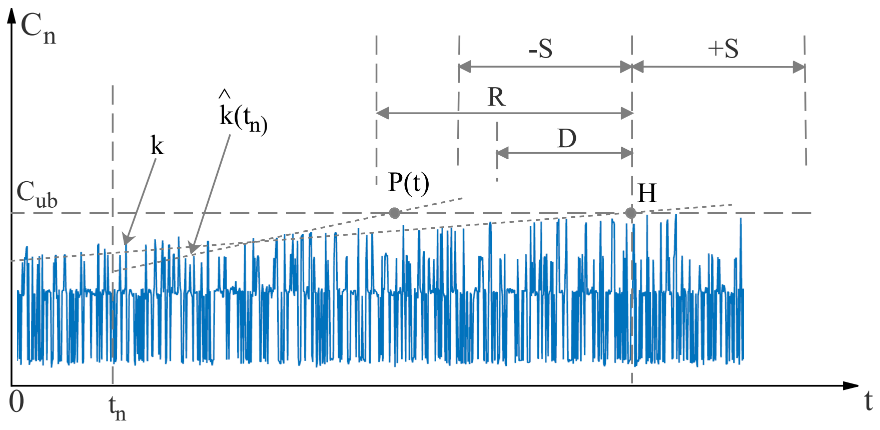


Figure 3.4: Definition of important points and regions

$t + R$, or more accurately if the prediction is run every T time, then the criterion is $P(t) < t + R - T$. To evaluate the effectiveness of each algorithm, positives and negatives are associated with whether an action is taken when it is supposed to do. A logic table discussed these scenarios is shown in Table 3.3.

Table 3.3: Definition of positives and negatives

	$t \in [0, H - R)$	$t \in [H - R, H - D)$	$t \in [H - D, H)$
Action Made	False Positive	True Positive	True Positive
No Action	True Negative	False Negative	False Negative

In reality, the numbers of false positives/negatives are found not providing enough information of the goodness of an algorithm, e.g., a false control action made close to the response region is at least better than the one made far earlier. Hence a penalty function is introduced as shown in Figure 3.5. The penalty of false positives is decreased when the time is approaching the response region $(H - R)$, and the penalty of false negatives is increasing from $(H - R)$ to the deadline $(H - D)$. When $t > H - D$, any false negative would score a higher penalty, as the deadline is already missed in this case.

3.9.2 Experiment Setup

As part of the evaluation, synthetic time-series data is generated using the model described in Section 3.7. In general, there are two categories of data: A) trend-free; B) with trend. For group A, data sets are generated that are trend-free but suffer from random noise. While for group B, both trends and

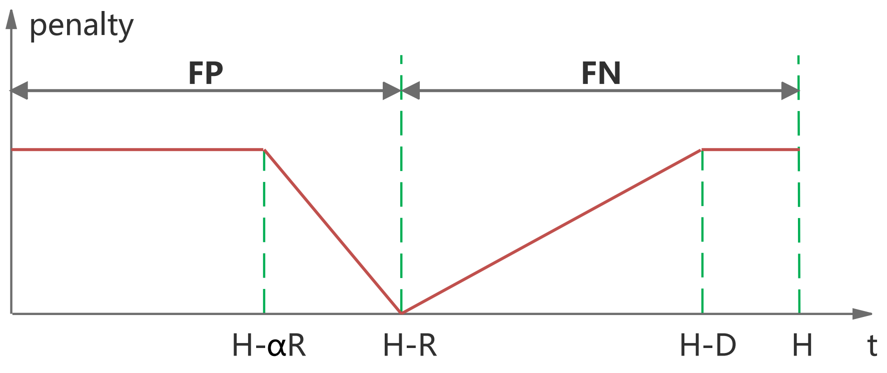


Figure 3.5: Penalties are given to false positives/false negatives.

noise are ejected. The same initial computation time C_i is used in group B, and four distinct magnitudes of trend which are gradually increasing: from 1% to 4%. For each value of trend, 10 data sets are generated independently, so overall there are 50 data sets. Each data set is generated till the point where a failure would happen, which is directly calculated from the actual trend. The size of the trend-free data set is made the same as data with 1% trend. A full list of data set is shown in Table 3.4.

Table 3.4: A table of generated data sets

Group	**Subgroup**	**data set Index**	**data set Size**	**Trend**
A	A1	1 - 10	5,000	0%
B	B1	11 - 20	5,000	+1%
	B2	21 - 30	2,500	+2%
	B3	31 - 40	1,667	+3%
	B4	41 - 50	1,250	+4%

For each data set in the table, the following evaluation steps are taken:

1. Define a sampling window W, and start to make the first estimation at time $t = W$.

2. Apply data selection process for samplings from $(t - W)$ to t. Fit pre-processed time series data with each trend analysis method to generate trend models.

3. Use the models to estimate the system failure point $P(t)$. Make a (dummy) control action if $P(t)$ satisfies $P(t) - t \leq R$.

4. Make an evaluation of each estimation, including prediction error, valid/ invalid of the estimation and the property of any made action. A cumulative penalty is added if a false positive/negative is presented.

5. Move to $t = t + M$ and repeat from step 2 until all data points are processed, where M is the step size. M controls the fraction of new data that is not overlapped in the training set. For example, if $M = 0.2W$, at each step 20% new data will be added into the analysis.

To evaluate the quality of an estimation, the knowledge of the actual failure time H can be used. Define the failure estimation error at time t as:

$$e_h(t) = H - P(t) \tag{3.15}$$

If $|e_h(t)| \leq S$, the estimation is satisfactory (valid). Otherwise, it is recognised as invalid. A smaller prediction error represents a better estimation, and an ideal predictor would have $e_h = 0$. In practice, it is preferred to have a predictor that would give a positive error (earlier) rather than a negative error (later), as in the former case, it gives more time for the system to process and make a reaction.

In addition to failure estimation error, trend estimation error is also considered, which is calculated as:

$$e_k(t) = k - \hat{k}(t) \tag{3.16}$$

Note that e_k and e_h are correlated, but e_k is more intuitive for evaluating the precision of estimated slopes. To study the absolute performance of each algorithm, a baseline algorithm is introduced: the *Ideal Predictor* (IDP), which has the foresight to know the HFP and associated time regions. For IDP, there is $\forall t : e_k(t) = 0$ and $\forall t : e_h(t) = 0$.

According to current time t and estimated failure point $P(t)$, there are several prediction scenarios:

- When $0 \leq t < H - R$, no action should be taken.
  - If the prediction has $P(t) > t + R$, no action will be taken, which is a correct behaviour and a true negative.
  - If $P(t) \leq t + R$, action is falsely taken, which is not correct and a false positive.
- When $t \geq H - R$, a control action should be taken.

- If $P(t) > t + R$, no action will be taken, which is an incorrect behaviour and a false negative.
- If $P(t) \leq t + R$, action will be taken, which is correct and a true positive.

Based on these scenarios, the number of correct behaviours equals the sum of all true positives and true negatives, and the number of incorrect behaviours equals the sum of all false results.

3.9.3 Impact of Data Pre-processing

Data pre-processing is an important procedure in processing time-series data. In this evaluation, the raw data is compared with two data pre-processing methods: block maxima and r-largest value ($r = 3$), which are both used in extreme value analysis.

For each method i, pre-processing method j and each data set κ, the mean estimated trend error $\bar{e}_k^{i,j,\kappa}$ is obtained of all evaluations over that data set:

$$\begin{aligned}\bar{e}_k^{i,j,\kappa} &= \frac{1}{N_\kappa}\sum_{n=1}^{N_\kappa} e_k^{i,j,\kappa}(W + n * M) \\ &= \frac{1}{N_\kappa}\sum_{n=1}^{N_\kappa}(k - \hat{k}^{i,j,\kappa}(W + n * M))\end{aligned} \tag{3.17}$$

where W is the sampling window, M is the step size and N_κ is the number of evaluations made over data set k. In this case, data sets with different magnitude of trends have different sizes. Hence N_κ of each subgroup is distinct, which can be calculated from:

$$N_\kappa = floor((\text{size\_of\_}\kappa - W)/M) + 1 \tag{3.18}$$

To analysis the result, $\bar{e}_k^{i,j,\kappa}$ are grouped by $\{i, j\}$ and are plotted out as box plots in Figure 3.6. There are overall 12 box plots (4 identification $\times$ 3 preprocessing methods), and each box plot consists of 50 data points that comes from all data sets. Definitions of method labels are the same as in Table 3.2.

From Figure 3.6, it can be clearly seen results that using *raw* data have worst performance, i.e., *olr-raw*, *tse-raw*, *svr-raw* and *evd-raw*. Compared with the other two methods *max* and *r*, methods using *raw* have a significant larger median and variance of mean error. This is reasonable as none of the

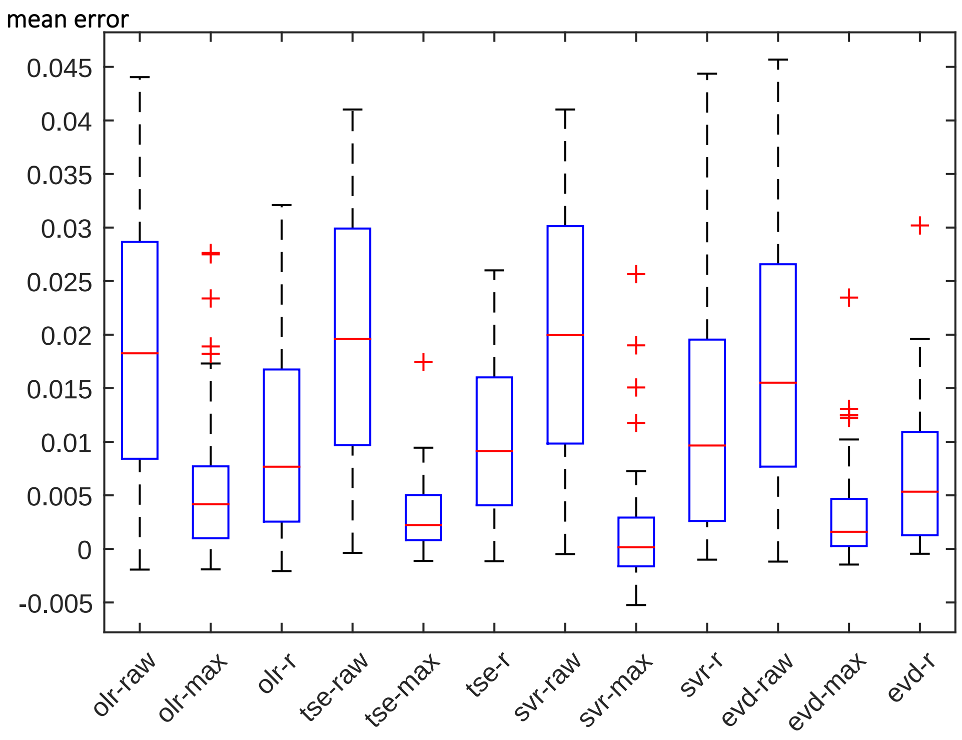


Figure 3.6: Box plot of mean trend estimation error

identification methods have an explicit objective to identify trend in extremes. If raw data are used as the training set, the trend of the extreme values will be overwhelmed by the data points that have no trend in them. Actually as it is observed during the experiment, $\hat{k}$ for any method with any magnitude of trend is approximately 0, i.e., no trend can be found.

If block-maxima and r-largest are further compared, it can be seen that even considering outliers, block-maxima still performs much better than r-largest with all four methods. To measure the improvement of block-maxima to r-largest, pairwise comparison for each identification method are made. Specifically, minima, mean, median, maxima and standard deviation are compared across all *-max* and *-r* methods. The result is shown in Table 3.5 (all numbers in the table are amplified by 1×10^3).

From the table, it can be seen the minimal errors are roughly the same, except *svr-max* which has slightly larger error. From *olr-max* and *olr-r*, it can be seen that *olr-r* has 85% larger median, 69% larger mean and 16% larger maximal error. For *tse-max* and *tse-r*, these values are 310%, 223% and 49%. *svr-max* outperformed *svr-r* with 69.5% improvement in mean and 1.87×10^{-2}

Table 3.5: Mean Error of $\hat{k}$ for Block Maxima and r-largest

	min	median	mean	max	σ
olr-max	-1.91	4.16	6.31	27.64	7.21
olr-r	-2.07	7.68	10.59	32.10	9.31
tse-max	-1.12	2.23	3.07	17.45	3.27
tse-r	-1.15	9.14	9.91	26.00	7.72
svr-max	-5.24	0.15	1.60	25.65	5.71
svr-r	-1.00	9.65	12.72	44.36	12.74
evd-max	-1.46	1.60	3.40	23.47	4.75
evd-r	-0.45	5.34	6.86	30.20	6.77

less in maxima. Considering the original trend which has a magnitude of 10^{-2}, this is a significant improvement. Finally for *evd-max* and *evd-r*, a similar conclusion is obtained: *evd-max* is about 100% better than *evd-r* in terms of mean error, and 6.73×10^{-3} less in maxima.

As a conclusion, data pre-processing can significantly improve identification performance, compared with using raw data. It can be seen that, in this particular data set and block size, block maxima performs the best.

3.9.4 Impact of Variations in Data Set

As part of the evaluation, the impact that the magnitude of trend would have on the performance of the methods is studied. In these data sets, there are five subgroups of data set, each of which has a distinct trend, ranging from 0% to 4%.

To make this comparison, mean trend errors of each method with all data sets are plotted as individual lines in Figure 3.7. The x axis represents the index of data set, and the y axis is the mean error of estimated trend for all predictions in that data set. From the figure it can be seen that mean errors tend to increase when the magnitude of trends increased. This can be clearly seen from the peaks of mean errors. It can also be seen that in each subgroup of data set, there exists a large variation between individual data sets. This suggests that estimation error is highly correlated to data set.

As a conclusion, estimation error is sensitive to data. When the magnitude of trend is increasing, the error will be increased in proportion. All of these methods are sensitive to the actual characteristics of the data set. From

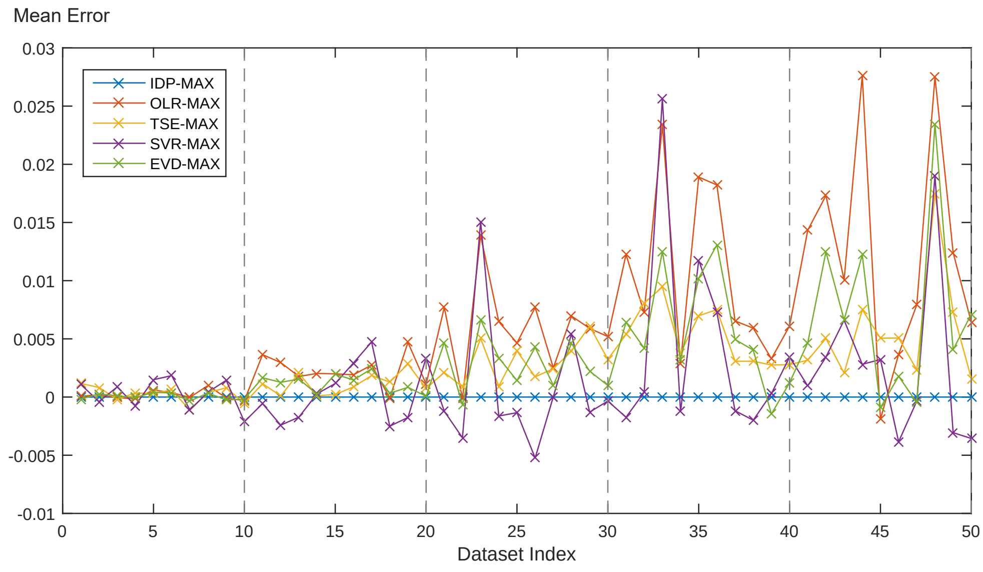


Figure 3.7: Estimated trend error of each data set (block maxima)

Figure 3.7, it can be seen that different methods have very similar patterns in terms of peaks and minimal. This indicates that these methods are sensitive to data sets, but in a similar way, so the characteristics of data set will not break the fairness of comparison as the same data sets are used across all methods. However, a large number of data sets should be used to average the effect of data set and reveal the actual performance of each method.

3.9.5 Comparison of Identification Methods

From previous sections, it is already shown block maxima performs the best among all data pre-processing methods. In this evaluation, trend identification methods are compared, and only block maxima will be considered in this section.

To compare the effectiveness of trend identification methods, an important index is the ability to detect trend. In this work, this is measured by two factors: the validation of an estimation, and the positiveness of a control action. A full diagram that shows relative performance is shown in Figure 3.8. Each bar plot shows a different metric of all four methods, plus the Ideal Predictor (IDP), separated by data set subgroups. For plots of valid, correct and true positives, data is normalised to [0,1] by IDP, while for plots of invalid, incorrect, false positive/negative, data is normalised by the worst method within the same subgroup.

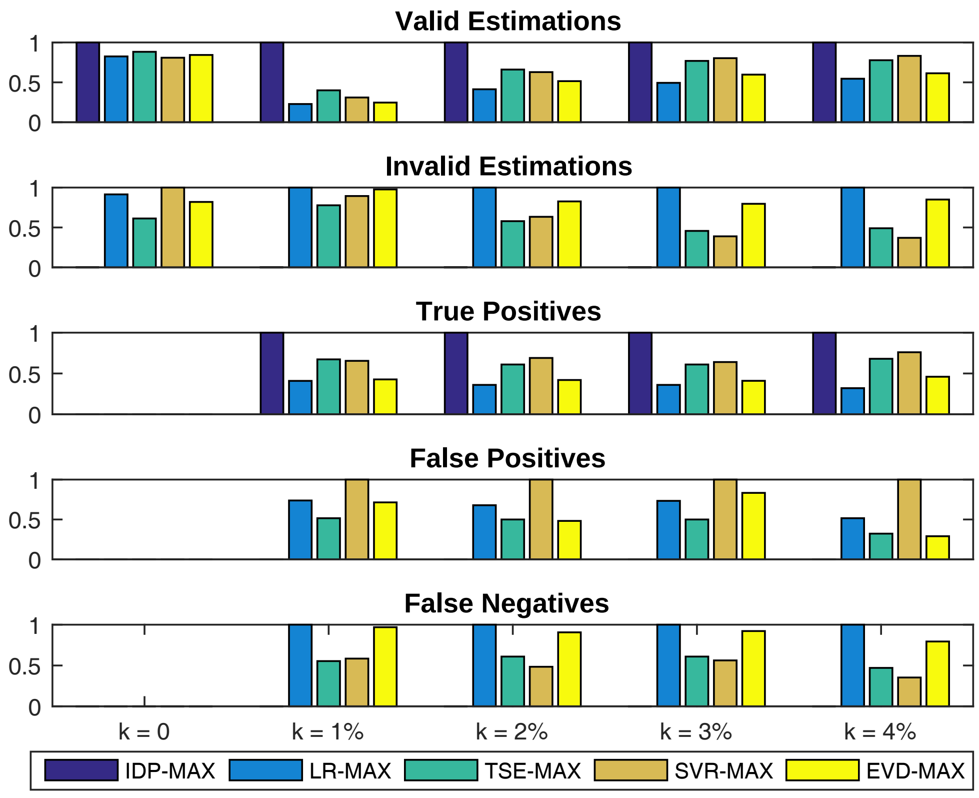


Figure 3.8: Experiment result - false negatives/positives

From the valid/invalid plots in the figure, it can be seen that *TSE* and *SVR* are the two best methods. *OLR* has the largest number of invalids for data set B1, B2, B3 and B4. *EVD* only performs slightly better than *OLR*. If further look at numbers of false positives, it could be seen *SVR* tends to give more false positives, and *OLR* gives more false negatives. All methods give no false positives and negatives when there is no trend in the data. *TSE* is consistent and gives the best performance on average.

To further compare these methods, penalties are summarised that come from the results of all data sets for each method, and this is shown in Table 3.6. From the table it can be seen that *TSE* has least mean penalties with all three data selection methods, comparing with other three methods. This is identical to the conclusion that is described earlier. For methods using block maxima, *OLR* obtained the largest penalty, while for *r-largest*, it is *SVR*.

Additionally detailed functionalities of these four methods are compared, as shown in Table 3.7. In the table, *TSE* is the most computational efficient

Table 3.6: Mean Penalty of All Methods with All Data Sets

	OLR	TSE	SVR	EVD
raw	62	62	62	62
maxima	58.28	29.02	42.26	49.68
r-largest	58.2	53.82	77.58	55.76

method, and it is least sensitive to characteristic of data set. *OLR* is median in computation, but it is sensitive to the composition of data set, and it will be biased if a large percentage of non-relevant data is involved. *SVR* is the only method that needs to provide parameters: the cost C that controls the trade off between errors of the SVM on training data and margin maximisation, and the epsilon ϵ that controls the size of insensitive region. The ability of supporting non-linear trend is supported by *SVR* as well. Other methods can be extended to support non-linearity, but *SVR* directly supports non-linear data by using Kernel functions. In this work, only one variable is considered: time, and it is assumed execution times are only affected by time. In terms of multivariable regression, if additional variable(s) need to be taken into consideration, both *OLR* and *SVR* can give enough support.

Table 3.7: Functional Comparison of Methods

	OLR	TSE	SVR	EVD
Computational Cost	++	+	+++	++
Sensitivity to data variation	+++	+	++	++
Sensitivity to data size	++	+	++	+++
Number of parameters	0	0	2	0
Support non-linear	No	No	Yes	No
Support multivariable	Yes	No	Yes	No

3.10 Summary

In this chapter, the issue of identifying long-term trends in execution times is studied to achieve timing fault predictions. Four different trend identification methods are discussed and their performances are compared. The results suggest that data pre-processing should be used as this can significantly improve estimation performance. It also can be seen that the Theil-Sen estimator, which is a non-parametric method, achieved the best performance in this particular experiment. It is robust against noise and outliers. The other non-parametric method, SVR, is also an outstanding method as it can predict non-linear trends and can be used in multivariable regression. The extreme value distribution does not perform well as it needs a large amount of data to fit the distribution, i.e., a large window size. However, this will decrease the ability of early detection of failures. Finally for OLR, the performance is not satisfactory as the assumption of data set is known to be violated. From the experimental result, it would suggest to use non-parametric methods with either block-maxima or r-largest method.

In the next chapters, two scheduling methods will be discussed that can reduce task utilization by varying control task parameters. The resulting spare capacity can then be made available to the tasks that have been identified as requiring additional resources.

Chapter 4

Period Adaptation of Real-Time Control Tasks

Cyber-Physical Control Systems (CPCS) often contain a number of control functions that require long-lived and non-stop execution. During the operation of a CPCS, considerable knowledge about its execution behaviour can be obtained, which can then be explored to improve its energy consumption and system resilience/robustness. If long-term trends in resource usage are identified, adaptation can be made to accommodate additional computation requirements. In this chapter, an adaptation method that uses online monitoring and model prediction is introduced, which can be used to reduce the resource requirements of the control tasks in a CPCS.

The control-related sub-system of a CPCS is often implemented as several control tasks. Each task is executed periodically with an interval h between consecutive control actions, which is known as the *control interval.* For current practice, the control interval is usually selected based on rule-of-thumb, e.g., $0.2 \leq \omega_0 h \leq 0.6$, in which ω_0 is the closed-loop bandwidth of the controlled plant [12]. However, a real-time controller can generally operate with a wide range of possible intervals, as long as the control performance is satisfactory. As limited resources are shared by multiple functional modules that are integrated on the same platform, it is important to make sure that the CPU resource allocated to control tasks is no more than necessary.

Before a system is deployed in the field, there is only limited knowledge about the system's dynamics and the interactions between different aspects of the external environment etc. Also the uncertainties introduced by scheduling are not able to be accounted for during the design phase, and the worst-case execution time is not always accessible or precise. This means that the system model is inevitably conservative. To improve the situation, the real behaviour of a system can be observed, and the control performance can be assessed at run-time. If there is evidence that control performance can be safely degraded, system schedulability can be improved by increasing the control intervals of control-related tasks. This released capacity can then be used to:

- cater for non-control tasks in the system that are experiencing evolutionary increases in demand (as discussed in Sections 3.1 and 3.2), e.g. tasks concerned with communication, data processing, system monitoring, diagnosis, fault recovery, decision making etc., or
- save energy in battery powered systems by reducing processor speed, core usage, activation time etc., [86] or
- improve the quality of other aspects of the system that have been implemented using anytime algorithms (see Section 2.3.4), or
- accommodate future system upgrades.

In this chapter, an online method is presented that uses model-based prediction based on a timing model and feedback measurements, to safely degrade control performance in a predictable and managed way. The desired control quality is defined statistically through a degradation degree parameter. It is assumed that one or more control tasks are initially running with conservative periods which are then gradually increased in small and controlled minor steps. During this adaptation process, performance predictions are made in advance, and a step change in period is only made if sanctioned by the predication. The consequence of the period change is monitored and feedback is used to improve the predictor.

The adaptation process is undertaken over a much longer time granularity than the system's control processes, e.g., hours, days or weeks compared to tens or hundreds of milliseconds in the later case. It is assumed that the embedded platform of the CPS has communication channels that are linked to a more powerful computational resource (e.g. a cloud facility) on which the modelling and prediction aspects of the adaptation process are undertaken. The adaptation iterates until no further changes can be sanctioned – although monitoring (with the potential to make further modifications) continues indefinitely.

As each period change is small and predictions are utilised, minimal disruption will be created to the ongoing operation of the system. The method proposed here is especially useful for applications that need to run for a long time and need non-stop operations, and for optimization-based controllers [107] [106] as they typically require a large amount of computation, and hence changes to task periods can release considerable capacity.

This chapter is organised as follows: background of real-time control is reviewed in Section 4.1. A general overview of the adaptation method is given in Section 4.2. The performance prediction and the run-time system support is discussed in Section 4.3 and Section 4.4, respectively. In Section 4.5 an evaluation based on a control-scheduling co-simulator is made to demonstrate and verify the effectiveness of the proposed method, followed by a discussion on some aspects of the approach in Section 4.6. Finally, a summary of the work and conclusions are given in Section 4.7.

4.1 Real-Time Digital Controller Implementation

A real-time control task is the entity that executes the software implementation of a digital feedback controller. In Section 2.4.1, the structure of a feedback controller for computer systems have already been reviewed. A more general structure of feedback control with a digital controller in the loop is given in Figure 4.1. To implement such a controller on an embedded platform, the control functions need to be abstracted into individual tasks and a few considerations need to be made. In this section, the implementation aspect of a control task is explained in detail.

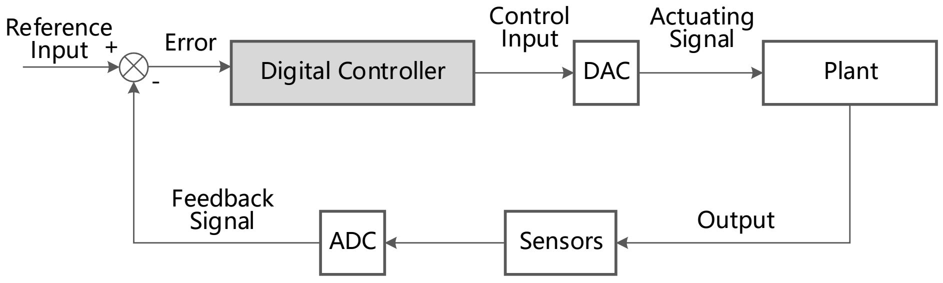


Figure 4.1: A digital controller in a feedback control system

To illustrate the problem, the timing of a control task's execution is firstly analysed in Section 4.1.1. Due to the scheduling effects of the real-time kernel, the timing behaviour of a control task is not exactly the same as the ideal periodic case, which will affect control performance. It is then demonstrated the relationship between control interval and control performance in Section 4.1.3, i.e., how would the changing of control task's period affect specific performance metrics.

4.1.1 Control Loop Timing

In the control community, it is often assumed that sampling and control are performed equidistantly and simultaneously with a fixed interval. However, as a digital controller is running on a computer, the controller will behave differently than the ideal periodic execution according to the applied scheduling algorithm.

Consider a system with three control tasks that are scheduled by a fixed priority scheduler (FPS), in which tasks are scheduled preemptively according to their priorities (a smaller task index indicates a higher priority). A timing diagram of this case is shown in Figure 4.2. Due to the nature of sharing resources in a multiprogramming environment, the timing of a control task is not fully deterministic. Among all three tasks, task 1 has the highest priority and hence is not suffering from interferences. However having the lowest priority, task 3 has the largest interferences and jitter. This example shows that in a multiprogramming environment, a control task without the highest priority will be occasionally preempted and suffer interferences from higher priority tasks in the same system. This process will introduce artefacts such as sampling jitter and control delay which will affect control outcomes [48] [11] [8].

To explain the details more, an illustration of the timing of a single control task is given in Figure 4.3. In the diagram, h_i is the task period, $\tau_{s,j}$ is jth sampling delay and $\tau_{\mathtt{io},j}$ is jth input-output latency. The sampling delay and jitter can be eliminated by using a programmable ADC that is synchronised to the task period. However, the scheduling-introduced input-output latency, or control delay, cannot be precisely predicted as it could be different for each job instance. From the scheduling point of view, if $\tau_{s,j} = 0$, this is equivalent to the control task's response time, which consists of its own execution time, the interference time from higher priority tasks and the blocking time from lower priority tasks.

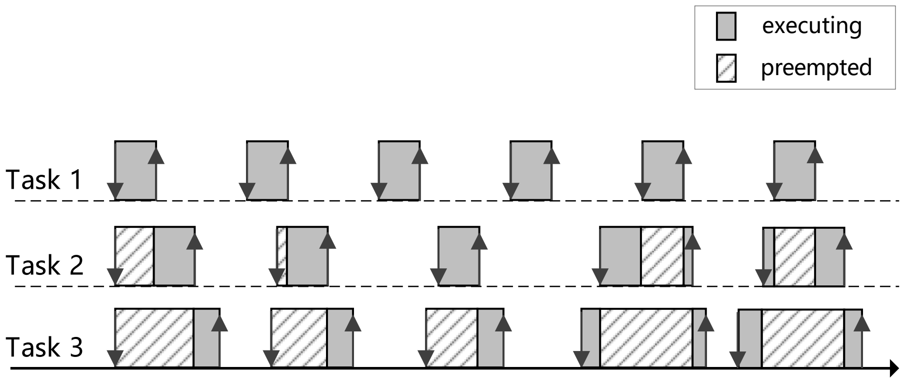


Figure 4.2: Task timing of multiple tasks scheduled by FPS

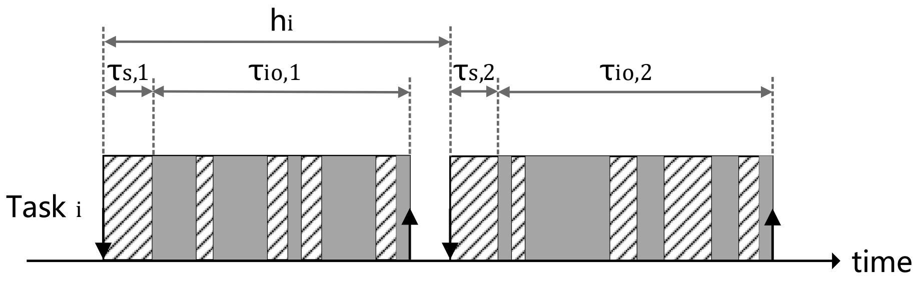


Figure 4.3: Control task timing (single task)

The influence of timing variations is dependent on the controlled plant and the controller. Some control systems are robust towards variations in control interval and latency, but others are less tolerant. A study on the influence of jitter on control performance is described in [49], where the notion of 'jitter margin' is introduced. Scheduling-introduced effects can also be analysed using Jitterbug [92] [47]. Overall, it is hard to analytically work out the exact effects on control that are introduced by task scheduling.

4.1.2 Control Performance Evaluation

To analyse the performance of a real-time controller, some numerical performance metrics need to be defined. There are many commonly used criteria for evaluating the performance of a designed controller, for example, percentage of overshoot, settling time, deviation from a reference output, control error variance, etc. [116]. A Performance Indicator (PI) can be seen as a numeric evaluation of the performance of the target control system. It should be able to reflect the true quality of a controller running under different conditions. Ideally, it should satisfy the following properties:

- it can be quantified and is numerically comparable;
- it can be normalised;
- it can reflect relative long-term behaviour;
- it is insensitive to initial states and noise.

Control error, i.e., the difference between the desired and the actual output, is a straightforward measure of the instant performance. However, this metric has large fluctuations as each of its evaluations is dependent on the current system state. To smooth short-term variations, some form of integral error is applied to evaluate controllers, e.g., integral of absolute error (IAE), integral of time-weighted absolute error (ITAE), integral of squared error (ISE), etc. In some circumstances this integral error is known as control cost.

Note that a higher control cost means worse performance. When the cost is measured at run-time, there could be inconsistent variations due to scheduling and system noise. Thus a range of possible costs could be observed, that can be represented as a distribution. This will be revisited later in Section 4.2.

4.1.3 Control Task Period

The control performance, or Quality-of-Control (QoC), of a digital control system is largely affected by the sampling period of the controller task. In the work of [119], it is shown that the performance of a digital controller has a monotonic decreasing relationship in regards to the control period. This claim is generally true for most systems, although a counterexample is given in [10] for a non-inverted pendulum.

The selection of a control task's period depends on the dynamics of the controlled plant, the desired control performance, and the resources that are available on the hardware platform. For a given digital control system, the quality-of-control is not a linear function of period. If the control interval is sufficiently small, increasing the control interval will have a negligible change on the control performance. However, as the period keeps increasing, the control performance will drop more significantly.

From the control point of view, running a controller faster (i.e., with smaller control intervals) would normally increase the control performance, as a higher control rate allows the dynamics of the system to be properly sampled and handled, and it makes the discrete-time implementation perform closer to its continuous-time counterpart. However, from the scheduling point of view, running a task too frequently could introduce more interferences to lower priorities tasks, making the system more difficult to schedule and reducing the system's resilience to timing faults. In order to make an optimal use of CPU resources, it is important to schedule control tasks in a way that could satisfy control performance as well as the timing requirements of other tasks in the system.

It is often hard for a control engineer to determine which is the right period to use at the system design stage. Following a rule-of-thumb or experimental simulations is good common practice. However as no scheduling effects are considered, the selected period could be too pessimistic and thus waste system resources. Ideally, a period should be adequate to satisfy control performance under required specifications, whilst using the least CPU resources. In the following sections, an adaptation method will be discussed that attempts to deliver this behaviour after deployment.

4.2 Method Overview

This work is identified as control-scheduling co-design [10] [7]. The research on co-design focuses on integration of design of control and scheduling systems, in which scheduling efforts are considered explicitly in the design process of real-time controllers. Resource constraints are also considered during the design of a digital controller.

This work is a form of feedback scheduling (Section 2.6), in which the scheduler has the ability to monitor system states, and to perform corresponding actions by adjusting scheduling parameters (e.g., task attributes such as task periods, execution times, deadlines and priorities). This work focuses on changing task periods only.

In contrast to optimising control periods using off-line analysis [119] [116], the method used in this work is an online adaptation method. The philosophy applied is to tune control task periods gradually and slowly, in which changes are made across a large time span, e.g., hourly or daily. This makes the approach in this work much less dynamic than other feedback scheduling methods, in which a change is basically made every tens or hundreds of milliseconds [66]. This work is also related to graceful degradation [122], in which planned and pre-designed degradation is made in order to avoid serious system failures.

As in each adaptation cycle only a small change is applied and the consequence of the change is also observed and considered, the method in this work is less aggressive than some of the existing adaptation methods, for example, state-aware and resource-aware feedback [70] [103] [96].

4.2.1 System Structure

In this system, it is assumed that each element of the physical plant is controlled by an individual control task executing on an embedded computer, which has connectivity to a more powerful machine 'in the cloud'. All tasks are executing concurrently and independently. Each task is responsible for sampling, updating system state and calculating control signals.

Figure 4.4 shows the basic structure of the proposed method. The system is composed of a server and one or more clients. The client/server structure distributes the computational load that is required, as the server has much more processing power than the local embedded computer. The traces of control and scheduling performance are measured at the local system with a monitor module, and transferred to the cloud server for processing and analysis. The planner on the cloud will make a decision if a longer period can be applied, based on a model-based predictor and the run-time observations. The observed data will also be used to update the prediction model which forms a feedback loop.

In order to apply this method, there are some general assumptions on the properties of the deployed system:

1. The embedded computer (local system) consists of a uniprocessor and a preemptive scheduler using fixed-priority scheduling (FPS).

2. Control tasks are released periodically, i.e., the system applies time-triggered rather than event-triggered controllers.

3. All tasks in the complete system are initially schedulable, given the control tasks are using periods that can satisfy control specifications.

4. The system has the ability to monitor task execution and response times, and control outputs. The kernel has the ability to change task periods

at run-time.

5. The system itself has limited resources but has connectivity to a more powerful cloud computer.

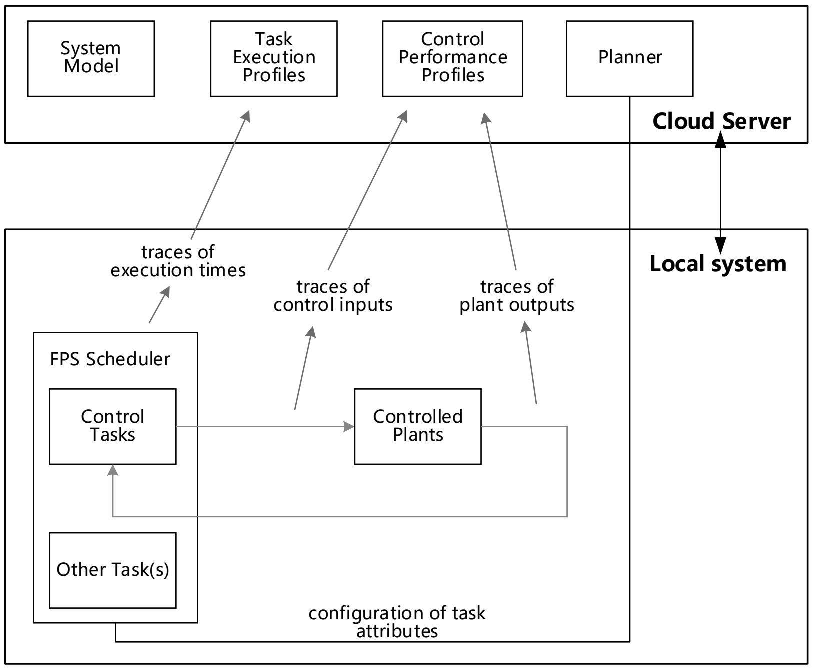


Figure 4.4: Block diagram of the proposed adaptation method

Assumption 1 is based on the fact that single core systems still form the majority of embedded control applications, and FPS is now extensively supported by real-time operating systems. Assumption 2 is valid as time-triggered periodic control is still the mainstream. Event-based control [6] is still an open research topic and more theoretic work is needed. Assumption 3 is necessary, because if the initial periods cannot satisfy the system requirements, increasing a period will only deteriorate the situation. It is notable that more conservative initial periods will lead to more potential of relaxing periods later on. For Assumption 4, the support of task execution time monitoring is already available in POSIX compliant run-time kernels. However, in order to change task attributes at run-time, some modifications may need to be applied, as this capability is not universally available in every real-time kernel. Finally, for Assumption 5, as the prediction and analysis need to be executed on a more powerful machine, a form of network connection to a local server or a cloud service provider is needed. There are many off-the-shelf commercial cloud service providers, e.g., IBM Bluemix, Amazon Web Services (AWS), Google Cloud or Microsoft Azure. In CPS, many monitoring and diagnosis functionalities have already been implemented on the cloud, so using a cloud would not necessarily increase the cost in terms of operation and maintenance. Note the link to the cloud does not need to be reliable or time predictable.

4.2.2 Task Model and Problem Formulation

Given a control application that is represented as a task set $\Gamma = \{\Gamma_c \cup \Gamma_{nc}\}$, in which Γ_c is the subset of all the control tasks and Γ_{nc} represents the subset of non-control-related tasks. For each control task $\tau_i \in \Gamma_c$, a flexible task model is used [52], of which the task period is a variable parameter. A task is defined, using the normal symbols as $\tau_i \equiv \{C_i, T_i^0, T_i, D_i\}$. T_i^0 is the initial period as well as the lower boundary on the period, and T_i is the current period. The schedulability of the initial task set is checked through response time analysis by using optimal priority assignment based on the initial periods.

The control aspect of a system can be represented as time-domain differential equations that describe the relationship between the control signal inputs and the system response. Define the controlled plant of control task τ_i as P_i. The dynamic model of P_i is represented in the standard state-space form:

$$\begin{cases} \dot{x}_i(t) = A_i x_i(t) + B_i u_i(t) + \omega_i(t) \\ y_i(t) = F_i x_i(t) + e_i(t) \end{cases} \tag{4.1}$$

in which $x_i(t)$ is the system states vector, $\dot{x}_i(t)$ is the first derivative of x_i, $u_i(t)$ is the control input, $y_i(t)$ is the system output; A_i is the system dynamic matrix, B_i is the input matrix, F_i is the output matrix; $\omega_i \sim \mathcal{N}(0, \sigma_\omega^2)$ is system process noise, and $e_i \sim \mathcal{N}(0, \sigma_e^2)$ is measurement noise. The symbol $\mathcal{N}(m, n)$ means normal distribution with mean m and variance n.

The optimization objective is to minimise the overall resources used by the control tasks under given control quality constraints, which is formulated as follows:

$$\begin{aligned} \underset{T_i}{\text{minimise}} \quad & \sum U_i = \frac{C_i}{T_i}, i \in \Gamma_c \\ \text{subject to} \quad & \frac{PI_i(T_i)}{PI_i^0} \geq 1 - \lambda_d(i), i \in \Gamma_c \end{aligned} \tag{4.2}$$

The Performance Index, PI, is defined as an inverse of a control cost J, i.e., PI $= 1/J$. The performance index is only used to evaluate the performance of a single control task running at different periods, rather than comparing across multiple control tasks. With the PI, the performance of period T_i and an arbitrary period T_i' can be compared.

The parameter λ_d is the degradation factor, which is defined as the fraction of the expected performance at a desired period, $PI_i(T_i)$, and the ideal expected performance PI_i^0 when $T_i = T_i^0$. This introduced design parameter is used to make trade-offs between task utilization and control performance. It is important that the performance index should be comprehensive and should also be a monotonic decreasing function with regard to the task period. By defining the degradation factor, the system designer can control the tolerance of QoC (Quality-of-Control) degradation as a consequence of manipulating periods. The subscript i in Equation (4.1) and Equation (4.2) will be omitted if there is only one control task.

4.3 Performance Prediction

It is important for the system to determine the consequence of applying a new period, and making advance predictions is a straight-forward way of estimating such influence. The adaptation relies on a performance prediction process, which is done through a model-based performance predictor using a Monte Carlo model that runs in the cloud server. The predictor has the ability to predict the performance distribution of a given digital controller when operating at a particular rate with a given task model. Monte Carlo is a method for evaluating a model that is complex, non-linear, or involves more than just a couple of uncertain parameters. Monte Carlo approximates results (i.e., the predictions) from a large number of repeated experiments through random sampling.

In this work, a Monte Carlo method is used for analysing how task scheduling uncertainties and variations would propagate to affect control system performance. It is used as the original control problem is hard to solve by a deterministic and analytical calculation. From a hybrid system point of view, each control action introduces a jump - i.e., a sudden change in system dynamics. Although control jobs are released periodically, the actual execution and outputs are not equally distributed in time. The overall control output is therefore the consequence of the contributions of multiple control job releases – as illustrated in Figure 4.5.

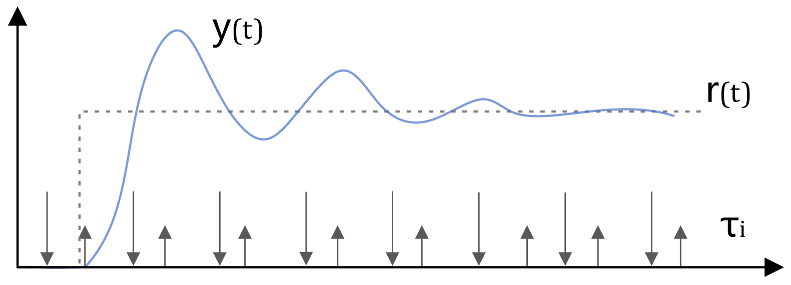


Figure 4.5: Timing of a control task. The control job releases periodically, but the control delay is different case by case, which will affect control performance. In the diagram, $y(t)$ is the control output and $r(t)$ is the expected output (reference).

4.3.1 Monte Carlo Predictor

The overall predictor structure is shown in Figure 4.6a. The predictor is formed of a simulator, a system dynamic model, a task set model, and a correction model to generate performance profiles. The Monte Carlo simulator module is a hybrid system that is formed of a discrete model and a continuous model, which are shown in Figure 4.6b and Figure 4.6c, respectively.

The discrete model is formed from a timed finite state machine, and each of the symbols is explained as follows:

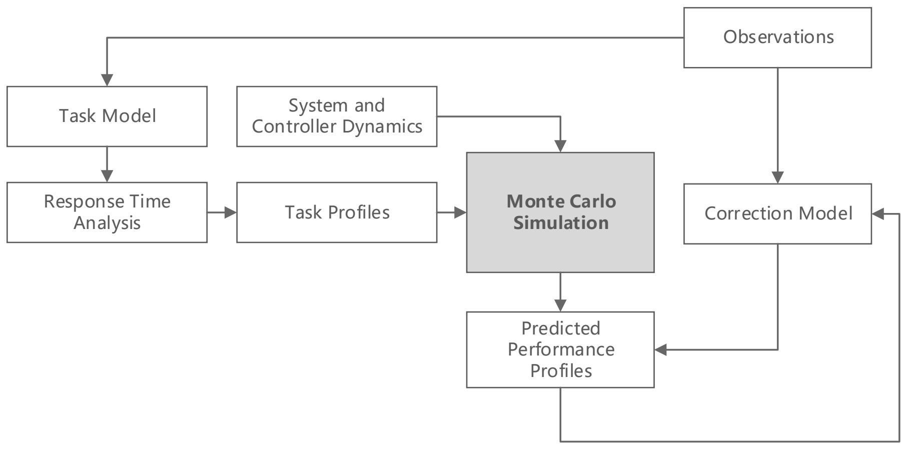
- t_1: is the delay due to phasing of the first released job after the operation point is changed. $t_1 \in (0, T_i)$. The worst-case is when the operational point changes right after the task is released. In this case, the control task will only be aware of the change after its next release.

- t_2: is the execution delay after the task is released due to interference from higher priority tasks.

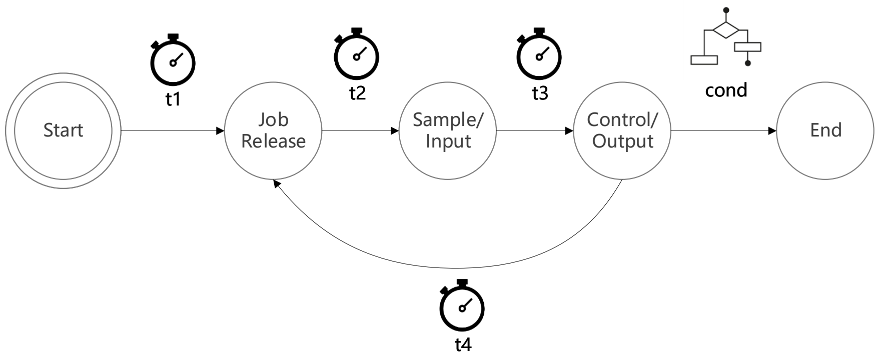
- t_3: is the input-output latency. This is partly from task interference and partly from task execution. Both t_2 and t_3 will result in controls that are not equal-distanced.

- t_4: the delay for the next release of the job: $t_4 = T_i - t_2 - t_3$.

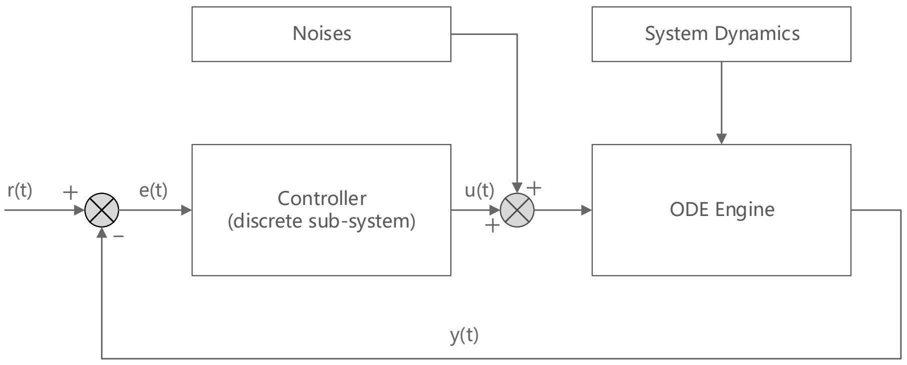
- *cond*: is a conditional to check if the simulation has finished. This termination criterion is either the system has reached 5% of steady state or the maximum allowed simulation time has passed.



(a)



(b)



(c)

Figure 4.6: The Monte Carlo Predictor: (a) An overview of the structure. The Monte Carlo Simulation module consists: (b) discrete sub-module which simulates a periodic task and generates control inputs; (c) continuous sub-module that takes control inputs and generate responses using the system dynamic model.

The continuous module simulates control system dynamics with an ODE solver. It takes inputs of control signals, and produces system response as output, which can then be used to calculate the performance index. The system dynamic model that is used can either be obtained from first principles or from empirical modelling. If the system model is in the form of a transfer function, it will be converted into a state-space model.

During the process of the simulation, the continuous model receives control inputs and sends plant outputs to the discrete model, while the discrete model decides when and how to update the input signal, according to the states of the discrete timing model. Collaboratively, these two subsystems simulate the real run-time behaviour of the digital controller.

4.3.2 Predicting Control Performance

The performance of a designed controller is quantitatively evaluated by control cost J. In this work, the Integral of Absolute Error (IAE) is used to describe the cost. It has the general form:

$$\begin{aligned} J &= \int_0^{\infty} |r(t) - y(t)| \cdot dt \\ &= \int_0^{\infty} |e(t)| \cdot dt \end{aligned} \tag{4.3}$$

The control error, $e(t)$, is defined as the difference between the desired reference $r(t)$ and the actual output $y(t)$. For practicability, the error is integrated from $t = 0$, when the reference starts to change, to $t = tss$ when the system is in steady-state (after which the margin of control error is within, for example, 5% representatively), so:

$$J = \int_0^{tss} |e(t)| \cdot dt \tag{4.4}$$

The integral operation smoothes the fluctuations in the system output due to short-term transients of system states. However, as there are variations due to scheduling and noise, each measure of J could be different. In this case, multiple runs of the simulation can be done to obtain a distribution for J. As there is no evidence to prove that the J distribution would follow a certain category of parametric distribution, the cumulative distribution function (CDF) is used to explain the data. Depending on the conservativeness of the requirement, the expectation of performance, $\bar{J}$, is determined from:

$$\bar{J} = \mathbf{E}[J] = \{X|\mathbf{cdf}(x < X) >= \alpha_d\} \tag{4.5}$$

in which $\mathbf{cdf}(\cdot)$ is the cumulative distribution function, and α_d is a decision threshold with $\alpha_d \in 0\ ..\ 1$ (with a typical value of 0.95). Note that the definition of expectation in this context is slightly different from the traditional explanation, which is to describe the average output. While in this case, more extreme cases will also need to be considered.

4.3.3 Prediction Refinement by Error Correction

Ideally, the predictions from the Monte Carlo model are expected to match the measurements of the actual system. However in reality, there are many factors that would affect the accuracy of the prediction, such as modelling error in the system dynamics, random processes and measurement noises, incorrect assumption of the response time distribution, and integral error due to numerical approximation. This will lead to imprecise and sub-optimal predictions, or even cause the adaptation process to fail.

Many of these errors are impractical to be directly measured or modelled. It is also difficult to analyse how these factors would translate into errors of performance measurements, even if the error source is identified. As a consequence, a correction model is proposed to refine the predictions in order to handle the errors and improve the utility of the prediction. It is assumed that the predictions ($\hat{J}$) are biased by a factor β with the addition of zero-mean Gaussian noise $\epsilon \in \mathcal{N}(0, \sigma^2)$. Hence the actual performance measure (J) is given by:

$$J = \hat{J} + \beta + \epsilon \tag{4.6}$$

In particular, the bias β parameter is estimated with the following criterion:

$$\begin{aligned} \underset{\beta}{\text{minimise}} \quad & D_n = \sup_x \left|\mathbf{cdf}_J(x) - \mathbf{cdf}_{\hat{J}}(x)\right| \\ \text{subject to} \quad & \forall x : \mathbf{cdf}_J(x) > \mathbf{cdf}_{\hat{J}}(x) \end{aligned} \tag{4.7}$$

in which $\mathbf{cdf}_J(\cdot)$ is the cumulative distribution function of the control cost, $\mathbf{cdf}_{\hat{J}}(\cdot)$ is the cumulative distribution of the estimated control cost, and D_n is the Kolmogorov-Smirnov (K-S) statistic [55] [134], which is the maximal distance between the CDFs of the two distributions. This criterion makes sure the predictions are more conservative than the actual measurements. It is assumed that the prediction error is sustained when making predictions for a small period change, as the main error sources are independent of task period.

4.4 The Run-Time System

The Monte Carlo simulation introduced in the previous section is executed in the cloud, which significantly reduces the computational load on the local system. In order to achieve adaptation, there is also a need for run-time support on the local computer. The run-time system is formed of two modules: monitor and executor. The monitor module runs on the client side. It is the process for collecting system observations and performing basic conformance analysis. The executor module communicates with the cloud, and is responsible for uploading observed traces, accepting adaptation decisions and informing the kernel to make changes.

The overall flow of the adaptation system is given in Figure 4.7. Unlike off-line period assignment methods, the period in this work is updated in multiple iterations. Each iteration only applies a small change to the period. The general work flow of this adaptation process is explained in the following steps:

1. Assuming the control task is running with its initial period T_i^0. When a new request of target utilization is received, a plan is made for changing the current period to the required period, by dividing the objective into fixed small step changes.

2. Based on the system model, a prediction is made by the Monte Carlo Predictor for the new period $T_i' = T_i + \Delta T_i$, in which ΔT_i is the step size.

3. If the predicted performance can satisfy the performance requirement defined by the system, the new period is passed to the scheduler, and the scheduler will change the period of the control task. However, the saved resources will not be immediately available to other tasks.

4. An evaluation phase is then involved to monitor if the system running at the new period can satisfy the performance requirements. If yes, the saved capacity $(U_i(T_i) - U_i'(T_i'))$ will be committed and can be used by other tasks in the system. If no, the task's period is returned to its previous value.

5. The prediction model is updated based on run-time observation.

6. Repeat the process until *a)* the required utilization is satisfied; or *b)* the control performance has reached its bound (i.e. future changes are not sanctioned or are rejected once evaluated on the plant).

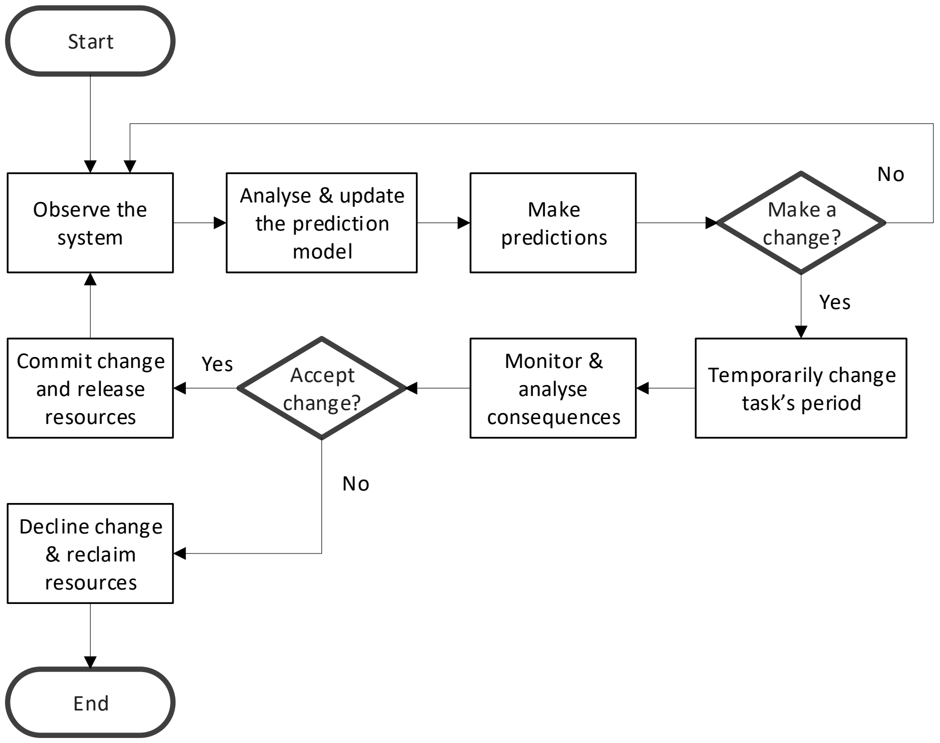

Start
Observe the system
Analyse & update the prediction model
Make predictions
Make a change?
No
Yes
Temporarily change task's period
Monitor & analyse consequences
Accept change?
Yes
No
Commit change and release resources
Decline change & reclaim resources
End

Figure 4.7: Flowchart of the proposed method.

The system monitor is implemented as a normal periodic task. It periodically makes performance measurements and checks if the actual performance is violated. If appropriate, it will raise exceptions to terminate the adaptation process. This module also checks if any task runs longer than the expected worst-case. In case of an anomaly, the system will roll back to use the previous period and reclaim resources that are not committed. This is to make sure the new period could be safely and permanently applied.

4.5 Evaluation

To demonstrate the effectiveness of the adaptation method, an illustrative example is given that uses a second-order system and a task set consisting of one control task and multiple non-control tasks. The effectiveness and robustness is also evaluated by investigating a range of design parameters. Unfortunately no currently available scheme attempts to address the issues identified in this thesis, and hence a comparative study is not possible.

The experiment is based on simulation using MATLAB/Simulink. Simulink is a block diagram simulation environment that supports model-based design and simulation of dynamic systems. The overall setup in Simulink is shown in Figure 4.8. The task scheduler (the A-FBS Kernel block in the diagram) is implemented as a discrete system using the MATLAB s-function in C++, and is called by the Simulink engine during simulation. The scheduler uses standard fixed-priority scheduling, and the deadline-monotonic policy is also used for task priority assignment.

In terms of the controller, a Linear-Quadratic-Regulator (LQR) controller is used. The control law of a LQR controller for reference tracking is defined as: $u(t) = \overline{N}r(t) - Kx(t)$. The optimal control gain K is calculated using a continuous-time model by solving a Riccati equation [61]. $\overline{N}$ is a precompensator scaling factor so that the output does equal the reference in steady state. The overall system closed-loop dynamic equation in a state-space form is:

$$\dot{x}(t) = (A - B \cdot K) \cdot x(t) + B \cdot \overline{N} \cdot r(t) \tag{4.8}$$

The task set used in this experiment is randomly generated using UUniFast [27], with log-uniform distributed periods. The schedulability of the task set is checked through response time analysis.

4.5.1 Demonstration

This experiment is started by evaluating a second-order system with one control task and five higher priority non-control tasks. The system dynamic equation is defined as:

$$\begin{gathered}\begin{bmatrix}\dot{x_1} \\ \dot{x_2}\end{bmatrix} = \begin{bmatrix}10 & 25 \\ -25 & 10\end{bmatrix}\begin{bmatrix}x_1 \\ x_2\end{bmatrix} + \begin{bmatrix}0 \\ 1.6\end{bmatrix} u \\ y = \begin{bmatrix}2.5 & 0\end{bmatrix}\begin{bmatrix}x_1 \\ x_2\end{bmatrix}\end{gathered} \tag{4.9}$$

in which the system has a complex conjugate pole pair: $p_{1,2} = 10 \pm 25j$. The closed-loop bandwidth of the system is 40.72 rad/s, which suggests an initial control period of 10 ms (middle value of the rule-of-the-thumb and rounded

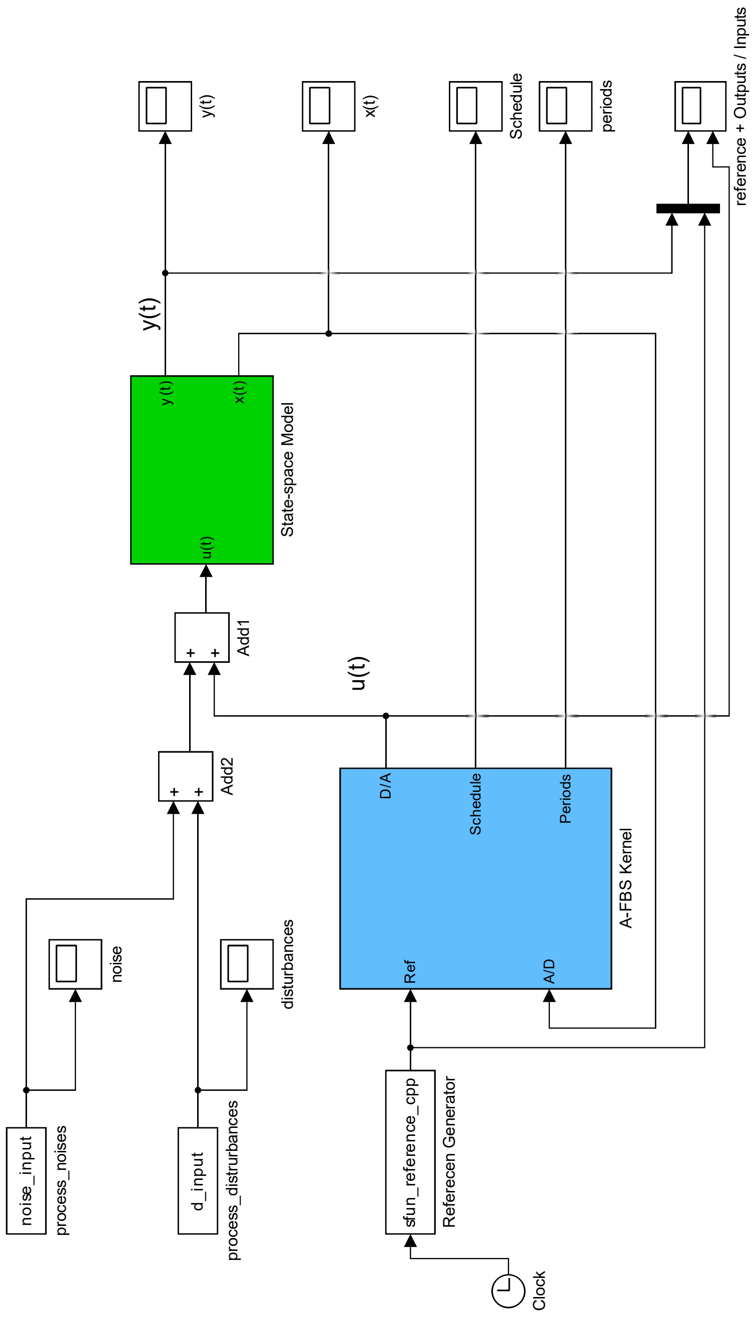


Figure 4.8: Experiment setup in MATLAB/Simulink

to the closest integer). The complete task set with other higher priority tasks in the system is given in Table 4.1. At run-time, these tasks are assumed to have variable execution times which are normal distributed from $C_i/2$ to C_i. The decision parameter α_d is set to be 0.95, and the period change ΔT_i in each step is 1.0 ms, which is 10% of the initial period.

Table 4.1: Experiment Task Set

Task	C_i (ms)	T_i (ms)	Control Task
τ_0	0.42	1.57	
τ_1	0.10	2.15	
τ_2	0.53	4.99	
τ_3	0.87	7.77	
τ_4	0.48	8.01	
τ_5	1.00	10.00	$\surd$

The degradation factor is set to be 0.7 while running the system. For each iteration, the actual system is observed for 1,000 seconds, which will give 400 - 500 measurements depending on the reference signal. The Monte Carlo predictor generates a prediction based on 3,000 randomised task executions and then makes an estimation of the PI for the next step.

The predicted performance is compared with the actual observed metrics in Figure 4.9, and the prediction bias is also shown in Figure 4.10. It can be seen that the predictor made a relatively precise prediction, i.e., the deviations between the predictions and observations are small (less than 10%), when $T_i \leq 30ms$. However, as the period increases, the prediction error is also increased. This is explained as the variation of the performance indices increases dramatically when the control period is becoming larger. Also the extremes that would rarely happen in a real system would still be used to produce expectations in the predictor. This can ensure the conservativeness of the predictor and reduce the chance of invalidation. For this experimental run, the period is terminated by the predictor at 39 ms, which is four times the initial period, i.e., the utilization is only 25% of the initial task utilization.

For an actual system, a degradation of 0.7 may be impractical as the control loss could be too high. To give a full spectrum of the system behaviour, a range of degradation factors are used from 0.05 to 0.70. From Figure 4.11, it can be seen that as the degradation factor increases, the period that the algorithm

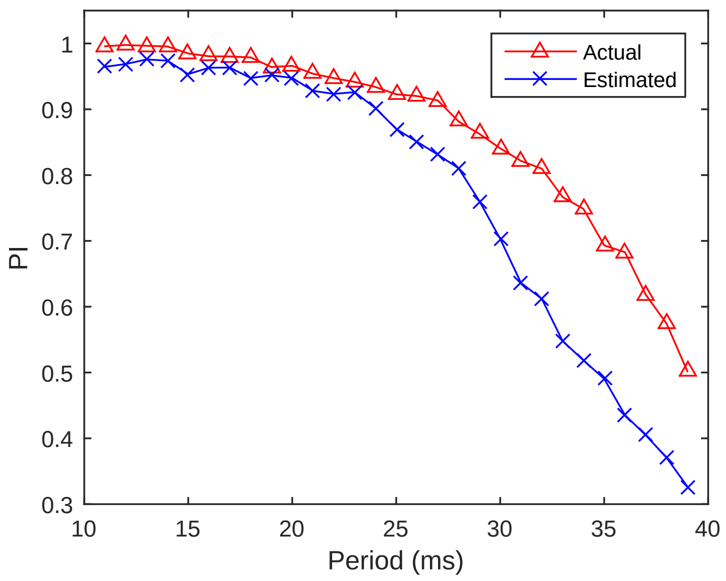


Figure 4.9: Predicted Control Performance v.s. Actual Performance. The prediction is one step ahead of the corresponding actual observation.

terminates at also dramatically increases. For example, if $\lambda_d = 0.1$, the period can be 25 ms, while if $\lambda_d = 0.5$, the period can be 34 ms. It can be seen that the degradation factor is an important design parameter as it determines when the adaptation process will have to be terminated.

To better illustrate the trade-off between utilization and performance, the two metrics against task period are compared in Figure 4.12. It can be seen that as the period increases, the control performance loss is also gradually increased. On the other hand, task utilization is reduced as a consequence of using a longer period. However, the benefit of utilization saved by increasing task period is exponentially decreased, while the penalty to control performance grows quadratically. This implies:

1. Increasing the control task's period could have a great positive impact on scheduling performance in terms of utilization, with just a small amount of control degradation as penalty;

2. However, over-extending the period of a control task could dramatically affect control performance while making limited contribution to utilization saving.

This trade-off is controlled by the degradation factor. The degradation

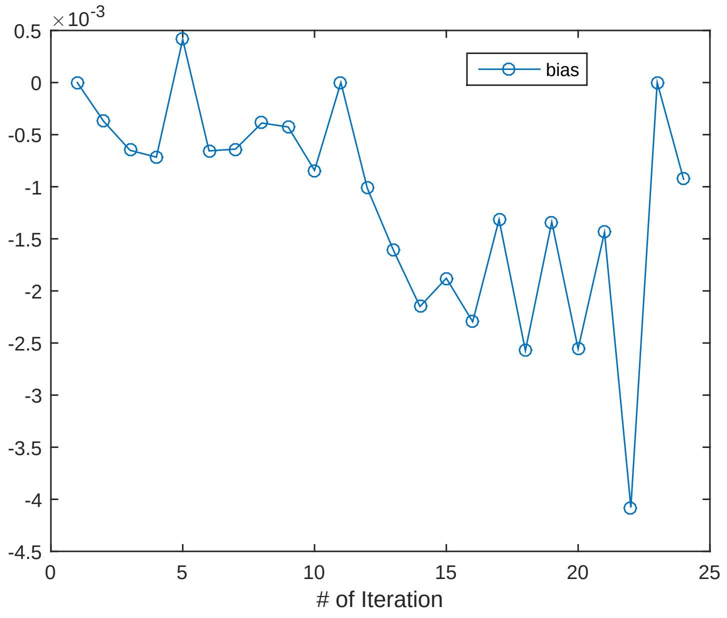


Figure 4.10: Prediction bias estimation. The bias is used as a feedback information to correct the prediction.

factor is an important design parameter in guiding the adaptation process. As it is a relative measure and it is used mainly for facilitating numerical computation, sometimes it is not straightforward to select a proper parameter. In this experiment, it is suggested that the degradation factor would be less than 0.3 to obtain a good trade-off. However, an unstable system is used in the experiment which is more sensitive to period changes. For other systems, the degradation factor would be higher. The selection of a proper degradation factor can be done by off-line simulation with conservative conditions, in which the control responses are examined under varies periods.

4.5.2 Robustness

In the previous demonstration, it is assumed the actual system is identical to the design model. In this experiment, the robustness of the algorithm will be explored by looking at the case in which model mismatch exists. Model mismatch is the phenomenon that the system model is deviated from the actual system. This is common in actual engineering systems, due to many factors such as modelling error, limited knowledge of the system, and simplification of the physical system.

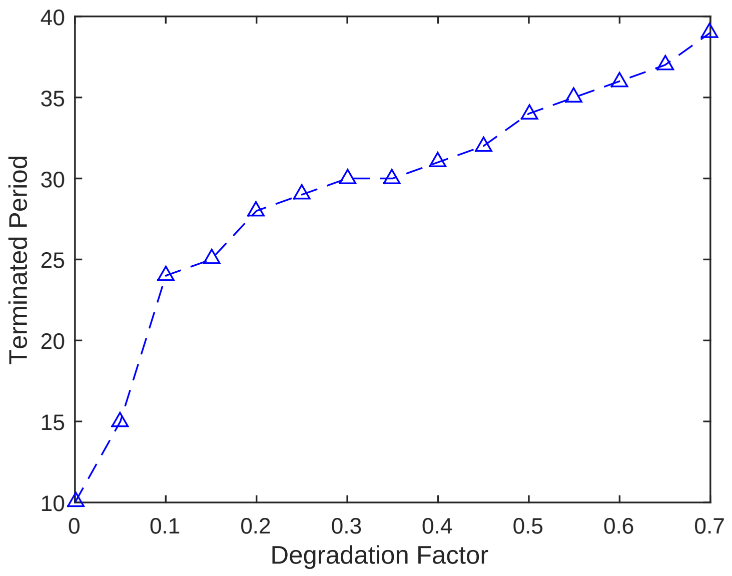


Figure 4.11: Terminated period with different degradation factors. The degradation factor will affect the period in which the system terminates at.

The robustness is evaluated by applying deviations into the actual system, compared to the design model. As model differences exist, the actual system behaviour will be different from expected. This includes both the system dynamic model and the task model.

The experiment configurations are given in Table 4.2. There are overall 7 experiments including a baseline E_1 (the one described above). The 'system dynamics' column in the table indicates how much percentage of error is added to the system dynamic matrix A. The 'task model' column indicates what task model is used in the simulation: 'WCET' for always using the worst-case execution times, 'BCET' for the best-case execution times, and 'Normal' for normally distributed between these two extremes.

The results are shown in Figure 4.13. From the figure it can be seen that the actual system outputs could be very different if there are errors in the original model. The worst-case is E_5 in which both system dynamics and task model are worse than ideal. The best two cases are E_6 and E_7 in which the best-case execution time model is applied. Nevertheless, in all cases, the predictions are more conservative than the actual observations, and it can be seen that the predictor is able to correct itself to accommodate different situations. For example, in E_6 and E_7, as the actual observations are much better than the model, the prediction corrects itself so it will not be too conservative.

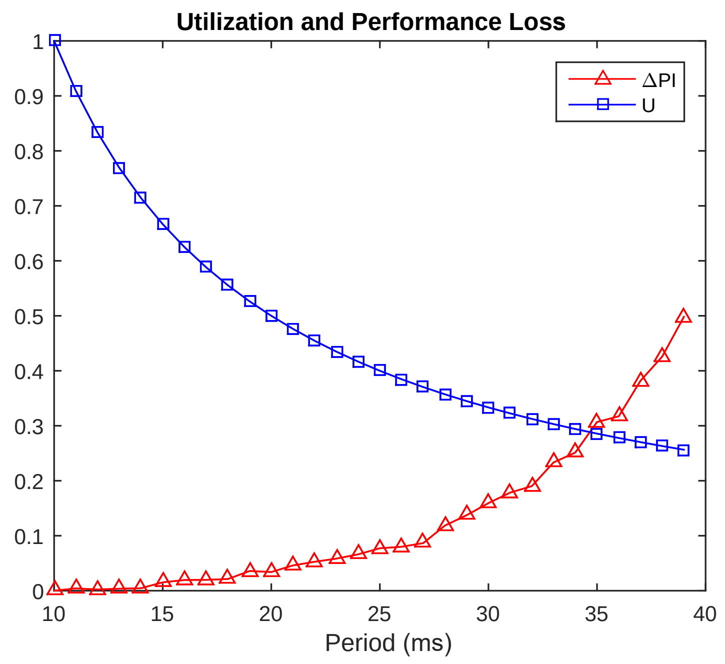


Figure 4.12: Utilization v.s. Performance Index. The utilization is defined in a relative percentage to the original utilization. The performance is illustrate as a relative percentage of loss.

4.6 Analysis and Discussion

In this section, some of the features of the introduced method will be discussed. Also some of the details that were not mentioned in the experiments will be explained.

4.6.1 Influence of Extending Periods

Given the standard response time analysis equation [18], which is (for constrained independent tasks):

$$R_i = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil C_j \tag{4.10}$$

it can be seen that extending the period of a task will not change the worst-case response time of that task. Also, it will have no influence on higher priority tasks. Extending period could, however, potentially improve schedulability of low-priority tasks; also this will not affect the schedulability of the system if the task set is initially schedulable. This is explained as scheduling sustainability [30].

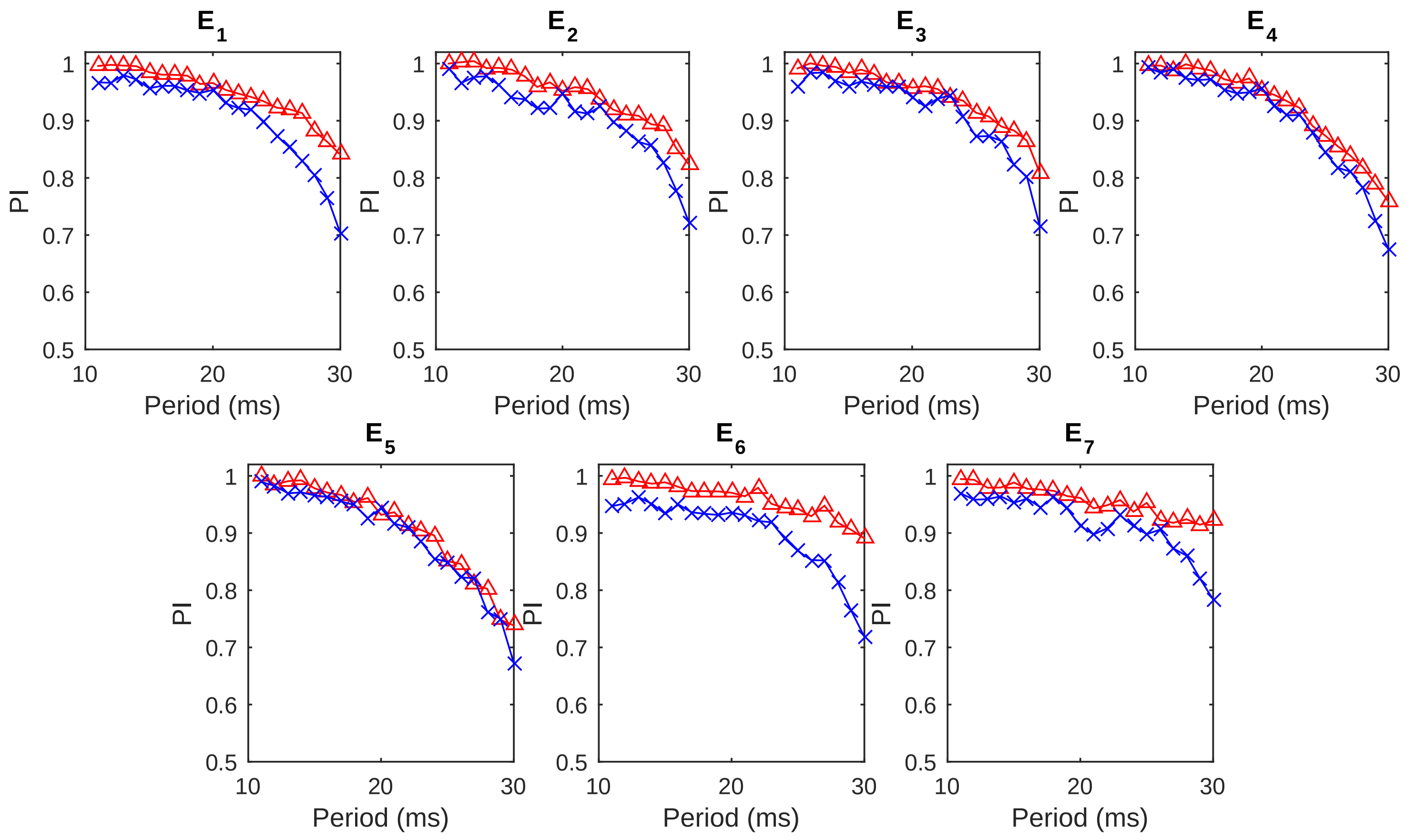


Figure 4.13: Results of robustness evaluation. The blue lines with ‘x’ marks are the predictions and the red lines with ‘$\triangle$’ marks are the actual observations. The experiment number E_n on top of each subgraph is defined in Table 4.2, in which E_1 is the experiment using perfect models.

Table 4.2: Experiment configuration for evaluating robustness

Experiment	**System Dynamics**	**Task Model**
E_1	Ideal	Normal
E_2	-5%	
E_3	+5%	
E_4	Ideal	WCET
E_5	+5%	
E_6	Ideal	BCET
E_7	-5%	

The priority ordering will be unaffected if the priority assignment policy is deadline-monotonic as task deadlines are not changed. However, if rate-monotonic is used (and deadlines change with period), extending the period without changing priorities would change the optimality of the priority ordering.

4.6.2 Discussion on Overheads

Although most of the computation load is distributed to the cloud server, the implementation of the method still requires additional computation and communication in the local embedded computer. In particular, these are:

Computation overhead: the additional overhead of computation mainly comes from calculating performance statistics and run-time monitoring. The process can be run as a low priority background service to create minimum interferences to other running tasks in the system.

Memory overhead: as traces and statistical data is buffered into memory before being sent to the cloud server, some memory storage is required. Depending on the sampling rate and reliability of the communication link, this size can range from a hundred bytes to a few kilobytes.

Communication overhead: the communication overhead is minor. As only packets containing statistical data are transferred to the cloud server, the communication bandwidth required by the method is negligible. Also as the communication is not in the control loop, the real-time and reliability requirements of the network are also low.

4.6.3 Dealing with Multiple Control Tasks

In the experimental evaluation, the case for only one control task in the system is considered. As it is common to have multiple control tasks in an control application, it is essential to prioritise each task for adaptation. To control the influence of adaptation, it is recommended that only one task at a time is in an adaptation cycle and can change its period.

Assuming all control tasks are equally important, control tasks could be prioritised according to one of the following potential policies:

Highest Priority First (HPF) The higher priority task has the largest margin and the highest interferences with non-control tasks, and also changing a higher priority task first can avoid the need for recalculating periods of lower priority task.

Least Sensitivity First (LSF) The task is selected according to the sensitivity of its performance index by changing its period. This is evaluated by a sensitivity function:

$$\Delta J = \frac{\partial J}{\partial h}\Delta h \,\Big|_{h=T_i(k-1)} \tag{4.11}$$

Least Uncertainty First (LUF) Select the controller that behaves the closest to its prediction model, i.e., minimal prediction bias and error.

If the importance, or critical levels, of the control tasks are not equal, it is always preferable to change the task with the least importance first.

4.6.4 Control Stability

The proposed method only checks control system stability through long simulations. In some situations, this may not be appropriate to demonstrate the stability of the system. One way of addressing this issue is to give an upper bound of the maximum allowed period, which could be a value from mathematical formulations or early stage experiments. Analysis of control system stability under random delays is still an open research question and can be found in related research, e.g., networked control systems [131]. In the adaptation process, it is assumed that the period will not be larger than a pre-designed safe boundary, otherwise the request will be rejected. In the simplest case this could be the upper bound of the rule-of-thumb, or this could come from simulations with extreme conditions.

4.7 Summary

In this chapter, an adaptation framework is proposed that gradually changes a control task's period at run-time, in order to compensate for additional computational requests emanating from other tasks in the system. The effectiveness and robustness of the method is demonstrated through multiple experiments. Although only one example system is used, the method itself is generalised and could potentially be applied to other systems. In summary, the following contributions have been made:

- A period adaptation framework is proposed that can accommodate additional computing requirements (or reduce power consumption) in a cyber-physical control system.
- A proposal to use cloud computing as a component in the loop of monitoring and improving control and scheduling performance.
- A decision making method that uses system dynamics and task timing models to make predictions and instigate future actions.
- A scheme which utilises run-time feedback information to improve the precision and robustness of performance predictions.
- The framework is demonstrated in an example system to show its usability.

In the next chapter, this framework will be extended through the use of a novel flexible model with two periods, the dual-period task model. It will be shown how resource saving can be made without compromising control performance.

Chapter 5

Dual-Period Task Model

In the last chapter, an adaptation method that uses a flexible task model is discussed. In that model, each control task has a flexible task period that can be changed on demand. In this chapter, this task model is extended by considering control modes. Specifically, each control task can run in two control modes: a fast mode and a slow mode, and for each mode there is an associated period.

This idea of multiple operation modes is inspired by mixed-criticality systems [21] [32] in which a system has multiple criticality modes that it can switch to. For instance, consider a system with two criticality modes: a high-critical mode and a low-critical mode. The system will operate in low-critical mode if all the tasks are executing within their expected WCETs; and the system will switch to high-critical mode and will terminate low-critical tasks if one or more tasks exceeds their budget.

Similarly, in a digital control system the controller task can operate in two modes: a slow mode (i.e., a longer control interval) if there is no significant ongoing perturbations, and a fast mode if there is an operational point change or a major disturbance. It is expected that the proposed model can have the following properties:

1. This method can save computational resources with minimal impact on the control performance.

2. This method can have predictable scheduling behaviour that can guarantee extra capacity for executing other tasks in the system.

3. This method can achieve graceful degradation if the system becomes

overloaded.

4. This method should have low run-time overheads, low memory footprint and minimal modification of the system.

The rest of this chapter is organised as follows: Section 5.1 gives a brief review on task models for real-time control. In Section 5.2, the method is formalised by introducing the definition of control performance and scheduling performance, and the collaborative optimization objective. A dual-period task model is then introduced and it is described how it can be scheduled by a fixed-priority scheduler. In Section 5.3, the dual-period model and its sensitivity on control performance regarded to scheduling parameters are explored. A heuristic searching method is also proposed in order to find the optimal periods and switching points. Finally the results are analysed and conclusions are presented.

5.1 Task Models for Real-Time Control

Many existing task models can be applied to real-time control systems, and some of these are covered in Chapter 2. For example, the elastic task model and the quality-of-service degradation.

There are also some other task models that are explicitly designed for control applications. For instance, task decomposition [73] can be used to improve schedulability of control tasks. In [45], a control task is split into two dependent sub-tasks: 1) an output calculation task and 2) a state update task. It is shown that by scheduling a control task as two sub-tasks, the computational delay could potentially be reduced significantly, and the control performance can be improved with maintained schedulability.

In [6], Årzén introduced an event-based controller as an alternative to the traditional time-triggered control paradigm. As a consequence, the corresponding task of the controller will be released aperiodically. The authors demonstrated that this method could lead to large reduction in the CPU utilization, while making minor control performance degradation. However, it is hard to verify the worst-case performance for the proposed event-based controller, both in terms of scheduling and for the control aspect.

Another commonly used model for scheduling control tasks is (m, k)-firm scheduling [76]. Instead of executing every job instance of the task, the scheduler only needs to schedule at least m job instances out of any k consecutive releases. This is feasible as occasional misses of the output update can be tolerated by most control applications. In [114], (m, k)-firm model is used as a less stringent guarantee than the hard deadline requirement. This work shows how the (m, k)-firm model can be used to achieve graceful degradation when a system is overloaded.

The distinctive feature of this method is that a task model is applied which has two flexible periods that it can switch to. Instead of decomposing the control task itself, the control system behaviour is decomposed as transient and steady phases. Similar to the event-based method, the proposed task model can achieve equivalent control performance with less CPU utilization. Compared to the (m, k) model, this method is equivalent to reducing control frequency rather than skipping some job instances. In the next section, the proposed task model will be discussed in detail.

5.2 Proposed Task Model

For the proposed dual-period model, each control task is associated with two periods: a fast period and a slow period. The actual task period will switch between these two periods, which is jointly determined by the switch period and the switching ratio. The switch period is the minimal cycle that consists of a fast and a slow mode. To formalise, the task model is defined as follows:

$$\tau_i \equiv (C_i, T_i^{\Gamma}, T_i^H, T_i^L, \alpha_i, D_i = T_i^H) \tag{5.1}$$

where C_i is the worst-case execution time; T_i^{Γ} is the switch period which is equal to the minimal interval between two disturbances; T_i^H and T_i^L are fast-mode and slow-mode periods respectively, with $T_i^H < T_i^L$; α_i is a ratio that defines when a mode switch will be performed, and D_i is the deadline of the task, which equals the fast-mode period T_i^H.

In this dual-period model, a task initially executes with period T_i^H. This phase, known as the 'fast mode', is used to bound the intersampling dynamics after a disturbance is made to the system. A switching from T_i^H to T_i^L is then made which is triggered at $t = \alpha_i T_i^{\Gamma}$. The phase in which $T_i = T_i^L$ is known as the 'slow mode', in which the system is likely to have less dynamics. As changing of task mode will not take effect until the next job release, the exact switching point happens at: $t_S = \lceil(\alpha_i T_i^{\Gamma})/T_i^H\rceil(T_i^H)$. So in every switch period T_i^{Γ} the task executes with period T_i^H and then executes with period

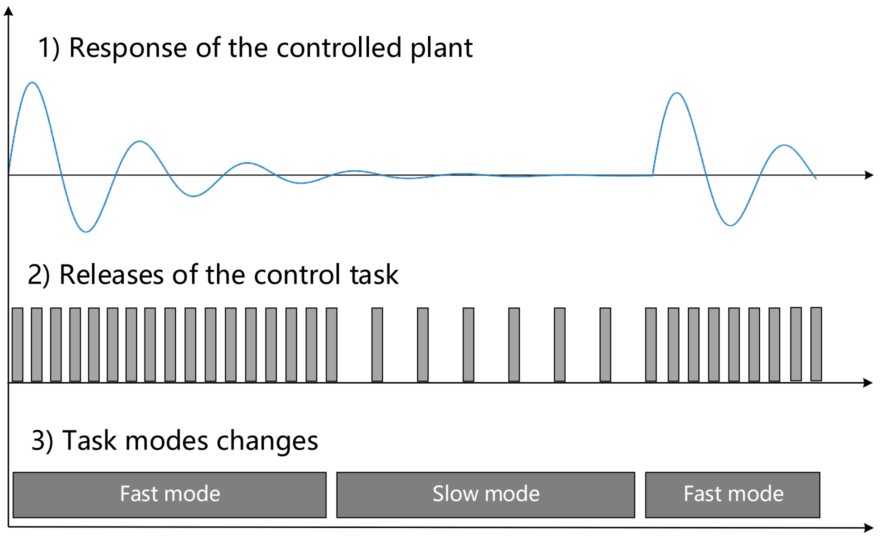

1) Response of the controlled plant
2) Releases of the control task
3) Task modes changes
Fast mode
Slow mode
Fast mode

Figure 5.1: An illustration of the dual-period task model

T_i^L. The task will stay in the slow mode and switch back to the fast mode if a new disturbance is detected. See Figure 5.1 for an illustration.

Both of the periods can adapt according to system requirements. However, under no circumstance should the system become unstable due to a period that is too large. Hence, an upper bound T_i^+ is defined, which is the period where the control system output will become unacceptable. Also, as the resource of the embedded hardware is limited, the period cannot be too small that would overload the system. Thus there is also a lower bound T_i^- on task period, which is determined by the maximum allowed utilization for task τ_i. Overall there is $T_i^- \leq T_i^H < T_i^L \leq T_i^+$, and for all cases, $T_i \in [T_i^-, T_i^+]$.

It is assumed that there is a minimum interval of time between perturbations. In the steady state the controller runs with the longer period. This will continue to be the case unless there is a significant perturbation or a change in set point. If this occurs then the controller will immediately switch to the shorter period. After a time computed from the parameter α_i the controller switches back to the longer period.

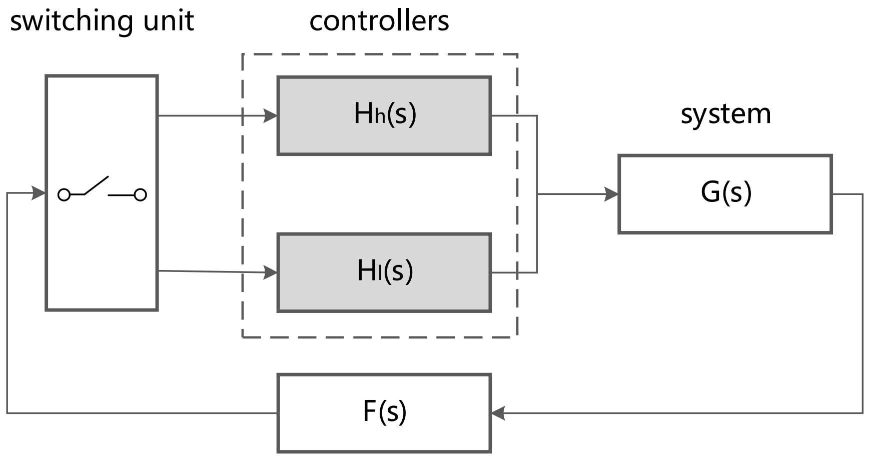


Figure 5.2: An illustration of the control model

5.2.1 Control Model

For a single-input-single-output (SISO) system, the open loop dynamics of a control system can be described as a transfer function in the s-domain [62]:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{a_n s^n + a_{n-1} s^{n-1} + ... + a_0}{b_n s^n + b_{n-1} s^{n-1} + ... + b_0} \tag{5.2}$$

where $U(s)$ represents the Laplace transform of system input and Y(s) represents system response. Here the symbol U is used by convention in control theory, it does not represent utilization in this context.

Assuming the transfer function of the controller is $H(s)$, and the feedback path is $F(s)$, the closed loop transfer function can be written as:

$$G_c(s) = \frac{G(s)H(s)}{1 + G(s)H(s)F(s)} \tag{5.3}$$

The controller $H(s)$ is designed in continuous time and then discretized. Depending on the control mode, two versions of controllers, $H_h(s)$ and $H_l(s)$ are used mutually. Both of the controllers use the same control law but with different time constants and parameters according to the associated task period. A diagrammatic representation that demonstrates the whole feedback loop is shown in Figure 5.2.

To quantify the control performance, a quadratic control cost function is applied that is widely used in the design of optimal controllers. The general form of this cost function is given by [26]:

$$J = \int_{t=0}^{\infty} (x^T(t)Q_1x(t) + 2x^T(t)Q_{12}u(t) + u^T(t)Q_2u(t))dt \tag{5.4}$$

where $x(t)$ is system state vector; $u(t)$ is controller input vector, and Q_1, Q_{12} and Q_2 are weighting matrices. Similar to the period adaptation model, in this work, the cost is only evaluated over a finite horizon up to T_i^Γ, i.e. the switch period of the control task:

$$J = \int_{t=0}^{T_i^\Gamma} (x^T(t)Q_1x(t) + 2x^T(t)Q_{12}u(t) + u^T(t)Q_2u(t))dt \tag{5.5}$$

5.2.2 Scheduling of Dual Period Tasks

To schedule dual-period tasks, an extension to standard fixed-priority scheduler is considered. Task priorities are assigned using the deadline monotonic policy. The deadline of a dual-period task is equal to the fast model period, i.e., $D_i = T_i^H$, and it is not affected by the current control mode. Hence at run-time, the task priorities will not change if the fast model period stays constant.

Scheduling a dual-period task requires a run-time monitor and a timer. The process of scheduling such a task is given in Algorithm 1. Note the control-related states are passed to this procedure from the monitor which runs independently and in parallel.

Algorithm 1: Scheduling a Dual-Period Task

```
Inputs : $T_i^H, T_i^L, T_i^\Gamma, \alpha_i, y(t), r(t)$
Initialise: $t_i \Leftarrow 0$
   loop:
1: scheduler → set_task_period($T_i^H$);
2: timer → start($t_i$);
3: timer → wait_until($\alpha_i T_i^\Gamma$);
4: scheduler → set_task_period($T_i^L$);
5: while $|y(t) - r(t) \leq b|$ and mode_change() == False do
6:     scheduler → stay_in_slow_mode();
7: end while
8: timer → reset();
9: goto loop
```

For operating systems that need permissions to access kernel functions, this routine should be executed in kernel mode. This process can also be integrated into the scheduler and be invoked in the scheduler handler.

5.2.3 Response Time Analysis for Dual-Period

For fixed-priority scheduling, the schedulability of a task can be checked through response time analysis. The standard equation for calculating response time for independent task is already introduced in Section 2.1.4, which is the addition of its own execution time, and the interference time due to preemptions from higher priority tasks:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{Ri}{T_j} \right\rceil C_j \tag{5.6}$$

From this equation it can be indicated that the worst-case response time of a dual period task is no different from a normal task, as the response time of the task is independent of its own period. Also for tasks with higher priorities, running a dual period will not change their response times. However, for lower priority tasks, their response time will potentially be influenced as the interference pattern is changed.

To make it more specific, assuming there is a dual period task τ_j and a normal periodic task τ_i that has a lower priority:

$$\tau_i \equiv (C_i, T_i, D_i = T_i) \tag{5.7}$$

$$\tau_j \equiv (C_j, \alpha_i, T_j^H, T_j^L, T_j^\Gamma, D_j = T_j^H) \tag{5.8}$$

and for the dual period task, the switching behaviour is defined as:

- switching point: $t_S = \lceil(\alpha_i T_i^\Gamma)/T_j^H\rceil(T_j^H)$;
- if $(t \mod T_i^\Gamma) \leq t_S \implies T_j = T_j^H$;
- if $(t \mod T_i^\Gamma) > t_S \implies T_j = T_j^L$.

The response time of τ_i is discussed under different conditions depending on if task τ_i finishes before or after the switching of τ_j:

1) If $R_i < t_S$, task τ_j always executes with T_j^H, and the resultant worst-case response time of τ_j is no different from the case if τ_j was a normal periodic task with period T_j^H:

$$R_i = C_i + \left\lceil \frac{R_i}{T_j^H} \right\rceil C_j$$

2) If $R_i \geq t_S$, after the switching task τ_j will use T_j^L which reduce the frequency of interferences:

$$R_i = C_i + \left(\frac{t_S}{T_j^H} + \left\lceil \frac{R_i - t_S}{T_j^L} \right\rceil \right) C_j$$

3) if $R_i > T_j^\Gamma$, the worst-case behaviour is that the dual period task switches immediately back to the fast mode after it reaches its switch period. Define the total interference time in a switch period as I_Γ, and the number of switch period that task τ_i suffers as n, there is:

$$R_i = C_i + \begin{cases} \left\lceil \frac{R_i - nI_\Gamma}{T_j^H} \right\rceil C_j + nI_\Gamma & \text{if } (R_i \mod T_j^\Gamma) < t_S \\ \left(\frac{t_S}{T_j^H} + \left\lceil \frac{R_i - nI_\Gamma - t_S}{T_j^L} \right\rceil \right) C_j + nI_\Gamma & \text{if } (R_i \mod T_j^\Gamma) \geq t_S \end{cases} \tag{5.9}$$

in which $n = \lfloor R_i / T_j^\Gamma \rfloor$. To generalise for all cases, the *max()* and *min()* functions are introduced which output the larger or lower value between two parameters, respectively. Overall, there is:

$$R_i = C_i + \left(\left\lceil \frac{\min(t_S, R_i - nI_\Gamma)}{T_j^H} \right\rceil + \left\lceil \frac{\max(R_i - nI_\Gamma - t_S, 0)}{T_j^L} \right\rceil \right) C_j + nI_\Gamma \tag{5.10}$$

5.2.4 Optimal Selection of Scheduling Parameters

The optimal scheduling of dual period is equivalent to optimally selecting task parameters (i.e., α_i, T_j^H, T_j^L) of control tasks. This can be transferred into an optimization problem. First a few assumptions need to be clarified:

A.1 The upper bound T_j^+ and the lower bound T_j^- of the period of a control task in Γ_c are known.

A.2 The whole task set Γ is guaranteed to be schedulable, if all control tasks $\tau_i \in \Gamma_c$ are executing in slow mode ($T_i = T_i^L$).

A.3 Switching between two period modes will not introduce control system instability.

A.4 The operating system has interfaces to allow the feedback scheduler to change task periods, and the kernel has information on the current control states.

Table 5.1: List of Symbols and Notations

Symbol	Description
Γ	the task set
Γ_c	the task set of all control tasks
Γ_{nc}	the task set of all non-control tasks
PI	system overall performance
J_c	control cost
J_s	scheduling cost
w_c	control weight
w_s	scheduling weight
α_i	switching point ratio
T_i^Γ	the switching period
T_i^H	the task period during fast mode
T_i^L	the task period during slow mode
T_i^+	upper bound of the period of control task τ_i
T_i^-	lower bound of the period of control task τ_i

The sampling interval of a control system is normally based on the desired speed of the closed loop system. To satisfy A.1, one practice would be let $T_j^+ = 10 \times T_r$ and $T_j^- = 4 \times T_r$, in which T_r is the closed-loop rising time. The second assumption A.2 is to make sure the task set is schedulable even in the worst case. Assumption A.3 is not always true. It is known in hybrid system theory that switching between two different controllers could introduce instability. As in this work, only two switches every T_i^Γ are performed, it is reasonable to think the switching interval is larger than the dwell time. Hence no instability should be introduced. Assumption A.4 is essential for observing the system and making changes to the period. The additional symbols and notations involved are defined in Table 5.1.

The overall system performance is judged by both the scheduling performance and the control performance. The optimization problem is formed as follows:

$$\begin{aligned}
\underset{\alpha_i, T_i^H, T_i^L}{\text{minimise}} \quad & \mathrm{J} = \{w_c J_c + w_s J_s\} \\
\text{s.t.} \quad & 0 < \alpha_i \leq 1 \\
& T_i^H \geq T_i^- \\
& T_i^L \leq T_i^+ \\
& T_i^H < T_i^L \\
& \forall i \in \Gamma_c
\end{aligned} \tag{5.11}$$

where w_s and w_c are constants that are used to decide trade-offs between control and schedulability. Relative cost is used as the metric for measuring control performance P_c:

$$P_c = \frac{J'}{J(0)}, J_c \geq 1 \tag{5.12}$$

in which $J(0)$ defines the cost of the best control scenario, i.e., $T_i = T_i^H$. A higher P_c indicates a worse control quality of the system.

In this initial investigation, schedulability performance is evaluated with CPU utilization. The lower the task set utilization is, the more likely the task set is schedulable. The performance of scheduling P_s can be measured by improvements in task utilization:

$$P_s = \frac{U_i}{U_i(0)}, 0 < J_s \leq 1 \tag{5.13}$$

here $U_i(0)$ is the initial utilization when the system has $T_i = T_i^H = T_i^-$. A smaller P_s represents a larger improvement in utilization.

5.3 Experiments and Evaluations of Dual Period

The purpose of doing evaluations is to reveal the performance of this dual-period model, and give insights into choosing proper scheduling parameters. All experiments in this section are performed using MATLAB R2015a and Simulink. A *mex* file is compiled with Microsoft Windows SDK 7.1. For all the control tasks, the same system dynamics is used, which is a second-order system with a transfer function $G(s)$ given as:

$$G(s) = \frac{200}{s^2 + 80s + 2000} \tag{5.14}$$

The controller used is a PID controller with proper timing compensation following details in [9]. A fixed-priority scheduler is simulated in Simulink with a kernel tick time of $0.1ms$. The range of allowed upper bound and lower bound is selected as $T_i^+ = 10ms$ and $T_i^- = 5ms$ respectively. The computation time is $C_i = 1ms$ for each task release. The switch period T_i^Γ is selected as $300ms$. All tasks are released at the critical instant, i.e., no task offset, unless explicitly mentioned. For simplicity, the dual-period task scheduling method is notated as DUAL-FPS. The Simulink block diagram used for this experiment is shown in Figure 5.3.

5.3.1 Performance of DUAL-FPS

In this experiment, there are three control tasks τ_0, τ_1, τ_2, and a background task τ_3. τ_0 applies a dual-period model with $T_0^H = T_i^-$, $T_0^L = T_i^+$ and $\alpha_i = 0.5$. τ_1 and τ_2 are periodic tasks with $T_1 = T_i^-$ and $T_2 = T_i^+$, respectively.

The system responses of each control plant are shown in Figure 5.4. From the figure, it can be seen that the output of the controller implemented by task τ_0 (Controller 0 for short) is similar to the output of Controller 1, which applies the shortest period. Specifically, the response is identical before the switching point ($0.15s$), and has noticeable degradations after the switch. However, when compared with Controller 2, the output of the dual-period task is still much better.

In terms of resource usage, the equivalent utilization of τ_0 over the switch period is 0.15, which lies exactly in the middle of τ_1 and τ_2 (see Figure 5.5). As the switching time instance is deterministic, the task utilization is expected to be:

$$U_i = (1 - \alpha_i)U_{(T_i^L)} + \alpha_i U_{(T_i^H)} \tag{5.15}$$

It can be seen that with DUAL-FPS, significant computational resources can be saved with a little compromise over the quality of control. Even in the worst-case, the output of DUAL-FPS is still bounded by the output of a single period task with $T_i = T_i^+$. The minimal resources that a DUAL-FPS uses is lower bounded by $U_{(T_i^L)}$ when $\alpha_i = 0$, and the maximal is when $T_i = T_i^L$ for $\forall t$ in which case $\alpha_i = 1$.

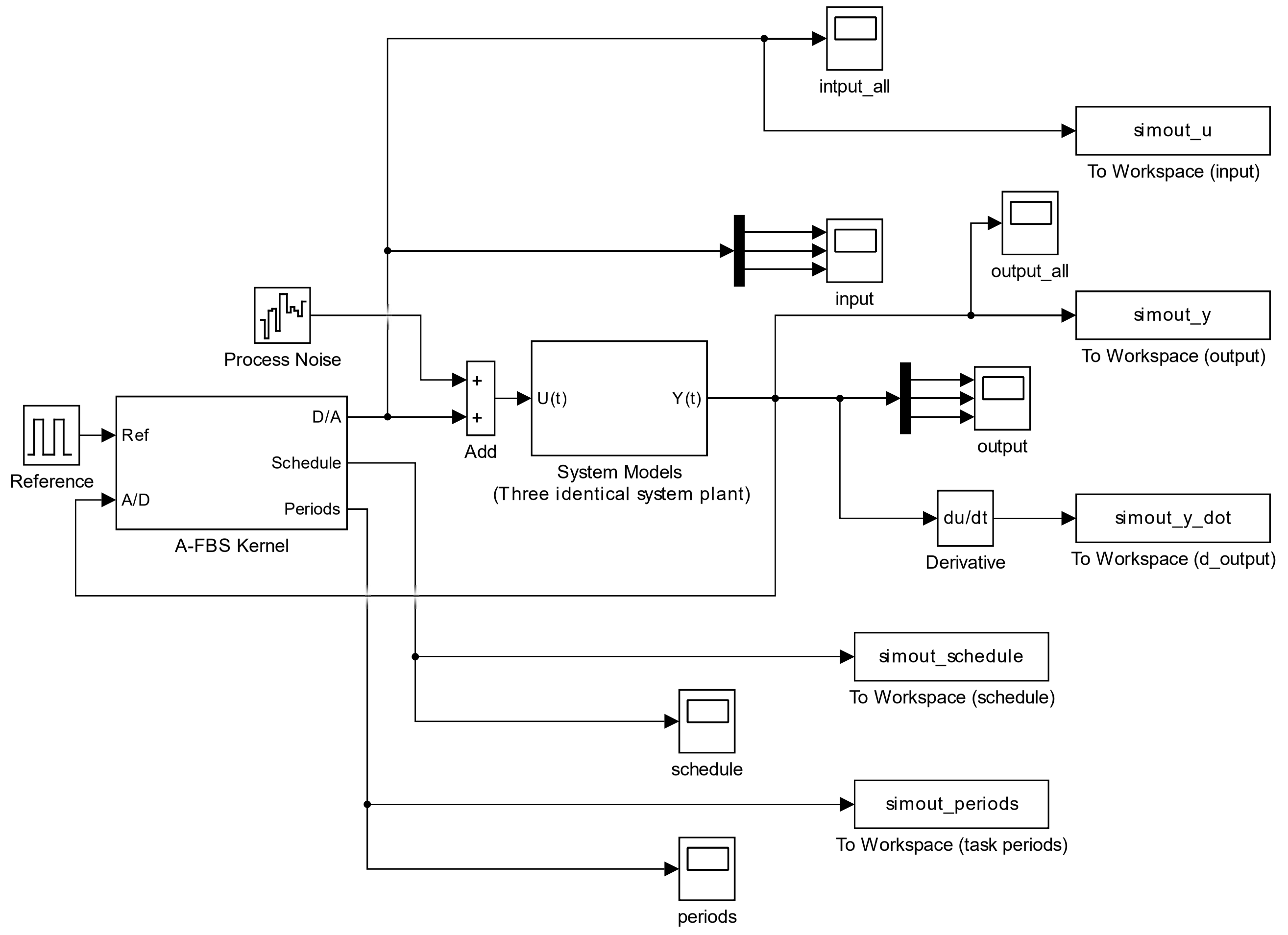


Figure 5.3: The experiment setup and block connections in Simulink

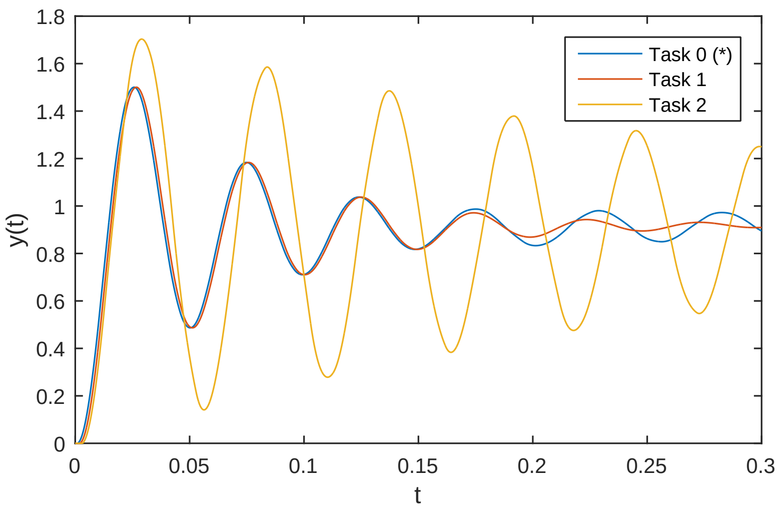


Figure 5.4: System outputs of each control task ($\alpha_i = 0.5$)

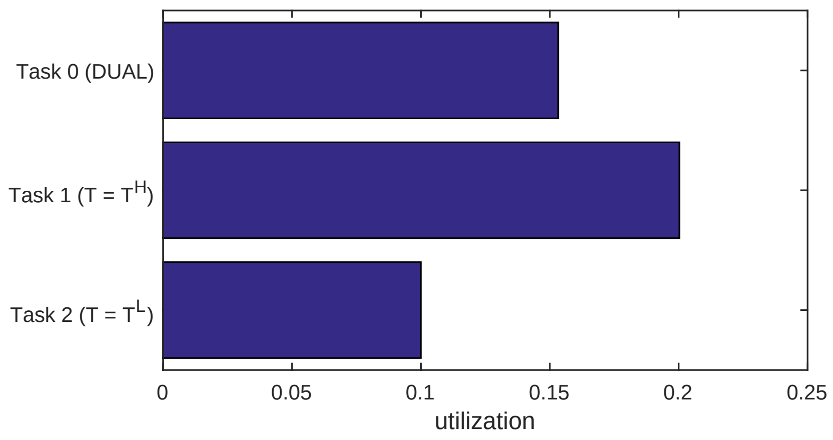


Figure 5.5: CPU utilization of each control task ($\alpha_i = 0.5$)

5.3.2 Parameter Sensitivity

In the previous evaluation, all of the three scheduling parameters are fixed. However, the selection of the parameters α_i, T_i^H and T_i^L has a large influence on both control and scheduling performance, thus needs more dedicated exploration.

To figure out how the variation of each parameter will change the control performance P_c and the scheduling performance P_s, a sensitivity analysis is performed for each scheduling parameter. This is done by exhaustive search in the parameter space formed by all possible values, while keeping the other two parameters fixed. Specifically, it is made $\alpha_i = 0.5$, $T_i^H = 50$ and $T_i^L = 100$ if they are not the variable of interest. In this experiment, only τ_0 and τ_3 are activated. The role of τ_3 is to measure unused capacities in order to calculate utilization.

The results of the experiment are shown in Figure 5.6 for T_i^H, Figure 5.7 for T_i^L, and Figure 5.8 for α_i. The first observation is that the scheduling and control performance do have a correlation with the three parameters. The only exception is T_i^L against P_c: increasing of T_i^L has little effect on P_c. This is because most of the system dynamic transition happens at the beginning of the switch period. After the system switches to the slow-mode, the interval between two successive controls has limited impact on the control performance. However, this is not the case for T_i^H. It can be seen from Figure 5.6, P_c is very sensitive to T_i^H. The correlation is not linear but P_c is monotonically increasing when T_i^H increases. It can be seen that α_i also controls P_c. However, after $\alpha_i = 0.3$, increments in α_i no longer decrease the control performance.

In terms of scheduling performance, because the existence of a deterministic switch, the result roughly follows Equation (5.15). Increasing T_i^L, T_i^L or decreasing α_i can all reduce resource usage. From these three figures, it can be seen P_s is equally sensitive to T_i^H and T_i^L, and is more sensitive to α_i.

One conclusion is that P_s and P_c have a negative correlation, i.e., increasing one will decrease the other. Theoretically, given an optimization objective, one or more balance points can be found that gives proper trade-offs between P_c and P_s. This gives motivation to do another experiment in the next subsection.

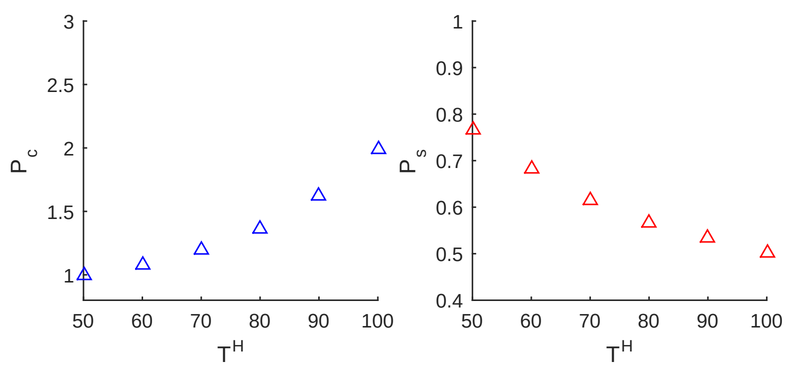


Figure 5.6: Sensitivity of T_i^H (when $\alpha_i = 0.5$ and $T_i^L = 100$)

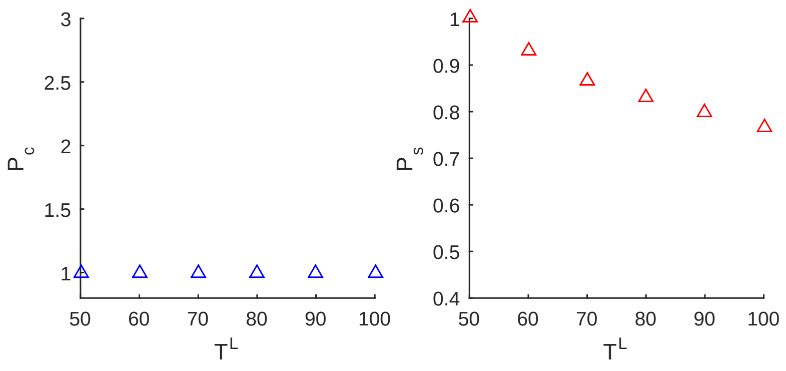


Figure 5.7: Sensitivity of T_i^L (when $\alpha_i = 0.5$ and $T_i^H = 50$)

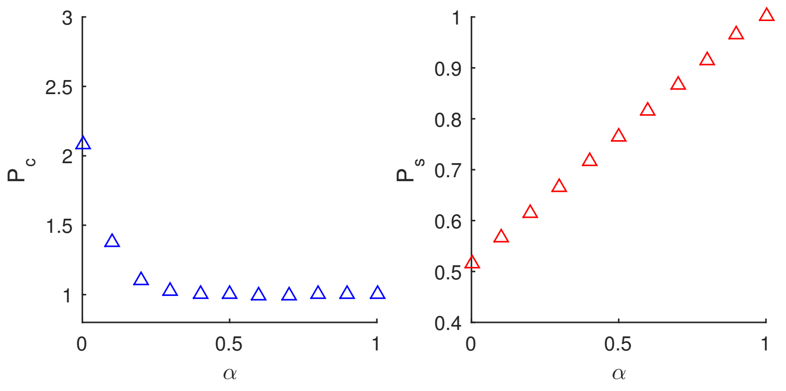
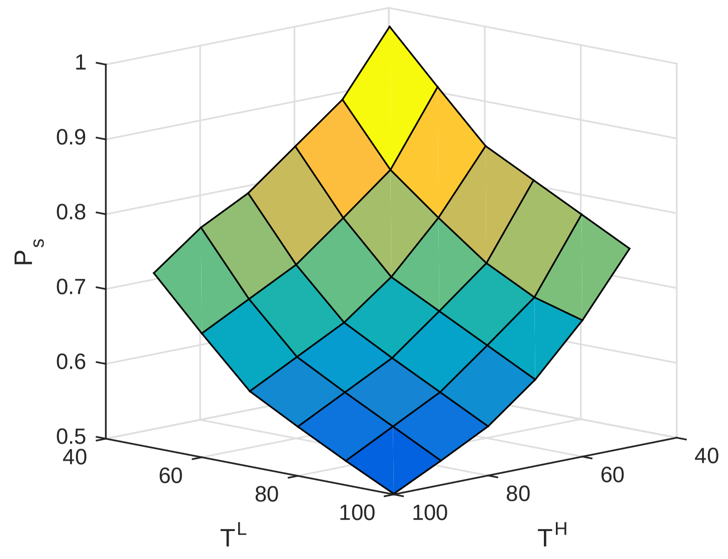


Figure 5.8: Sensitivity of α_i (when $T_i^H = 50$ and $T_i^L = 100$)

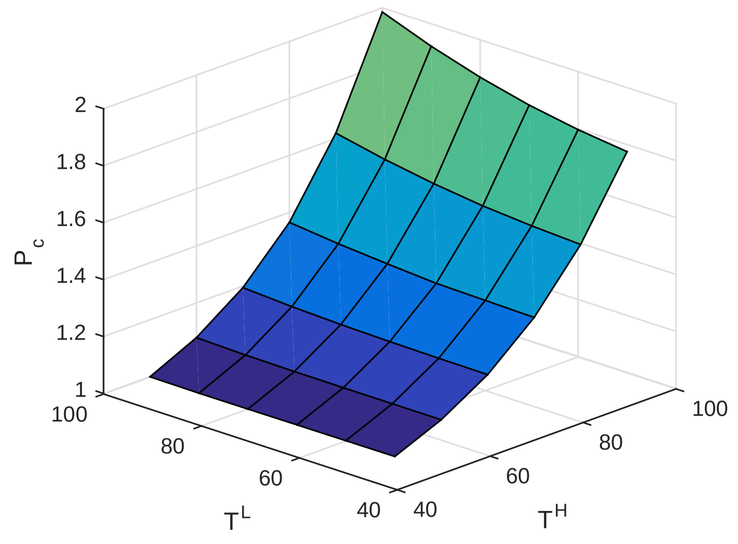
5.3.3 Control and Scheduling Trading-off

To further explore the trade-off that can be made between control and scheduling, two more studies will be done in this subsection. In the first study, the cost spaces of P_s (Figure 5.9a) and P_c (Figure 5.9b) are explored that are determined by varying the two periods when α_i is fixed to 0.5. From these two diagrams, it can be seen T_i^L and T_i^H have equal influences in terms of the scheduling performance. However, for the control performance, T_i^H (quadratic) is more influential than T_i^L (linear). This indicates that if the same utilization needs to be achieved, increasing T_i^L is less expensive than increasing T_i^H.

For the second analysis, the objective is to explore how α_i could change P_s and P_c. This is done by plotting a series of scatter points with different parameters. In Figure 5.10, each point represents a unique combination of T_i^H, T_i^L and α_i. Points with the same α_i are grouped, marked with the same colour.



(a) Periods and Control Performance



(b) Periods and Scheduling Performance

Figure 5.9: The plot of performance when $\alpha_i = 0.5$

From this diagram, it can be seen α_i does not have a clear correlation to P_s and P_c. An α_i that has a small P_s and a large P_c can also occasionally have a large P_c and a small P_s. From the distribution of these scatter points, a lower bound of all points on the figure can be found, which is also a boundary that indicates all optimised parameters. If there is no constraint on the utilization that the control task can use, the corner point $(P_s = 0.66, P_c = 1.05, \alpha_i = 0.4)$ can be selected that represents the best option: resource is saved with little degradation in control. If more resources are required, then degradation in control is unavoidable. If this is the case, then decreasing T_i^L is preferred rather than T_i^H as learned from Figure 5.9a and 5.9b.

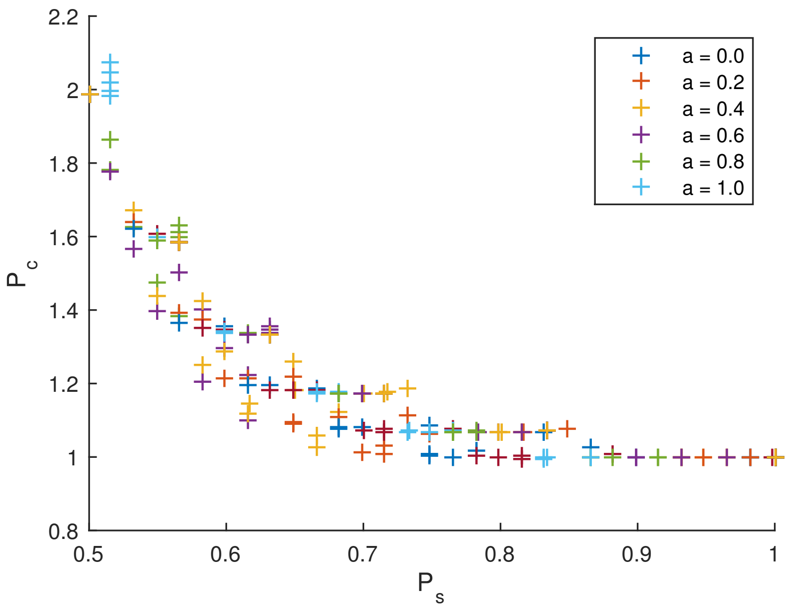


Figure 5.10: Parameter α_i against P_c and P_s

All of these experiments reveal the performance of the dual-period task model, and provide insights of how parameter changes will affect P_s and P_c. Figure 5.11 shows how the three parameters collaboratively affect both control and scheduling performance. The way of getting an optimised point can be found from exhaustive simulation by applying all possible combinations of parameters.

As exact optimization through full search of the space is costly and not always tractable, some heuristic assignment methods can also be used. One possible guidance is given here, which tries to minimise the utilization without

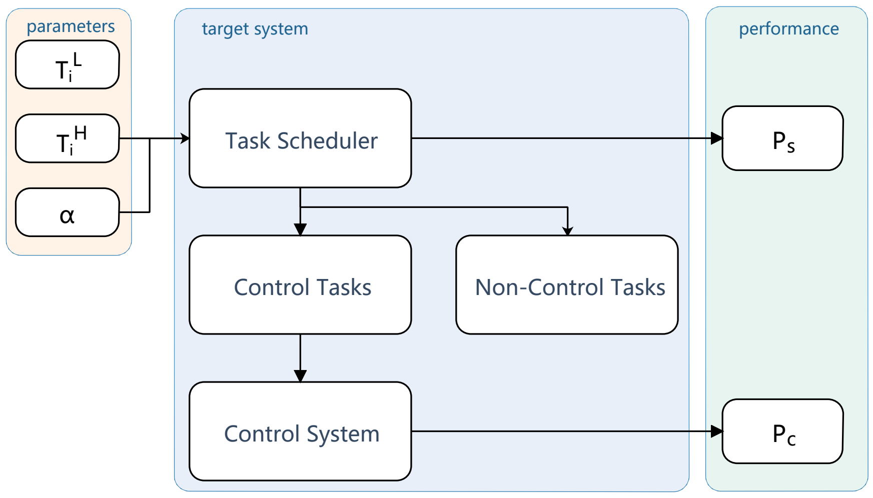


Figure 5.11: Influence of task parameters on the control and scheduling performance

compromising the control performance:

1. Initially set $T_i^H = T_i^L = T_i^-$ and $\alpha_i = 1.0$. In this case, period switching is effectively disabled. Let the task period $T_i = T_i^H$ (fastest).

2. Simulate the step response of the closed-loop system. Record the settling time T_s and the quadratic cost J_0 in $[0, T_s]$. Let $T_i^\Gamma \geq T_s$.

3. Now enable period switching. Increase T_i^H and re-run the simulation ($T_i^L = T_i^H$). Assuming the new cost is J', repeat the process until $J' - J_0 > 0.1J_0$ (i.e., a 10% performance drop).

4. Decrease α_i. Run a simulation and record the control cost as J''. Repeat this process until $J'' - J' > 0.1J'$.

5. Increase T_i^L and run a simulation. Record the control cost as J'''. Repeat this process until $J''' - J'' > 0.1J''$ or $T_i^L \geq T_i^+$.

This strategy prioritised the searching of the three parameters in the order $T_i^H \succ \alpha_i \succ T_i^L$. By utilising this searching method with a step size of 10 for periods, and 0.1 for α_i, a solution is found: $\{\alpha_i = 0.1, T_i^H = 60, T_i^L = 80\}$ that leads to $P_c = 1.18$ and $P_s = 0.65$. It can be seen from Figure 5.10 that this solution is very close to the corner point.

Note a different solution can be obtained if there are constraints on the utilization or control performances. To handle scenarios in which there are dynamic requirements on different utilization and/or quality of control, multiple solutions can be solved off-line and be provided to the scheduler, in order to enable it to choose the proper parameters under different conditions at run-time.

5.4 Summary

In this work, it is first motivated the requirement of a method to improve the resource usage of an over-controlled control system. A novel scheduling model is then presented: the dual-period task model. This model is designed explicitly for control applications. Three parameters T_i^H, T_i^L and α_i are associated with each control task. It can be seen from the experiment that for a particular second-order system, the resulted control performance is competitive to that of using the shortest control period.

It can be also seen that, by properly choosing the scheduling parameters, the system can have different set-points that result in different scheduling and control performances. This is to say, if a solution exists, this method could find a balance point that makes a trade-off between the control and the scheduling performances in a predictable and controllable way. In scenarios where dynamic resource allocation is required, this method can also produce good trade-offs under different constraints on utilization or quality of control. This can be done by making performance mappings off-line with extensive simulations in the parameter space.

Note it is possible to adapt the two periods T_i^H and T_i^L at run-time, which is similar to the period adaptation model. Once outline analysis has determined T_i^H and T_i^L, the run-time strategy (described in the previous chapter) can be applied to each of these periods. First the T_i^H mode is monitored and T_i^H would be potentially extended. Then T_i^L is considered a candidate for extension.

Chapter 6

Conclusions and Future Work

In this chapter, the work previously presented will be evaluated in terms of the primary objectives of this research. This chapter will also summarise the thesis and re-emphasis its contributions. The insight into further work that can be performed is also provided at the end of this chapter.

The development of the next generation of cyber-physical systems requires all system components to be more flexible and robust. As task scheduling has an important role in coordination of computer resources, it is vital for a scheduling system in a cyber-physical system to have higher flexibility and adaptiveness. All of the work presented in this thesis tries to address these arising issues by proposing a novel scheduling framework that could potentially be used for emerging systems.

The proposed framework, Adaptive Task Scheduling Framework, or ATAS, is a general adaptive task scheduling framework that has many novel features. There are three major properties that make ATAS distinctive from traditional scheduling methods:

1) The system performance, including both scheduling and control performance, is monitored and used in a feedback loop.

2) The task parameters, e.g., period of a task, are allowed to be changed to make resource and performance trade-offs.

3) The scheduling system is hierarchical, with the lower-level scheduler being a traditional FPS or EDF, and the higher-level scheduler being a feedback scheduler that is deployed in the cloud.

The framework can cope with scenarios in which the resource demand of a system increases over its operation, due to additional function requirements or hardware ageing. Within this context, it is shown a case where task worst-case execution times are increasing and a method for trend identification is developed through a comparison study. To accommodate for these additional resources, a period adaptation model is proposed, which applies a flexible model and cloud based analysis. The period of a control task can be extended to release computational resources, if that control system can still meet its specification after the adaptation process. Finally, this model has been extended to one in which a task has two flexible periods that a task can switch between.

These two opposing dynamic behaviours of a long-lived CPS are addressed and balanced through the framework. The integration of these two opposites is straightforward, which can be done through CPU utilization, i.e., the resource saved from period adaptation or dual period can be allocated to tasks with increased demand. When applied in accordance with other adaptive methods, this method can improve the resilience of a CPS, making a system run longer without stopping for maintenance, and save energy and resources. Although the context of this work is merely for control applications, the generality of this framework makes it not limited to control, but also applicable to other applications that have quantifiable performance or utility that is correlated to task periods.

6.1 Contributions

In this thesis, a framework for scheduling tasks in cyber-physical systems is presented: the Adaptive Task Scheduling Framework, or ATAS. This framework is unique in its ability to adapt to changes by manipulating task parameters at run-time. The framework is based on the IBM MAPE-K architecture and is designed explicitly for handling real-time tasks. The argument starts from the identification of dynamic worst-case execution times, or dWCET. It is demonstrated that both software and hardware aspects will influence task worst-case execution times.

The dWCET will often cause increased computational requirements in the system. It is essential that the dWCET should be monitored and any trend should be predicted which forms the next contribution of this work. A review of trend identification methods is given which results in three candidate methods that are then used for a comparative study. It is shown that by introducing a wide range of performance measures, the effectiveness of the trend identification methods can be compared in a context that is close to real scheduling scenarios.

The next contribution of this thesis is to explore methods that can accommodate such changes in the system load caused by varied worst-case execution times. In particular, the problem of actively changing task periods in real-time controllers is explored. The periodical real-time control tasks apply a flexible task model, in which case the task period can be changed and reconfigured at run-time according to the actual system requirement. The use of cloud-based monitoring and prediction makes it feasible for embedded real-time controllers with limited resources to make complicated decisions. The adaptation decision is made based on both a simulated model and run-time information.

Finally, a dual-period task model is proposed to enhance the case where only one period can be adapted. The dual-period model switches between two operational modes according to the scheduling parameters and control conditions. The use of dual-period makes it easier in terms of making tradeoffs between control and scheduling. The selection of scheduling parameters in the dual-period model is discussed by using a heuristic. The response time analysis is also given, along with the scheduling mechanisms on a fixed-priority scheduler.

All of these contributions are evaluated and analysed by experiments on a hybrid simulation environment based in Matlab and Simulink. The research hypothesis, defined in Chapter 1, is revisited below:

Flexible and adaptive task scheduling can improve the schedulability of long-lived cyber-physical systems. This can be achieved by using novel flexible models for making design tradeoffs, utilising statistical learning techniques for supervision and analysis, and using cloud computing facilities for adaptively managing resource reclaiming.

All of the contributions discussed provide evidence to sustain the hypothesis of this thesis. It can be seen that the hypothesis is sufficiently satisfied.

6.2 Future Research

Although sufficient discussions and evaluations are already made in previous chapters, the author is aware that there is still plenty of potential work that can be incorporated into the framework to either improve or enhance the usability of ATAS. To be more specific, future work is discussed in the following subsections. Some of this work have already been looked at, but not fully investigated due to time limitation.

6.2.1 Trend Analysis

For this work, it is only explored the case where a linear and deterministic trend is considered. In reality, many trends are non-linear and stochastic. Some of them can be decomposed into sub linear trends but may exhibit different characteristics. It is also worth to consider more dependent variables that could have an impact on execution times to improve the precision of prediction. The use of ensemble learning to combine two or three identification methods could also benefit the result of analysis, and multiple successive predictions should be considered to make a control decision with a high confidence. This approach can also be applied to real data which is obtained from industrial applications. All these issues form topics for future work.

6.2.2 Period Adaptation

There are still many other aspects that can be further explored for period adaptation. These include consideration of the case in which the system has dependent control tasks, for example a multi-loop controller that has cascaded control tasks. Another example is a motion control system with multiple degrees of freedom, e.g., a humanoid robot, in which more than one control task has to cooperate and be synchronised. This method can also be extended to satisfy multiple objectives, as there may be more than one design consideration and even conflicted constraints in additional to the performance requirement.

6.2.3 Dual Period

As part of future work, different forms of control systems and controllers will be studied. A scheduling method that better matches the dual-period model also needs to be explored, e.g., Dual Priority Scheduling [58]. Also in this work, utilization is used to measure the scheduling performance. However, utilization does not provide all the information of the effectiveness of scheduling. In future work, a more precise timing analysis will be used to better describe the resource usage and improvements in terms of providing additional capacity to lower-priority tasks. The adaptation of T_i^H and T_i^L is also important. It may also be possible to modify the α parameter by monitoring the behaviour of the system as it executes.

6.2.4 Real-World Case Study

In this thesis, control and scheduling system co-simulations are applied for the purpose of demonstration and evaluation. However, there is a limitation on simulations as the fidelity of simulation depends on the precision of the system model. Hence a real-world case study will provide stronger support of the usability of the framework. To transform the proposed method into a real-world application, some details related to software development, kernel modification and communication with the cloud need to be formalised. The other interesting topic is what methodology can be applied for validation and verification of the deployed system with the ATAS integrated.

All of those work discussed can be formed into future research.

List of Abbreviations

ATAS	Adaptive Task Scheduling Framework
AFBS	Adaptive Feedback Scheduler
BCET	Best-Case Execution Time
CDF	Cumulative Distribution Function
CPS	Cyber-Physical Systems
CPCS	Cyber-Physical Control Systems
EDF	Earliest Deadline First
FPS	Fixed-Priority Scheduling
IAE	Integral of Absolute Error
LQR	Linear-Quadratic Regulator
OLS	Ordinary Least Square
PI	Performance Indicator
QoC	Quality-of-Control
QoS	Quality-of-Service
RTA	Response Time Analysis
RTS	Real-Time Systems
SVR	Support Vector Regression
WCET	Worst-Case Execution Time

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Real-Time Systems Symposium, 1998. Proceedings. The 19th IEEE*, pages 4–13. IEEE, 1998.

[2] L. Abeni and G. Buttazzo. Stochastic analysis of a reservation based system. In *null*, page 30092a. IEEE, 2001.

[3] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[4] L. Abeni, G. Lipari, and J. Lelli. Constant bandwidth server revisited. *SIGBED Review*, 11:19–24, 2014.

[5] S. Arunachalam, T. Chantem, R. P. Dick, and X. S. Hu. An online wear state monitoring methodology for off-the-shelf embedded processors. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 114–123. IEEE Press, 2015.

[6] K.-E. Årzén. A simple event-based PID controller. In *Proc. 14th IFAC World Congress*, volume 18, pages 423–428, 1999.

[7] K.-E. Årzén, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha. Integrated control and scheduling. *Report ISRN LUTFD2/TFRT–7586–SE, Department of Automatic Control*, 1999.

[8] K.-E. Årzén and A. Cervin. Software and platform issues in feedback control systems. *Cyber-Physical Systems*, pages 165–195, 2017.

[9] K.-E. Årzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Proceedings of the 39th IEEE Conference on Decision and Control, 2000.*, volume 5, pages 4865–4870. IEEE, 2000.

[10] K.-E. Arzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 5, pages 4865–4870. IEEE, 2000.

[11] K.-E. Årzén, A. Cervin, and D. Henriksson. Implementation-aware embedded control systems. *Handbook of networked and embedded control systems*, pages 377–394, 2005.

[12] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Prentice-Hall, 1997.

[13] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[14] T. Atdelzater, E. M. Atkins, and K. G. Shin. QoS negotiation in real-time systems and its application to automated flight control. *Computers, IEEE Transactions on*, 49(11):1170–1183, 2000.

[15] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 123–132. IEEE, 1998.

[16] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[17] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes*, 24(2):127–132, 1991.

[18] N. C. Audsley, A. Burns, M. M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[19] S. D. Balkin and J. K. Ord. Automatic Neural Network modeling for univariate time series. *International Journal of Forecasting*, 16(4):509–515, 2000.

[20] C. Ballabriga, J. Forget, and G. Lipari. Context-sensitive parametric wcet analysis. In *WCET*, 2015.

[21] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43, 2011.

[22] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*, pages 21–28. IEEE, 1999.

[23] S. Bennett. A brief history of automatic control. *IEEE Control Systems Magazine*, 16(3):17–25, 1996.

[24] G. Bernat, A. Colin, and S. Petters. pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems. *REPORT-UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE YCS*, 2003.

[25] G. Bernat, A. Colin, and S. M. Petters. Wcet analysis of probabilistic hard real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 279–288. IEEE, 2002.

[26] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2008.

[27] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[28] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[29] G. E. Box and G. M. Jenkins. Time series analysis: Forecasting and control. In *Holden-Day series in time series analysis.* Holden-Day, 1976.

[30] A. Burns and S. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[31] A. Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. In *Embedded Software*, pages 1–15. Springer, 2003.

[32] A. Burns and R. Davis. Mixed criticality systems - a review. *Department of Computer Science, University of York, Technical Report*, pages 1–69, 2013.

[33] A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, pages 89–96. IEEE, 2000.

[34] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46(4):305–325, 2000.

[35] A. Burns and A. J. Wellings. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*, pages 365–420. Pearson Education, 2001.

[36] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Using harmonic task-sets to increase the schedulable utilization of cache-based preemptive real-time systems. In *rtcsa*, page 195. IEEE, 1996.

[37] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.

[38] G. Buttazzo and L. Abeni. Smooth rate adaptation through impedance control. In *null*, page 3. IEEE, 2002.

[39] G. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 90–99. IEEE, 1995.

[40] G. C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, pages 233–279. Springe, 2005.

[41] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 286–295. IEEE, 1998.

[42] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *Computers, IEEE Transactions on*, 51(3):289–302, 2002.

[43] S. Bygde, A. Ermedahl, and B. Lisper. An efficient algorithm for parametric wcet calculation. *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 13–21, 2009.

[44] A. Cervin. Improved scheduling of control tasks. In *ECRTS*, 1999.

[45] A. Cervin. Improved scheduling of control tasks. In *ECRTS*, 1999.

[46] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1-2):25–53, 2002.

[47] A. Cervin, D. Henriksson, B. Lincoln, and K.-E. Årzén. Jitterbug and truetime: Analysis tools for real-time control systems. In *Proceedings of the 2nd Workshop on Real-Time Tools*. Copenhagen, Denmark, 2002.

[48] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Arzen. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. *IEEE control systems*, 23(3):16–30, 2003.

[49] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzén, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 1–9. Gothenburg, Sweden, 2004.

[50] S. Chakravarthi, A. Krishnan, V. Reddy, C. Machala, and S. Krishnan. A comprehensive framework for predictive modeling of negative bias temperature instability. In *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International*, pages 273–282. IEEE, 2004.

[51] T. Chantem, X. S. Hu, and M. D. Lemmon. Generalized elastic scheduling for real-time tasks. *Computers, IEEE Transactions on*, 58(4):480–495, 2009.

[52] T. Chantem, X. Wang, M. D. Lemmon, and X. S. Hu. Period and deadline selection for schedulability in real-time systems. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 168–177. IEEE, 2008.

[53] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai,

H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, 2009.

[54] V. Cherkassky and Y. Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, 2004.

[55] J. S. Cho, M.-H. Park, and P. C. Phillips. Practical kolmogorov–smirnov testing by minimum distance applied to measure top income shares in korea. *Journal of Business & Economic Statistics*, pages 1–15, 2017.

[56] J. B. Dabney and T. L. Harman. *Mastering simulink.* Pearson, 2004.

[57] X. Dai. Qualifying dissertation: The role of flexible models and feedback in real-time scheduling. *University of York*, 2015.

[58] R. Davis and A. Wellings. Dual priority scheduling. In *RTSS*, 1995.

[59] R. DeCarlo, M. S. Branicky, S. Pettersson, B. Lennartson, et al. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE*, 88(7):1069–1082, 2000.

[60] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 289–300. IEEE, 2002.

[61] R. C. Dorf and R. H. Bishop. *Modern Control Systems.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 9th edition, 2000.

[62] R. C. Dorf and R. H. Bishop. *Modern control systems.* Pearson, 2011.

[63] P. Dziurzanski, H. A. Ghazzawi, and L. S. Indrusiak. Feedback-based admission control for task allocation. In *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, pages 1–5. IEEE, 2014.

[64] K. Ecker, F. Drews, and J. Lichtenberg. QoS-based management of multiple shared resource in dynamic real-time systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 6–pp. IEEE, 2006.

[65] S. Edgar and A. Burns. Statistical analysis of wcet for scheduling. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 215–224. IEEE, 2001.

[66] J. Eker, P. Hagander, and K.-E. Årzén. A feedback scheduler for real-time controller tasks. In *In IFAC Control Engineering Pratice*, 2000.

[67] S. Esterby. Trend analysis methods for environmental data. *Environmetrics*, 4(4):459–481, 1993.

[68] R. A. Fisher and L. H. C. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 24, pages 180–190. Cambridge Univerisity Press, 1928.

[69] C. Frei and C. Schär. Detection probability of trends in rare events: Theory and application to heavy precipitation in the Alpine region. *Journal of Climate*, 14(7):1568–1584, 2001.

[70] M. M. B. Gaid, A. Cela, Y. Hamam, and C. Ionete. Optimal scheduling of control tasks with state feedback resource allocation. In *American Control Conference, 2006*, pages 6–pp. IEEE, 2006.

[71] M. K. Gardner and J. W. Liu. Analyzing stochastic fixed-priority real-time systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 44–58. Springer, 1999.

[72] R. Gerber and S. Hong. Slicing real-time programs for enhanced schedulability. *ACM Trans. Program. Lang. Syst.*, 19:525–555, 1997.

[73] R. Gerber and S. Hong. Slicing real-time programs for enhanced schedulability. *ACM Trans. Program. Lang. Syst.*, 19:525–555, 1997.

[74] H. A. Ghazzawi. A control-theoretic approach for scheduling soft real-time tasks with dependencies. *University of York Technical Report*, 2014.

[75] M. Grottke, R. Matias, and K. S. Trivedi. The fundamentals of software aging. In *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pages 1–6. Ieee, 2008.

[76] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE transactions on Computers*, 44(12):1443–1451, 1995.

[77] M. G. Harbour. Real-time posix: an overview. In *VVConex 93 International Conference, Moscu*. Citeseer, 1993.

[78] M. G. Harbour, M. A. Rivas, J. G. García, and J. P. Gutiérrez. Implementing and using execution time clocks in ada hard real-time applications. In *International Conference on Reliable Software Technologies*, pages 90–101. Springer, 1998.

[79] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback control of computing systems*. John Wiley & Sons, 2004.

[80] A. Hess, H. Iyer, and W. Malm. Linear trend analysis: a comparison of methods. *Atmospheric Environment*, 35(30):5211–5222, 2001.

[81] T. Hill, M. O'Connor, and W. Remus. Neural Network models for time series forecasts. *Management Science*, 42(7):1082–1092, 1996.

[82] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Online workload monitoring with the feedback of actual execution time for real-time systems. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 764–769. European Design and Automation Association, 2017.

[83] D. Hull, W.-c. Feng, and J. W. Liu. Operating system support for imprecise computation. *Flexible Computation in Intelligent Systems: Results, Issues, and Opportunities*, 1996.

[84] IEEE Standards Project P1003.4b. Portable operating system interface (posix)— part 1: Realtime system api extension. In *Draft Standard for Information Technology*. The Institute of Electrical and Electronics Engineers, 1993.

[85] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[86] F. Kerasiotis, A. Prayati, C. Antonopoulos, C. Koulamas, and G. Papadopoulos. Battery lifetime prediction model for a wsn platform. In *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, pages 525–530. IEEE, 2010.

[87] K. E. Kunkel, K. Andsager, and D. R. Easterling. Long-term trends in extreme precipitation events over the conterminous United States and Canada. *Journal of Climate*, 12(8):2515–2527, 1999.

[88] E. A. Lee. Cyber physical systems: Design challenges. *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, 2008.

[89] J. P. Lehoczky. Real-time queueing theory. In *Real-Time Systems Symposium, 1996., 17th IEEE*, pages 186–195. IEEE, 1996.

[90] B. Li and K. Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *Selected Areas in Communications, IEEE Journal on*, 17(9):1632–1650, 1999.

[91] K.-J. Lin and S. Natarajan. Expressing and maintaining timing constraints in flex. In *Real-Time Systems Symposium, 1988., Proceedings.*, pages 96–105. IEEE, 1988.

[92] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 2, pages 1319–1324. IEEE, 2002.

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

[94] J. W. Liu. *Real-time systems.* Prentice Hall, 2000.

[95] J. W. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao. *Algorithms for scheduling imprecise computations.* Springer, 1991.

[96] C. Lozoya, P. Martí, M. Velasco, and J. M. Fuertes. Control performance evaluation of selected methods of feedback scheduling of real-time control tasks. *IFAC Proceedings Volumes*, 41(2):10668–10673, 2008.

[97] C. Lu, Y. Lu, T. F. Abdelzaher, J. Stankovic, S. H. Son, et al. Feedback control architecture and design methodology for service delay guarantees in web servers. *Parallel and Distributed Systems, IEEE Transactions on*, 17(9):1014–1027, 2006.

[98] C. Lu, J. Stankovic, G. Tao, S. H. Son, et al. Design and evaluation of a feedback control edf scheduling algorithm. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 56–67. IEEE, 1999.

[99] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms*. *Real-Time Systems*, 23(1-2):85–126, 2002.

[100] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *Parallel and Distributed Systems, IEEE Transactions on*, 16(6):550–561, 2005.

[101] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 156–163. ACM, 2002.

[102] Y. Ma and L. S. Indrusiak. Hardware-accelerated analysis of real-time networks-on-chip. *Microprocessors and Microsystems*, 53:81–91, 2017.

[103] P. Marti, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes. Optimal state feedback based resource allocation for resource-constrained control tasks. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 161–172. IEEE, 2004.

[104] A. I. McLeod, K. W. Hipel, and B. A. Bodo. Trend analysis methodology for water quality time series. *Environmetrics*, 2(2):169–200, 1991.

[105] J. Miranda and M. G. Harbour. A proposal to integrate the posix execution-time clocks into ada 95. In *International Conference on Reliable Software Technologies*, pages 344–358. Springer, 2003.

[106] R. M. Murray. Optimization-based control. *California Institute of Technology, CA*, 2009.

[107] R. M. Murray, J. Hauser, A. Jadbabaie, M. B. Milam, N. Petit, W. B. Dunbar, and R. Franz. Online control customization via optimization, based control. *Software-Enabled Control Inf. Technol. Dyn. Syst*, page 149, 2003.

[108] E. Nassor and G. Bres. Hard real-time sporadic task scheduling for fixed priority schedulers. In *Proceedings International Workshop on Responsive Systems*, pages 44–47, 1991.

[109] K. M. Obenland. The use of posix in real-time systems, assessing its effectiveness and performance. *The MITRE Corporation*, 2000.

[110] P. Pedreiras, L. Almeida, and P. Gai. Ftt-ethernet: A platform to implement the elastic task model over message streams. In *Factory Communication Systems, 2002. 4th IEEE International Workshop on*, pages 225–232. IEEE, 2002.

[111] M. Qi and G. P. Zhang. Trend time–series modeling and forecasting with Neural Networks. *IEEE Transactions on Neural Networks*, 19(5):808–816, 2008.

[112] R. Rajkumar, I. Lee, L. Sha, and J. A. Stankovic. Cyber-physical systems: The next computing revolution. *Design Automation Conference*, pages 731–736, 2010.

[113] P. Ramanathan. Graceful degradation in real-time control applications using (m, k)-firm guarantee. *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, pages 132–141, 1997.

[114] P. Ramanathan. Graceful degradation in real-time control applications using (m, k)-firm guarantee. *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, pages 132–141, 1997.

[115] G. C. Reinsel and G. C. Tiao. Impact of chlorofluoromethanes on stratospheric ozone: A statistical analysis of ozone data for trends. *Journal of the American Statistical Association*, 82(397):20–30, 1987.

[116] M. Ryu and S. Hong. Toward automatic synthesis of schedulable real-time controllers. *Integrated Computer-Aided Engineering*, 5(3):261–277, 1998.

[117] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2):14, 2009.

[118] P. K. Sen. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American Statistical Association*, 63(324):1379–1389, 1968.

[119] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Real-Time Systems Symposium, 1996., 17th IEEE*, pages 13–21. IEEE, 1996.

[120] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155, 2004.

[121] H. Shang, J. Yan, M. Gebremichael, and S. M. Ayalew. Trend analysis of extreme precipitation in the Northwestern Highlands of Ethiopia with a case study of Debre Markos. *Hydrology and Earth System Sciences*, 15(6):1937–1944, 2011.

[122] K. G. Shin and C. L. Meissner. Adaptation and graceful degradation of control system performance by task reallocation and period adjustment. In *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*, pages 29–36. IEEE, 1999.

[123] D. Simon, A. Seuret, and O. Sename. On real-time feedback control systems: requirements, achievements and perspectives. In *International Conference on Systems and Computer Science*, 2012.

[124] K. A. Smith and M. I. Seltzer. File system aging - increasing the relevance of file system benchmarks. In *SIGMETRICS*, 1997.

[125] R. L. Smith. Extreme value analysis of environmental time series: an application to trend detection in ground-level ozone. *Statistical Science*, pages 367–377, 1989.

[126] A. J. Smola and B. Schölkopf. A tutorial on Support Vector Regression. *Statistics and Computing*, 14(3):199–222, 2004.

[127] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, C. Lu, et al. Feedback control scheduling in distributed real-time systems. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 59–70. IEEE, 2001.

[128] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *OSDI*, volume 99, pages 145–158, 1999.

[129] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium, 1995. Proceedings*, pages 164–173. IEEE, 1995.

[130] G. Tiao. Use of statistical methods in the analysis of environmental data. *The American Statistician*, 37(4b):459–470, 1983.

[131] Y. Tipsuwan and M.-Y. Chow. Control methodologies in networked control systems. In *Control Engineering Practice*, pages 1099 – 111, 2003.

[132] C. Tres, L. B. Beck, and E. Nett. Real-time tasks scheduling with value control to predict timing faults during overload. In *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC'07. 10th IEEE International Symposium on*, pages 354–358. IEEE, 2007.

[133] H. Visser and J. Molenaar. Trend estimation and regression analysis in climatological time series: an application of structural time series models and the Kalman filter. *Journal of Climate*, 8(5):969–979, 1995.

[134] C. Whitnall, E. Oswald, and L. Mather. An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis. In *International Conference on Smart Card Research and Advanced Applications*, pages 234–251. Springer, 2011.

[135] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al. The worst-case execution-time problem: overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.

[136] W. H. Wolf. Cyber-physical systems. *IEEE Computer*, 42(3):88–89, 2009.

[137] Y. Xu, A. Cervin, and K.-E. Årzén. Harmonic scheduling and control co-design. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*, pages 182–187. IEEE, 2016.

[138] R. Yan, H. Sun, and Y. Qian. Energy-aware sensor node design with its application in wireless sensor networks. *IEEE transactions on instrumentation and measurement*, 62(5):1183–1191, 2013.

[139] W. Yan. Toward automatic time-series forecasting using Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1028–1039, 2012.

[140] K. Zeng, Y. Lu, H. Wan, and J. Shu. Efficient storage management for aged file systems on persistent memory. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 1773–1778. European Design and Automation Association, 2017.

[141] G. P. Zhang and M. Qi. Neural Network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514, 2005.

[142] R. Zhang, C. Lu, T. F. Abdelzaher, J. Stankovic, et al. Controlware: A middleware architecture for feedback control of software performance. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 301–310. IEEE, 2002.

[143] X. Zhang, K. D. Harvey, W. Hogg, and T. R. Yuzyk. Trends in Canadian streamflow. *Water Resources Research*, 37(4):987–998, 2001.

[144] X. Zhang, F. W. Zwiers, and G. Li. Monte Carlo experiments on the detection of trends in extreme values. *Journal of Climate*, 17(10):1945–1952, 2004.

[145] Y. Zhu and F. Mueller. Feedback edf scheduling exploiting dynamic voltage scaling. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 84–93. IEEE, 2004.