REAL-TIME SCHEDULING OF SENSOR AND ACTUATOR NETWORKS

By

PADMAVATHY PAGILLA

Bachelor of Engineering

Osmania University

Hyderabad, India

1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2005

REAL-TIME SCHEDULING OF SENSOR AND ACTUATOR NETWORKS

Thesis Approved:

Dr. N. Park

Thesis Adviser

Dr. K.M. George

Committee Member

Dr. V. Sarangan

Committee Member

Dr. A. G. Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincerest appreciation to my major advisor, Dr. Park for his encouragement, inspiration, guidance and his understanding of my situation throughout my course work and during this thesis work. I would like to thank my committee members, Dr. George and Dr. Sarangan, for their help and constructive comments during the proposal presentation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Advances in sensing and actuation have led to increased capabilities in design, monitoring and manipulation of complex physical systems. These advances offer enormous opportunities for applications in every scientific discipline. Embedded sensors and actuators can be found in all walks of our lives, in our houses, transportation, communication, entertainment, and manufacturing. Embedded systems such as sensors, actuators, and controllers interact with the physical world to perform specific functions. Most modern engineering systems contain embedded sensors and actuators, and it is expected that reliance of large number of such devices working together in future applications will be enormous. With growing number of applications, there is a need to network a number of embedded sensors and actuators to perform complex tasks. One application that has increasingly used embedded systems is the automotive industry. Modern automobiles contain a network of embedded sensors and actuators to provide a safe, comfortable driving experience.

A sensor network is a passive system capable of monitoring the state of the environment or a physical system, but it is unable to change it. Actuators can change the behavior of the underlying physical system or environment based on information from sensor networks. Although there are situations where only sensor networks are needed, for example security monitoring, surveillance, etc., there are a number of physical engineering applications where network of actuators must be used in conjunction with sensors.

Recent advances in sensor and actuator technology have provided us with cheap, customizable, and embedded sensor and actuator systems which are capable of wireless communication with each other. Existing physical systems are using SANs to operate efficiently and improve performance. For example, modern aircraft control systems extensively use SANs and make autonomous decisions without intervention from the human operators. There are a number of emerging applications areas which were conceived with SANs as a primary backbone. For example, formation of low cost unmanned aerial vehicles (UAVs) that are being conceived to perform surveillance and reconnaissance operations will require extensive use and development of SANs.

Research activities in SANs is growing at a tremendous pace due to their importance in growing number of applications. In particular there has been a surge in research activity in wireless sensor networks due to their enormous advantages; deploying and maintaining a wired communication network of a large number of nodes is impractical. A recent compilation of results dealing with a number of issues in wireless sensor networks can be found in [1]. Discussions, existing results, and future challenges on a number of key issues in wireless sensor networks is given by researchers in the field. Issues such as network protocols, data storage and manipulation, security, energy efficiency, localization and management. The book also discusses two useful applications of sensor networks, surveillance and habitat monitoring.

Some key issues arising from the use of sensor networks in control applications are discussed in [2]. It discusses how the classical control theory is insufficient when modeling distributed control problems that involve issues such as communication delay and time synchronization between components. It presents a model for the distributed system composed of continuous time-trigger components at the low level and discrete event-triggered components at the high level. Opportunities and challenges related to mobile SANs with micro sensing and actuation are discussed in [3]; existing and emerging technologies are discussed giving future directions.

Recent advances in wireless technology has fueled the growth of wireless sensor networks which has attracted considerable research attention. Low-cost sensors equipped with wireless network interfaces have paved the way for their use in large-scale sensor networks consisting of hundreds to thousands of sensor nodes. Growing applications of such large sensor networks are arising in many areas such as military, industry, and homes. Protection of military establishments, security of nuclear installations, power plants, border surveillance are conceived as some critical areas where sensor networks can provide significant benefits. A general model of a real-time system consisting of a number of sensors and actuators is shown in Fig. 1.1.

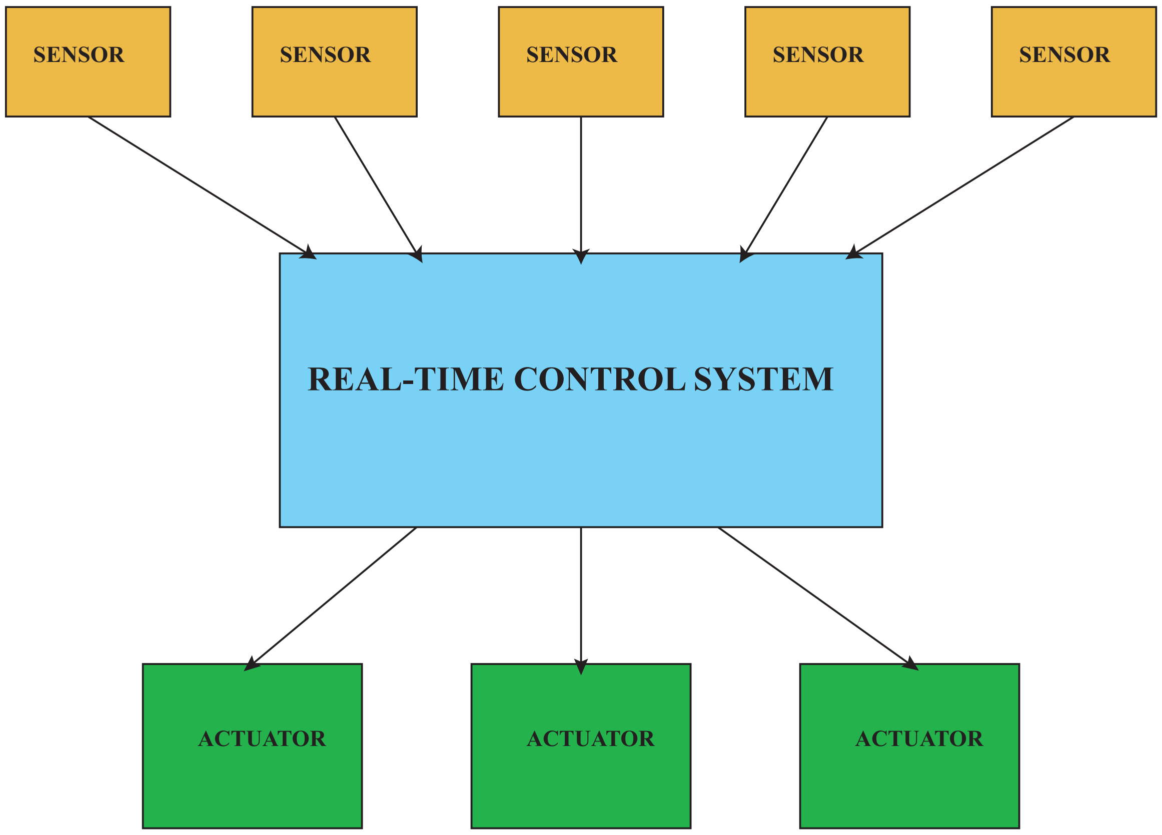

Figure 1.1: Real-time system model

Sensing and actuation are prevalent in nature. Sensors detect the environment, whereas actuators affect the environment. Combined together, a network of sensors and actuators can mutually complement the capabilities of sensors and actuators which can be used intelligently used in a number of applications. Sensors do not have the ability to change the environment. Actuators can enhance the sensing capability by opening valves to transport fluid at a prescribed rate, pointing cameras, aiming antennae at correct locations, and in some cases repositioning of sensors to enhance sensing in a particular direction. As a result, the idea of a network of sensors and actuators to perform certain key functions have captured the imagination of a lot of researchers. Contributions from a number of fields are necessary to build efficient and reliable sensor and actuator networks.

Sensors and actuators play an essential part in almost all engineering systems. Manufacturing plants, chemical plants, air traffic control systems, transportation vehicles such as airplanes, ships, automobiles, and many more have sensor and actuators embedded in them. Feedback is an integral part of many of these engineering systems and involves the use of sensors and actuators. Feedback is responsible for normal operation of a process even in the event of a number of variations such as poor model of the process, disturbances acting on the process, etc. The sensor senses the output of a process or a plant and the actuators provide adequate correction to keep a required output variable at a constant value or vary it in a prescribed way. Therefore, all feedback control systems involve sensors and actuators in their loop. For example, a modern aircraft consists of a large number of sensors and actuators and as a result a substantial number of feedback control loops which render modern aircraft to take-off, fly, and land with minimal human intervention.

Although there is a potential for substantial use of real-time sensor and actuator networks in a large number of applications, our ability to make these networks reliable, robust, and perform in a satisfactory and safe manner pose significant challenges. Some of the challenges include power consumption, data storage and manipulation, network protocols, security, timeliness of obtaining data, scheduling tasks in real-time without missing deadlines, communication, and configuration. Substantial research activity is being carried out in these areas to design and develop efficient sensor and actuator networks.

The focus of research in this study is on real-time scheduling of sensor and actuator networks. These networks can improve their performance by increasing the sampling and actuation rates in the feedback control systems. In sensor and actuator networks, and in many other real-time applications, tasks need to be accomplished before their deadlines within the available amount of resources. There is definitely a trade-off between the schedulability of a given set of task and task rates with that of the available resources; if we have a large amount of resources, we can schedule any task set at higher rates without any problem. Therefore, the problem in real-time scheduling is how do we optimally assign task rates with higher resource efficiency without missing the task deadlines. Assigning tasks based on worst case scenarios is computationally feasible when resources are available but is highly inefficient. Dynamically assigning task rates as well as resources can generally achieve higher resource efficiency and good performance.

Real-time scheduling of tasks to achieve higher sampling rates has been studied in the recent past. Some key contributions include computation of optimal task frequencies by minimizing a performance index. Typically, worst case execution times were used in the analysis and scheduling of tasks. This leads to inefficient use of resources when the worst case task execution times is seldom encountered. In this research, use of a execution times that are below the worst case times but above the best case execution times is investigated. It is shown that higher frequencies can be achieved while minimizing the performance index. One problem that arises when considering execution times smaller than worst case is that when the worst case execution of a task arises, then such a task overrun must be handled and deadlines must be guaranteed. We investigate several overrun handling strategies and show that the proposed approach leads to higher frequencies and more efficient usage of resources.

The remainder of the thesis is organized as follows. In Chapter 2, background description of three key areas, real-time systems, feedback control systems, and real-time scheduling, is given. Real-time task scheduling of sensor and actuator networks in feedback control systems is given in Chapter 3. Real-time control system performance is described in terms of minimization of a performance index. Real-time scheduling and how the task execution times affect the performance index as well as optimal frequencies is described. Efficient overrun handling approaches are discussed and analyzed; these allow scheduling of tasks with average execution times and efficiently handling overruns while guaranteeing hard deadlines for the tasks. A summary of the work accomplished and some potential future work in this area is given in Chapter 4.

CHAPTER 2

Background

In the following, we give a description of three key areas: Real-time systems, feedback control systems, and scheduling. Section 2.1 discusses key aspects of real-time systems, such as its definition, various components, and its importance in many practical applications. Section 2.2 gives a brief review of real-time feedback control systems including a summary of key components that are involved in the design and implementation of a feedback control system. Section 2.3 gives a brief description of the scheduling theory with most commonly available algorithms, their characteristics, and advances made in scheduling theory over the last thirty years.

2.1 Real-Time Systems

Real-time systems are defined as those systems in which the correctness of computation depends not only on the logical result of the computation, but also on its timeliness [4]. Real-time systems vary from a small micro-controller and a limited software situated within products such as a microwave oven to a number of microprocessors working in parallel to control a large factory automation floor where a variety of time-critical tasks are scheduled. The main difference between real-time systems and the other computer systems is the importance of correct timing behavior in addition to providing correct logical results. Conventional real-time systems have been designed for applications such as process control, command and control, automation, etc. Real-time systems are becoming increasingly more complicated with increasing demand for higher productivity and improved performance from physical engineering systems.

The two main components that characterize real-time systems are time and reliability. Time is a very important resource to manage in real-time systems. The assignment and scheduling of the tasks must be done before their deadlines. Messages must be sent and received in a timely manner between the interacting real-time tasks. Reliability is an important component of any real-time system since a failure of executing a task may lead to adverse situations such as extreme economical disaster and loss of human life.

Many engineering applications involve a controlling and a controlled system. The controlled system is typically a physical engineering system where a process or a product is being made. The controlling system is a real-time system that provides inputs to the process based on available sensor signals. The controlling system interacts with its environment using information about the environment available from various sensors. Timely processing of sensed information and decisions made based on this information is critical for the environment to behave in the required manner. The most common constraints [5, 13] for real-time computer tasks can be classified according to their relative importance, which is the priority level, and further according to how they are timed, that is, periodic, aperiodic, or sporadic. The damage that can occur when timing constraints are not met depends on the application; it can be disastrous for a real-time system that is controlling a nuclear power plant.

The main focus areas in which research is being actively pursued in real-time systems can be categorized into the following [13]: (1) Scheduling, (2) Fault tolerance, (3) Real-time computer architectures, (4) Real-time operating systems, and (5) Real-time communication. The scheduling and task assignment problem in real-time systems involves the process of determining when (time) and where (processor) each task will be executed given a set of real-time tasks and the resources in the system. A real-time system is said to be fault tolerant if it delivers the expected service, i.e., the execution of tasks in a timely and reliable manner, even in the presence of faults in its hardware and/or software. Predictable timing behavior even in the presence of sensors and/or actuator faults is sought of fault tolerant real-time systems.

Real-time architectures must provide the speed and predictability in executing tasks in a timely manner, handling interrupts, and monitoring and sensing the physical process it is controlling. Dedicated real-time architectures can differ significantly from architectures used in other computing environments. For example, real-time systems try to avoid the use of caches because of uncertainty of cache hit/miss causes unpredictable memory access delays and thus delays in execution of timed tasks.

A real-time operating system differs from the general purpose operating system in that many of the issues of general purpose operating systems may not be relevant to real-time systems. For example, issues such as file system support, virtual memory, and security are not very important for real-time systems. However, it is crucial to have reliable handling of interrupts, and scheduling with timing constraints. Time constrained communication is essential for all real-time communications. For example, in a physical process an actuator input is generated in a periodic manner based on signals from different sensors; so communication between sensors and the processor, processing of sensor signals, communicating the processed input to the actuators must all be performed within a given time period. Hence, communication rate and its reliability is crucial in all real-time tasks.

2.2 Feedback Control Systems

In most physical systems, the dynamics of the underlying process is continuous; for example, aircraft systems, automobile systems, robots, manufacturing processes, chemical processes, etc. Although the dynamics is continuous, the control algorithm implementation, to achieve the desired behavior for the physical systems, almost always is digital due to widespread use of digital computers. For large-scale control systems consisting of SANs, where a number of small systems are being controlled, such as control of an aeroplane, there are a number of tasks that need to be performed by the controller, i.e., the digital computer that is controlling the physical system. These tasks generally have different priority of implementation and for some it is critical to maintain exact timing. Issues such as acquisition of sensor data, processing of the data, design of the control algorithm, computation of the control algorithm, and task scheduling affect the system performance. The critical part of all real-time control systems is task scheduling based on their priorities. An approach that would enhance the system performance requires consideration of the control algorithm, sensor and actuator networking issues, task scheduling in conjunction with the available computing resources.

2.2.1 Components of Real-Time Control Systems

A typical real-time control system is shown in Figure 2.1. The function of each com-

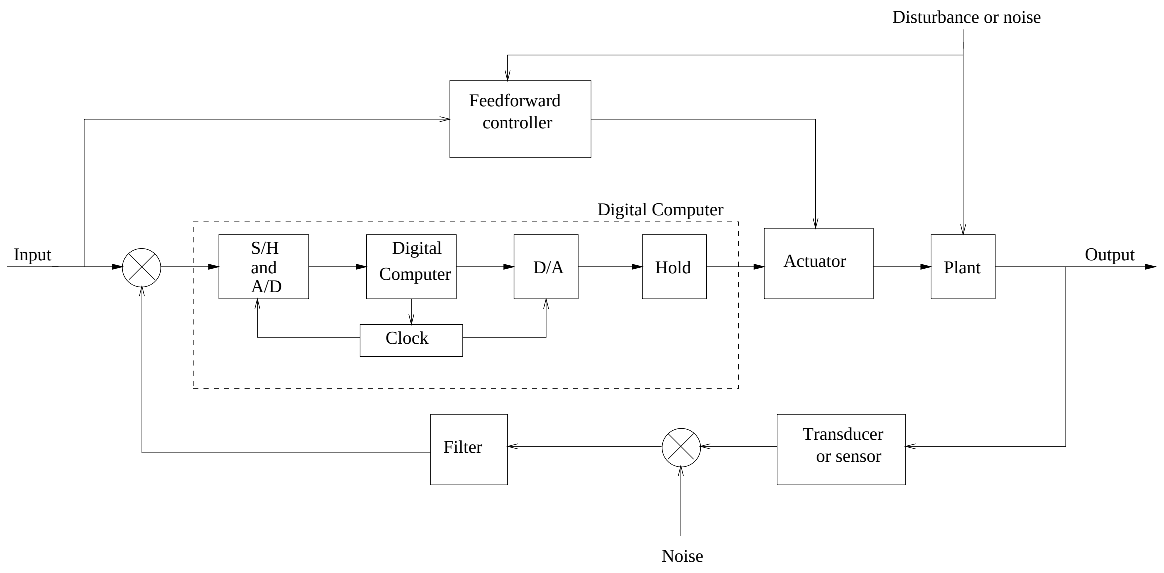

Figure 2.1: Block diagram of a real-time control system

ponent of the real-time control system shown in Figure 2.1 are explained briefly in the following:

- **Sample and Hold Circuits (S/H)**: Most of the physical systems are analog systems and the variables associated with the system are continuous in nature and as such cannot be fed directly to the digital computer for processing. The analog signal must be converted into a form suitable to the digital computer. *Sample* and *hold* operations are the first operations in the process of conversion. A sampler in a digital system converts an analog signal into a series of pulses. And the hold circuit holds the value of the sampled pulse over a specified period of time.

- **Analog to Digital Converter (A/D)**: An analog-to-digital converter, also called an encoder, is a device that converts an analog signal into a digital signal, usually a numerically coded signal. The analog signal can (theoretically) take values with infinite precision whereas the digital signal can take a value from a finite set (as decided by the "word" capacity of the processor used). So, the process of converting an analog signal into a digital signal is an approximation. This approximation process is called *quantization*. Sample and hold circuit mentioned earlier is often an integral part of commercially available A/D converters.

- **Digital Computer**: Digital computer or microprocessor shown in performs all the functions such as processing of data, computation of the control algorithm, and scheduling of all the tasks in a timely manner. Often, the reference signal that the output of the plant is supposed to follow is also generated in the digital computer or microprocessor. The digital computer issues a control signal to the actuator through the necessary D/A conversion in a timely sequenced fashion. Such systems are also called computer-controlled systems.

- **Digital to Analog Converter (D/A)**: The Digital-to-Analog Converter is an interface between digital computer and the actuator, which is an analog device in many applications.

- **Hold Operation**: Sampling operation mentioned earlier produces a series of pulses which are instantaneous values of the signal being sampled. The function of the hold circuit is to reconstruct the analog signal that has been sampled and fill the spaces between the sampling periods. The hold circuit extrapolates the output signal between the successive points according to some prescribed manner. When the signal is held constant between samples, the hold circuit is called a *zero order hold*, which is generally used in control systems. First order and higher order hold circuits are also available.

- **Clock**: The clock shown in the figure serves to streamline the operation of the digital computer and the A/D and D/A operations. The idea is that one operation is performed for each clock-tick. All the time periods are evaluated in terms of clock-ticks.

- **Actuator**: Actuator is an element that manipulates one or more of the variables of the plant being controlled in such a way that the required output is achieved. Actuator receives a signal proportional to the difference between the reference input and the output.

- **Plant or Process**: A plant is any physical object to be controlled. The objective is to maintain one or more of the variables associated with the plant at their desired values. These variables could be temperature, pressure, flow rate, position, velocity, etc. For example, a furnace (temperature is the controlled variable), chemical reactor (concentration and amount of the reactants could be controlled variables), aircraft dynamics, automobile dynamics, etc.

- **Transducer or Sensor**: A transducer/sensor is a device that converts an input signal in one form into an output signal of possibly another form. Sensors are required in control systems to measure the output.

- **Filter**: The feedback signal coming from the sensor of a controlled variable contains useful information related to disturbances (external disturbances and variations in hardware parameters). It may often include high frequency noise introduced in the process of measurements of the output taken using sensors. Such noise signals could be too fast for the control system to correct. Therefore, *low-pass filtering* is often needed to allow good control performance.

- **Feedforward controller**: When a disturbance enters the process, the controlled variable deviates from its desired value and, on sensing the error, the feedback controller manipulates the process input in such a way favoring the reduction of error. The main limitation of a feedback control system is that in order for it to compensate for disturbances, the controlled variable must first deviate from its desired value. Feedback control acts upon an error between the set-point and the controlled variable. This means that once a disturbance enters a process, it must propagate through the process and force the controlled variable to deviate from the set-point before corrective action can be taken. Feed-forward control compensates for disturbances before they affect the controlled variable. In this scheme, the disturbances are measured before they enter the process and required value of the manipulated variable to maintain the controlled variable at its desired value is calculated; implementation of such a scheme often results in undisturbed controlled variable. Success of *disturbance feed-forward control scheme* depends on the ability to:

  - measure the disturbance; and
  - estimate the effect of the disturbance on the controlled variable, so that it can be compensated.

  In many real-time control systems, the feed-forward control scheme also resides in the computer controlling the process.

2.3 Scheduling

Scheduling theory addresses the problem of meeting the specified timing requirements. To satisfy these timing requirements of real-time systems, the scheduling algorithms should be considered with the timing behavior of the system. Early fundamental contributions to scheduling was made in [12]; both the Rate Monotonic Algorithm (RMA) and the Earliest Deadline First (EDF) algorithm were introduced in this paper for the first time. A number of extensions of the two algorithms have been proposed since then [14]. A recent article [15] gives a historical perspective and a comprehensive survey of most of the existing real-time scheduling algorithms. Recent research activity includes scheduling on multiple processors [18]. The power of high performance computing is increasingly being used in real-time signal processing applications, such as in factory automation, radar and sonar [8].

2.3.1 Classification of real-time systems for scheduling

In most operations research scheduling problems [4], there is a fixed system having completely specified and static characteristics. The characteristics of real-time tasks that affect the scheduling algorithm are periodic or aperiodic tasks, preemptible and non-preemptible tasks and independence, resource and placement constraints and the deadlines (hard or firm deadlines). A classification of real-time systems for scheduling is shown in Figure 3.1 [16].

If the result of not meeting a deadline is catastrophic, then the real-time tasks are said to be *hard* otherwise they are *soft*. If the scheduler makes the scheduling decisions at run time, then the scheduler is called *dynamic*, and if the scheduler makes decisions during the compile time, then it is called *static*. Scheduling is said to be *preemptive* scheduling if the current task can be interrupted for the execution of another higher priority task. If the current task cannot be interrupted, it is said to be *nonpreemptive* scheduling.

A real-time application is usually comprised of a set of cooperating tasks. A task is said

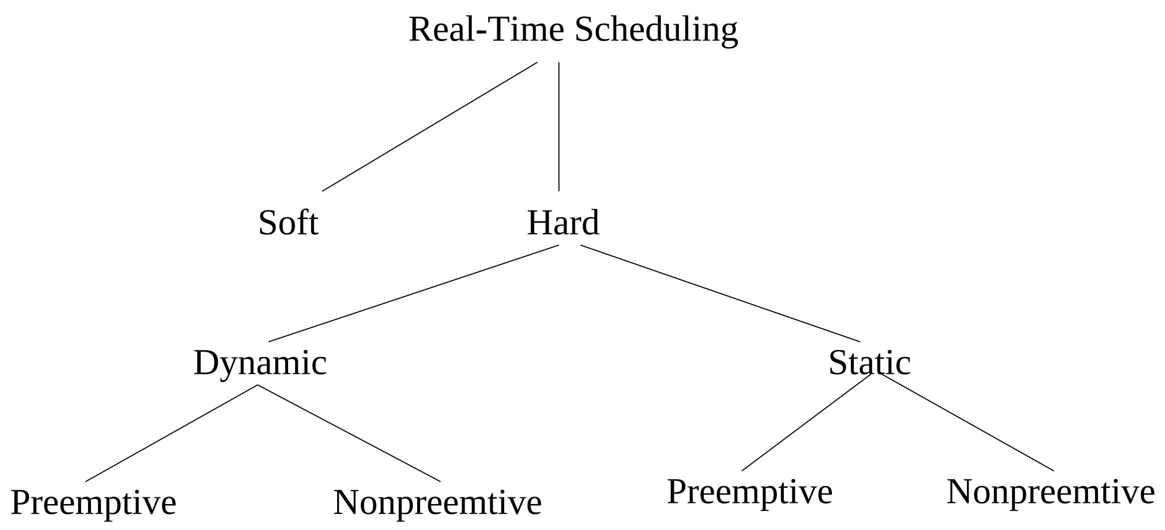

Figure 2.2: Classification of Real-Time Systems for Scheduling

to be *Periodic* if the system cannot function without completing the tasks in time, i.e., it is time-critical. Deadlines of critical tasks should be met in the system for periodic tasks. The tasks that are activated when some event is triggered are said to be *Aperiodic* tasks. When the task is time-critical, then the aperiodic task will have a deadline and if the task is not time-critical, it will not have any deadline by which it must complete its execution.

2.3.2 Scheduling algorithms

A dynamic scheduler decides which task is to be scheduled at run time. In dynamic scheduling the scheduler may schedule independent or dependent tasks. The focus in this study will be on scheduling of independent and periodic tasks. The two scheduling algorithms that fall into this cateogory are the Rate Monotonic Assignment (RMA) and Earliest Deadline First (EDF) algorithms. In RMA, the assignment of priorities to the tasks is made with respect to their request rates without depending on the run time; in particular, tasks with higher request rates will have higher priorities. The RMA algorithm is a static algorithm, i.e., it assigns static task priorities. Tasks with urgent request will be assigned higher priority in rate monotonic priority assignment. It turns out that the RMA algorithm is optimum in the sense that if the task cannot be scheduled by RMA, no other fixed priority assignment can schedule the task. In RMA all the tasks must meet their deadlines. The EDF algorithm is an optimal dynamic preemptive algorithm that is based on dynamic priorities. In EDF, priorities are assigned to tasks according to the deadlines of their current requests. If the deadline of a particular request is the nearest, then it will be assigned the highest priority, and the task with the furthest deadline is assigned the lowest priority. As opposed to the RMA in which priorities do not change one, the priorities of tasks in EDF changes with time, and hence, it is a dynamic scheduling algorithm. It turns out that when EDF algorithm is used to schedule a set of tasks there is no processor idle time prior to an overflow.

Based on the seminal work of [12], extensive progress was made to modify and improve based on specific applications or conditions [15]. The real-time system model of [12] started with the following assumptions:

1. all tasks are periodic,
2. all tasks are released at the beginning of period and have a deadline equal to their period,
3. all tasks are independent, i.e., tasks have no resource or precedence relationships,
4. all tasks have fixed computation time, which is less than or equal to their period,
5. no task may voluntarily suspend itself,
6. all tasks are fully preemptible,
7. all overheads are assumed to be 0,
8. there is just one processor.

The model assumes that a periodic task is time triggered, and the length of time between releases of successive jobs of task τ_i is a constant T_i, which is called the period of the task. Each job also has a deadline D_i after release. A task is said to have a hard deadline if every job of the task must meet its deadline.

The schedulable bound of a task set is defined as the maximum CPU utilization for which the set of tasks can be guaranteed to meet their deadlines. One disadvantage of the RMA is that the schedulable bound is less than 100%. The CPU utilization of the task τ_i is defined as the ratio of the worst case computing time C_i to the period T_i. The total utilization for n tasks is given by

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.1}$$

In [12], it was shown that for RMA the least upper bound to processor utilization is given by

$$U_n = n(2^{1/n} - 1) \tag{2.2}$$

All the tasks will meet their deadlines if processor resources equal or greater than the upper bound given in (2.2) are available. From (2.2), we obtain $U_1 = 100\%, U_2 = 83\%$, and $U_3 = 78\%$. It was shown that in the limit $U_\infty = 69\%$. Therefore, a set of tasks, irrespective of the number of tasks, whose total CPU utilization is less than 69% will always meet all the deadlines.

The advantage of the EDF algorithm over RMA is that the schedulable bound is 100% for all tasks. The problem with the EDF algorithm, as given in [12], is that there is no way to guarantee which tasks will fail in a overload situation. In RMA, it is clear that the low priority tasks are always the first to fail. However, the EDF algorithm does not have any such priority assignment. Hence, it is possible using the classical EDF algorithm that critical tasks will fail at the expense of a lower priority task. One can also conceive of a mixed algorithm that is a combination of the two discussed algorithms. Discussions and detailed explanations of these algorithms can be found in [7, 12, 14, 15]. Developments in the past thirty years can be summarized into the following key aspects:

- Feasibility analysis: necessary and sufficient conditions have been developed to show that higher utilization levels can be achieved, and overrun sensitivity analysis has been extensively studied.
- Analysis of task interaction: priority inheritance protocols have been developed to analyze task interaction and blocking.
- Inclusion of aperiodic tasks.
- Overload management: techniques were developed to handle variations in task execution times, which led to relaxation of worst case execution times and still ensure that the deadlines of critical tasks are guaranteed.
- Implementation simplifications.
- Multiprocessors and distributed systems: analysis tools were developed for applications involving multiple processors.

CHAPTER 3

Real-Time Task Scheduling of Sensor and Actuator Networks

The main focus of this chapter is to present a real-time scheduling algorithm that can efficiently and systematically handle overruns, achieve optimum frequencies of task execution with given resources, and guarantee the hard deadlines associated with the tasks. The proposed real-time scheduling algorithm is based on the earliest deadline first algorithm, that is, tasks with the earliest deadlines are executed first. Optimal frequencies are computed by minimizing a performance index under resource constraints. It is shown that by assuming execution times lower than worst case execution times significant reduction in performance loss index can be achieved. Further, since average case execution times are considered, overrun handling must be done in the event task execution take longer. Two approaches are given to handle overruns. Simulations on a number of examples are given to illustrate and validate the approach.

The chapter is organized as follows. First, real-time control system performance in terms of minimization of a performance index is described in Section 3.1. Computation of optimal frequencies based on different execution times are discussed in Section 3.2. Section 3.3 gives handling of overruns when execution times less than the worst case times are used. Section 3.4 gives simulations on a number of tasks to illustrate the advantages of considering task execution times less than the worst case times for computing the frequencies of execution of tasks.

3.1 Real-Time Control System Performance

A typical control system consists of development of a continuous-time control algorithm which optimizes a certain performance index. For example, in a radar system the objective is to track a target and the performance index is generally a function of the tracking error, i.e., the difference in the position of the target and the desired value of the position. For this example, a control algorithm that minimizes the performance index is sought. It may be possible in other situations where the performance index may have to be maximized and the objective may be to develop control algorithm that maximizes the performance index. The continuous-time control algorithm is discretized for implementation. To get a good match between the continuous-time control algorithm and the discretized version, the time period used in this process must be as small as possible. Thus, the sampling frequency, defined as the reciprocal of the time period, must be as high as possible. But computing resources place a limitation on how high one can choose the sampling frequency for a particular task.

The problem of optimizing a performance index for a control system is generally stated as follows.

$$\min_{u\in\Omega} J(u) = S(x(t_f),t_f) + \int_0^{t_f} L(x(t),u(t),t)dt \tag{3.1}$$

$$\text{subject to : } \dot{x}(t) = Ax(t) + Bu(t) \tag{3.2}$$

where $J(u)$ is the system performance index, t_f is the final time, $S(x(t_f),t_f)$ and $L(x(t),u(t),t)$ are cost functions that depend on the control input, states and final time. Eqn. (2) is the model of the plant or the physical system that is being controlled. Now, suppose the above minimization problem results in an optimal control $u^*(t)$ and the corresponding optimal performance index given by J^*. The continuous-time optimal control input, $u^*(t)$, is now discretized for digital implementation. The discretized equivalent of the optimal performance index J^* is

$$J_D^*(f) = S(x^*(t_f), t_f) + \sum_{k=0}^{n-1} \int_{kT}^{(k+1)T} L(x^*(t), u^*(kT), t)\,dt \tag{3.3}$$

where T is the period of discretization, i.e., $f = 1/T$ is the frequency at which the continuous-time is sampled, and $u^*(kT)$ is the discrete-version of the optimal control input. In the block diagram of the real-time control system shown in Figure 2.1, all of the tasks such as measurement of output by the sensor, conversions, computation of the control algorithm, communication of the control input to the actuators must be achieved within one sampling period. Notice that $J_D^*(f)$ is a function of the sampling frequency f. The higher the sampling frequency, the more accurate the discretization will be. As the sampling frequency is increased, the discrete time optimal control that is actually implemented will converge to the continuous-time optimal control. But the sampling frequency is directly related to the computing resources used in the digital implementation of the control algorithm. The difference $\Delta J^*(f) = J_D^*(f) - J^*$ called the performance loss index (PLI) must be as small as possible. It is clear that as $f \to \infty$ the magnitude of this difference converges to zero whereas as $f \to 0$ the magnitude of the difference converges to infinity.

The frequency f cannot be arbitrarily increased and is limited by the amount of available computing resources. The main goal of task scheduler for real-time control systems is to maximize this frequency within the constraints of the process capacity and utilization of the available computer.

Noting that the PLI, $\Delta J^*(f)$ exponentially decreases with frequency, for each control task, it was approximated in [17] by

$$\Delta J_i(f_i) = \alpha_i e^{-\beta_i f_i} \tag{3.4}$$

where α_i is the magnitude coefficient, β_i is the decay rate, and i represents the index of the physical system in SANs.

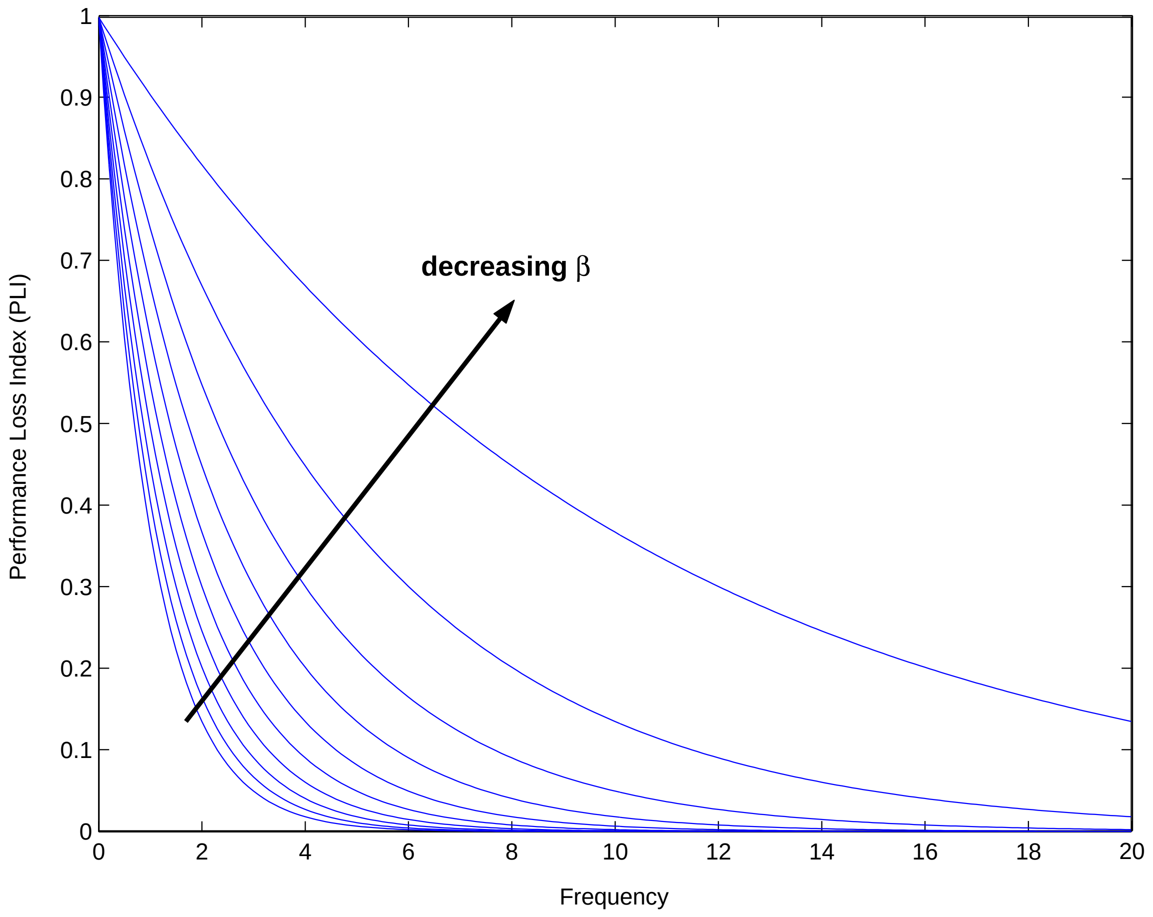

Figure 3.1: Performance loss index with varying β

The performance loss index of the overall large-scale network is defined as

$$\Delta J(f_1,\ldots,f_n) = \sum_i w_i \Delta J_i(f_i) \tag{3.5}$$

where w_i is the i-th design parameter which weighs the influence of the PLI of the i-th system on the overall performance index; equal weighting of all tasks would mean $w_i = 1$ for all i.

3.2 Real-Time Scheduling

The problem now reduces to the efficient scheduling of the tasks within the given computing constraints and control system constraints while minimizing the performance loss index. Based on the given minimum frequency f_{mi} and the worst case execution time C_i for each i-th task, an algorithm to compute the optimal frequencies f_i^{opt}, which minimizes the overall PLI of the system and guarantees the schedulability constraints, was given in [17]. The following optimization problem was posed for minimization of the overall PLI with respect to the task frequencies:

$$\min_{f_1,\dots,f_n} \Delta J = \sum_{i=1}^{n} w_i \alpha_i e^{-\beta_i f_i} \tag{3.6}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} C_i f_i \leq A,\ 0 < A \leq 1$$

$$f_i \geq f_{mi},\ i = 1,\dots,n \tag{3.7}$$

where $C_i, i = 1,\dots,n$ are the task execution times, f_{mi} lower bound on the i-th frequency f_i, and A is the processor utilization factor. The following proposition gives the solution to the above problem.

Proposition 1: [17] Given the objective function (3.6) and the constraints (3.7), there exists a unique optimal solution given by

$$f_i = f_{mi},\ \ i = 1,\dots,p \tag{3.8}$$

$$f_j = \frac{1}{\beta_j}(\ln\Gamma_j - Q), j = p+1,\dots,n,$$

where f_k's are ordered as f_{mk} which are arranged according to the following inequality

$$\Gamma_1 e^{-\beta_1 f_{m1}} \leq \Gamma_2 e^{-\beta_2 f_{m2}} \leq \cdots \leq \Gamma_n e^{-\beta_n f_{mn}} \tag{3.9}$$

and p is the smallest integer such that

$$\sum_{l=1}^{p} C_l f_{ml} + \sum_{l=p+1}^{n} \frac{C_l}{\beta_l}\left(\beta_p f_{mp} + \ln\frac{\Gamma_l}{\Gamma_p}\right) \geq A, \tag{3.10}$$

and

$$\Gamma_j = \frac{w_j \alpha_j \beta_j}{C_j}$$

$$Q = \frac{1}{\sum_{l=p+1}^{n} \frac{C_l}{\beta_l}}\left(\sum_{l=1}^{p} C_l f_{ml} + \sum_{l=p+1}^{n} \frac{C_l}{\beta_l}\ln\Gamma_l - A\right)$$

Proposition 1 gives a result with an implicit assumption that there is flexibility in choosing the sampling frequency of real-time control systems provided it is chosen above a certain minimum bound. This method gives computation of the optimal sampling frequencies within the given levels of processor utilization while guaranteeing schedulability of all tasks. Both the EDF and RMA algorithms are used for scheduling.

The optimal frequencies in [17] were determined based on the worst case execution times. If there are large variations between the worst case execution times (WCETs) and the best case execution times (BCETs), then the optimal frequencies computed in [17] will be substantially less than achievable frequencies for many instances. So, a strategy that can adaptively determine optimal frequencies based on the best case execution times and the worst case execution times is desirable. In real-time systems, BCETs and WCETs are generally available. The real-time control system as shown in Figure 2.1 has a number of components which will provide the constraints to determine BCETs and WCETs. There are four key factors that will determine BCETs and WCETs:

1. **Actuator Constraints:** The response of the actuator plays an important role in the choice of the sampling period. In many engineering systems the actuators are typically dynamic devices such as electromechanical systems which have their own response time. This should be taken into account while implementing a real-time control system and computation of the sampling period used for digital implementation of the controller.

2. **Sensor Constraints:** The sensors and its associated hardware/software play a critical role in all real-time feedback control systems. They not only acquire output data but convert it into a form that can be input to the controller/computer. In some applications, to measure a signal with desired accuracy, multiple sensors are used to measure the output and some sort of averaging is done to get the actual output. In applications such as triangulation of an object in the workspace, several sensors need to

be used to pinpoint the actual position of the object, and thus, data from each sensor must be processed to get the object position before transmitting it to the controller. All these functions of the sensor take time to accomplish, and must be included in the calculation of BCETs and WCETs. Further, in some cases, data from the sensors need to be filtered before it can be used by the controller to generate the control input. Processing of data must be done in real-time, so, it must be accounted for in the sampling period.

3. **Conversion and Sample/Hold Operations:** There are time constraints associated in converting analog data to digital data, and vice-versa. Further, sample/hold operations take-up a portion of the sampling period in which they are performed.

4. **Control Computation:** This will include execution of the relevant real-time program code to generate the control input. The code is generally modular and several sections are triggered based on the difference between the output and input (error) and other logical events. In some cases, only a small section of the code is triggered and the control input is generated based on running that code in real-time; this will generally give the best case time associated with control computation. And in some cases all of the real-time code needs to be implemented, which will give the worst case time for implementing the code.

The above factors can be used to get the BCETs and WCETs for all the systems/tasks. Now, the key question is how does one schedule tasks so that processor utilization is at a maximum while satisfying the constraints. One way is to adjust the frequencies to optimize the performance index and meet the scheduling constraints, that is, one has to ensure that the performance index is minimized and when the tasks require worst case execution times or close to it, then they do not miss the deadlines. Two issues that must be considered in developing a scheduling algorithm based on BCETs and WCETs: (1) What is the initial frequency assignment for the tasks, and how does one adjust it? (2) What is the strategy to handle overruns? One strategy is to use the optimal frequencies generated using BCETs as the initial assignment, and stretch it as overruns happen. Though one can expect strong performance improvement using this strategy, it has the potential to generate too many overruns. Another strategy is to consider an average of BCET and WCET for each task and use that to compute the initial optimal frequency assignments for the tasks, and then devise a strategy to handle overruns. Denote such an execution time as Average Case Execution Time (ACETs). It is expected that the use of ACETs for initial optimal frequency computation will result in considerably less overruns than using BCETs. Further, another strategy is to use a line joining the BCETs and WCETs such as

$$C_{ci} = (1-\gamma_i)C_{wi} + \gamma_i C_{bi} \tag{3.11}$$

where C_{wi} and C_{bi} are the WCET and BCET, respectively, of the i-th task, C_{ci} is the weighted combination of BCET and WCET, and γ_i is the weight. Notice that $C_{bi} \leq C_{ci} \leq C_{wi}$. Also, notice that the average execution time is obtained by setting $\gamma_i = 0.5$. The weight γ_i for each task can be assigned based on some type of confidence level on the task being able to perform either close to the worst case or best case. Further, γ_i can also be chosen based on the weights assigned (w_i) to each task in the performance index, that is, choosing γ_i closer to one (execution time closer to the best case) for a heavily weighted task in the performance index.

Denote the optimal frequencies obtained from the BCETs and WCETs as f_{bi} and f_{wi}, respectively, and the ones obtained using C_{ci} as in (3.11) as f_{ci}. Since the execution times are interlaced, that is, $C_{bi} \leq C_{ci} \leq C_{wi}$, we can also show that the number of optimal frequencies assigned using these three cases are also interlaced. Let p_w, p_c and p_b be the number of optimal frequencies that take the minimum frequency bounds for the execution times, C_{wi}, C_{ci} and C_{ci}, respectively. Then, $p_w \geq p_c \geq p_b$. That is, when BCETs are used to compute the optimal frequencies, fewer number of optimal frequencies, f_{bi}'s, will take the values of minimum frequency bounds, f_{mi}'s. We can show this using the condition for setting the number of optimal frequencies as minimum frequencies, which is given by the following for the three execution time cases:

$$\sum_{j=1}^{p_w} C_{wj} f_{mj} + \sum_{j=p_w+1}^{n} \frac{C_{wj}}{\beta_j} \left(\beta_{p_w} f_{mp_w} + \ln \frac{\Gamma_{wj}}{\Gamma_{p_w}} \right) \geq A \tag{3.12}$$

$$\sum_{j=1}^{p_c} C_{cj} f_{mj} + \sum_{j=p_c+1}^{n} \frac{C_{cj}}{\beta_j} \left(\beta_{p_c} f_{mp_c} + \ln \frac{\Gamma_{cj}}{\Gamma_{p_c}} \right) \geq A \tag{3.13}$$

$$\sum_{j=1}^{p_b} C_{bj} f_{mj} + \sum_{j=p_b+1}^{n} \frac{C_{bj}}{\beta_j} \left(\beta_{p_b} f_{mp_b} + \ln \frac{\Gamma_{bj}}{\Gamma_{p_b}} \right) \geq A \tag{3.14}$$

where $\Gamma_{wj}, \Gamma_{cj}, \Gamma_{wj}$ are given by the following:

$$\Gamma_{qj} = \frac{w_j \alpha_j \beta_j}{C_{qj}}, \;\; q = w, c, b \tag{3.15}$$

Notice that $\Gamma_{bj} \geq \Gamma_{cj} \geq \Gamma_{wj}$. Further, we can also see that the optimal frequencies themselves are interlaced, that is, $f_{bi} \geq f_{ci} \geq f_{wi} \geq f_{mi}$ for $i = 1, \ldots, n$.

Proposition 2: The number of optimal frequencies obtained from using BCETs, WCETs, and ACETs, that take the minimum frequency bounds as their values, satisfy the following:

$$p_w \geq p_c \geq p_b. \tag{3.16}$$

Further, the optimal frequencies obtained from these three cases are also interlaced, that is,

$$f_{bi} \geq f_{ci} \geq f_{wi}. \tag{3.17}$$

3.3 Overrun Handling

In this section, we address the issue of how to schedule overruns for the execution time cases discussed above. One can handle overruns in two ways: (1) By reserving a part of the processor utilization locally for each task based on the optimal frequencies obtained using BCETs or ACETs; that is, overruns are handled for an individual task by decreasing its optimal frequency so that the processor bandwidth assigned to that task is not exceeded and the deadline is met. (2) By handling overruns in a global sense, that is, the bandwidth for the overrun task is created by decreasing the frequencies of all the tasks in an adaptive manner. In either case, it is expected that the interlacing property of the optimal frequencies as well as the number of minimum frequencies assigned will aid in the development of an algorithm to decrease the frequencies locally or globally across all tasks to meet the deadlines as well as maximize processor utilization. It is assumed that the task set is scheduled by the EDF algorithm, which assigns higher priorities to tasks with earlier deadlines.

Overrun's need to be handled in a systematic way such that hard deadlines are not missed. The following example [23] shows that the result of increasing the frequencies without properly considering overruns is a missed deadline. The example consists of scheduling two tasks whose WCET's (C_{wi}), normal executing times and minimum frequency bounds are given in Table 3.3. Assume that the two tasks are executing normally.

Task	WCET: C_{wi} (ms)	Normal Execution Time: c_i (ms)	f_{mi} (Hz)
τ_1	25	20	9.9
τ_2	25	20	20

Also, assume that τ_2 is executing at frequency $f_2 = 40$ Hz and τ_1 is executing at its minimal frequency $f_1 = 9.9$ Hz. The CPU utilization factors are $U_1 = c_1 f_1 = 0.198$ and $U_2 = c_2 f_2 = 0.8$ (approx. 20% and 80% of the available CPU power). Let's assume in an adhoc way that if the worst case arises, task τ_2 could slow down from 40 Hz to 20 Hz to release bandwidth. This adhoc approach would not work as illustrated in Fig. 3.2. Notice that at $t = 100$ ms any overrun of task τ_1 of more than 1 ms will result in missing the hard deadline for task τ_1.

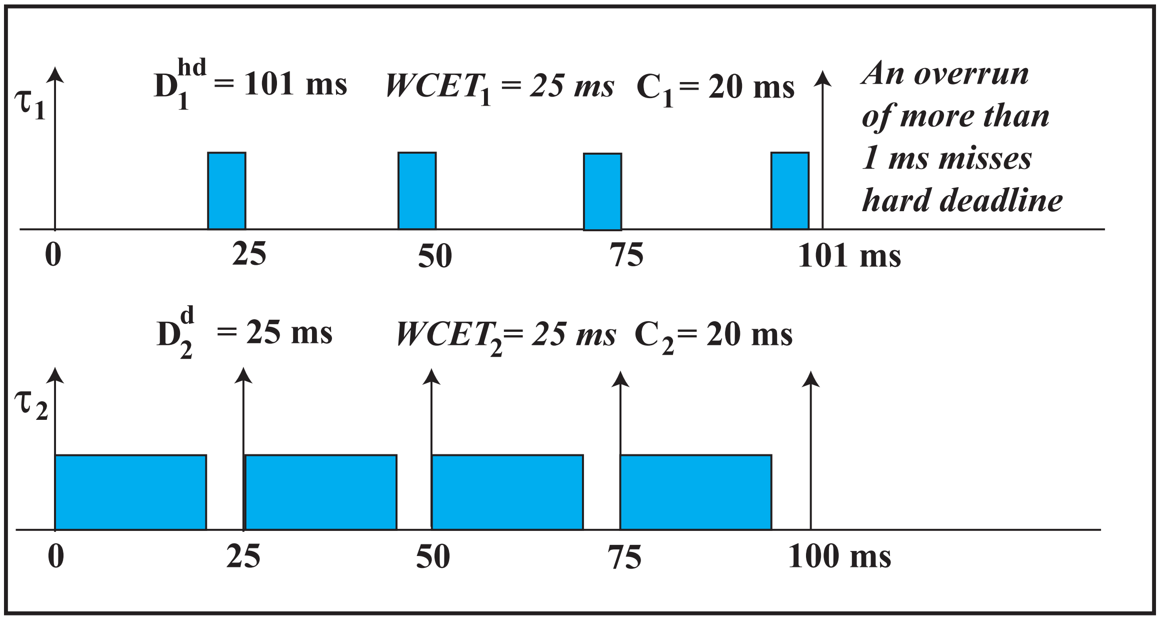

Figure 3.2: Example: Overrun handling using adhoc approach – results in missing hard deadlines

3.3.1 Local Approach

First, some key terms are defined to simplify the understanding of the approach taken in the subsequent material. Each periodic task is described by

$$\tau_i(C_{bi}, C_{wi}, f_{mi}, \Delta J_i(f_i)). \tag{3.18}$$

Further, each task consists of a sequence of jobs $\tau_{i,j}$, which are characterized by the release time $r_{i,j}$, an execution time $c_{i,j}$, and a dynamic deadline $d_{i,j}$. Note that the execution time lies anywhere between the best case and the worst case, that is, $c_{i,j} \in [C_{bi}, C_{wi}]$. It is also required that the dynamic deadline $d_{i,j}$ must be less than or equal to the hard deadline D_i^{hd}, that is, $d_{i,j} \leq D_i^{hd}$. The hard deadline is related to the minimum frequency as

$$D_i^{hd} = 1/f_{mi}. \tag{3.19}$$

Now, the task scheduling goal is to determine the optimal frequencies f_i^{opt} which minimize the overall PLI while guaranteeing that the dynamic deadline of each task ($d_{i,j}$) never exceeds the hard deadline, thus assuring that the minimum frequency for each task is achieved. To achieve this goal, a local approach was suggested by [23] using the normal execution times c_i^n; this approach of overrun handling is described first, followed by a modification of the approach which will give higher frequencies while efficiently handling overruns. The advantage of the local approach is that execution overrun of a job $\tau_{i,j}$ is handled locally, that is, when $c_{i,j} > c_i^n$, its execution is systematically delayed so that other tasks are not affected. Now, suppose that $d_{i,j}^{last}$ is the last deadline used by the server, after handling all the overruns, to schedule job $\tau_{i,j}$, the next job $\tau_{i,j+1}$ of task τ_i will be released at time $r_{i,j+1} = d_{i,j}^{last}$. As a result each job has a variable time period $T_{i,j} = r_{i,j+1} - r_{i,j}$. By ensuring that the variable time period $T_{i,j}$ satisfies $T_{i,j} \leq 1/f_{mi}$, for all j, the hard deadline D_i^{hd} of each task τ_i is guaranteed.

The idea is to reserve a portion of the entire processor bandwidth to each task using a bandwidth server. It is assumed that the optimal frequency f_i^{opt} for each task τ_i is calculated using the SLSS algorithm given by Proposition 1 based on the normal execution time c_i^n. Using the computed optimal frequencies each task τ_i is reserved a bandwidth given by

$$U_i = f_i^{opt} c_i^n. \tag{3.20}$$

The deadline assignment rule of the bandwidth server sets the initial deadline, $d_{i,j}^0$, of the job $\tau_{i,j}$ assuming normal execution time c_i^n as

$$d_{i,j}^0 = r_{i,j} + \frac{c_i^n}{U_i} \tag{3.21}$$

If the job $\tau_{i,j}$ overruns, that is, if the actual execution time $c_{i,j}$ is greater than the normal time c_i^n, then the deadline is extended to

$$d_{i,j}^{last} = r_{i,j} + \frac{C_{wi}}{U_i} = d_{i,j}^0 + \frac{C_{wi} - c_i^n}{U_i} \tag{3.22}$$

This results in the job using the worst case execution time if there is an overrun. Since bandwidth U_i is reserved for the task τ_i, it is not exceeded. The guaranteeing of the deadline depends on the reliability of the bandwidth server in assigning the bandwidth U_i correctly to task τ_i. It is noted that any overrun is treated as the maximum overrun, that is, the new job deadline will take the worst case execution time in the event of any amount of overrun. This approach is very conservative and leads to task frequencies close to minimum frequencies whenever an overrun occurs. In [23], it is shown that the following necessary and sufficient condition on the reserved bandwidth U_i of task τ_i for every i:

$$U_i \geq f_{mi} C_{wi} \tag{3.23}$$

will ensure that all tasks are schedulable with a frequency $f_i \geq f_{mi}$, that is, hard deadlines will be met even when worst case execution times are required. Though the proof of the above condition is given in [23], it is intuitively clear that when the reserved bandwidth is greater than or equal to the worst case bandwidth obtained using minimum frequency and worst case execution time will result in at least assigning the minimum frequency to the task if an overrun occurs.

To incorporate the constraint during the optimization stage, notice that the constraint given by (3.23) must be satisfied by each task. Since $U_i = f_i^{opt} c_i^n$, the constraint (3.23) for every task τ_i becomes

$$f_i^{opt} \geq \frac{C_{wi}}{c_i^n} f_{mi} \tag{3.24}$$

Notice that the original optimization problem given by Proposition 1 already has the constraint that the optimal frequencies must be greater than or equal to the minimum frequencies, that is,

$$f_i^{opt} \geq f_{mi}. \tag{3.25}$$

The constraint given by (3.24) will be the same as in the original optimization problem if the minimum frequencies, f_{mi}, for each task τ_i in the original optimization problem are replaced by new lower bounds $\tilde{f}_{mi}$, which are given by

$$\tilde{f}_{mi} = \frac{C_{wi}}{c_i^n} f_{mi} \tag{3.26}$$

Therefore, overruns can be taken into account in this local approach by computing the optimal frequencies from the optimization problem given by the SLSS algorithm (Proposition 1) using normal execution times c_i^n and modified minimum frequencies bounds given by (3.26).

The local approach described above is conservative because whenever an overrun occurs for a particular task, it will be considered as a worst case overrun, that is, the deadline is postponed by an amount equal to the difference between the worst case execution time (C_{wi}) and the normal execution time (c_i^n). Although this strategy meets the hard deadlines, it is conservative and results in poor usage of resources. In the following, a local method is suggested which considers extension of deadlines when overruns occur in a less conservative manner and thus results in higher efficiency in terms of usage of resources and has the potential of achieving higher frequencies when tasks overrun their normal execution time.

Instead of the normal execution time as considered in the above approach, it is assumed that the best case (C_{bi}) and worst case (C_{wi}) execution times are available and the user has the freedom to choose a linear combination of the two times based on the confidence level on a particular task being able to execute either closer to the best case time or worst case time. The average or normal execution time for each task is taken as

$$C_{ci} = (1-\gamma_i)C_{wi} + \gamma_i C_{bi} \tag{3.27}$$

The optimal frequency f_i^{opt} for each task τ_i is computed using the execution time C_{ci}. For each task τ_i, the following bandwidth is reserved:

$$U_i = f_i^{opt} C_{ci} \tag{3.28}$$

The initial deadline for job $\tau_{i,j}$ of task τ_i is set to be

$$d_{i,j}^0 = r_{i,j} + \frac{C_{ci}}{U_i} \tag{3.29}$$

When an overrun occurs, i.e., the job $\tau_{i,j}$ takes longer time to execute than C_{ci}, then the deadline is extended in the following manner:

$$d_{i,j}^{1} = d_{i,j}^{0} + \frac{1}{2}\frac{(C_{wi} - C_{ci})}{U_i} \tag{3.30}$$

Notice that, at the first instance of the overrun of job $\tau_{i,j}$, the job deadline is extended by a bisection of the difference between the worst case execution time (C_{wi}) and the normal execution time (C_{ci}). Suppose in the event that this first extension of the deadline is not adequate to complete the job, then the deadline of the job $\tau_{i,j}$ is further extended for the second time as

$$d_{i,j}^{2} = d_{i,j}^{1} + \frac{1}{2^2}\frac{(C_{wi} - C_{ci})}{U_i} \tag{3.31}$$

Therefore, when the second overrun occurs, the deadline is extended by a bisection of the remaining time to the worst case execution time. If there are m such overruns before the job $\tau_{i,j}$ is executed, then the last deadline for the job $\tau_{i,j}$ is given by

$$d_{i,j}^{m} = d_{i,j}^{m-1} + \frac{1}{2^m}\frac{(C_{wi} - C_{ci})}{U_i} \tag{3.32}$$

The deadline $d_{i,j}^{m}$ can be simplified to the following:

$$\begin{aligned} d_{i,j}^{m} &= d_{i,j}^{0} + \left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right)\frac{(C_{wi} - C_{ci})}{U_i} \\ &= r_{i,j} + \frac{C_{ci}}{U_i} + \left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right)\frac{(C_{wi} - C_{ci})}{U_i} \end{aligned} \tag{3.33}$$

Notice that for this approach to work we still have to show that the hard deadline is met irrespective of the number of overruns. Recall that the hard deadline for each task τ_i is given by

$$D_i^{hd} = \frac{1}{f_{mi}} \tag{3.34}$$

where f_{mi} is the minimum frequency of the task τ_i. Hence, we have to show that the following is true:

$$\frac{C_{ci}}{U_i} + \left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right)\frac{(C_{wi} - C_{ci})}{U_i} \le D_i^{hd} \tag{3.35}$$

By noticing that

$$\sum_{j=0}^{m} \frac{1}{2^j} = \frac{1-\frac{1}{2^m}}{1-\frac{1}{2}} = 2 - \frac{1}{2^{m-1}},$$

we can write

$$\left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right) = 1 - \frac{1}{2^{m-1}} \tag{3.36}$$

The left-side of (3.35) can be written as

$$\begin{aligned} \frac{C_{ci}}{U_i} + \left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right)\frac{(C_{wi}-C_{ci})}{U_i} &= \frac{C_{ci}}{U_i} + \left(1 - \frac{1}{2^{m-1}}\right)\frac{(C_{wi}-C_{ci})}{U_i} \\ &= \frac{C_{wi}}{U_i} - \left(\frac{1}{2^{m-1}}\right)\frac{(C_{wi}-C_{ci})}{U_i} \end{aligned} \tag{3.37}$$

Since $U_i = f_i^{opt} C_{ci}$,

$$\frac{C_{ci}}{U_i} + \left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right)\frac{(C_{wi}-C_{ci})}{U_i} = \frac{1}{f_i^{opt} C_{ci}}\left(C_{wi} - \frac{(C_{wi}-C_{ci})}{2^{m-1}}\right) \tag{3.38}$$

If the reserved bandwidth for each task τ_i is such that

$$U_i \geq f_{mi} C_{wi}, \tag{3.39}$$

then by using $U_i = f_i^{opt} C_{ci}$, we obtain

$$f_i^{opt} \geq \frac{f_{mi} C_{wi}}{C_{ci}} \tag{3.40}$$

Since $\left(C_{wi} - \frac{(C_{wi}-C_{ci})}{2^{m-1}}\right) > 0$ for all $m \geq 1$, using (3.40) in (3.38), we obtain

$$\frac{C_{ci}}{U_i} + \left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^m}\right)\frac{(C_{wi}-C_{ci})}{U_i} \leq \frac{1}{f_{mi} C_{wi}}\left(C_{wi} - \frac{(C_{wi}-C_{ci})}{2^{m-1}}\right) \tag{3.41}$$

Notice that for all m,

$$\frac{1}{f_{mi}} - \left(\frac{1}{2^{m-1}}\right)\frac{(C_{wi}-C_{ci})}{f_{mi} C_{wi}} \leq \frac{1}{f_{mi}} \tag{3.42}$$

Therefore, (3.35) is true for all m, and hence, for each task τ_i, the hard deadline D_i^{hd} is always met using this strategy. Notice that worst case situation occurs when $m \to \infty$, that is,

$$\lim_{m\to\infty} \left(\frac{1}{2^{m-1}}\right)\frac{(C_{wi}-C_{ci})}{f_{mi} C_{wi}} = 0 \tag{3.43}$$

The following proposition summarizes the results of the proposed approach:

Proposition 3: Given a set of n periodic tasks $\tau_i(C_{wi}, C_{bi}, f_{mi}, \Delta J_i(f_i))$ with total bandwidth $\sum_{i=1}^{n} U_i = A$ and reserved bandwidth $U_i = f_i^{opt} C_{ci}$ with $U_i \geq f_{mi} C_{wi}$, each task τ_i is schedulable meeting its hard deadline $D_i^{hd} = 1/f_{mi}$ when the following bisection strategy is used to extend the deadlines of the m-th overrun of the job $\tau_{i,j}$:

$$d_{i,j}^{m} = d_{i,j}^{m-1} + \frac{1}{2^m} \frac{(C_{wi} - C_{ci})}{U_i} \tag{3.44}$$

where the initial deadline is given by

$$d_{i,j}^{0} = r_{i,j} + \frac{C_{ci}}{U_i} \tag{3.45}$$

Remark 3.3.1 *Notice that instead of using* (3.44), *one can use the following:*

$$\begin{aligned} d_{i,j}^{0} &= r_{i,j} + \frac{C_{ci}}{U_i} \\ d_{i,j}^{m} &= d_{i,j}^{m-1} + \varepsilon_i^m \frac{(C_{wi} - C_{ci})}{U_i} \end{aligned} \tag{3.46}$$

where $0 < \varepsilon_i \leq 0.5$ *for all* $i = 1:n$*. As a result, the left-side of* (3.35) *satisfies*

$$\begin{aligned} \frac{C_{ci}}{U_i} + \left(\varepsilon_i + \varepsilon_i^2 + \cdots + \varepsilon_i^m\right) \frac{(C_{wi} - C_{ci})}{U_i} &= \frac{C_{ci}}{U_i} + \left(\frac{1 - \varepsilon_i^m}{1 - \varepsilon_i} - 1\right) \frac{(C_{wi} - C_{ci})}{U_i} \\ &= \frac{C_{ci}}{U_i} + \left(\frac{\varepsilon_i - \varepsilon_i^m}{1 - \varepsilon_i}\right) \frac{(C_{wi} - C_{ci})}{U_i} \\ &= \frac{C_{wi}}{U_i} + \left(\frac{\varepsilon_i - \varepsilon_i^m}{1 - \varepsilon_i} - 1\right) \frac{(C_{wi} - C_{ci})}{U_i} \\ &= \frac{C_{wi}}{U_i} - \left(\frac{1 + \varepsilon_i^m - 2\varepsilon_i}{1 - \varepsilon_i}\right) \frac{(C_{wi} - C_{ci})}{U_i} \end{aligned}$$

Therefore, using the same arguments as before and $\varepsilon_i \leq 0.5$, *we obtain*

$$\frac{C_{ci}}{U_i} + \left(\varepsilon_i + \varepsilon_i^2 + \cdots + \varepsilon_i^m\right) \frac{(C_{wi} - C_{ci})}{U_i} \leq \frac{1}{f_{mi}} - \left(\frac{1 + \varepsilon_i^m - 2\varepsilon_i}{1 - \varepsilon_i}\right) \frac{(C_{wi} - C_{ci})}{f_{mi} C_{wi}} \leq \frac{1}{f_{mi}} \tag{3.47}$$

Remark 3.3.2 *Implementation considerations dictate that if too many overruns occur (i.e., large m) for a job* $\tau_{i,j}$*, then one has to assume that the worst case overrun has taken place at a fixed value of m and the deadline must be extended to consider the worst case execution time.*

3.3.2 Example

Consider an example task set consisting of two periodic tasks, τ_1 and τ_2, with minimum frequencies $f_{m1} = 1/20$ and $f_{m2} = 1/16$, worst case execution times $C_{w1} = 5$ and $C_{w2} = 8$, best case execution times $C_{b1} = 1$ and $C_{b2} = 1$, respectively. To compare with the algorithm given in [23], let us suppose that the normal execution time (c_i^n) and the linear combination execution time ($C_{ci} = (1-\gamma_i)C_{wi} + \gamma_i C_{bi}$) are the same and are given by $C_{c1} = c_1^n = 3$ and $C_{c2} = c_2^n = 2$. Suppose that the optimal frequencies computed by Proposition 1 are $f_1^{opt} = 1/6$ and $f_2^{opt} = 1/4$. Therefore, each task is assigned a bandwidth $U_1 = U_2 = 0.5$. We consider the following cases to highlight the two local approaches discussed above, i.e., local approach which assumes every overrun as a worst case overrun and the local approach which uses a bisection method to compute the new deadlines.

- **No overruns:** Figure 3.3 shows a sketch of the two tasks running at or below their normal execution times, i.e., without any overruns.

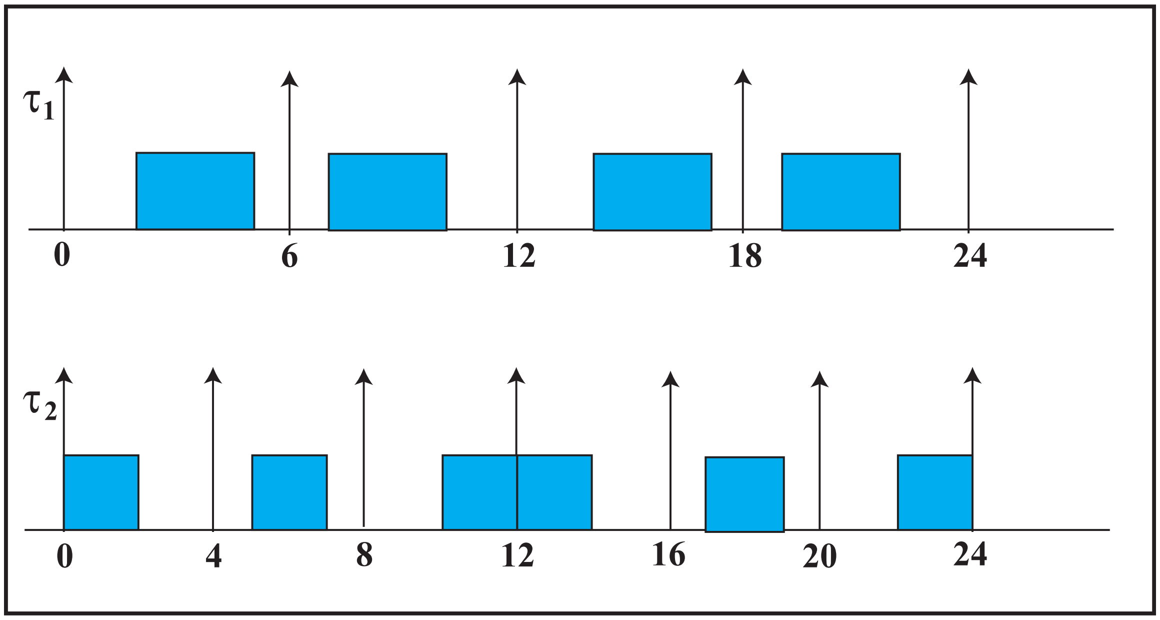

Figure 3.3: Normal execution times: No overruns

- **Deadlines and task execution using the worst-case overrun (Job $\tau_{2,2}$ has an overrun of 3 time units):** Figure 3.4 shows a sketch of this case. Now, suppose that the job $\tau_{2,2}$ has an overrun at time $t = 7$. Notice that the initial deadline is $t = 8$. Assuming the overrun is a worst-case overrun, using the local worst case algorithm described before, the job deadline is postponed to

$$d_{2,2}^{last} = r_{2,2} + \frac{C_{w2}}{U_2} = d_{2,2}^{0} + \frac{C_{w2} - C_{c2}}{U_2} = 20.$$

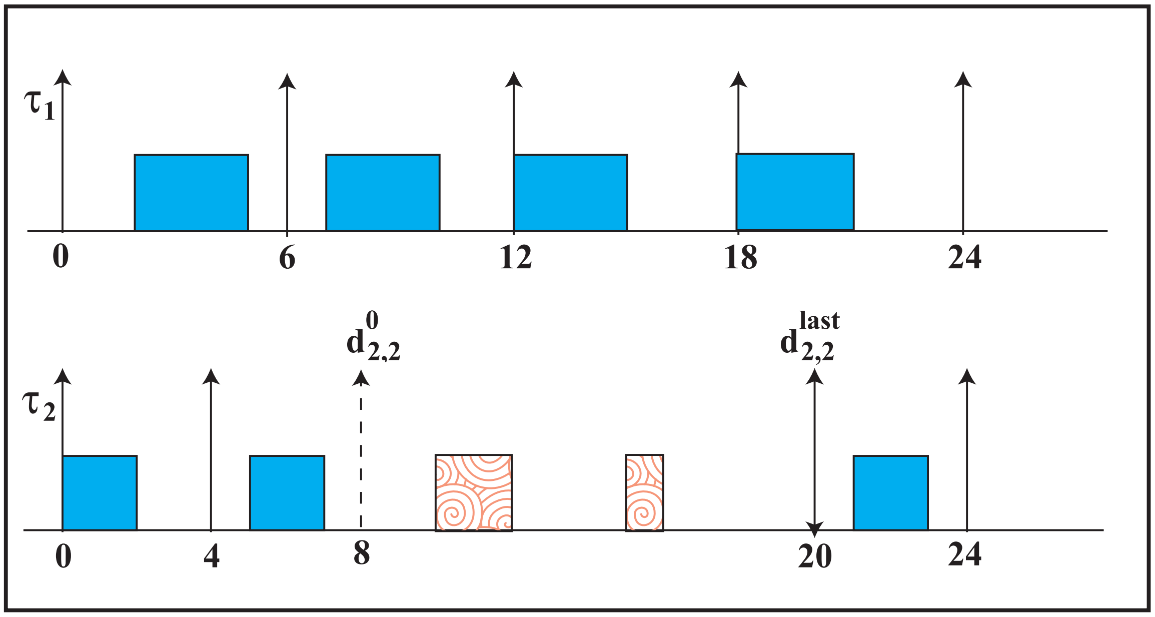

Figure 3.4: Worst case method: Job $\tau_{2,2}$ overrun = 3

- **Deadlines and task execution using bisection method (Job $\tau_{2,2}$ has an overrun of 3 time units):** Figure 3.5 shows a sketch of the execution of tasks as a result of overrun handling using the local bisection algorithm. Using the proposed local bisection approach, the job deadline is postponed to

$$d_{2,2}^{1} = d_{2,2}^{0} + \frac{1}{2}\frac{(C_{w2} - C_{c2})}{U_2} = 14.$$

Notice that for the same length of overrun, 3 time units, using the bisection approach, as shown in the sketch in Figure 3.5, the job $\tau_{2,2}$ is completed before the deadline, and the normal execution cycle continues after that. Notice that the time length for job $\tau_{2,2}$ including the overrun using the bisection approach is 10 units and for the worst case overrun scenario shown in Figure 3.4 is 16 units.

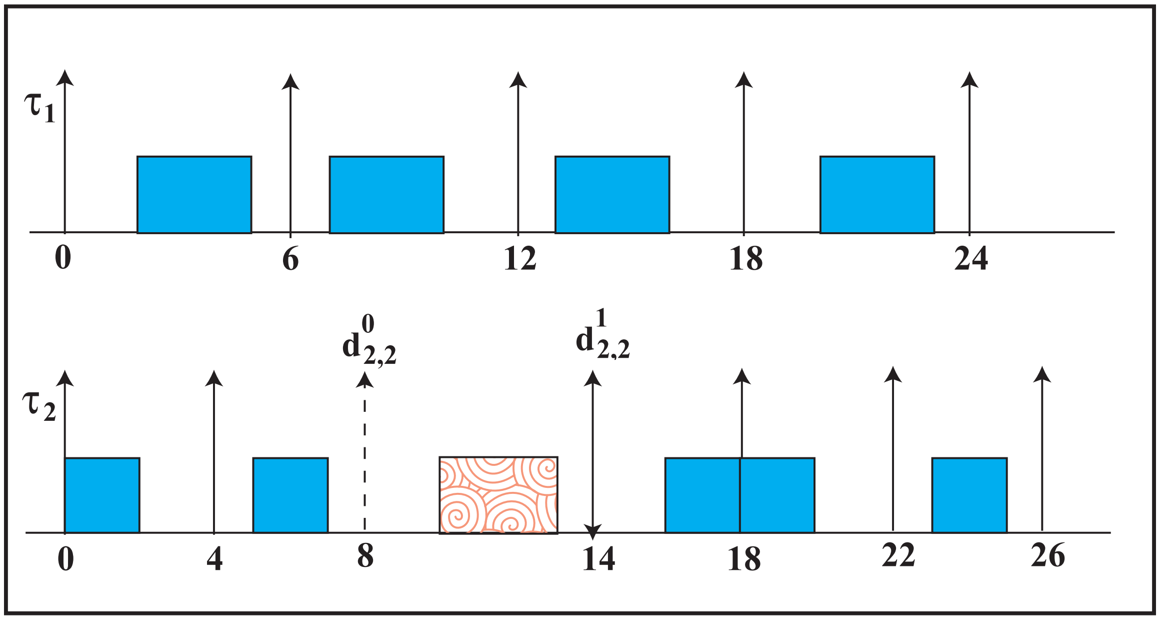

Figure 3.5: Bisection method: Job $\tau_{2,2}$ overrun = 3

- **Deadlines and task execution using the bisection method (Job $\tau_{2,2}$ has an overrun of 5 time units):** Figure 3.6 shows a sketch of the execution of tasks as a result of an overrun of 5 time units for job $\tau_{2,2}$. Notice that the bisection algorithm is invoked twice before the task execution is completed, i.e., $d^2_{2,2}$ is the last deadline of the job.

- **Deadlines and task execution using the bisection method (Job $\tau_{2,2}$ has an overrun of 2 time units and $\tau_{2,2}$ has an overrun of 3 time units):** Figure 3.7 shows a sketch of the task execution for this case. Notice from the figure that for job $\tau_{1,2}$ the last deadline is $d^2_{1,2} = 15$ and for job $\tau_{2,2}$ the last deadline is $d^1_{2,2} = 14$.

- **Deadlines and task execution using the bisection method (Job $\tau_{2,2}$ has an overrun of 2 time units and $\tau_{2,2}$ has an overrun of 5 time units):** A sketch of task exe-

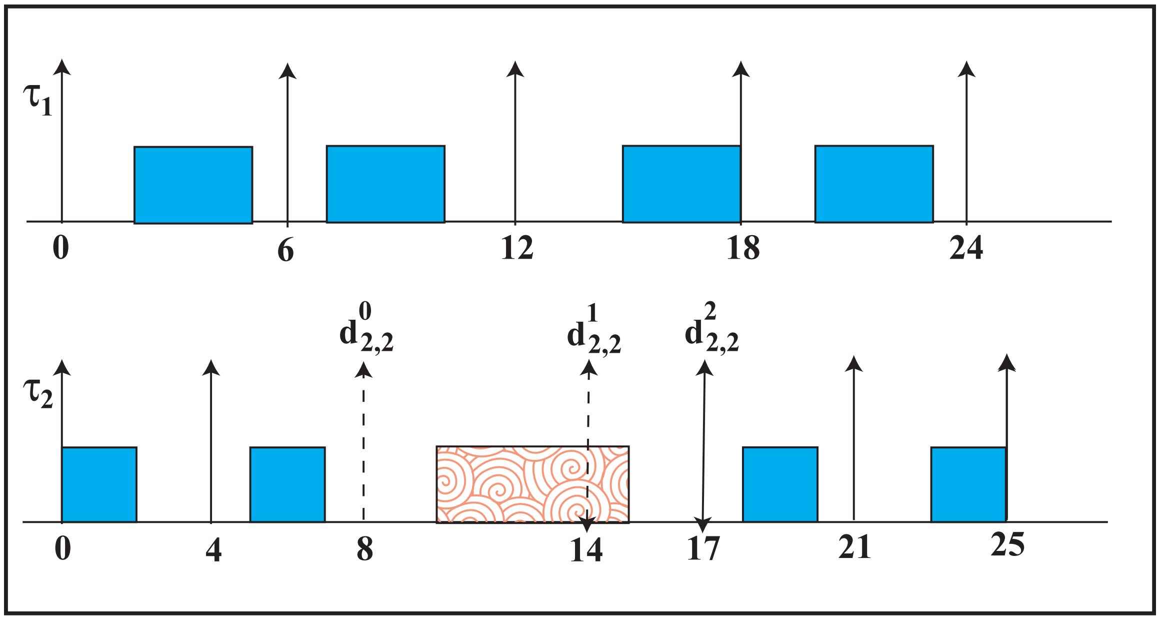

Figure 3.6: Bisection method: Job $\tau_{2,2}$ overrun = 5

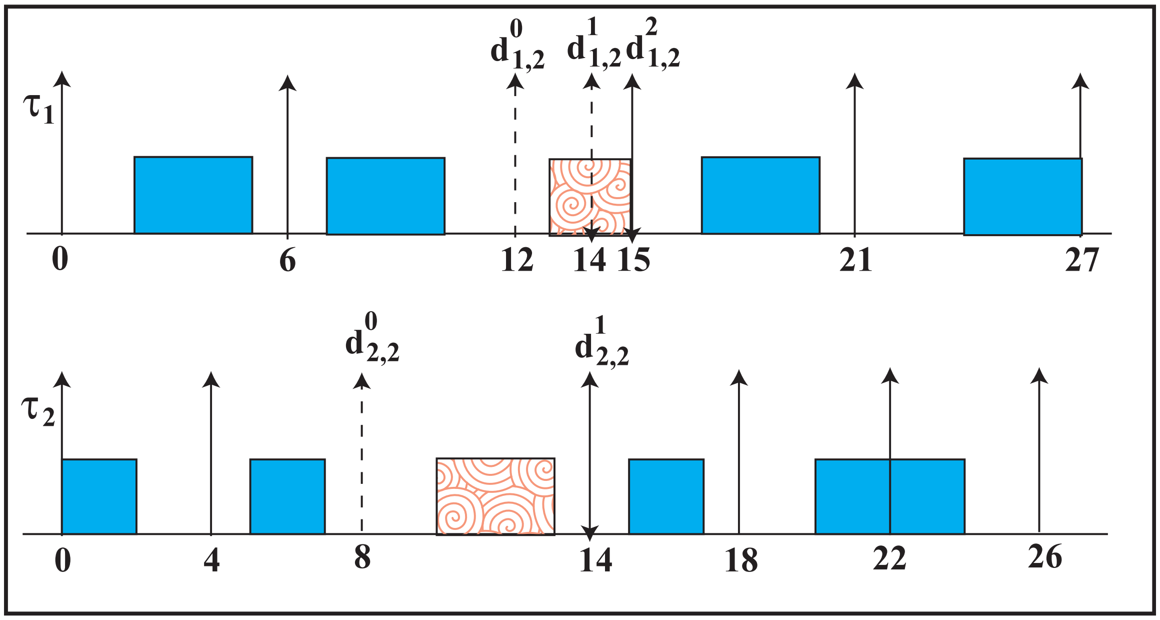

Figure 3.7: Bisection method: Job $\tau_{1,2}$ overrun = 2 and job $\tau_{2,2}$ overrun = 3

cution for this case is shown in Figure 3.7. The last deadline for job $\tau_{2,2}$ is $d^3_{1,2} = 16$ and for job $\tau_{2,2}$ the last deadline is $d^2_{2,2} = 17$.

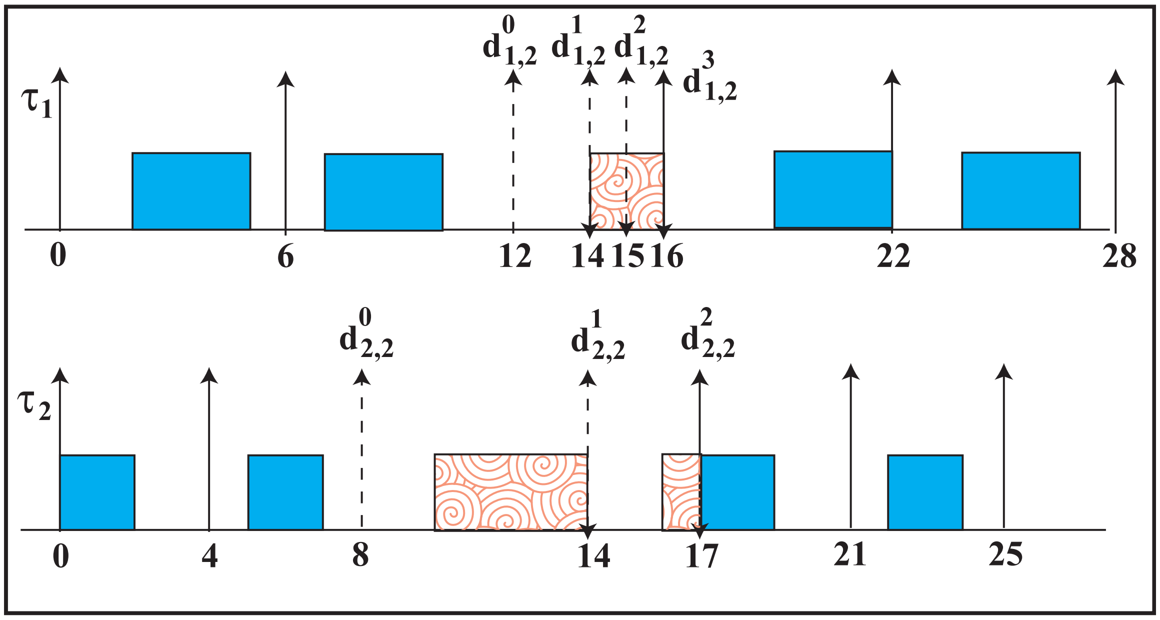

Figure 3.8: Bisection method: Job $\tau_{1,2}$ overrun = 2 and job $\tau_{2,2}$ overrun = 5

3.3.3 Global Approach

In the local approach, overruns and deadlines were restricted to individual tasks, that is, when a job $\tau_{i,j}$ has an overrun, the job deadline was extended so that the job could be executed within the bandwidth reserved for the task τ_i. In the global approach, when an overrun occurs, one can decrease the frequencies of all tasks to handle the overrun. One must decrease the frequencies so as to meet all the hard deadlines in the available total bandwidth. We consider two methods for handling overruns using the global approach. In the first method, whenever an overrun occurs, decrease all the frequencies to their minimal values for one period, i.e., until the overrun is handled. As evident, this method turns out to be conservative, and results in poor usage of resources. In the second method, extend the deadline of the job which overruns by a certain amount and extend the deadlines of all other jobs by the same amount; ensure that hard deadline is met by taking the worst case in the event that the amount of extension results in not meeting the minimum period. For example, when an overrun occurs for the job $\tau_{k,j}$, decrease the frequencies of the task by a small amount which is achieved by a smaller choice of ε_k, that is, extend the deadline by an amount equal to

$$d^1_{k,j} = r_{k,j} + \frac{C_{ck}}{U_k} + \varepsilon_k \frac{(C_{wk} - C_{ck})}{U_k} \tag{3.48}$$

where, for example, $\varepsilon_k \leq 1/4$. Further, decrease the frequencies of all other jobs by the same amount as for the task τ_k, that is,

$$d^1_{i,j} = r_{i,j} + \frac{C_{ci}}{U_i} + \varepsilon_k \frac{(C_{wk} - C_{ck})}{U_k} \tag{3.49}$$

for all $i \neq k$.

The same example as in the local approach is taken to evaluate the global approach. The following cases highlight the approach.

- Job $\tau_{2,2}$ has an overrun of 3 time units: Figure 3.9 shows a sketch of the task execution and deadlines in the event of an overrun of job $\tau_{2,2}$ of 3 units.

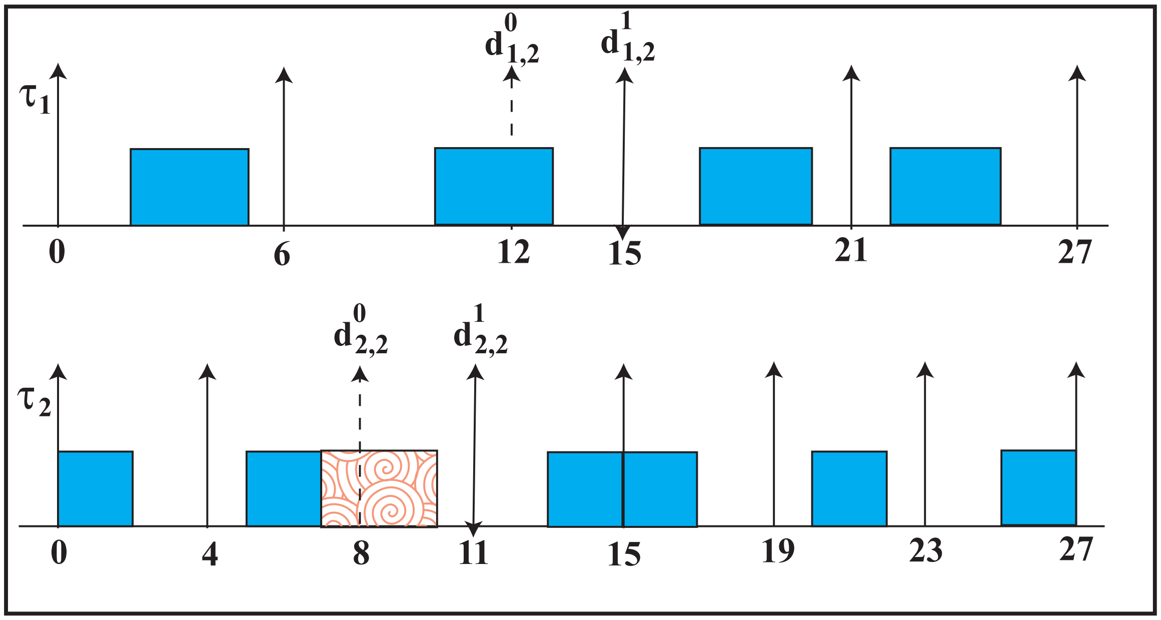

Figure 3.9: Bisection method: Job $\tau_{2,2}$ overrun = 3

- Job $\tau_{2,2}$ has an overrun of 5 time units: Figure 3.10 shows a sketch of the task execution and deadlines.

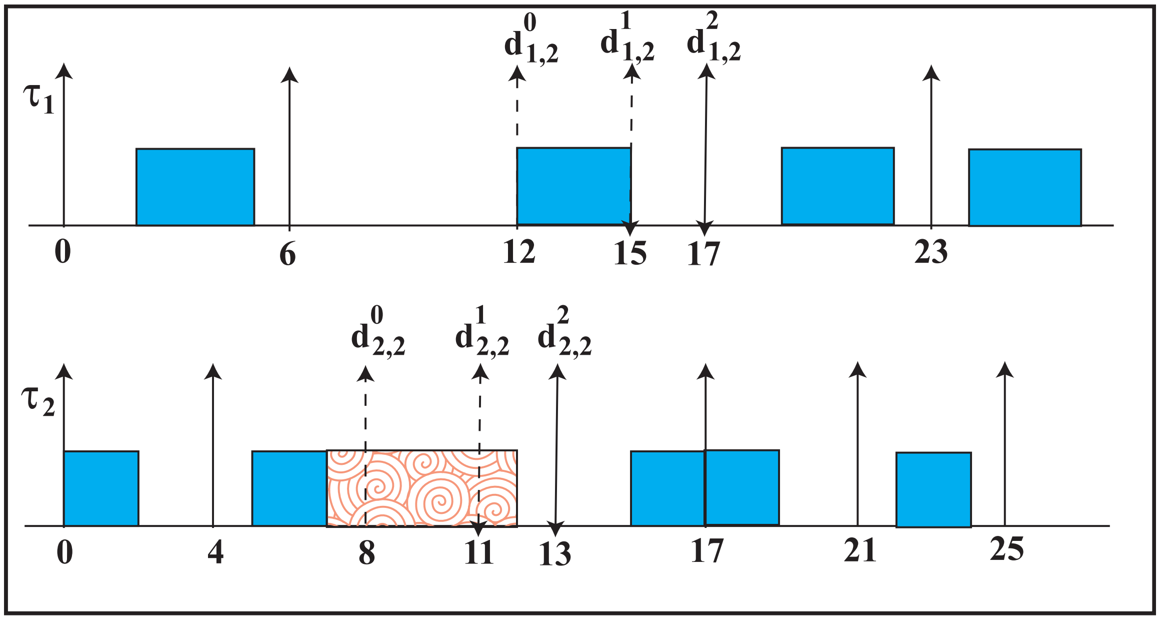

Figure 3.10: Bisection method: Job $\tau_{2,2}$ overrun = 5

- Job $\tau_{1,2}$ has an overrun of 2 time units and job $\tau_{2,2}$ has an overrun of 5 time units: Figure 3.11 shows a sketch of the task execution and deadlines.

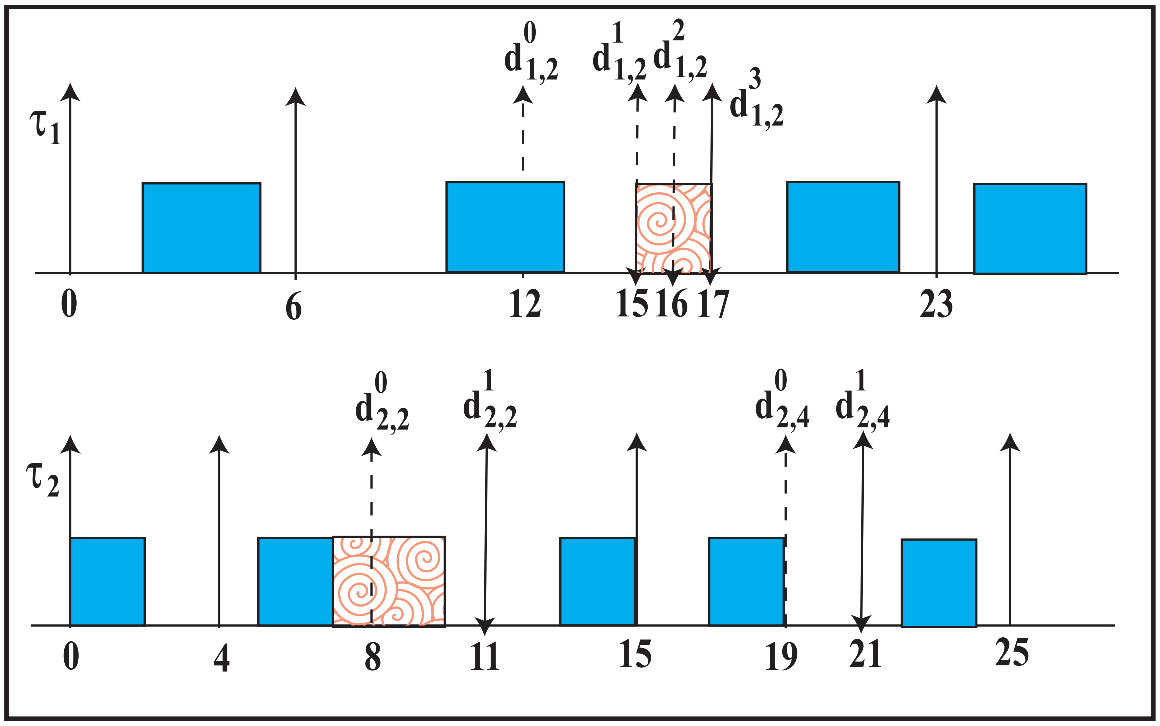

Figure 3.11: Bisection method: Job $\tau_{1,2}$ overrun = 2 and job $\tau_{2,2}$ overrun = 3

3.4 Simulations

We verify the results using simulations on a number of examples with variations in number of tasks, execution times, and minimum frequencies, etc.

The data for a real-time system with five tasks is given in Table 3.1. Figure 3.12 shows the minimum frequencies and the optimal frequencies obtained using BCETs, WCETs, and ACETs. Notice that the optimal frequencies are interlaced as discussed previously.

	β_i	C_{bi} (ms)	C_{wi} (ms)	f_{mi} (Hz)	w_i
Task 1	0.4	2	5	20	1
Task 2	0.5	5	8	12	2
Task 3	0.6	8	11	10	3
Task 4	0.7	11	14	6	4
Task 5	0.8	14	17	4	5

Table 3.1: Example 1: Data for a real-time system with five tasks

Figure 3.13 shows the optimal frequencies for two additional execution times obtained by using $\gamma = 0.25$ and $\gamma = 0.75$ in Equation (3.11). Notice that the optimal frequencies are interlaced as per the execution times. Figure 3.14 shows the performance loss indices corresponding to the frequencies shown in Figure 3.13. It can be observed that the performance loss index decreases as the optimal frequencies are increased (execution times are decreased) as expected. Further, one can observe that there is a considerable variation of the PLI for the WCET and the one with the execution time given by Equation (3.11) with $\gamma = 0.25$. This reinforces our approach to handle overruns using the bisection method as opposed to the method that characterizes any task overrun as a worst case overrun.

We have tested real-time examples with different number of tasks. Table 3.2 shows a seven-task example. The optimal frequencies are shown in Figure 3.15; notice that the first four optimal frequencies computed for the worst case take the minimum frequency bounds. For the example considered in Table 3.3, the optimal frequencies are shown in Figure 3.16; it is observed that the first three optimal frequencies obtained for the average case take the values of the minimum frequency bounds. A nine task example is given in Table 3.4; the optimal frequencies for this example are shown in Figure 3.17; notice that the first seven worst case frequencies take the values of the minimum frequency bounds. In all the cases tested the optimal frequencies are interlaced as per the execution times as given in Proposition 2.

	β_i	C_{bi} (ms)	C_{wi} (ms)	f_{mi} (Hz)	w_i
Task 1	0.3	4	5	20	1
Task 2	0.4	6.5	7.5	17	2
Task 3	0.5	9	10	14	3
Task 4	0.6	11.5	12.5	12	4
Task 5	0.7	14	15	9	5
Task 6	0.8	16.5	17.5	6	6
Task 7	0.9	19	20	4	7

Table 3.2: Example 2: Real-time system with seven tasks

	β_i	C_{bi} (ms)	C_{wi} (ms)	f_{mi} (Hz)	w_i
Task 1	0.1	6	9	15	1
Task 2	0.2	8	11	13	2
Task 3	0.3	10	13	10	3
Task 4	0.4	12	15	9	4
Task 5	0.5	14	17	6	5
Task 6	0.6	16	19	4	6
Task 7	0.7	18	21	2	7

Table 3.3: Example 3: Real-time system with seven tasks

	β_i	C_{bi} (ms)	C_{wi} (ms)	f_{mi} (Hz)	w_i
Task 1	0.1	6	9	15	1
Task 2	0.2	8	11	12	2
Task 3	0.3	10	13	9	3
Task 4	0.4	12	15	8	4
Task 5	0.5	14	17	7	5
Task 6	0.6	16	19	6	6
Task 7	0.7	18	21	5	7
Task 8	0.8	20	23	3	8
Task 9	0.9	22	25	2	9

Table 3.4: Example 4: Real-time system with nine tasks

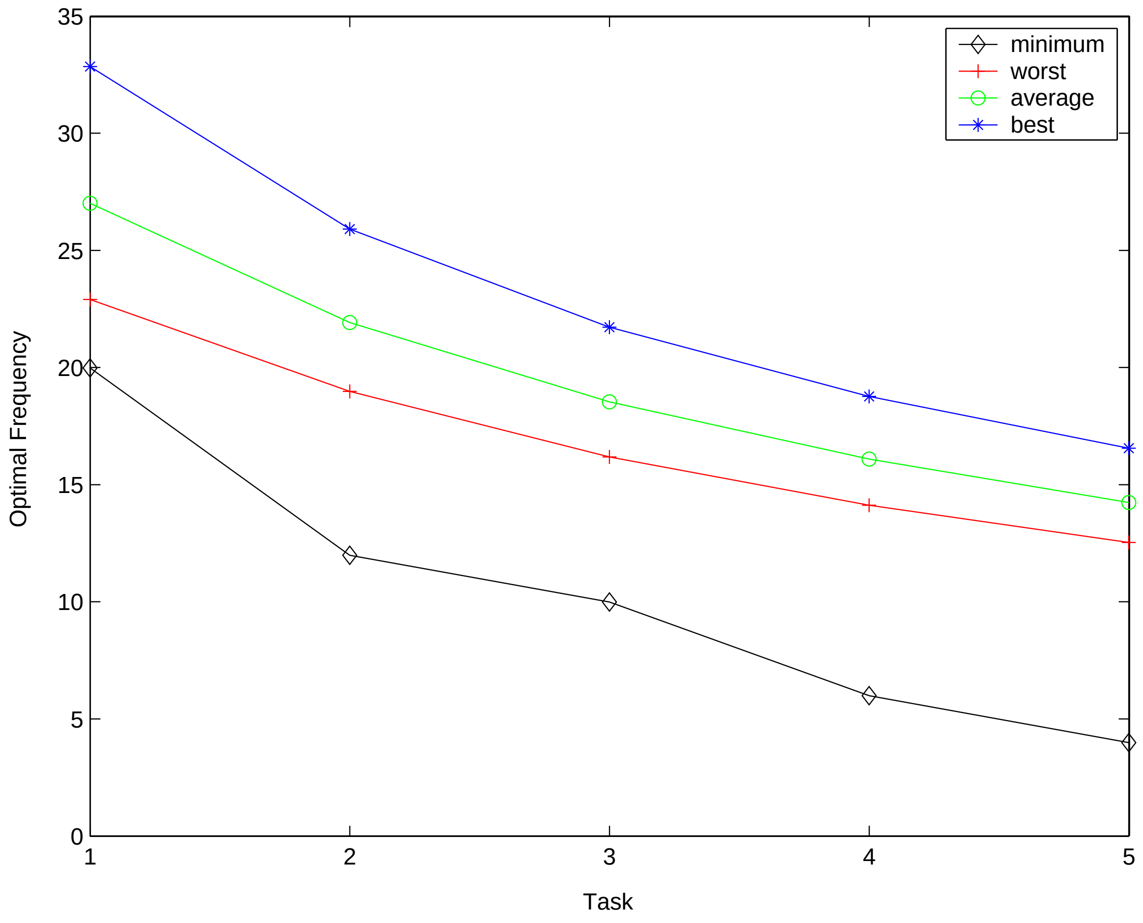

Figure 3.12: Frequencies for example 1

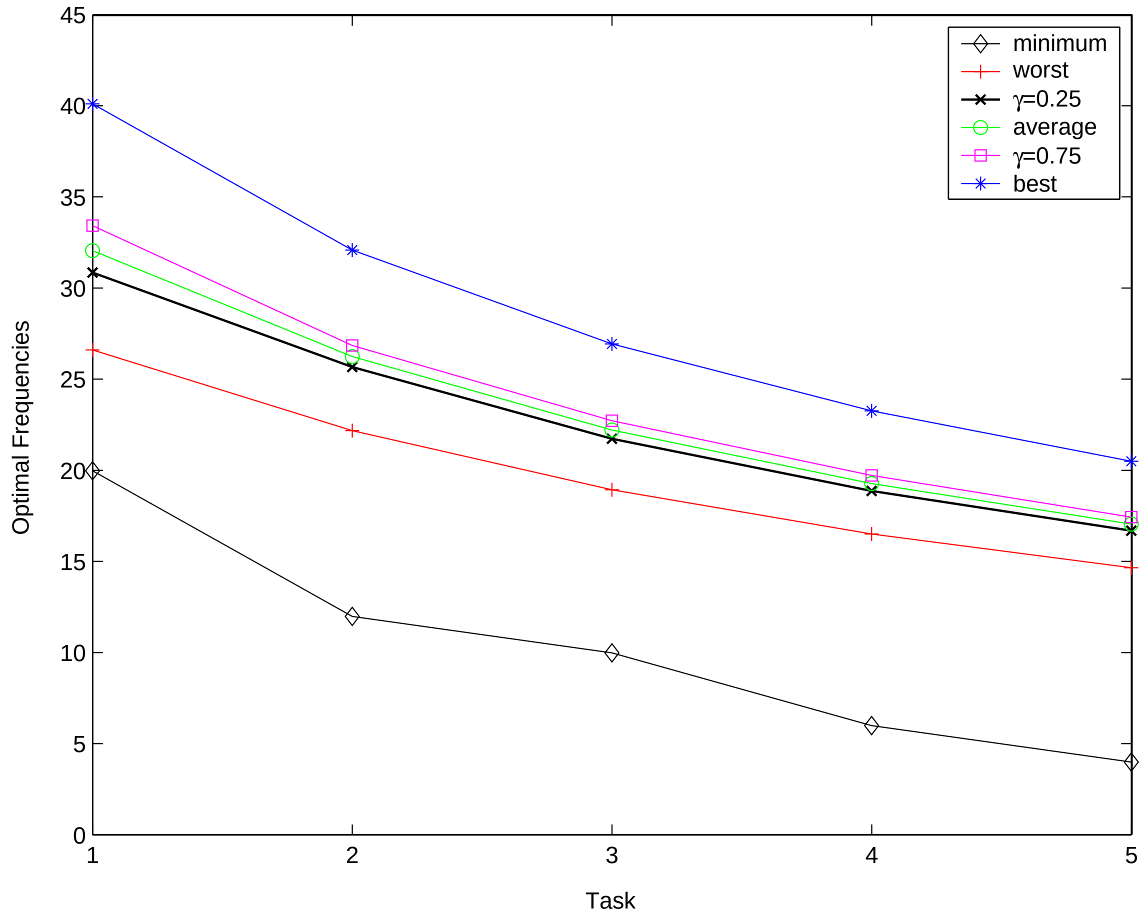

Figure 3.13: Frequencies for example 1 with two more execution times ($\gamma = 0.25$ and $\gamma = 0.75$)

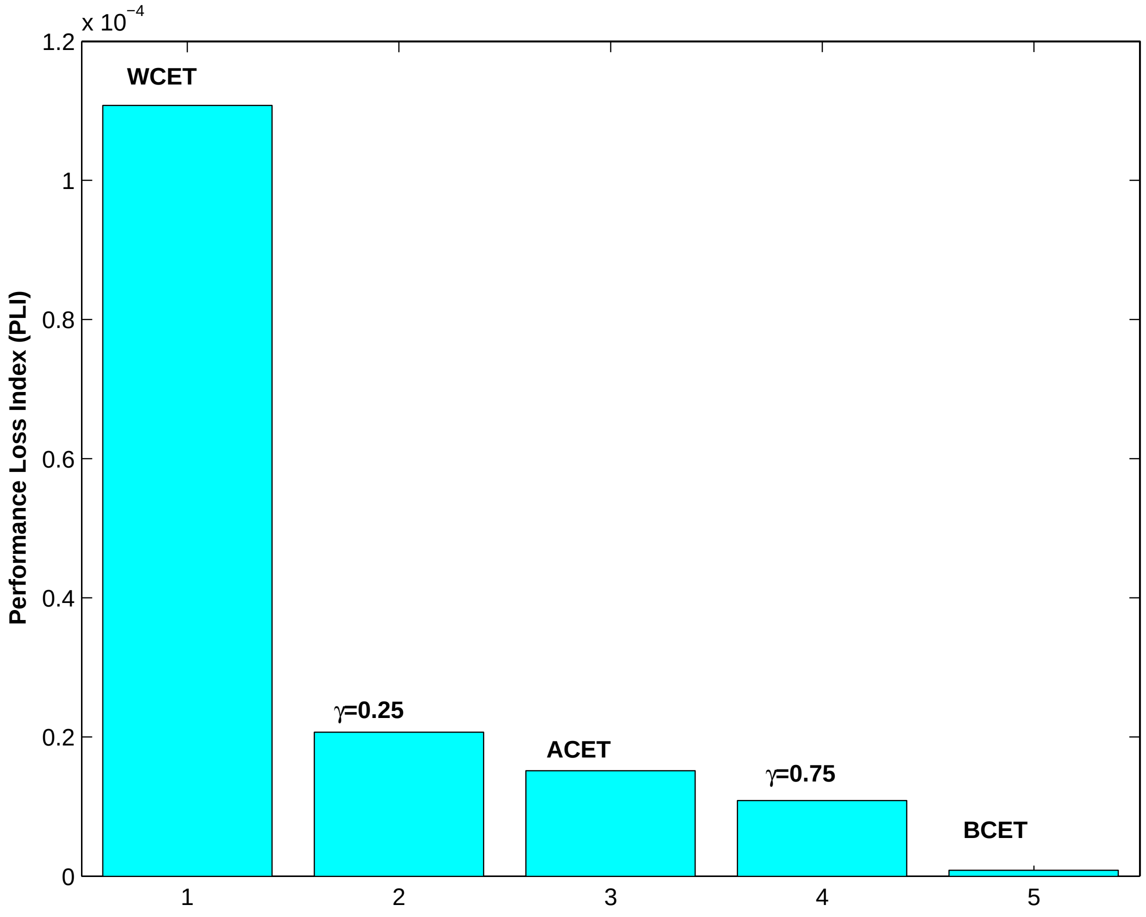

Figure 3.14: Performance loss indices for example 1

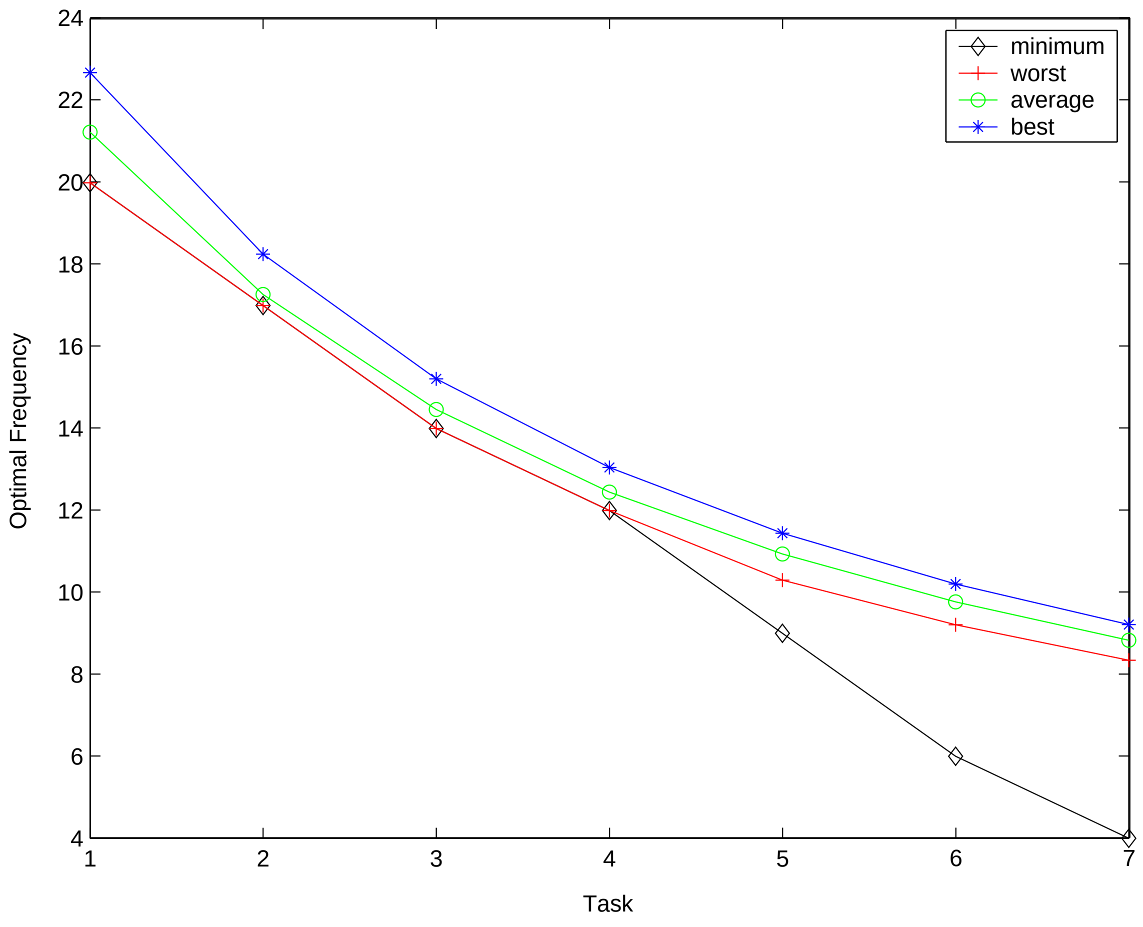

Figure 3.15: Frequencies for example 2

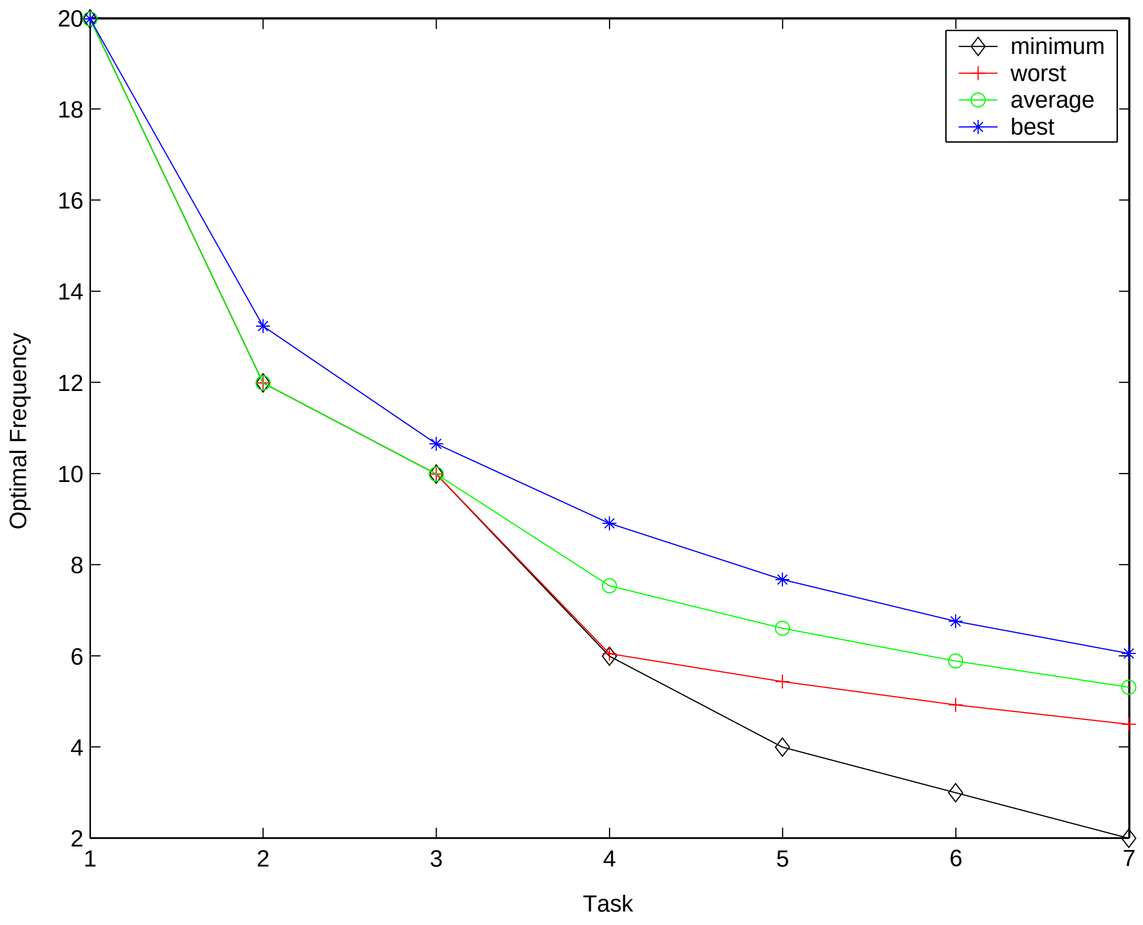

Figure 3.16: Frequencies for example 3

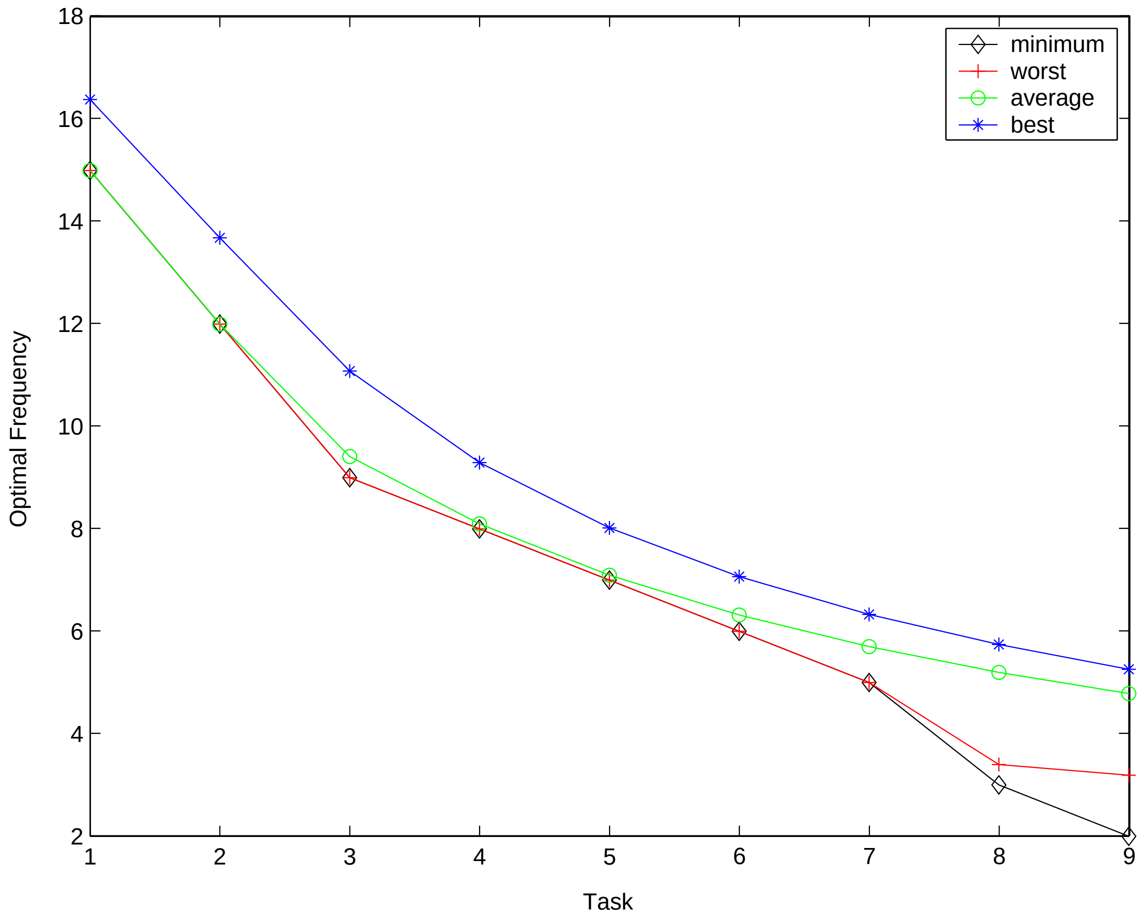

Figure 3.17: Frequencies for example 4

CHAPTER 4

Summary and Future Work

In this work, we have investigated scheduling of real-time sensor and actuator networks. Efficient scheduling of real-time sensor and actuator networks can enable their use in a large number of applications. The goal is to achieve higher task execution rates and maximization of the use of resources while maintaining hard deadlines of all tasks, which was the focus of this study. A summary of the work accomplished in this thesis is given in the following:

- An extensive background review was conducted to bring the key areas that influence real-time scheduling of sensor and actuator networks. The key areas are real-time systems and their applicability to sensor and actuator networks, feedback control systems and their components, and real-time scheduling algorithms.

- Performance of real-time control systems in terms of minimizing a performance index was studied.

- Real-time task scheduling while minimizing a performance loss index function subject to resource and task rate constraints was investigated. The effect of the task execution times on the optimal task frequencies and performance loss index was investigated. It is shown that higher task frequencies and lower performance loss index can be achieved if task execution times lower than worst case execution times are used in the computation of optimal frequencies. A discussion of key factors that determine the best case execution times and worst case execution times for tasks was given.

- Since execution times lower than worst case are used in the computation of optimal frequencies, methods that allow for overruns while maintaining hard deadlines were investigated. Two approaches were investigated, namely, local and global. In the local approach, a task overrun is restricted to that particular task, i.e., its frequency is adaptively changed to execute the task within the given bandwidth while meeting the hard deadline, and other task frequencies are unaffected. Two methods using the local approach were investigated. One is an existing approach that treats every overrun as a worst case overrun, where as the second is the proposed bisection method. The bisection method handles overruns by extending the deadlines by an amount equal to the bisection of the normal execution time and the worst case execution time. It was shown that all deadlines will be met by the bisection method. In the global approach, when an overrun occurs for a particular task, then the frequencies of all the tasks are reduced to release bandwidth for the overrun task.

- Examples are given throughout to illustrate the proposed strategies.

In this work, it was assumed that the available resources do not change over time. It is conceivable that in wireless sensor and actuator networks, the amount of available resources on all the devices may be varying. Re-configuration and re-allocation of dynamic resources is necessary in a changing network environment where devices are added and/or removed. Resource allocation and task scheduling in this environment will be a good and challenging future topic in this area. In particular, applicability of the bisection algorithm in a dynamically changing resource environment may form a good topic of research.

BIBLIOGRAPHY

[1] C.S. Raghavendra, K.M. Sivalingam, and T. Znati, Editors, Wireless Sensor Networks. *Kluwer Academic Publishers*, Boston, MA, 2005.

[2] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S.S. Sastry, “Distributed Control Applications Within Sensor Networks,” *Proc. of the IEEE*, Vol. 91, No. 8, August 2003.

[3] M. Haenggi, “Mobile Sensor Actuator Networks: Opportunities and Challenges,” *Proc. of the 7th IEEE Intl. Workshop on Cellular Neural Networks and Their Applications*, 2002.

[4] J.A. Stankovic. “A Serious Problem for Next-Generation Systems.” *IEEE Computer*, Vol. 21, No. 10, October 1988.

[5] J.A. Stankovic, “Real-Time and Embedded Systems,” *ACM Computing Surveys*, Vol. 28, No. 1, pp. 205–208, March 1996.

[6] D.D. Gajski and F. Vahid, “Specification and Design of Embedded Hardware-Software Systems,” *IEEE Design & Test of Computers*, Vol. 12, No. 1, pp. 53–67, Spring 1995.

[7] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications. *Kluwer Academic Publishers*, Boston, 1997.

[8] W. Liu and V. K. Prasanna, “Utilizing the Power of High-Performance Computing,” *IEEE Signal Processing Magazine*, Vol 15, No 5, pp. 85–100, September 1998.

[9] T. Wills, “Real-Time Computing Demands - New Tools, Methods, and Design,” *Computer Design*, pp. 58–59, August 15, 1988.

[10] J.A. Stankovic, T.F. Abdelzaher, C. Lu, L. Sha, and J.C. Hou, “Real-Time Communication and Coordination in Embedded Sensor Networks,” *Proc. of the IEEE*, Vol. 91, No. 7, July 2003.

[11] K. Ogata, Discrete-Time Control Systems. Prentice-Hall, Upper Saddle River, New Jersey, 1995.

[12] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” *J. Assoc. Comput. Mach*, Vol. 20, No. 1, pp. 46–61, January 1973.

[13] K.G. Shin and P. Ramanathan, “Real-Time Computing: A New Discipline of Computer Science and Engineering,” *Proceeding of the IEEE*, Vol. 82, No. 1, pp. 6–24, January 1994.

[14] L. Sha, R. Rajkumar, and S.S. Sathaye, “Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems,” *Proceedings of the IEEE*, Vol 82, No. 1, pp. 68–82, January 1994.

[15] 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*, Vol. 28, pp. 101–155, 2004.

[16] S.C. Cheng, “Scheduling Algorithms for Hard Real-Time Systems – A Brief survey,” *IEEE Press*, Los Angeles, 1987.

[17] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin, “On Task Schedulability in Real-Time Control Systems,” *Proc. of the 17th IEEE Real-Time Systems Symposium*, pp. 13–21, 1996.

[18] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," *Proc. of the 22nd IEEE Real-Time Systems Symposium*, pp. 193–202, 2001.

[19] L. Sha and J.B. Goodenough. "Real-Time Scheduling Theory and Ada." *IEEE Computer*, Vol. 23, No. 4, pp. 53–62, April 1990.

[20] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic Feedback Control," *Proc. of Euromicro 2000, 12th Intl. Conf. on Realtime Systems*, June 2000.

[21] L. Sha, X. Liu, M. Caccamo, and G. Buttazzo, "Online Control Optimization Using Load Driven Scheduling," *Proc. of the 39th IEEE Conf. on Decision and Control*, December 2000.

[22] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm," *Proc. of the IEEE Real-Time Systems Symposium*, Pheonix, AZ, December 1999.

[23] M. Caccamo, G. Buttazzo, and L. Sha, "Handling Execution Overruns in Hard Real-Time Control Systems," *IEEE Trans. on Computers*, Vol. 51, No. 7, pp. 835–849, July 2002.

[24] G. Lipari and E. Bini, "Resource Partitioning among Real-Time Applications," *Proc. of the 15th Euromicro Conference on Real-Time Systems*, June 2003.

[25] G. Lipari, G. Lamastra, and L. Abeni, "Task Synchronization in Reservation-Based Real-Time Systems," *IEEE Trans. on Computers*, Vol. 53, No. 12, pp. 1591–1601, December 2003.

[26] M.K. Gardner and J.W.S. Liu, "Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overload," *Proc. of the 11th Euromicro Conference on Real-Time Systems*, June 1999.

[27] S. Giannecchini, M. Caccamo, and C. Shih, “Collaborative Resource Allocation in Wireless Sensor Networks,” *Proc. of the IEEE Real-Time Systems Symposium*, 2004.

[28] M. Caccamo, G. Buttazzo, and D.C. Thomas, “Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times,” *IEEE Trans. on Computers,* Vol. 54, No. 2, pp. 198–213, February 2005.

APPENDIX A

MATLAB Program

```
%
N=5; % number of tasks
alpha=2/3;
b(1)=0.3; cw(1)=10*0.001; cb(1)=8*0.001;; w(1)=1;
ca(1)=(cw(1)+cb(1))/2;
fm(1)=20; fm(2)=12.5; fm(3)=10; fm(4)=6; fm(5)=4;
sum_cwfm=fm(1)*cw(1);
sum_cbfm=fm(1)*cb(1);
sum_cafm=fm(1)*ca(1);
Gw(1)=w(1)*alpha*b(1)/cw(1);
Gb(1)=w(1)*alpha*b(1)/cb(1);
Ga(1)=w(1)*alpha*b(1)/ca(1);
%
for i=2:N,
    b(i)=b(i-1)+0.1;
    cw(i)=cw(i-1)+5*0.001;
    cb(i)=cw(i)-2*0.001;
    ca(i)=(cw(i)+cb(i))/2;
    w(i)=w(i-1)+1;
    Gw(i)=w(i)*alpha*b(i)/cw(i);
    Gb(i)=w(i)*alpha*b(i)/cb(i);
    Ga(i)=w(i)*alpha*b(i)/ca(i);
    sum_cwfm=sum_cwfm+fm(i)*cw(i);
    sum_cbfm=sum_cbfm+fm(i)*cb(i);
    sum_cafm=sum_cafm+fm(i)*ca(i);
end
%
% computing p in the SLSS proposition 3.1
U=1; % utilization factor
% Computing pw such that (11) is satisfied
fpw=sum_cwfm;
pw=N;i=0;
while fpw<U,
    fpw1=0; fpw2=0;
    pw=pw-1;
    for j=1:pw,
        fpw1=fpw1+cw(j)*fm(j);
    end
    for j=(pw+1):N,
        fpw2=fpw2+(cw(j)/b(j))*(b(pw)*fm(pw)+log(Gw(j)/Gw(pw)));
    end
    fpw=fpw1+fpw2;
end
%
%Computing frequencies using worst case
Qw_num1=0; Qw_num2=0; Qw_den=0;
```

```
for j=1:pw,
    Qw_num1=Qw_num1+cw(j)*fm(j);
end
for j=(pw+1):N,
    Qw_num2=Qw_num2+cw(j)*log(Gw(j))/b(j);
    Qw_den=Qw_den+cw(j)/b(j);
end
Qw=(Qw_num1+Qw_num2-U)/Qw_den;

for j=1:pw,
    fw(j)=fm(j);
end
for j=(pw+1):N,
    fw(j)=(log(Gw(j))-Qw)/b(j);
end

% performance loss index (worst case)
pliw=0;
for i=1:N,
    pliw=pliw+w(i)*alpha*exp(-b(i)*fw(i));
end

% Computing pb such that (11) is satisfied
fpb=sum_cbfm;
pb=N;i=0;
while fpb<U,
    fpb1=0; fpb2=0;

    pb=pb-1;

    for j=1:pb,

        fpb1=fpb1+cb(j)*fm(j);

    end

    for j=(pb+1):N,

        fpb2=fpb2+(cb(j)/b(j))*(b(pb)*fm(pb)+log(Gb(j)/Gb(pb)));

    end

    fpb=fpb1+fpb2;

end

%

%Computing frequencies using best case

Qb_num1=0; Qb_num2=0; Qb_den=0;

for j=1:pb,

    Qb_num1=Qb_num1+cb(j)*fm(j);

end

for j=(pb+1):N,

    Qb_num2=Qb_num2+cb(j)*log(Gb(j))/b(j);

    Qb_den=Qb_den+cb(j)/b(j);

end

Qb=(Qb_num1+Qb_num2-U)/Qb_den;


for j=1:pb,

    fb(j)=fm(j);

end

for j=(pb+1):N,
    fb(j)=(log(Gb(j))-Qb)/b(j);
end

% performance loss index (best case)
plib=0;
for i=1:N,
    plib=plib+w(i)*alpha*exp(-b(i)*fb(i));
end

% Computing pa (average case) such that (11) is satisfied
fpa=sum_cafm;
pa=N;i=0;
while fpa<U,
    fpa1=0; fpa2=0;
    pa=pa-1;
    for j=1:pa,
        fpa1=fpa1+ca(j)*fm(j);
    end
    for j=(pa+1):N,
        fpa2=fpa2+(ca(j)/b(j))*(b(pa)*fm(pa)+log(Ga(j)/Ga(pa)));
    end
    fpa=fpa1+fpa2;
end
%
%Computing frequencies using best case
Qa_num1=0; Qa_num2=0; Qa_den=0;
```

```
for j=1:pa,
    Qa_num1=Qa_num1+ca(j)*fm(j);
end
for j=(pa+1):N,
    Qa_num2=Qa_num2+ca(j)*log(Ga(j))/b(j);
    Qa_den=Qa_den+ca(j)/b(j);
end
Qa=(Qa_num1+Qa_num2-U)/Qa_den;

for j=1:pa,
    fa(j)=fm(j);
end
for j=(pa+1):N,
    fa(j)=(log(Ga(j))-Qa)/b(j);
end

% performance loss index (best case)
plia=0;
for i=1:N,
    plia=plia+w(i)*alpha*exp(-b(i)*fa(i));
end

i=1:N;
%plot(i,fw,'o',i,fa,'+',i,fb,'*')
plot(i,fw,i,fa,i,fb)
legend('worst','average','best')
```

VITA

Padmavathy Pagilla

Candidate for the Degree of

Master of Science

Thesis: REAL-TIME SCHEDULING OF SENSOR AND ACTUATOR NETWORKS

Major Field: Computer Science

Biographical:

Education: Received the B.Engg. degree from Osmania University, Hyderabad, India, in 1999, in Civil Engineering. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July, 2005.