Schedulability Tests for Real-Time Uni- and Multiprocessor Systems

Focusing on Partitioned Approaches

HABILITATION

Der Fakultät für Informatik der
TU Chemnitz
zur Erlangung der Lehrbefähigung im Fach Informatik und
des akademischen Grades eines
Dr.-Ing. habil.

eingereichte Habilitation

von
Herrn Dr.-Ing. Dirk Müller
aus
Reichenbach/V.

Datum der Einreichung: 2013/01/18

Gutachter 1: Prof. Dr.-Ing. habil. Matthias Werner
Gutachter 2: Prof. Dr. sc. (ETH) Samarjit Chakraborty
Gutachter 3: Prof. Dr.-Ing. Robert Baumgartl

Tag der mündlichen Prüfung: 2014/02/19

Für Beata und Stefan.

Danksagung

Im April 2008 wechselte ich an die TU Chemnitz und wurde sofort in die Lehre eingebunden. Unter der Anleitung von Prof. Robert Baumgartl und meinem Betreuer Prof. Matthias Werner, denen ich in besonderem Maße zu Dank verpflichtet bin, arbeitete ich mich in das faszinierende Gebiet der Echtzeitsysteme ein. Nach zwei Jahren, von 2010 an, wurde meine Forschung auf diesem Gebiet durch mehrere Veröffentlichungen belohnt. Im Literaturverzeichnis sind meine Beiträge durch Fettdruck hervorgehoben. Als ein umfassendes Werk, das diese Publikationen in einen großen Zusammenhang stellt, ist die vorliegende Habilitationsschrift zu verstehen.

Während meiner Forschungstätigkeit lernte ich auf der ECRTS 2010 in Brüssel Prof. Samarjit Chakraborty kennen und schätzen. Auch ihm danke ich für viele anregende Diskussionen und Kommentare.

Für das Korrekturlesen danke ich meiner Schwester Katja. Durch ihre Hilfe konnten einige sprachliche und typographische Fehler und Ungereimtheiten behoben werden.

Schließlich danke ich außerordentlich meiner Frau Beata und meinem Sohn Stefan sowie meinen Eltern Erika und Bernd für die Unterstützung und Aufmunterung, die ich durch sie während der manchmal nicht einfachen Zeit des Schreibens dieser Arbeit erhielt.

Abstract

Real-time systems are computer systems for which the benefit of a result in addition to its correctness and accuracy crucially depends on the time when it is available. Frequently, they occur in embedded systems, i.e., as computers which are not identified as such prima facie. Real-time systems are of outstanding relevance in our globalized and mobile world. Important sectors are transportation with road, railway and aerospace traffic as well as communication and consumer electronics. While the first group typically is characterized by *hard* deadlines, *soft* deadlines are dominating in the latter one.

Schedules are known from everyday life, e.g., in the form of time-tables for trains. Clearly, the correct creation of such schedules is a poorly scaling problem. Similarly difficult is the scheduling of real-time tasks on processors. In the simplest case, only one processor is available. Even here, there is a multitude of algorithms and concepts. Essential ones are prioritization according to *absolute* deadlines (Earliest Deadline First, EDF) and prioritization according to *relative* deadlines (Rate-Monotonic Scheduling, RMS).

Since about 2005 multicore and multiprocessor systems have been gaining more and more importance. New PCs and mobile computers are almost exclusively sold based on these architectures. Also, the above mentioned embedded systems are undergoing such a technology change or are at least intended to be changed. While real-time scheduling on uniprocessors is solved to a large extent, there are a lot of open problems in real-time scheduling on multiprocessors. Easily implemented greedy algorithms are no more optimal there which was already published by Dhall and Liu in 1978. These considerations illustrate the necessity of developing specific real-time multiprocessor scheduling algorithms.

There is a difference between global and partitioned scheduling. Global scheduling uses a common queue. The partitioned approach mainly considered in this thesis, in contrast, reduces the problem to that of static distribution of tasks to processors or cores and several uniprocessor scheduling problems which are already well solved. Thus, the NP-Hard problem of task-to-processor assignment remains. For larger task sets, the combinatorial explosion is only manageable by a fallback to approximative approaches using heuristics like *First Fit* or *Next Fit*. Note the relation of the problem to *bin packing*. However,

partitioned RMS is more complicated since periods as additional parameters have an influence on the maximum possible utilization.

Accelerated Simply Periodic Tasks Sets. Simply periodic (harmonic) task sets are characterized by pairwise integer ratios of periods. This situation is especially beneficial for schedulability according to RMS since then no waste occurs and a utilization of 100% can be achieved. This is an enormous advantage compared to the general Liu/Layland utilization bound of ca. 69.3%. Since a reduction of periods corresponds to a tightening of requirements (sustainability), the detection of Accelerated Simply Periodic Tasks Sets, ASPTSs) is well-suited for the proof of RMS schedulability of general task sets. Such sufficient schedulability criteria are appropriate since exact analysis using Time-Demand Analysis (TDA) has pseudo-polynomial run-time which is not manageable for larger task sets. When using the partitioned approach, ASPTSs are also suited for multiprocessor scheduling. The challenge in this context is the search for good combinations of uniprocessor schedulability tests and allocation heuristics.

Circular Period Similarity. An established algorithm taking tasks' periods into account for a powerful partitioned RMS is Rate-Monotonic Small Tasks (RMST) by Burchard *et al.* from 1995. It will be shown how this algorithm can be improved by, e.g., integrating the concept of simply periodic task sets. The more general view on presorting as changeover from online to offline approaches is interesting in this context. Here, the impact of the sorting criterion (or criteria for two-stage approaches) on the minimization of idle times and, thus, on the performance of the algorithm is of major concern.

It will be shown how the performance of RMST can be increased by substituting linear by circular range. As a side effect, the Burchard schedulability test is simplified. An up to now necessary case-by-case analysis turns out redundant. To the author's best knowledge, this is the first application of circular statistics in real-time scheduling.

Sharp Thresholds. Partitioned EDF is characterized by sharp phase transitions at a threshold of total utilization of tasks. A steady increase of total utilization leads to an abrupt change from the state "almost guaranteed schedulable" to the state "almost guaranteed not schedulable". Often, these thresholds can only be obtained approximately in the form of bounds. For some special cases, determination of exact values succeeds. The knowledge of such thresholds or of the probability-utilization function enables then to decide in constant time whether a task can be accepted or not. This approach is suited for soft real-time systems with an online admission control. The risk of a false positive or a false negative decision (e.g. 5%) can be preset. Knowing the probability-utilization function enables the derivation of appropriate bounds.

Goal. The goal of this habilitation is thus the presentation of new, advanced methods for the realization of real-time scheduling on multiprocessor systems. It focuses on partitioned approaches and concentrates on the method of sufficient schedulability tests. Target is the maximization of possible utilization in worst or average case under a given number of processors. This scenario is more realistic than the dual case of minimizing the number of necessary processors for a given task set since the hardware is normally fixed. Sufficient schedulability tests may be useful for quick estimates of task set schedulability in automatic system-synthesis tools.

scheduling, multiprocessor, real-time, partitioned, rate-monotonic, circular similarity measure, phase transition, threshold

Zusammenfassung

Echtzeitsysteme sind Computersysteme, bei denen der Wert eines Ergebnisses neben seiner Korrektheit und Genauigkeit in entscheidendem Maße vom Zeitpunkt seines Vorliegens abhängt. Häufig treten sie in eingebetteten Systemen auf, also als Computer, die nicht auf den ersten Blick als solche erkennbar sind. Echtzeitsysteme haben in unserer globalisierten und mobilen Welt eine starke Bedeutung. Wichtige Bereiche hierbei sind das gesamte Verkehrswesen mit Straßen- und Schienenverkehr sowie Luft- und Raumfahrt und die Kommunikations- und Unterhaltungselektronik. Während die erste Gruppe häufig durch *harte* Deadlines gekennzeichnet ist, dominieren in der zweiten Gruppe *weiche* Deadlines.

Pläne oder Schedules sind uns aus dem Alltag z.B. in Form von Zugfahrplänen bekannt. Klar dürfte sein, dass die korrekte Erstellung solcher Schedules ein schlecht skalierendes Problem darstellt. Ähnlich schwer ist die Planung von Echtzeittasks auf Prozessoren. Im einfachsten Fall steht nur ein Prozessor zur Verfügung. Bereits dafür gibt es eine Vielzahl von Algorithmen und Konzepten. Grundlegend sind hierbei die Priorisierung nach der *absoluten* Deadline (Earliest Deadline First, EDF) sowie die nach der *relativen* Deadline (ratenmonotones Scheduling, RMS).

Seit etwa 2005 haben Mehrkern- und Mehrprozessor-Systeme ihren Siegeszug angetreten. Neue PCs und mobile Computer werden fast nur noch mit derartigen Architekturen verkauft. Auch für die oben erwähnten eingebetteten Systeme läuft die Umstellung bzw. ist sie geplant. Während das Echtzeit-Scheduling auf Einprozessorsystemen heute weitgehend gelöst ist, bestehen noch viele offene Probleme des Echtzeit-Schedulings auf Mehrprozessor-Systemen. Leicht zu implementierende Greedy-Algorithmen wie RMS oder EDF sind hier nicht mehr optimal, wie bereits 1978 von Dhall und Liu publiziert. Diese Betrachtungen verdeutlichen die Notwendigkeit der Entwicklung spezifischer Echtzeit-Scheduling-Algorithmen für Mehrprozessor-Systeme.

Dabei unterscheidet man globales und partitioniertes Scheduling. Globales Scheduling verwendet eine gemeinsame Warteschlange. Der partitionierte Ansatz, der in dieser Arbeit vorwiegend betrachtet wird, reduziert dagegen das Problem auf das der statischen Verteilung von Tasks auf die Prozessoren bzw.

Kerne und mehrere Einprozessor-Scheduling-Probleme, welche bereits gut gelöst sind. Somit bleibt das NP-schwere Problem der Verteilung der Tasks auf die Prozessoren. Für größere Taskmengen lässt sich die kombinatorische Explosion der Möglichkeiten nur durch den Rückzug auf Näherungslösungen, die mit Hilfe von Heuristiken wie z.B. *First Fit* oder *Next Fit* gefunden werden können, handhaben. Zu beachten ist die Verwandtschaft des Problems zum Behälterproblem. Allerdings ist partitioniertes RMS komplizierter, da die Periodendauern als zusätzliche Parameter die maximal mögliche Last beeinflussen.

Beschleunigte einfach-periodische Taskmengen. Einfach-periodische (harmonische) Taskmengen sind durch paarweise ganzzahlige Verhältnisse von Perioden gekennzeichnet. Diese Situation ist besonders günstig für die Planbarkeit nach RMS, da dann kein Verschnitt auftritt und somit eine Last von 100% erreicht werden kann. Dies ist ein enormer Vorteil gegenüber der allgemeinen Liu-Layland-Lastgrenze von ca. 69,3%. Da eine Verkürzung von Periodendauern einer Verschärfung der Anforderungen entspricht (Sustainability, Nachhaltigkeit), kann das Finden von beschleunigten einfach-periodischen Taskmengen (Accelerated Simply Periodic Task Sets, ASPTSs) hervorragend zum Beweis der Planbarkeit von allgemeinen Taskmengen unter RMS genutzt werden. Solche hinreichenden Planbarkeitskriterien sind angebracht, da die exakte Analyse mittels TDA (Time-Demand Analysis) pseudo-polynomielle Laufzeit benötigt, was für größere Taskmengen nicht mehr handhabbar ist. Bei Nutzung des partitionierten Ansatzes sind ASPTSs ebenso für das Multiprozessor-Scheduling geeignet. Die Herausforderung hierbei ist das Finden von günstigen Kombinationen aus Einprozessor-Einplanungstest und Zuordnungsheuristiken.

Zirkuläre Periodenähnlichkeit. Ein etablierter Algorithmus, der die Ähnlichkeit zwischen den Perioden von Tasks für ein leistungsfähiges partitioniertes RMS in einem Vorsortierschritt nutzt, ist Rate-Monotonic Small Tasks (RMST), 1995 entwickelt von Burchard *et al.* Es wird gezeigt, wie dieser Algorithmus u.a. durch ein Einpassen des Konzepts der einfach-periodischen Taskmengen verbessert werden kann. Interessant ist in diesem Zusammenhang auch der allgemeinere Blick auf die Vorsortierung als Übergang von einem Online- zu einem Offlineansatz. Dabei interessiert der Einfluss des Sortierkriteriums (bzw. der Sortierkriterien bei zweistufigen Ansätzen) auf die Minimierung der Leerlaufzeiten und damit die Leistungsfähigkeit des Algorithmus.

Es wird gezeigt, wie die Leistungsfähigkeit von RMST im Sinne der Minimierung von Leerlaufzeiten auf Prozessoren durch den Übergang von einer linearen zu einer zirkulären Spannweite gesteigert werden kann. Zugleich vereinfacht sich dabei der Burchard-Einplanungstest für Einprozessorsysteme. Eine bisher notwendige Fallunterscheidung wird entbehrlich. Nach bestem Wissen des Autors ist dies die erste Anwendung zirkulärer Statistik im Echtzeitscheduling.

Plötzliche Phasenübergänge. Partitioniertes EDF zeigt plötzliche Phasenübergänge an einem Schwellwert der Gesamtlast der Tasks. Eine kontinuierliche Erhöhung dieser führt zu einem plötzlichen Umschlagen vom Zustand „fast sicher planbar“ in den Zustand „fast sicher nicht planbar“. Häufig können diese Schwellwerte nur näherungsweise in Form von Schranken ermittelt werden. Für einige Fälle gelingt jedoch eine Bestimmung der exakten Werte. Die Kenntnis solcher Schwellwerte bzw. der Wahrscheinlichkeits-Last-Funktionen ermöglicht es dann, in konstanter Zeit zu entscheiden, ob ein Task noch angenommen werden kann oder nicht. Dieses Vorgehen eignet sich für weiche Echtzeitsysteme mit einer Online-Zugangskontrolle. Das Risiko einer falsch-positiven bzw. falsch-negativen Entscheidung (z.B. 5%) kann dabei vorgegeben werden. Die Kenntnis der Wahrscheinlichkeits-Last-Funktion ermöglicht die Ableitung passender Schranken.

Ziel. Das Ziel der Habilitation ist also das Aufzeigen von neuen, fortgeschrittenen Verfahren, um Echtzeitscheduling auf Mehrprozessor-Systemen zu realisieren. Der Schwerpunkt liegt bei partitionierten Ansätzen, die Arbeit konzentriert sich auf hinreichende Einplanungstests. Zielrichtung ist dabei die Maximierung der möglichen Last im ungünstigsten bzw. mittleren Fall bei einer gegebenen Anzahl von Prozessoren. Dieses Szenario ist realitätsnaher als der duale Fall der Minimierung der Anzahl notwendiger Prozessoren unter einer gegebenen Taskmenge, da die Hardware in der Regel fixiert ist. Hinreichende Einplanungstests können für schnelle Abschätzungen der Taskmengen-Planbarkeit in automatischen Systemsynthese-Werkzeugen nützlich sein.

Scheduling, Mehrprozessorsystem, Echtzeit, partitioniert, ratenmonoton, zirkuläres Ähnlichkeitsmaß, Phasenübergang, Schwellwert

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Part I

Foundations

This part first gives fundamental terms and concepts. Next, basics of uniprocessor scheduling are presented.

Chapter 1

Basic Terms and Concepts

This Chapter serves as an introduction to the fascinating field of real-time scheduling. Readers already familiar with real-time scheduling might tend to skip it. But I would recommend to read it at least in a cursory manner since the framework of terms and concepts used later on is built up here. The structure of the thesis is given at the end of this Chapter in Subsection 1.9.2.

1.1 Real Time

1.1.1 Abstraction and Time in Programming Languages

Architectures

The principles of the famous *Von Neumann architecture* were published by John von Neumann [228] in 1945. This publication can be regarded as a breakthrough for the construction of digital computers. One of the central points of his concept is the storage of code and data processed by the central control (CC) in one and the same physical memory (M).

> "The orders which are received by CC come from M, i.e. from the same place where the numerical material is stored.", [228], Section 14.1

Note that this relatively simple idea enormously increases flexibility since programs can be regarded as a special kind of data and, thus, easily be manipulated. It enables also the implementation of *self-modifying code.* Although this is nowadays generally seen as a bad practice, its use might be acceptable in the form of *function pointers* since this allows for a more structured and, thus, controllable modification. The opposite approach of a (physical) separation of code and data is the constitutive feature of the so-called *Harvard architecture*. It follows the mighty computer science principle *Separation of Concerns* (SoC). Today's computers often are a mixture of these opposing approaches. Interestingly, a flag to discriminate between data and code called NX bit (from "No eXectue") was introduced on the Von Neumann architecture to fight the threat by computer viruses in the form of malicious data interpreted (and then executed) as code.

Synchronization

Next to that, von Neumann recommended to use a central clock for synchronization of the units, see "Synchronism, gating by a central clock", Section 6.3 of [228]. He was highly aware of the fact that a proper timing is one of the key factors to build a well-working digital computer.

> "All questions of timing and of speed, and of the relative importance of various factors, must be settled within the framework of these considerations.", [228], Section 3.1

On the level of machine language and assembler, each instruction is either associated with a constant timing (e.g., CWD - Convert Word to Doubleword) or highly dependent upon argument data (e.g., MUL - Unsigned Multiply). This already suggests that an estimation of the total running time of a given program is *data-dependent* and might be difficult. Note that even the decision whether a program finishes on a particular input or runs forever is undecidable over *Turing machines* and was coined the *Halting problem* [269].

High-level Programming Languages

Later, from the 1950s on, *high-level programming languages* were developed on top of assembler. Tools like compilers and interpreters are needed to translate programs written in them to machine-oriented assembler. They are intended to make software development easier and more human-oriented by using (in most cases English) natural language keywords and to make software portable. Only the above mentioned translation tools have to remain machine-dependent. This abstraction is undoubtedly a great achievement and was only recently outplayed in terms of abstraction by *model-driven software development* where programmers just create the models. Tools known as generators then bridge the gap to source code in high-level programming languages.

Loss of Time and Getting it Back by WCETs

But the in the synchronous von Neumann approach inherent concept of time has been *abstracted away* to a great deal in popular higher programming languages. This is considered a problem in situations were worst-case timing and not only average-case timing becomes an issue. Today's computers are very quick in terms of *clock rate* and average *cycles-per-instruction* (CPI) value. But this average is of only restricted or even no worth in the context of (hard) real-time systems where *worst-case execution times, WCETs*, see 1.1.2, are required in order to guarantee meeting deadlines.

A WCET time analysis is only valid for a concrete hardware platform. An entire computer science branch with own conferences and workshops has evolved in order to research the practical and theoretical methods for determination and estimation of WCETs. A good survey on this topic can be found in [283].

Synchronous Programming and Modeling Languages

A second solution to the problem that timing is hard to analyze in high-level programming languages is the implementation of the synchronous paradigm. The basic characteristic in synchronous systems is that each computation step takes exactly one time step and that the results are available to all units immediately. This is motivated by the synchronous abstraction of electronic circuits where the time response of transistors and the signal propagation speed is abstracted away, too.

So, program execution becomes deterministic even for concurrent program parts. This allows for deterministic semantics ideally suited for formal specification, analysis and verification. The obtained coincidence of worst-case and average-case timing has the capability of making real-time scheduling less pessimistic since the padding of the actual execution time up to the WCET then has a zero contribution. Hence, the above mentioned science branch of WCET analysis becomes obsolete by switching to the synchronous paradigm.

Examples of synchronous programming languages are *Lustre*, *Esterel*, *Signal* and *Atom*. On the next level of abstraction, the model level, popular synchronous modeling languages are SCADE and UML MARTE.

Summary

As we could see, time aspects are not covered by standard high-level programming languages. Time has been abstracted away. This is also true for the instruction sets of microprocessors. For general-purpose computing, this approach turned out to be powerful since it provides portability of code. Reducing response times is a matter of *performance* which can be achieved by hardware upgrades.

Predictable timing can be achieved to some degree by taking WCETs into account or by the use of synchronous programing and modeling languages. But there is still the problem that the underlying processor architecture can have an instruction set with varying execution times[1]. Hence, new processor architectures like *Precision-Timed Processors* (PRET) [176] might be a significant part of the solution. For more details on temporal aspects in computing, see [116] and [176].

1.1.2 Timing Constraints and Deadlines

Real-time System

For real-time systems, timing belongs no more to performance, but to *correctness*. A widely agreed definition of a real-time computer system is given by Kopetz:

> "A *real-time computer system* is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced. [..]", [169], p. 2

[1]For *distributed* real-time systems, the network timing comes as a third factor into play. Timing needs again to be considered as a correctness and not as a performance criterion. This can be achieved by using the *time-triggered* approach.

Such a real-time computer system is always embedded in a larger system called *real-time system* or *cyber-physical system* [169], p. 2. An interpretation of this statement could be that all real-time systems are embedded systems. But usually, the term *embedded system* is applied in the narrow sense. Constitutive characteristics of an embedded system are its static structure optimized for a special purpose and its mass production[2] among others [169], p. 2. For *mobile* embedded systems, features like the ability to communicate with a a larger system or the Internet and the limited amount of energy provided by a battery have to be added. According to the definition by Kopetz, timing constraints are constitutive for real-time systems.

Job and Task

A job is a "[..]unit of work that is scheduled and executed by the system[..]" [190], p. 26. A task is defined as a "[..]set of related jobs which jointly provide some system function[..]" [190], p. 26.

Deadline, Release and Response Time

The most typical timing constraint is the not-too-late completion of a job. This means that *predictability* is the primary concern for real-time systems.

The maximum tolerable point in time where a job completes is called the job's (absolute) *deadline* d_i. Opposed to that, the relative deadline D_i of a job is the interval of time between its *release time* r_i and its absolute deadline d_i. In situations where the context clarifies whether absolute or relative deadlines are meant, we omit the explicit use of the long form with the adjective. Summing up, the relationship between the terms introduced is given in (1.1).

$$d_i = r_i + D_i \tag{1.1}$$

The *actual* time between release at r_i and completion of a job at c_i, see (1.2), is called *response time* R_i. So, the basic requirement for a hard real-time job, see Subsection 1.1.4, is (1.3) or, equivalently, (1.4).

[2]The term mass has to be understood in total. A car company could sell 5 million cars each worth 20,000 € per year making an annual sales volume of 100 billion €. An aircraft producer could achieve the same volume of sales by selling 500 airplanes each worth 200 million €.

$$R_i := c_i - r_i \tag{1.2}$$

$$R_i \leq D_i \tag{1.3}$$

$$c_i \leq d_i \tag{1.4}$$

The actual response times may vary for different task instances (jobs). So, it is useful to consider the *worst-case response time* (WCRT) of a task, the maximum over all response times of a task's job. Then, only this maximum needs to be compared with the relative deadline[3]. Transitivity of the greater-than-or-equal-to relation ensures than condition (1.3).

WCET

As already discussed in 1.1.1, the notion of worst-case execution times (WCETs) of jobs is fundamental for real-time systems. We coin it e_i. Then, (1.5) is valid for all possible actual job execution times $e_{i,act}$.

$$e_{i,act} \leq e_i \tag{1.5}$$

Note that there is a generalization of WCET called WCET-matrix [235] which is less pessimistic by specifying WCETs in a case-by-case analysis of the concrete execution environment.

For a consideration of WCETs on multiprocessors, see Subsection 1.6.6.

Laxity, Lateness and Tardiness

The *laxity* or *slack* of a job at a certain time t is the difference between the time to the absolute deadline—which can be interpreted as the current relative deadline $D_i(t)$—and the remaining execution time $e_i(t)$, see (1.6). Thus, a job's laxity decreases (with slope -1) when it is running and remains constant (slope 0) when not.

$$l_i(t) := (d_i - t) - e_{i,rem}(t) = D_i(t) - e_i(t) \tag{1.6}$$

[3]We are assuming here the typical situation of a per-task relative deadline D_i.

The initial laxity l_i of a job is then defined by the initial laxity at the job's release time, see (1.7).

$$l_i = l_i(r_i) = d_i - r_i - e_i = D_i - e_i \tag{1.7}$$

When not meeting the deadline, the job completes with a negative laxity. The final laxity's inverse value is then aka *lateness* [190], p. 54, see (1.8).

$$late_i := -l_i(c_i) = c_i - d_i \tag{1.8}$$

Opposed to lateness, *tardiness* (1.9) can never become negative [190], p. 28.

$$tardi_i := \max(0, -l_i(c_i)) = \max(0, c_i - d_i) \tag{1.9}$$

The basic parameters of a job belonging to a task are summarized in Figure 1.1. Note that a job's release is visualized by an upward pointing arrow while a job's absolute deadline is illustrated by a downward pointing arrow. If a job's absolute deadline coincides with the release of a subsequent job in a task[4], a double arrow or just a vertical line are commonly used.

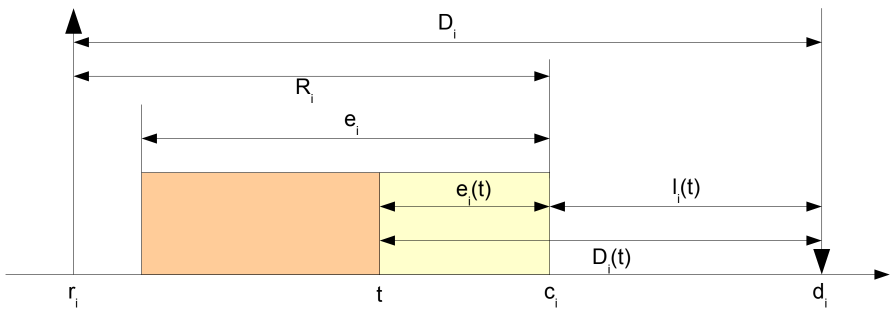

Figure 1.1: Basic Parameters of a Job: Release Time r_i, Completion Time c_i, Response Time R_i, Absolute Deadline d_i, Relative Deadline D_i, Execution Time e_i, Remaining Execution Time $e_i(t)$, Initial Laxity l_i and (Remaining) Laxity $l_i(t)$

Real-time and High-performance computing

It has to be stressed that real-time is not equivalent to high-performance computing. The deadlines set requirements to be met even in the worst case. This

[4]This situation occurs regularly for implicit-deadline tasks, cf. Subsection 1.2.5.

is consistent with limiting the execution time on job level where the most important real-time parameter is the job's WCET e_i. Real-time systems favor *predictability* over (average-case) performance. This makes them quite different from a PC with a desktop operating system. Both your PC hardware and your standard operating system are optimized for average-case performance. In order to narrow the gap between CPU and memory access in terms of speed[5], an entire hierarchy of caches has been added. These caches are using policies like, e.g., Least Recently Used (LRU) to decide which data to be kept in the cache. Cache behavior is very hard to analyze and can be seen as non-deterministic since it largely depends on the initial state of all the caches involved[6]. The difference between high-performance and real-time computing will be explained in the following using some scenarios and examples.

When comparing two jobs A and B on concrete platforms, it is well possible that job A has a lower average-case execution time (ACET), but a greater WCET than job B. In the extreme case, the WCET e_i could even be unbounded, i.e., it shall be set to infinity. For implementing a real-time system and having a choice between jobs A and B, job B should be preferred over job B due to its better worst-case behavior.

A computer-based scientific simulation of e.g., a double pendulum, can require—depending on the level of detail used in the model—high-performance computing. But it is not time-critical[7], it is not a real-time system.

An anti-lock braking system (ABS) has to meet all its deadlines. But it is not useful to further speed-up the involved calculations. Due to the cost-pressure in the automotive sector no reasonable company would exceed the requirements. Hence, it is a (hard) real-time system, but there is no high-performance computing involved.

Computerized weather forecast is based on modeling and running ensemble simulations where the initial state is varied slightly [100]. Then, based on the proportions of the outcomes of the simulations, probabilities of the

[5]This is well-known as the *memory wall*, see, e.g., [285].

[6]This is the reason that some benchmarks refer to cold or warm caches. Typically, an application runs faster on a warm cache where relevant data has already been loaded.

[7]It has to be admitted that the narrowness of human patience sets a kind of limit comparable to a (soft) deadline. On the other hand, scientists are known to be thorough and patient; even a continuation of work by one's successors is imaginable.

weather types can be computed. In the 1950s, available computers were slow and ensemble simulations were not yet discovered. Even a single model simulation for forecasting tomorrow's weather took a long time, often more than a day. Thus the result had only some theoretical, but no practical value since observation results were more up-to-date than simulation ones. The real-time requirement of the one-day forecast with a relative deadline of one day was not met. Summing up, computer-based weather forecast is *both* high-performance and real-time computing, only the relative deadlines are of another order of magnitude than engineers are used to in typical real-time system design.

1.1.3 Scheduling

Scheduling is the creation of a *schedule* for a set of jobs mapping them in a time-dependent manner to a set of processors. So, the schedule is a time table for jobs running on processors. Goals of classical scheduling are high fairness, high resource efficiency, high throughput and low latency. Note that these parameters are typically optimized on an *average-case* basis.

A *valid* schedule is defined by the following conditions [190], p. 53.

1. At any time, a processor can run at most one job.
2. Every job runs at at most one processor at any time.
3. The earliest point in time a job can run is its release time.
4. The amount of processor time assigned to a job corresponds to its actual or its maximum execution time.
5. All precedence and resource usage constraints are met.

Note that conditions 1 and 2 determine an at most one-to-one relationship between job and processor at any time. Precedence constraints are discussed in Section 1.3. The problem of resource constraints is discussed in Section 1.5.

Real-time Scheduling

Real-time scheduling additionally requires that deadlines (or more generally, timing constraints) have to be taken into account. If all timing constraints are met, we call a valid schedule a *feasible* schedule. Deadlines are limits in *worst-case* scenarios. This requirement is often in conflict to the classical goals of scheduling mentioned above. The only parameter still being important in the real-time context is resource efficiency, meaning that as few processors as possible shall be used for solving the given problem.

A summary of the different types of schedules and their interrelationships is given in Figure 1.2. In the following, the focus is on feasible schedules, on constructing them and proving their existence, cf. Section 1.7.

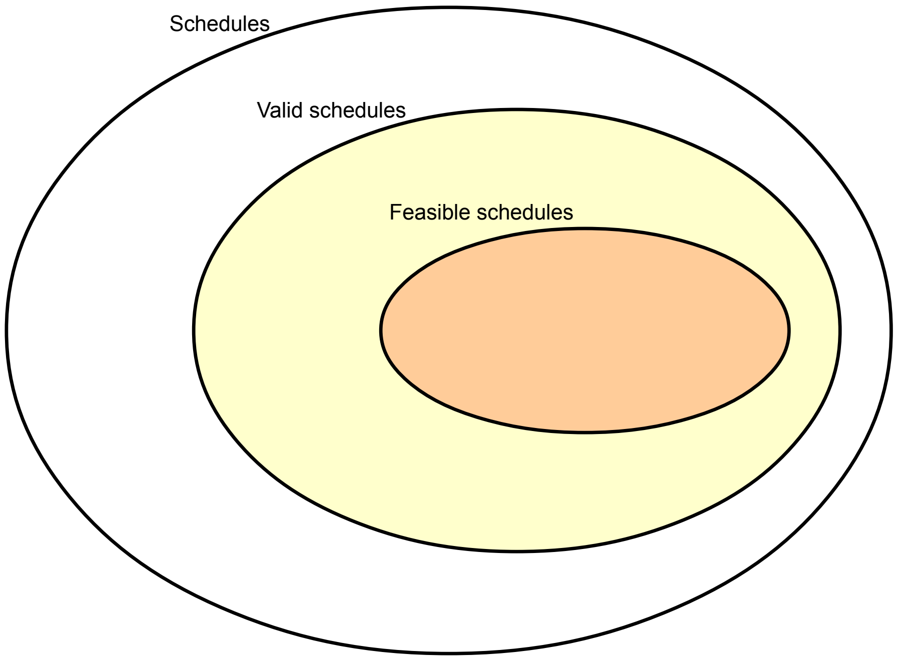

Figure 1.2: Valid and Feasible Schedules

1.1.4 Hard, Soft and Firm Deadlines; TUFs

Hard Deadlines

In a strict interpretation, the given deadlines have to be met under all circumstances since the violation of a single deadline can result in a catastrophe. This could be the loss of human lives, severe injuries or substantial property damage. Then, the deadlines are *hard* ones and we call such a system a hard real-time system. Examples are pacemakers as medical equipment and avionics systems in transportation.

Soft Deadlines

An important application of IT is multimedia including audio and video communication and entertainment. Here, the violation of deadlines results only in a degradation in the Quality of Service (QoS). The degree of this degradation often depends on the lateness, see 1.1.2. Here, the deadlines are *soft* ones and we call such a system a soft real-time system. So, a typical goal in soft real-time systems is the minimization of the *average tardiness*[8] where negative lateness counts as zero, see 1.1.2, since it should be impossible to compensate for deadline violations by earlier responses. The omission of some pixels or even of single frames can often be tolerated since only the quality of the video perceived on the user side is reduced.

Jitter. For some applications like film dubbing and digital video streaming, the minimization of *jitter* is a primary goal since the human visual system is more sensitive to relative track offsets (audio vs. video) than to common absolute delays. According to [281], jitter is "[..]the undesired deviation from true periodicity[..]". Usually, jitter denotes the maximum tolerable absolute value of such a deviation. So, it is a maximum time difference.

Jitter can be be reduced by using a *jitter buffer*, a FIFO-based[9] temporary storage compensating for arrival jitter at the expense of increased delays. In many cases, a huge jitter buffer is desirable. But in the challenging application *IP telephony*, a compromise between jitter reduction and delay restriction has to be found since a delay is also bothersome in this application.

Physiology. Note that an error being small enough might even not be *detectable* by the human visual system. Besides reduced temporal accuracy, reduced spatial accuracy can be tolerated sometimes as well. An example is the reduced color resolution of the human visual system compared to its black-

[8]Note that the simpler approach of average lateness is not reasonable since arbitrary great fluctuations can cancel out.

[9]*First In, First Out*

and-white resolution. This physiological fact can serve as a basis for advanced 3-D display implementations, cf. [215].

Firm Deadlines

A third group are systems with *firm* deadlines which are sometimes subsumed under soft RT systems. The sporadic violation of deadlines is tolerable although the usefulness of results beyond a deadline is zero. A burst of deadline misses is not tolerable. Examples are route guidance systems. Loosing the GPS signal for a single polling time is not critical since the absolute position can still be sufficiently approximated by extrapolation from the former position using relative movement indicators like velocity and steering angle. In this context, the concept of *sensor fusion*, or, more general, *data fusion* becomes important. A fundamental tool for achieving a reasonable fusion of data is the Kalman filter [149].

Time-Utility Functions

Time-Utility Functions (TUFs) were developed in the military context by Jensen in the 1970s and published in 1985, see [146]. Their strength lies in their ability to generalize and, thus, to integrate hard and soft deadlines. A TUF maps the utility of a result to the time when it is delivered, the completion time of the job. This utility can be positive (reward), negative (penalty) or zero (neutral).

An example is given in Figure 1.3. Such a qualitative course of a TUF both penalizing a too late and a too early delivery can be found, e.g., for the *ignition timing* in *spark ignition internal combustion engines* used in the form of Otto engines in cars. The lower the gas consumption, the higher the utility[10].

A hard deadline can be characterized by a positive constant utility up to the deadline and a downward step to negative infinity at the deadline, see Figure 1.4.

[10]Note that the optimal ignition timing is dependent upon engine speed and engine load, i.e., the current operating point of the engine. Further, engine damage can happen both with a much too early and a much too late ignition. Hence, ignition timing combines soft and hard deadlines showing the practical relevance of the generalization to TUFs.

A soft-deadline TUF coincides with a hard-deadline TUF at times before the deadline. But beyond the deadline, it is typically non-increasing in a smooth manner down to zero utility, cf. Figure 1.5.

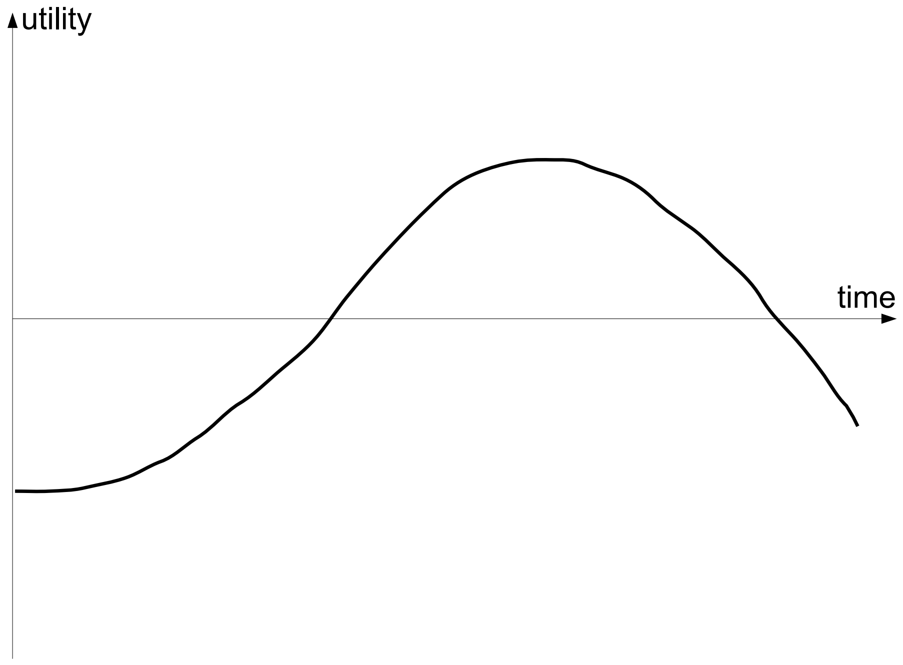

Figure 1.3: General Case of a Time-Utility Function (TUF); Here, e.g., Both a too Early and a too Late Delivery of the Result is Penalized.

Profit and Penalty Aware Scheduling

Recently, Li *et al.* proposed an extension of the TUF concept [188]. There, different TUFs for profit and penalty are suggested in order to adapt to situations where completion and abort of a job yield different utilities. Then, the reduction to a single TUF as difference of gain and loss is not appropriate[11]. These two time-dependent functions are also called *Completion TUF* for the gain and *Abort TUF* for the loss [188].

An example explained in [188] is an *online travel planning service provider*. Such a gain/penalty-time relationship makes it crucial for the system to estimate the odds for job completion before its deadline as early as possible

[11]Note that the utility is the negative value of the loss in case of aborting the job, but the positive value of the gain in case of completing it.

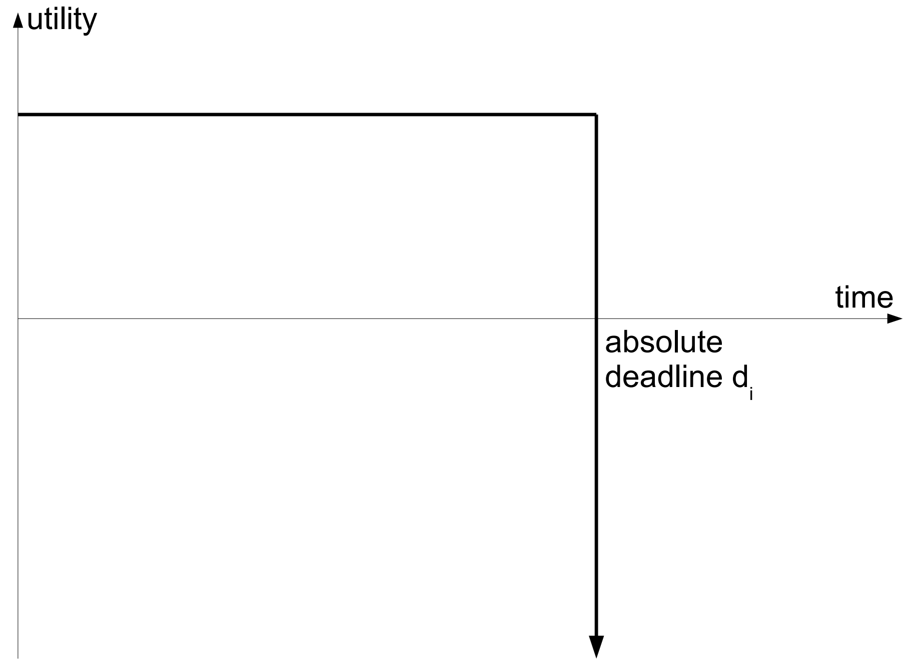

Figure 1.4: Typical Time-Utility Function (TUF) of a Hard Deadline

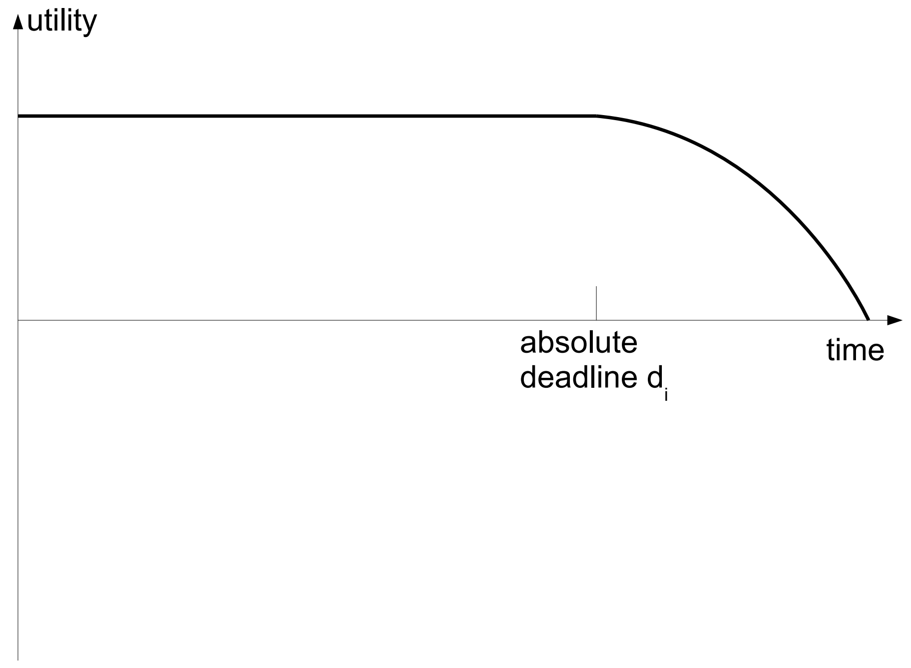

Figure 1.5: Typical Time-Utility Function (TUF) of a Soft Deadline

since penalties due to job abortion can then be reduced. As an example, in Figure 1.6, two linear functions of gain and loss are shown.

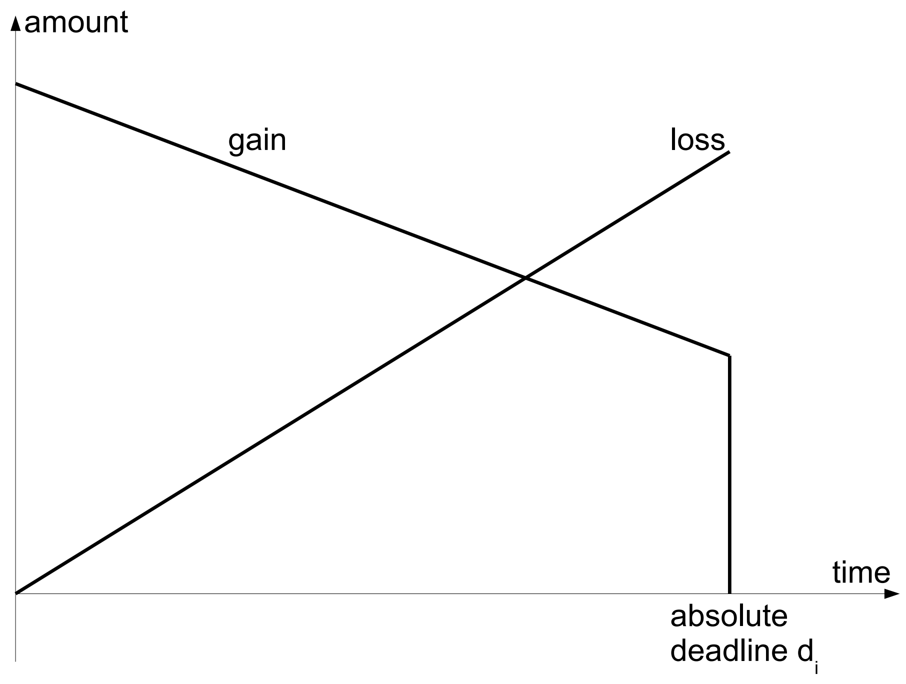

Figure 1.6: Time-dependent Gain and Loss

In [188], "[t]he ratio between a potential loss against an expected gain is hence used as an index to measure the risk of processing a task." Then, a critical time based on a pre-set maximum tolerable risk can be calculated. Both preemptive and non-preemptive scheduling algorithms are proposed with an objective to maximize system's total utility [188].

An alternative approach based on the economic concept of *opportunity cost* presented in [191] turned out to be less successful compared to scheduling based on expected utility.

1.2 Basic Event Models: Activation Patterns

A finite *task set* consists of n tasks indexed from 1 to n and coined T_1 to T_n. The activation behavior of jobs belonging to a task is described by the *event model*. These models are considered from the most specific one, the periodic model, to the general model of *real-time calculus*.

1.2.1 Periodic Event Model

One of the simplest and most widely used event models is the *periodic* one. It is motivated by closed-loop control applications where sensors measure and actuators act periodically. There, the deadlines are a result of the *stability* requirement for closed-loop control. It was described in the seminal paper of Liu and Layland in 1973 [189].

Here, releases of subsequent jobs happen regularly separated by a *constant* interval of time p_i called *period*[12]. The release time of the first job of a task is called the *phase* φ_i of this task. Summing up, the release times of a task's jobs are given by (1.10).

$$r_{ij} = \varphi_i + jp_i \tag{1.10}$$

The j-th released job of task i has an individual absolute deadline d_{ij} calculated according to 1.11 as a result of combining (1.1) and (1.10).

$$d_{ij} = r_{ij} + D_i = \varphi_i + jp_i + D_i \tag{1.11}$$

Utilization

The utilization of a task is defined as the ratio of its WCET and its period. This is given in (1.12). More precisely, u_i is called the processor utilization factor of a task, describing "[..]the fraction of processor time spent in the execution of the task set." [189] More intuitively, utilization can be explained as "[..] one minus the fraction of idle processor time." [189]

$$u_i := \frac{e_i}{p_i} \tag{1.12}$$

The (total) utilization u of a task set is simply the sum of all utilization values of its tasks, see (1.13).

$$u := \sum_{i=1}^{n} u_i \tag{1.13}$$

[12]Note that in [190], p. 40, periodic tasks subsume tasks with an interarrival time bounded from below. This is justified by the wide applicability of scheduling and validation algorithms for the periodic event model also for this more general model corresponding to sporadic tasks. But this approach mixes definitions and results. Thus, the principle of *Separation of Concerns* (SoC) would be violated. We prefer to keep these issues separated.

Next, the maximum utilization of a task in a task set u_{max} is an important parameter of a task set, see (1.14).

$$u_{max} := \max u_i \tag{1.14}$$

Synchronism

A *synchronous* task set is a task set where all tasks' phases are equal. Since a global offset is only an index shift of time yielding an equivalent system, all phases can be assumed to be zero, $\forall 1 \leq i \leq n : \varphi_i = 0$ for a synchronous task system.

Hyperperiod

The *hyperperiod* H of a task set represents the period of the smallest major cycle of the system. It is calculated as the *least common multiple* of all period values [184], see (1.15).

$$H = \text{lcm}(\{p_i | 1 \leq i \leq n\}) \tag{1.15}$$

Integer divisibility relationship creates a complete lattice based on the finite set of period values with the *supremum* least common multiple (lcm) and the *infimum* greatest common divisor (gcd). For an example, see Figure 1.7.

Simply Periodic Task Sets and Semi-harmonic Task Sets

Often, it is interesting to consider special cases with a short major cycle in order to make task systems better analyzable and to increase schedulability for certain scheduling algorithms.

The shortest possible hyperperiod equals the maximum period of all tasks in the task set. Then, each period has to be an integer fraction of the longest period. Such a task set is called a *semi-harmonic* task set, cf. [75]. An example of this situation can be found in Figure 1.8.

Tightening restrictions leads to the notion of *simply periodic* [216] or, synonymously, *harmonic* [75] task sets. Here, the complete lattice based on the divisibility relationship is degenerated to a chain describing a totally ordered set of periods. I.e., each period is an integer part of all non-shorter periods, see (1.16).

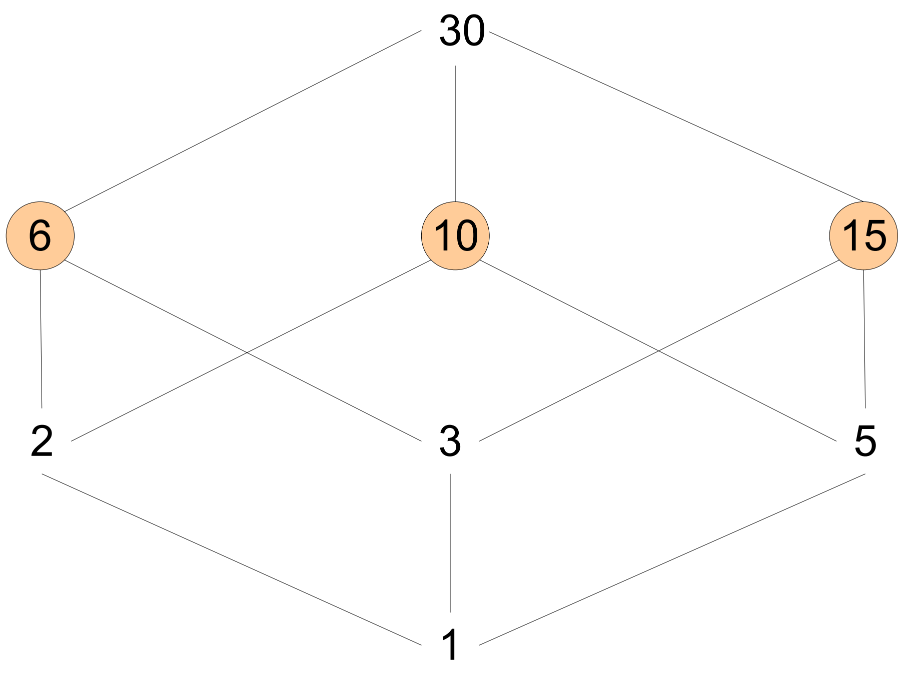

Figure 1.7: Complete Lattice of a General Task Set with Periods {6,10,15} and Hyperperiod 30

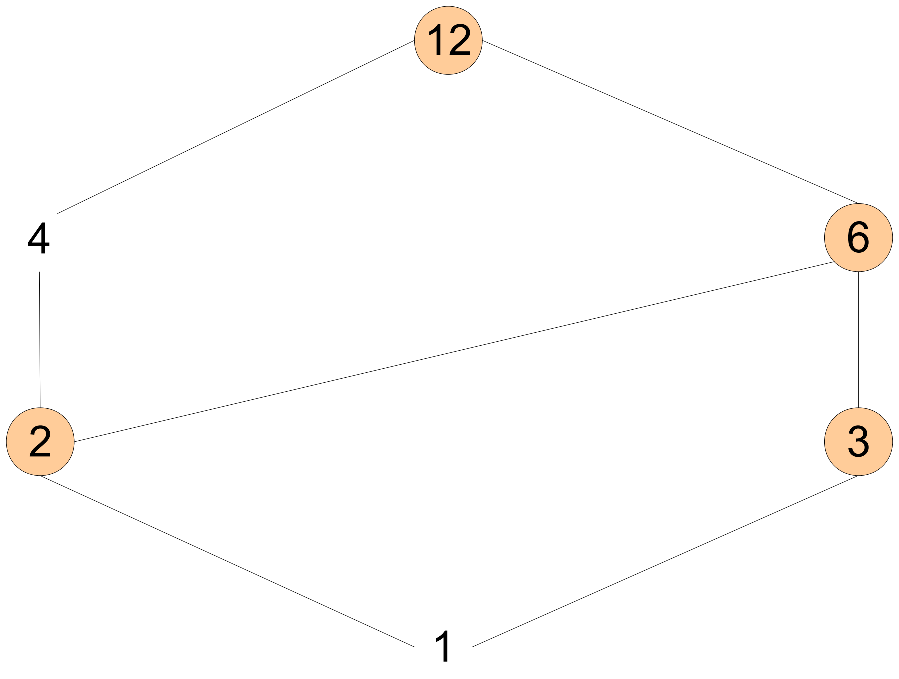

Figure 1.8: Complete Lattice of a Semi-harmonic Task Set with Periods {2,3,6,12} and Hyperperiod 12

$$\forall 1 \leq i,j \leq n : p_i \leq p_j \implies p_i \mid p_j \tag{1.16}$$

An example of a simply periodic task set is shown in Figure 1.9.

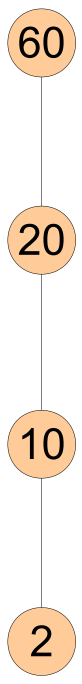

Figure 1.9: Complete Lattice of a Simply Periodic Task Set with Periods {2,10,20,60} and Hyperperiod 60

Strongly Simply Periodic Task Sets and Binary Task Sets

In an article on non-preemtive scheduling, cf. Subsection 1.4.2, two useful specializations were introduced [68].

A *strongly simply periodic task set* is a simply periodic task set with no two tasks having the same period (1.17), see [68].

$$\forall 1 \leq i,j \leq n : i \neq j \implies p_i \neq p_j \tag{1.17}$$

This means that duplicates are excluded, denoted by the term "strongly". We can obtain strongly simply periodic task sets from simply periodic task sets by merging tasks of the same period to compound tasks. In many situations, this yields an equivalent task set.

A *binary task set* is a task set with each period being twice the next shorter period (1.18), see [68].

$$\forall 1 \leq i \leq n-1 : p_{i+1} = 2p_i \tag{1.18}$$

Note that duplicates are already excluded by this definition. Implicitly, (1.18) constrains task sets to be sorted according to increasing periods. Since this recursive constraint (1.18) contains no recursion base, the shortest period can be chosen arbitrarily. Notably, each single-task task set ($n = 1$) is a binary one by this definition.

1.2.2 Sporadic Event Model

The rigid event model of task activations exactly each period was relaxed by introducing a *minimum interarrival time* which can be regarded as a generalization of the concept *period*. Tasks of such an activation pattern are called *sporadic* tasks. The model was introduced in 1983 by Mok [212].

1.2.3 Aperiodic Event Model

If there are no constraints on the activation pattern of the jobs of a task, we call it *aperiodic* task. This model is presented in [259].

Then, hard real-time is no longer possible since a single aperiodic task could prevent all other tasks from execution by a dense activation of jobs. Only soft or firm deadlines are reasonable here. The usual goal for aperiodic tasks is to minimize their *average* response times in spite of the lack of guarantees. Note that this is contrary to the defining property of real-time systems: predictability. Such an approach is also known as *best-effort* scheduling.

Nevertheless, aperiodic tasks are important in the RT context since in practice, often a *co-existence* of periodic or sporadic tasks and aperiodic ones is required. Then, the average-case goal of the aperiodic jobs has to be achieved under the restriction that the worst-case goals of the periodic and sporadic tasks remain untouched. Such a co-existence might be effective in using the CPU resources temporarily available due to the pessimism of the WCET approach for hard-deadline periodic tasks, cf. [190], p. 39f.

Due to the lack of restrictions in the activation pattern of the jobs belonging to the task, this model is sometimes sourced out from the task models and referred to as an aperiodic job model.

1.2.4 Generalizations: Real-Time Calculus and Busy Window Method

An excellent generalizing approach is Real-Time Calculus (RTC) presented in [267] for uniprocessors and in [182] for multiprocessors. There, concepts like Max-Plus Linear System Theory[13], see [23], and Network Calculus are used to analyze RT systems in compositional manner by request and delivery curves. A key strength of RTC is the natural coverage of several scheduling policies.

A good overview on existing response-time analysis methods is given in [243]. There, a synthesis of the *busy window method* [180] and RTC is proposed as a mighty concept of response-time analysis also accounting for hierarchical scheduling relevant to the upcoming virtualization approaches.

As a starting point, the definition of a level-i busy period is given.

> "A level-i busy period is a time interval [a,b] within which jobs of priority i or higher are processed throughout [a,b] but no jobs of level i or higher are processed in $(a-\epsilon, a)$ or $(b, b+\epsilon)$ for sufficiently small $\epsilon > 0$.", [180]

For a basic understanding of priorities, see Subsection 1.4.1. Note that the exact schedulability test TDA, see Subsection 2.2.1, is a special case of the busy window method.

1.2.5 Implicit, Constrained and Arbitrary Deadlines

For the periodic and sporadic task model, we distinguish three different types of relative deadlines according to their relationship to periods or minimum interarrival times.

Implicit Deadlines

The easiest situation is that all relative deadlines coincide with the respective periods, see (1.19).

$$\forall 1 \leq i \leq n : D_i = p_i \tag{1.19}$$

Such a configuration ensures that there is always at most *one* task instance (job) ready under maximum margin of placement of a job in a period.

[13]aka (max,+) algebra

Density

For the case of deadlines non-equal to periods, the *density* δ_i of a task i becomes an important parameter. It is defined as (1.20).

$$\delta_i := \frac{e_i}{\min(D_i, p_i)} \tag{1.20}$$

The (total) density δ of a task set is correspondingly the sum of all density values of its tasks, see (1.21).

$$\delta := \sum_{i=1}^{n} \delta_i \tag{1.21}$$

Another important parameter is the maximum density δ_{max} , see (1.22).

$$\delta_{max} := \max \delta_i \tag{1.22}$$

Constrained Deadlines

Relaxing the condition that the jobs shall have their maximum margin in a period leads to the case of constrained deadlines, see (1.23).

$$\forall 1 \leq i \leq n : D_i \leq p_i \tag{1.23}$$

Such requirements could be useful for modeling necessary I/O operations between completion of a job and release of the subsequent one, see [190], p. 41. The density equation (1.20) specializes for constrained deadlines to (1.24)

$$\delta_i = \frac{e_i}{D_i} \tag{1.24}$$

Arbitrary Deadlines

In the most general case, there are no constraints on positioning the deadlines. This has the drawback that there can be several instances (jobs) of the same task ready at the same time.

1.3 Basic Task Models: Precedence Constraints

Next to the different activation patterns discussed in 1.2, another dimension of the classification is based on dependencies between the jobs.

1.3.1 A Task of Independent Jobs

In the simplest case, all jobs in the system are completely independent from each other. Then, their sequence of execution can be chosen arbitrarily, only release times and deadlines establish restrictions. According to [190], p. 44, this is the usual assumption for the periodic event model.

1.3.2 Task Chain: Serial Job Order

A restriction becoming relevant in overload situations with the constrained-deadline model and for arbitrary deadlines in general is the allowance of the start of the next job of a task only if the previous task instance has been completed. This creates a linear dependency relationship or a *task chain* with the task's jobs in serial order.

1.3.3 Task Graph: General Job Order Description

A *precedence graph* is the Hasse diagram of the partial order on the jobs induced by the precedence relation. It is a directed acyclic graph (DAG) representing the transitive reduction of the precedence relation.

A *task graph* is an extension to a precedence graph allowing additionally concepts like AND/OR precedence constraints, conditional branches and pipeline relationships, cf. [190], p. 42ff. Note that, according to [190], p. 46, OR jobs with incoming k-out-of-l constraints are powerful enough to model practically relevant concepts like Triple Modular Redundancy (TMR) by a 2-out-of-3 choice, and two-version computation by a 1-out-of-2 choice. By supporting these two relevant redundancy techniques [278], p. 67ff., task graphs are even suitable for modeling some properties of dependable systems.

1.3.4 Generalizations

Generalizations only briefly discussed here are the multiframe, the generalized multiframe (GMF) [35], the non-cyclic GMF [214], the recurring RT (RRT) [37], the non-cyclic RRT [31], the Digraph[14] RT [264] and the Task Automata model [107]. For an overview on them, see Figure 1.10 and [264] for a more

[14]Digraph is a short form of *directed graph.*

detailed description. They are characterized by an increasing level of expressiveness. But this comes at the cost of an ever-higher schedulability analysis effort.

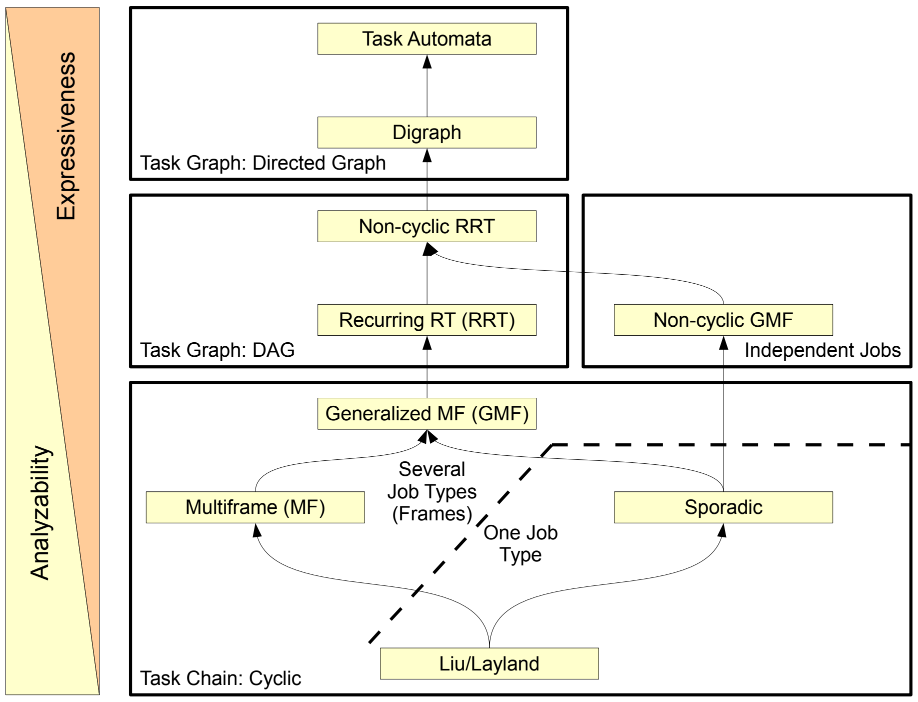

Figure 1.10: Important Task Models: Arrows Indicate the Direction of Generalization

The generalizations of the Liu/Layland task model given in Figure 1.10 involve several directions. First, deadlines D_i are decoupled from the corresponding periods p_i (sporadic)[15]. Next, the cyclic variation of WCETs (multiframe), deadlines and periods (GMF) is allowed. All these four models are of cyclic nature and can be modeled by task chains, cf. Subsection 1.3.2.

Subsequently, the allowance of conditional branches modeled by *directed acyclic graphs*, DAGs, (recurring RT) and of job sequences of any order (non-cyclic GMF) are two different directions of generalization of the GMF and of the sporadic task model. They are unified in the non-cyclic RRT model which both allows for branches and for non-cyclic behavior simulated by restarts

[15]Actually, this is the change from implicit to arbitrary deadlines, see Subsection 1.2.5. But in this context, it is common to attribute the sporadic task model, see Subsection 1.2.2 implicitly with arbitrary deadlines.

triggered by reaching sink vertices. The allowance of arbitrary graphs also permitting *dynamic* cycles (digraph) is a further step to increase expressiveness. In the most general model of *task automata* [107] as an extension of *timed automata* [7], advanced dependencies and sophisticated concepts like task synchronization are supported. Clearly, we need task graphs, cf. Subsection 1.3.3, to describe these approaches, except for the non-cyclic GMF model where a task consists of independent jobs[16], see Subsection 1.3.1.

1.4 Prioritization and Preemption

1.4.1 Prioritization

A *priority* is a time-dependent measure attached to an *execution unit* and a point of time. It expresses a precedence relationship among such units at a certain time. Usually, a lower numerical value expresses a higher priority. Then the resource-sharing problem of the CPU, CPU scheduling, can be solved by running always the task with the currently lowest priority value corresponding to the highest priority. This turned out to be a mighty concept.

[16]This might seem confusing since the simpler model of independent jobs is used for a higher abstraction level. But the task chain in the Liu/Layland and in the sporadic model degenerates to a single job. So, non-cyclic GMF adds a level in the hierarchy and groups jobs of one and the same task into job types called frames. An application of the non-cyclic GMF is *software-defined radio* (SDR) [214].

Priority-driven[17] real-time scheduling belongs to the group of online scheduling algorithms. It is widely accepted that priorities are the best approach to ensure timeliness for all jobs in an online situation.

Based on the level of execution units to which the priorities are attached as fixed values, typically three types of prioritization are distinguished.

In the case of a tie among jobs in terms of priorities, this tie has to be resolved in a fixed manner. Often, this can happen in an arbitrary but deterministic way. In such cases, it is most appropriate to use the *task index* as second, tie-breaking sorting criterion.

Static Priorities

The easiest, straightforward approach, is to assign priorities fixed to tasks. Then, they are *static* since they do not change their value during the entire runtime of the system. Synonymously, we are using the term *fixed priority* (FP). Such a fixed priority belonging to task i shall be termed $Prio_i$ in the following. The inequality $Prio_i > Prio_j$ means then that task i has a higher priority than task j although widespread formalisms and implementations characterize higher priorities by a *lower* numerical value. The approach is also known as *fixed-priority* scheduling.

Dynamic Priorities Fixed on Job Level

Next, priorities can be dynamic, but *fixed on job level.* This choice can be regarded as a compromise between static and fully-dynamic priorities.

[17]Clock-driven scheduling, cf. [190], p. 85ff., is an alternative. One possibility is *timeline scheduling* (TS) which precalculates a mapping of jobs to time slots offline, e.g. by solving a maximum-flow problem [55], p. 30f. and [190], p. 108ff. It is simple and has a very low runtime overhead due to the lack of both preemptions, see Subsection 1.4.2, and intra-frame scheduling decisions. On the other hand, TS is inflexible in overload situations and for handling aperiodic jobs and application changes. These issues can be mitigated by priority-driven scheduling.

Sometimes even *weighted round robin* scheduling is used for hard real-time communication [245]. With *round robin scheduling* as an online clock-driven policy, a time slice of a fixed size is offered to the active jobs in turn. For the generalization weighted round robin, time slice sizes are dependent upon the job type (task). Then, a proper weight allocation is crucial for ensuring RT behavior [245].

Fully-dynamic Priorities

Finally, priorities can be *fully-dynamic*. Then, after each quantum of time, the priority of a job can change which gives a maximum of flexibility. On the other hand, the overhead of a scheduling algorithm handlig fully-dynamic priorities might be high since there are many more scheduling points compared to the two other approaches.

1.4.2 Preemption

Preemption is the temporary interruption of a job in favor of another job which is considered more important at the time of interruption, based on the current priorities. Only higher-priority jobs can preempt a lower-priority job. This relation makes preemption a dual concept to blocking, cf. Subsection 1.5.1.

Preemption can improve the responsiveness of the system. Thus, this concept is widely used in modern desktop operating systems like Mac OS X, Linux and Windows. But relative deadlines occurring in real-time systems are upper bounds on the response time. Thus, it seems natural to apply preemption also to real-time systems which was well formulated by Saksena and Wang:

> "Preemptability is considered a necessary pre-requisite to meet timing requirements in real-time system design.", [254]

Preemptive, Non-preemptive and Cooperative Scheduling

But the situation is more complicated. The two extremes of non-preemptive and fully preemptive systems are not optimal in terms of schedulability. Schedulability can be increased by allowing only restricted preemptions. Already in 1992, Jeffay conjectured

> "[..]that if preemption among tasks is required for feasibility, it will be limited to a few tasks.", [144], p. 10

Besides this possibility to restrict preemptions on the task interaction level, one can select so-called *scheduling points* on the time axis. This is called *scheduling with deferred preemption* [61] or *cooperative scheduling*. For more details on restricted-preemption algorithms on uniprocessor, see Figure 2.7 in Subsection 2.7.1.

Overhead

A preempted job's state has to be saved before and restored after the interruption. This can be implemented using a stack. Regardless of the concrete implementation, the management of the preemption induces overhead. The easiest (but also most pessimistic) way is the add-up of the worst-case preemption overhead to WCETs, yielding new, increased WCET values.

Security

When running task sets of mixed criticality on one and the same hardware, a strict isolation of critical tasks is desirable. Unfortunately, when using fixed-priority schedulers, see 1.4.1, in combination with resource blocking, see Subsection 1.5.1, critical-task behavior can be deduced from the execution order of lower-priority tasks as shown in [271]. This is a *side-channel attack* where the system is no longer considered as a black, but as a gray[18] box [193]. This form of attack is of gaining popularity and has to be fought in terms of security.

When analyzing the security of mixed-criticality systems, the (max,+) algebra [23] can be used as suggested in [279]. This work uses clearance levels and models the data flow by a communication graph.

1.5 Resource Usage Constraints

The CPU is by far the most important resource of a digital computer. But other resources like input and output devices should not be ignored in a model of sufficient generality. Today's prominent devices are networks with the Internet as their most comprehensive form, hard disks and solid-state drives (SSDs)[19]. But even in programming, a lock on a data object can be regarded as a resource.

It is a challenge in resource scheduling to take into account that resources are non-preemptable (see 1.4.2) in many cases. *Resource requirement graphs* are a convenient means to specify the resource requirements of tasks or jobs by

[18]The metaphor expresses that side-channel data like energy consumption or timing are allowed to be taken into account.

[19]For an analysis of the real-time capabilities of flash media in general, see, e.g., the study [239].

a bipartite graph with tasks and resources as nodes and resource requests by edges, cf. [190], p. 283f. Such a type of graph has to be strictly distinguished from a *resource graph* describing the physical configuration of resources and the costs for communication, see [190], p. 52.

1.5.1 Blocking and Priority Inversion

When requiring one and the same resource at once, a lower-priority job can *block* a higher-priority job, see Subsection 1.4.1. This precedence is an inversion of the normal priorities. Hence, it is known as *priority inversion*. An often unwanted phenomenon in this context is *uncontrolled priority inversion* where its duration is unbounded from above due to a preemption of the low-priority blocking job by a medium-priority job. The most famous accident of uncontrolled priority inversion is the Mars Pathfinder mission [249].

Important approaches restricting priority inversion are Non-preemptive Critical Sections (NPCS) [212], the Priority Inheritance Protocol (PIP) [255] and the Priority Ceiling Protocol (PCP) [255]. PIP and PCP are less pessimistic than NPCS.

However, this reduced pessimism comes at the price of an increased complexity of the algorithms. Additionally, PIP is prone to deadlocks and does not prevent a job from being blocked several times, cf. [190], p. 289. Stack Resource Policy (SRP) [24] is an improvement of PCP since it supports a clear *Separation of Concerns* (SoC) by distinguishing preemption level from priority, multi-unit resources, an early blocking of jobs on arrival (dynamic priority scheduling) and a stack sharing [24]. For more details on these protocols, see, e.g., Chapter 7 in [65].

1.6 Multiprocessor

1.6.1 The Number of Processing Units, Multi- and Many-cores

The multiprocessor shall consist of m processors. Starting from $m = 2$, we use the term multiprocessor or, equivalently, *multicore* system. A multicore system differs from a multiprocessor system in a tighter integration of the m

processing units, e.g., regarding the cache levels. Since our abstraction level does not consider complicated cache effects, cf. 1.1.2, and focuses on the pure number of processing units, both terms are used synonymously in the following.

Another related term is *manycore* system. It highlights a particular high number of involved cores. As a threshold, 16 or 32 is an appropriate choice since then, effects of traditional multicore approaches become shadowed by new ones due to the large number of cores. As an example, modern graphic cards follow the manycore paradigm. They serve as GPUs and contain dozens or even hundreds of shading units which are increasingly used for calculations traditionally handled by CPUs. This is also called *general-purpose computing on graphics processing units* (GPGPU), see [225] for a survey on this topic.

1.6.2 Identical, Homogeneous and Heterogeneous Multiprocessors

Multiprocessors can be classified according to the dependence of execution time of a job on the processor it is executed on.

Identical Multiprocessors

In the simplest situation, all task-processor pairs result in an only task-specific WCET e_i. From the hardware point of view, this means that all processors are identical.

Homogeneous Multiprocessors

If all processors are of the same type[20] but run at different speeds in the general case, then the WCETs e_i of the tasks are scaled by the speedup factor S_j of the processor according to (1.25).

$$e'_{ij} = e_i / S_j \tag{1.25}$$

Using column vectors and matrix notation, the matrix of (worst-case) execution times of task i on processor j is a $n \times m$ matrix defined by (1.26).

$$(e'_{ij}) := (e_i)({S_j}^{-1})^T \tag{1.26}$$

[20]The wording "homogeneous" clearly refers to this type coincidence, not to the speed.

Due to its construction, this matrix is collinear in the homogeneous multiprocessor case. Note that the rows of the matrix refer to the tasks and its columns correspond to the processors. In the terminology proposed in [55], these processors are called *uniform*.

Heterogeneous Multiprocessors

The most general situation occurs when we have processors not only of different speeds but also of different types. Such a situation can, e.g., occur with GPUs and CPUs mixed in one and the same system. In the terminology of [55], these processors are called *unrelated*. Błażewicz even considers a fourth group of multiprocessors with *dedicated* processors.

> "Unrelated processors, on the contrary, are specialized in the sense that they prefer certain types of tasks for example numerical computations, logical programs or simulation procedures, since the processors have different instruction lists. Of course, they can process tasks of each type, but at the expense of longer processing time. A different type are the dedicated processors which may process only certain types of tasks.", [55], p. 6

Such a complication is not necessary since the execution time of a task i on processor j not being dedicated to it can easily be set to *infinity* by $e_{ij} = +\infty$, expressing the impossibility of allocating certain tasks to certain processors.

A Venn diagram summing up the different processor types and their alternative names is given in Figure 1.11. Note that a hierarchical structure was chosen, embedding more specific multiprocessor models in more general ones. At first glance, this seems to violate the dualism suggested by the wording "hetero" and "homo". But the apparent conflict can be resolved by accepting that heterogeneity refers to a potential one.

1.6.3 Priorities on MPs

With a *global* view on the system, cf. 1.6.7, the prioritization concept known from uniprocessors, see 1.4.1, can be extended in a straightforward manner to multiprocessors. Since there are now $m \geq 2$ processing units available, the m highest-priority tasks are selected for execution. This general approach

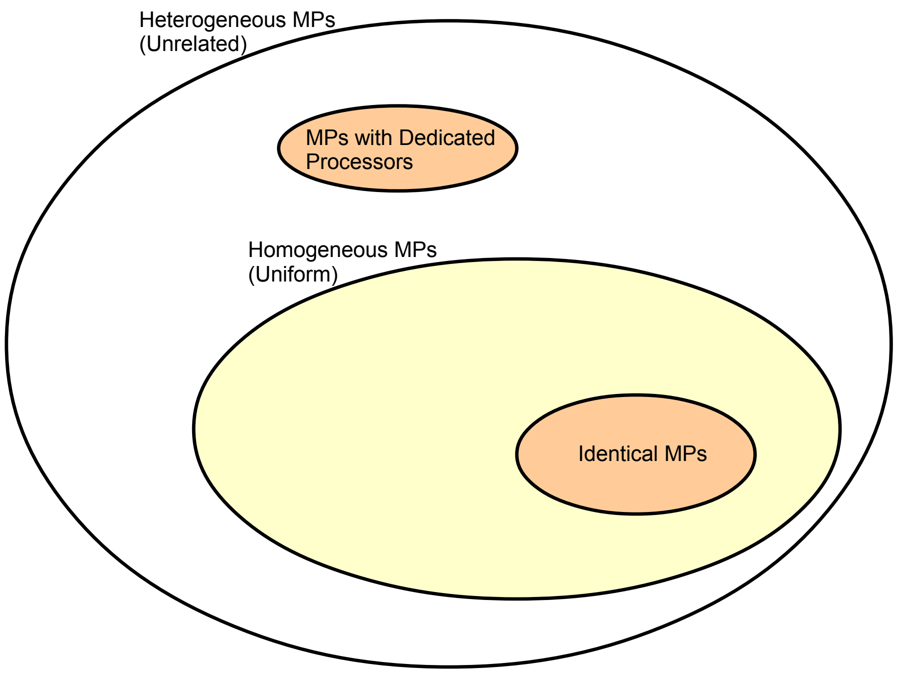

Figure 1.11: Heterogeneous, Homogeneous and Identical Multiprocessors

extends the basic uniprocessor prioritization and embeds it as a special case with $m = 1$.

1.6.4 Migration

A *migration* of a job belonging to a task is characterized by its preemption and a later resume of the preempted job on a different than the original processor [190], p. 65. The easiest strategy is to use the current rank number of each job in the priority list directly for mapping the jobs to the processors. This simple-to-implement approach usually results in a high number of migrations. Hence, more advanced schemes include a reordering of the priority queue in order to reduce the number of migrations.

Due to the necessary inter-processor (or -core) data transfers, migrations are considered slower than preemptions. But on modern architectures, there is only a marginal gap remaining, as recent studies show [41]. Bastoni *et al.* used a 24-core Intel Xeon processor with a three-level cache hierarchy and a real-time Linux running on it. They have shown "[..]that there is no substantial difference between preemption and migration costs in a system under load." [41]

The overhead of migrations can be included in WCETs as it has been suggested for preemption overheads in 1.4.2.

1.6.5 Utilization on MPs

Sometimes, it is useful to consider a related concept to the utilization. Since the maximum utilization on a multiprocessor system with m processors is m, a normalization can be regarded. Then, we obtain the system utilization u_{sys} defined in (1.27).

$$u_{sys} := \frac{u}{m} = \frac{\sum_{i=1}^{n} \frac{e_i}{p_i}}{m} \tag{1.27}$$

1.6.6 WCRTs on MPs under Resource Conflicts

Resource conflicts are typically regarded to increase response times due to blocking, see Subsection 1.5 and, thus, to decrease schedulability. But, as recently shown by Kollmann *et al.* [167], resource competition can also lead to a *decrease* in worst-case response times (WCRTs). This happens due to a competition for a resource which reduces peak load by limiting event streams. Conventionally, WCRTs are calculated assuming dense periodic job releases which is no longer appropriate in a resource-coupled system. A holistic analysis is, thus, preferable on multiprocessor systems [167].

1.6.7 Global, Partitioned and Semi-partitioned Scheduling

According to the allowance of migrations, the two extremes global scheduling with migrations allowed and partitioned scheduling with banned migrations are distinguished. As a hybrid solution, additionally semi-partitioned scheduling with both global and partitioned tasks is considered.

Global Scheduling

Global scheduling uses a *common* job queue where migrations are allowed. This is typically managed by the principles presented in 1.6.3 and 1.6.4. A possible restriction of minor practical use is to allow migrations only at job boundaries, cf. [38]. This is called *restricted migration* compared to full migration. Note the analogy to dynamic priorities fixed on job level described in 1.4.1.

According to the scheme introduced in [70], restricted migration makes its own category besides full migration and partitioned approaches. But restricted migration is less popular than the upcoming hybrid scheduling approaches to be discussed in Chapter 5 with semi-partitioned scheduling (see below) as an exponent.

Partitioned Scheduling

With partitioned scheduling, there are per-processor job queues used. Migrations are completely excluded. Then, the remaining problem is the static allocation of tasks to processors.

Semi-Partitioned Scheduling

A combination of global and partitioned scheduling is obtained by a subdivision of tasks into partitioned scheduled ones and globally scheduled ones. Often, there is additionally *task splitting* allowed in order to fill remaining capacities better.

Partitioned scheduling suffers from *external fragmentation*[21], i.e., “[..]as tasks get assigned to processors, the remaining available capacity gets fragmented among the processors.” [14] Hence, the size of the greatest chunk of available processor capacity constrains the task-to-processor assignment much more than it would be the situation without fragmentation. According to [14], “[t]ask splitting circumvents this problem by permitting that a task be split across multiple processors.”

[21]The complement is *internal* fragmentation well-known from main memory allocation. There, wasted memory is contained *within* allocated regions. This kind of waste happens due to the fact that memory typically can only be accessed in the form of memory blocks of a fixed (minimum) size.

Note that such a task splitting breaks the basic model of a job as an execution unit and might be semantically dangerous. On the other hand, this objection is as well appropriate for global scheduling with full migration. The difference is that with semi-partitioned scheduling, the migration points are predetermined inside a job due to the task splitting. Using global scheduling with full migration, they can occur everywhere in a job. So, this argument is attenuated since a fixed point of splitting a job is semantically better to control than an arbitrary one. Semi-partitioned scheduling belongs to hybrid scheduling approaches which will be regarded in more detail in Chapter 5.

1.6.8 Visualization of Schedules by Gantt Charts

The *Gantt chart* was proposed by Karol Adamiecki under the name *harmonogram* in Polish as early as in 1896, cf. [213], p. 7, but reached world-wide popularity only after publication in English by Henry L. Gantt in 1910, see [118].

Gantt charts is a useful tool in project management since it reflects the timely progress of a project and its elements which are called jobs. The start and completion times of the jobs are visualized by horizontal bars of width proportional to a job's execution time.

For a uniprocessor system, there is only one useful type of schedule, the processor-to-job mapping. A three-task example of priority-driven scheduling with preemptions is given in Figure 1.12 . But for a multiprocessor system, it is both useful to consider time-dependent job-to-processor and processor-to-job mappings. An example of these two variants is given in Figure 1.13.

1.7 Schedulability, Optimality and Dominance

1.7.1 Schedulability and Schedulability Tests

A task set is *schedulable* by a particular scheduling algorithm if the schedule obtained using this algorithm is feasible, cf. Subsection 1.1.3.

The positive outcome of a *sufficient* schedulability test implies schedulability. On the other hand, a negative outcome of a sufficient test yields no valuable result. Conversely, the negative outcome of a *necessary* schedula-

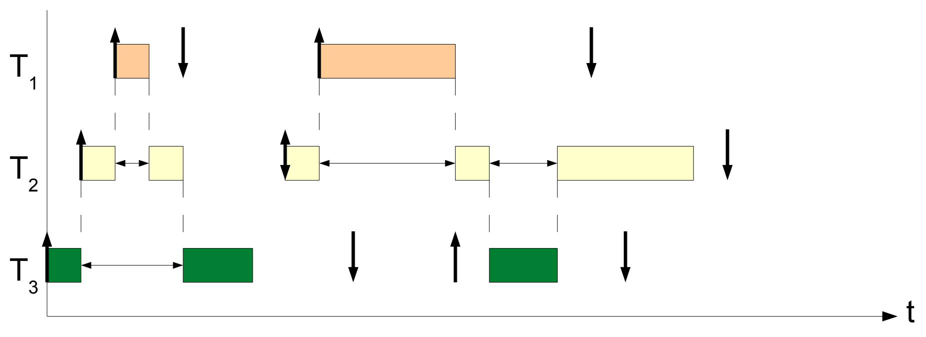

Figure 1.12: Schedule (Gantt Diagram) of 3 Tasks With Preemptions on a Uniprocessor

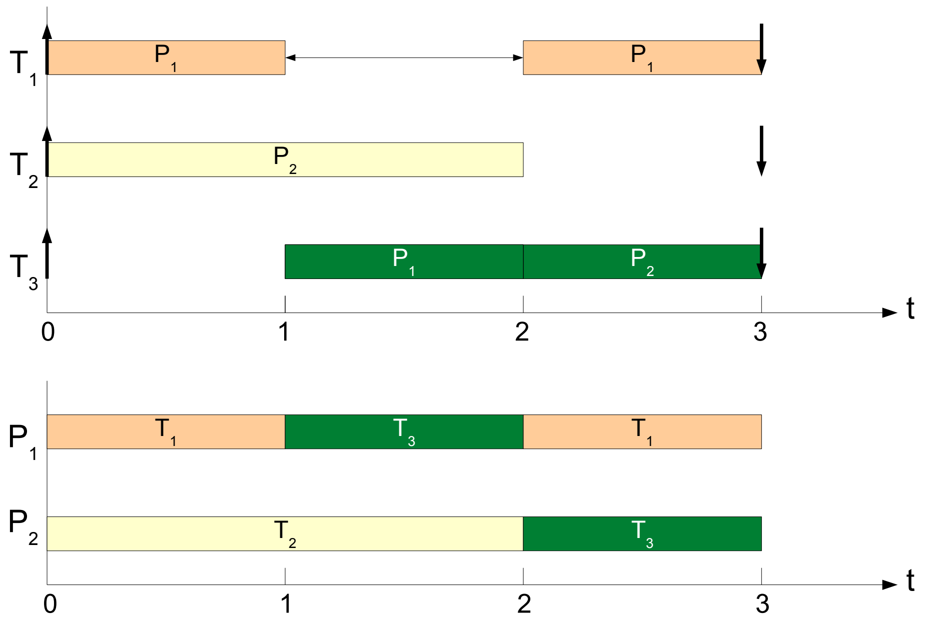

Figure 1.13: Schedule (Gantt Diagram) of 3 Jobs With a Migration of T_3 at Time 2 on a Multiprocessor System With 2 Processors: Job-to-Processor Mapping (Upper Part) and Processor-to-Job Mapping (Lower Part)

bility test implies non-schedulability. But a positive outcome of a necessary test leaves the schedulability question open. A test being both necessary and sufficient is called an *exact* one.

1.7.2 Feasibility

A task set is feasible if there is a valid schedule providing that all deadlines are met, cf. Subsection 1.1.3. Note that there are algorithms for which exist *feasible* task sets not being *schedulable* by these algorithms. This fact highlights the difference between schedulability which refers to a particular scheduling algorithm, and feasibility which refers to the existence of a feasible schedule, completely ignoring the question for the method to obtain such a schedule.

1.7.3 Sensitivity

A quantification of a sufficient schedulability test is given by its *sensitivity*, the ratio between the number of task sets deemed schedulable and the number of task sets actually being schedulable. The best value is 1 corresponding to an exact test like TDA, cf. Subsection 2.2.1. The worst value is 0 describing a completely useless test. Alternatively, sensitivity is the proportion of the true positives among the really schedulable task sets which comprise true positives and false negatives. For a summary of this situation, see Figure 1.14.

Note that the parameter sensitivity is only well-defined and reproducible when referring to a set of (explicitly) given task sets or at least task sets following given parameter distributions, or, more practically, synthetic task sets with a given procedure of generating them. Without such a reference, the metrics becomes blurry and prone to biases.

Suitable procedures for generating appropriate synthetic task sets will be presented in Chapter 6.

1.7.4 Utilization Bounds

A very convenient and popular kind of a schedulability test is the utilization bound. It relies on the total utilization u of a task set and is in effect a sim-

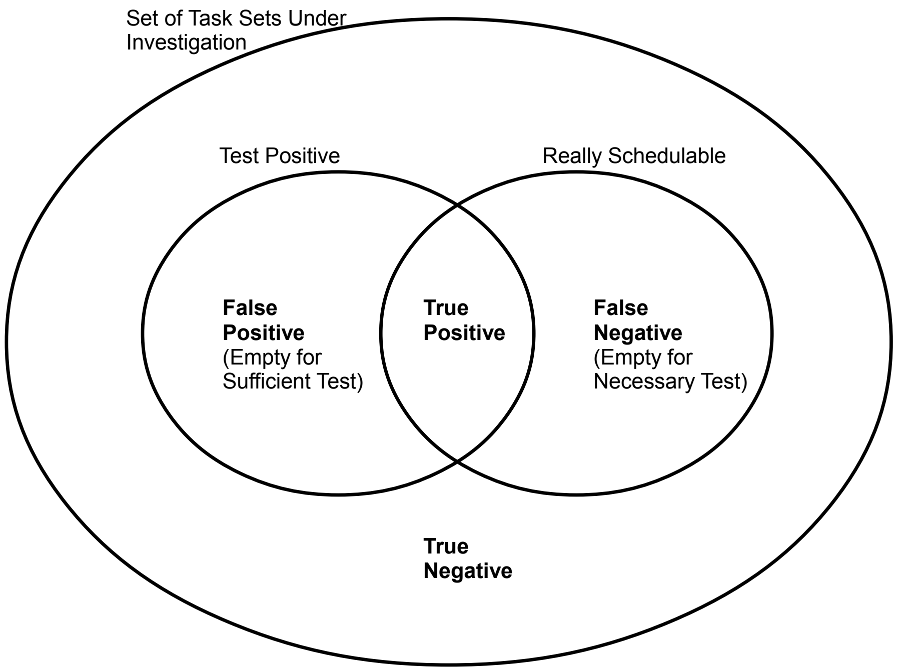
Set of Task Sets Under Investigation
Test Positive
Really Schedulable
False Positive (Empty for Sufficient Test)
True Positive
False Negative (Empty for Necessary Test)
True Negative

Figure 1.14: Necessary and Sufficient Schedulability Tests

ple comparison with a precalculated[22] constant u_b meaning *bound*, see (1.28). Typically, it is the ultimate goal to determine a *tight* upper bound. For sufficient schedulability tests, it is called *least upper bound* u_{lub} or *exact utilization bound*[23]. This u_{lub} is the maximum[24] we can get and, thus, the most useful among all utilization-bound-based sufficient schedulability tests for the particular problem (1.29). See also Figure 1.15 for summing up the concept of utilization bound and its relationship to sufficient and necessary schedulability tests.

[22]Note that finding and proving such a constant can be very complicated. This type of problems belongs to the very core of traditional real-time scheduling theory.

[23]Note that, in general, this does not imply an exact schedulability test. But it is rather an exception that the exact discrimination into schedulable and non-schedulable task sets can be performed solely taking the total utilization value into account.

[24]At first glance, this might be confusing. The “least” in *least upper bound* refers to the minimization of the total utilization u of task sets which *fully utilize* the processor. A schedulable task set fully utilizes the processor if any increase of an e_i by an $\epsilon > 0$ of any task T_i results in a deadline violation, cf. [189]. In effect, the utilization minimization of the dangerous task sets and the utilization maximization of the safe task sets meet exactly at u_{lub}.

$$u \leq u_b \tag{1.28}$$

$$u \leq u_{lub} := \max u_b \tag{1.29}$$

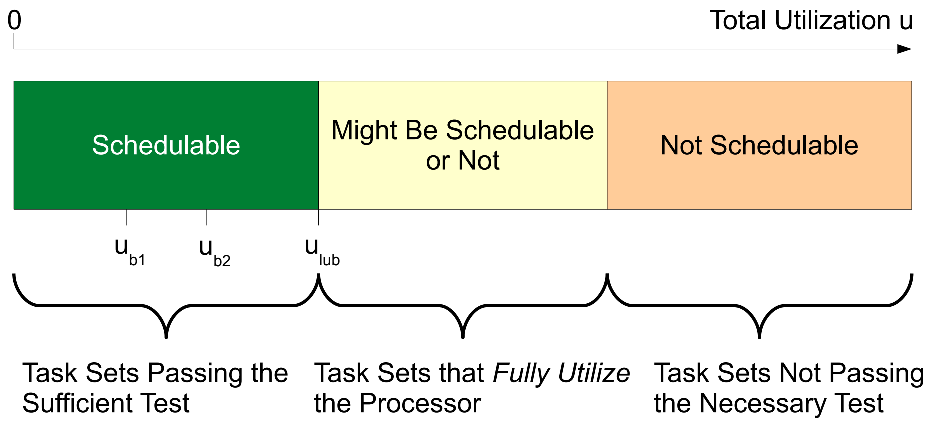
0
Total Utilization u
Schedulable
Might Be Schedulable or Not
Not Schedulable
u_{b1}
u_{b2}
u_{lub}
Task Sets Passing the Sufficient Test
Task Sets that *Fully Utilize* the Processor
Task Sets Not Passing the Necessary Test

Figure 1.15: Utilization Bound u_b and Least Upper Bound u_{lub}, aka Exact Utilization Bound

The approach has linear computational complexity, $\mathcal{O}(n)$, when applied offline and only constant complexity, $\mathcal{O}(1)$, when used for online admission control of tasks. The linear complexity has its origin in summing up the task utilizations to the total utilization[25]. Admission control checks the conformance to a utilization bound by adding up the new task's utilization to the old total utilization value. This update operation requires only constant complexity.

For a sufficient schedulability test for a particular scheduling algorithm, the bound is typically an upper bound. Up to a total utilization of this bound, all task sets are schedulable by the specified algorithm. For a necessary schedulability test, the bound is typically a lower bound. Beyond this total utilization, there is no task schedulable by the specified algorithm.

Besides their application for schedulability tests, utilization bounds allow for an schedulability analysis [197]. The impact of parameters like number of

[25]Interestingly, the same complexity classes hold for the Hyperbolic Bound, see 2.2.2, which uses a *product* of incremented utilizations. The quest for further schedulability tests based on products is a challenge.

tasks n, maximum task utilization u_{max}, number of processors m (in a multiprocessor setting), or the period configuration $\{p_i | i = 1..n\}$ (see Chapter 7) can be studied. A utilization bound does not only give a binary decision on schedulability, but it also *quantifies* how far the configuration is from being optimal [197] where the total utilization equals the number of processors, $u = m$, see (3.1).

As a generalization, so-called *non-utilization bounds* were introduced in [192]. Since their use is restricted to aperiodic tasks, utilization bounds remain the most important schedulability tests, see also Subsection 8.8.2.

A general framework for the derivation of utilization bounds was recently given in [284]. Since this framework relies on *network calculus* similar to *RT calculus*, see Subsection 1.2.4, it is very general and powerful enough to reprove several known utilization bounds.

1.7.5 Optimality

A scheduling algorithm is called *optimal* referring to a set of task sets iff it can schedule all task sets in the set which are schedulable by any algorithm. This means for an optimal algorithm A, there is no difference between A-schedulable and feasible task sets. To find optimal scheduling algorithms is one of the central goals in scheduling theory.

Note that the corresponding concept for schedulability tests ist exactness. An exact schedulability test filters exactly the schedulable task sets, cf. Subsection 1.7.1, there are neither false positives nor false negatives.

1.7.6 Dominance

There are three important binary relationships between scheduling algorithms. Algorithm A *dominates* algorithm B iff schedulability of a task set by algorithm B implies schedulability by algorithm A. Two algorithms are *equivalent* iff they dominate each other. Algorithm A *strictly dominates* algorithm B iff A dominates B and A is not equivalent to B.

Since the universal quantifier leads only to a *partial order* induced by the dominance relationship, there is a third case of two algorithms being *incom-*

parable. Then, there is a task set schedulable by A, but not by B and one schedulable by B, but not by A.

A summary of relationships between scheduling algorithms using a Venn diagram is given in Figure 1.16.

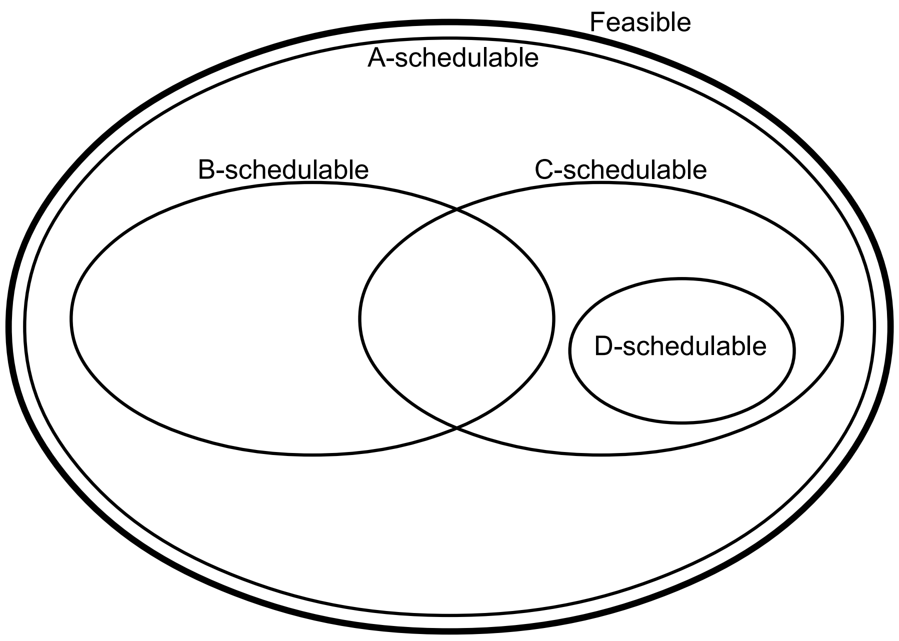

Figure 1.16: Optimality, Dominance and Incomparability of Scheduling Algorithms (here called A, B, C and D) Operating on a Set of Feasible Task Sets: A is Optimal, C Dominates D, B and C are Incomparable

This framework of terms can also be applied to schedulability tests in an analogous way. There, “schedulable” has to be replaced by “deemed to be schedulable”.

1.7.7 Approximation Ratio

There are non-optimal algorithms. They often use heuristics. A means of quantifying the goodness of such algorithms is the *approximation ratio.* This is the ratio of the approximated solution to the exact solution. For minimization problems being typical in our context, the definition of the approximation ratio $R(w)$ is (1.30) where w denotes the *cost* of a solution.

$$R(w) := \frac{w}{w_{\mathrm{opt}}} \tag{1.30}$$

Thus, the best approximation ratio is 1. The greater the $R(w)$ value, the worse. As an example, we give the Traveling Salesman Problem (TSP). Here the *minimum spanning tree heuristics* has an approximation ratio of 2, but the *Christofides heuristics* [77] achieves 1.5 and is, thus, better.

The *asymptotic* performance ratio is the limit of $R(w)$ as the problem size approaches infinity. This metric is well-defined, but sometimes criticized for only having theoretic value. A reason is that the optimal value might be hard to or even impossible to access. Next, a heuristics can behave well for large problem sizes yielding a very good approximation ratio.

1.8 Sustainability

By an intuitive argument, it is often assumed that changing a schedulable task set to another one with *weaker* requirements keeps its schedulability. In [34], the concept is defined and applied to uniprocessor scheduling. Only very rarely, task parameters are known exactly at design time. Sustainability enables that task parameters can be interpreted as bounds instead of exact values which is much more appropriate to real-world situations. Hence, sustainability is a fundamental concept to be considered in real-time system design.

The following task set modifications are typical ones:

Decreasing execution times. Reducing the execution time is the fundamental process behind the notion of WCETs, cf. 1.1.2. Thus, sustainability in terms of execution time reduction is very important.

Increasing periods/minimum interarrival times. Reducing the frequency of releases (or its maximum) is a typical relaxation.

Increasing relative deadlines. Increased relative deadlines result in later absolute deadlines. Already transitivity of time seems to ensure this type of sustainability.

The limits of sustainability become apparent when considering asynchronous task sets with non-zero phase values, cf. (1.10). The most important uniprocessor schedulig algorithms RMS and EDF are not sustainable with respect to incrasing periods for asynchronous task sets, cf. [34].

Global multiprocessor scheduling is prone to unsustainability. While both global EDF and global RMS are sustainable with respect to decreased execution times, global RMS is not sustainable with respect to increased periods as shown in [15] by counter-example.

1.8.1 Schedulability Test Sustainability

Because of the unsustainability of some global multiprocessor scheduling policies, the concept of sustainability was applied to (sufficient) schedulability tests in [28]. There, Baker and Baruah distinguish into non-sustainable, sustainable and self-sustainable schedulability tests. While a self-sustainable test needs to deem the modified task set schedulable by itself, sustainable tests are allowed to prove schedulability of the resulting task sets by some other means.

Sufficient schedulability tests for global EDF are particularly prone to non-sustainability. Test sustainability can be violated both with respect to decreased execution times and with respect to increased relative deadlines [28] which clearly shows that sustainability cannot be taken as a matter of course.

1.8.2 Accelerated Task Sets

Accelerated task sets are based on the results of the inverse operation of the second typical task transformation mentioned above. Thus, they are characterized by *decreased* periods or minimum interarrival times, cf. [216].

Once having established period sustainability for a task set under a scheduling algorithm, accelerated task sets can, thus, be used to prove schedulability of the original task set, i.e., that one with the weaker frequency conditions.

Besides its use for schedulability proofs, task sets with a requirement of maximum periods can be accelerated and then be scheduled in this transformed manner. An application of such an approach to a radar system is discussed in [256].

1.9 Models Used in this Thesis and Structure

1.9.1 Models Used

We are now in the position to specify the model mainly to be used in this thesis. For short, it is based on the Liu/Layland model [189]. Refining this, it is characterized by *preemptive scheduling of periodic, synchronous, independent tasks with implicit, hard or soft deadlines without any self-suspension on identical multiprocessors*. The memory hierarchy is assumed to be degenerated to a flat one, i.e., we do not consider cache and swapping effects. Preemption and migration overheads are assumed to be integrated in the WCETs. The same is assumed for *communication* overheads. For a survey on real-time communication, see, e.g., [218]. Finally, even the scheduler *itself* needs processor time. The cleanest solution to this is to run the scheduler on an extra dedicated processor. But often, this is not a viable option. Then, for running the scheduler on the platform where the actual workload is processed, WCETs have to be enlarged in order to account for this overhead. The only two remaining parameters of a task T_i in this model are its period p_i and its WCET e_i. This shall be denoted by $T_i(p_i, e_i)$ in the following.

Further, *priority-based* and *partitioned* scheduling is in the main focus of this thesis.

1.9.2 Structure of the Thesis

After having introduced basic terms and concepts in this Chapter, Part I will be completed with Chapter 2 on uniprocessor real-time scheduling. The famous real-time scheduling algorithms EDF and RMS are discussed there. The focus lies on sufficient schedulability tests.

An overview of the state-of-the-art in multiprocessor real-time scheduling including global (Chapter 3), partitioned (Chapter 4) and hybrid (Chapter 5) approaches is given in Part II. It will be completed by Chapter 6 on the appropriate generation of synthetic task sets. Such methods are required for a correct and fair evaluation of real-time scheduling approaches. Their impact was often underestimated. It will be shown that the generation procedures of synthetic task sets have an impact on schedulability statements.

Part III contains new results in multprocessor scheduling. Partitioned rate-monotonic scheduling (RMS) with hard deadlines will be discussed in Chapter 7. There, the Rate-Monotonic Small Tasks (RMST) algorithm is improved at several stages. The most innovative one is the change from a *linear* to a *circular* similarity measure when describing period similarity.

Partitioned Earliest Deadline First (EDF) scheduling is investigated in Chapter 8. The total utilization of a task set will turn out to be a convenient parameter for estimating the schedulability property of the task set. EDF schedulability has a sharp threshold behavior when controlled by the total utilization. Here, thresholds act as a generalization of utiliation bounds. Both the proof of existence of sharp thresholds and their determination are goals of this Chapter.

Part IV will provide a comprehensive discussion and an outlook.

In Appendix A, an introduction to the fascinating field of circular statistics needed for Chapter 7 is given. This might be interesting and relevant for readers not yet familiar with it.

Chapter 2

Uniprocessor Scheduling

2.1 Necessary Schedulability Tests

An obvious necessary condition for feasibility of a task set on a uniprocessor is a maximum total utilization of 1, cf. (2.1). Otherwise the load demand cannot be covered by CPU time since using the entire CPU leaving no idle time is the maximum possible.

$$u \leq 1 = 100\% \tag{2.1}$$

2.2 Static Priorities and RMS for Implicit Deadlines

Rate-monotonic scheduling (RMS) is a static priority scheduling policy with priorities assigned the higher the shorter the period p_i (or, equivalently, the higher the rate) of a periodic or sporadic task. It was shown to be optimal for implicit-deadline task sets [189].

Deadline-monotonic. For the generalization to tasks with constrained deadlines, the changeover to the more general deadline-monotonic (DM) priority assignment is optimal [185]. DM assigns priorities the higher the shorter the relative deadline D_i of a task is, see Section 2.3 for more details. In spite of the similar naming, DM is completely different to the task-level dynamic scheduling approach EDF, see Section 2.4. DM must not be confused with EDF.

Arbitrary deadlines. In the general case of arbitrary deadlines, or with a relaxation to allowing asynchronous task sets, neither RM nor DM priority assignments are optimal [185]. Instead, the more complex algorithm of Audsley [19] [20], a greedy algorithm of $\mathcal{O}(n^2)$ complexity[1], is required to achieve an optimal priority assignment often referred to as *Audsley's optimal priority assignment algorithm* (OPA).

Further strategies. Other strategies which are of less importance in the uniprocessor case are slack-monotonic scheduling (SMS) with a utilization bound of 50% [12] and the adaptive TkC priority assignment, see Subsection 3.3.2 and [15], which was designed for global fixed-priority scheduling. Note that TkC reduces to RM in the uniprocessor case since the adaptive formula (2.2) calculating an optimal k dependent upon the number of processors m gives $k = 0$ for $m = 1$. The generalization DkC, see Subsection 3.3.7, for constrained-deadline task sets [84] analogously collapses to DM in the uniprocessor case.

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m} \tag{2.2}$$

This shows that ingenious multiprocessor approaches sometimes have the power to integrate uniprocessor ones and can be helpful in understanding established uniprocessor concepts. For a general discussion of the TkC approach, see Subsection 3.3.1.

In the following, we will consider exact (Subsection 2.2.1) and sufficient (Subsection 2.2.2) RMS schedulability tests. For a discussion on the probabilistic concept of a threshold utilization (to be presented in Chapter 8), applied to uniprocessor RMS, see Section 8.6.

2.2.1 Exact Schedulability Tests

There has been a challenge for a long time: to find an exact schedulability test for static-priority real-time scheduling on a uniprocessor.

[1]Note that this is much faster than the brute-force approach of priority $n!$ permutations.

Constructing the Schedule

An obvious approach suggested early is the check for deadline violations in the hyperperiod H. If there are no violations in it, there will be no violations at all since the hyperperiod is the major cycle in a synchronous task system. In the general case of an asynchronous system, the check has to be extended to $2H + \max_{i=1}^{n} p_i$ as shown in [184].

This easy constructive approach has the drawback that it is NP-Hard [184] and can take a long time since, in the worst-case of all periods being co-prime[2], the hyperperiod is the product of all periods.

Further, the approach of schedule construction is only possible for completely specified instances or proven worst-case instances like maximum execution time (the WCET) in the Liu/Layland task model or minimum interarrival time (as minimum period) in the sporadic task model, both on uniprocessor. The property exploited here is sustainability, see Section 1.8, with respect to decreased execution times and increased periods for fixed-priority preemptive scheduling on uniprocessor [34].

Lehoczky's Test

Lehoczky *et al.* [181] found that an analysis of the *demand* on the processor gives an exact characterization of the exact RMS schedulability of task i, too. The condition is (2.3).

$$\exists t, 0 < t \leq D_i = p_i : e_i + \sum_{Prio_j > Prio_i} \left\lceil \frac{t}{p_j} \right\rceil e_j \leq t \tag{2.3}$$

Here, only multiples of the task periods, aka as *scheduling points*, need to be tested [181]. By applying the universal quantifier $\forall i$ on (2.3), an exact RMS schedulability test for a task set is obtained.

Equation (2.3) sums over all higher priority tasks their WCET times the maximum number of jobs which can preempt the task i up to time t. This *interference* term is also called a (per-task) *time-demand function*[3] or, according to [108], *request-bound function* (RBF). The RBF is the maximum cumulative

[2]This means that their gcd equals 1.

[3]Sometimes, the term *time-demand function* is used for the entire LHS of (2.3). In order to point out this difference, the attributes *per-task* and *cumulative* can be used, cf. [108].

execution requirement of all jobs belonging to T_i with their release times in the interval $[0, t]$, see (2.4). It has a rise of e_i and a going of p_i. For an example of an RBF, see Figure 2.6.

$$RBF(T_i, t) := \left\lceil \frac{t}{p_i} \right\rceil e_i \tag{2.4}$$

Time-Demand Analysis (TDA)

A further improvement according to the effort needed was independently found by Joseph and Pandya [147] and Audsley *et al.* [21]. Here, the result is even the exact worst-case response times (WCRTs), see 1.1.2, of all tasks. They can be compared with the relative deadlines, all response times must be no greater than the respective deadline. This finally gives the schedulability result. The formula (2.5) works in an iterative manner with the WCET e_i as a start value for $WCRT_i$, i.e., $WCRT_{i,0} = e_i$.

$$WCRT_{i,k+1} = e_i + \sum_{Prio_j > Prio_i} \left\lceil \frac{WCRT_{i,k}}{p_j} \right\rceil e_j \tag{2.5}$$

If static-priority schedulability is the problem, the iteration stops as soon as (i) a fixed point not beyond the deadline D_i is reached or (ii) a value beyond the deadline occurs. The equation (2.5) is sometimes called Time-Demand Analysis (TDA), cf. [190], p. 134ff. Note that TDA in this popular form makes use of the general approach of busy windows, see also Subsection 1.2.4.

If we continue with the iteration (2.5) until a fixed point is reached ignoring the deadline D_i which is a requirement, the WCRTs are obtained. Applied in this form, the TDA is also known as response time analysis (RTA).

Note that the TDA approach can be generalized to global multiprocessor scheduling, see Subsection 3.3.5.

Hyperplanes Exact Test (HET)

The *hyperplanes exact test* was proposed by Bini and Buttazzo in 2004 [52]. There, the TDA condition (2.3) is rewritten in a more expressive form as several inequalities first ORed and then ANDed. By each inequality, a half-hyperspace at one side of a hyperplane is selected. This gives a polytope of complex nature as schedulability region [52]. HET reduces the set of scheduling points used for (2.3) and is, thus, an improvement in terms of effort.

On the other hand, according to [210], it turns out to be slower than the iteration method (2.5). So, in spite of its conceptual elegance[4], its application might be restricted.

Complexity

Recently [96], it was shown that the response time calculation is NP-Hard. But the problem of schedulability with only a binary outcome might be easier. The presented exact tests are all of pseudo-polynomial complexity.

Thus, the exact tests may take a too long time. Hence, it is reasonable to develop sufficient schedulability tests of polynomial complexity.

Exact Tests for Two Tasks

For the special case of $n = 2$ tasks and, w.l.o.g. $p_1 \leq p_2$, an explicit formula was developed [261]. It is sufficient to combine only two terms by a logical OR operator (2.6). This can be equivalently expressed using a maximum operation [92], see (2.7).

$$\left\lfloor \frac{p_2}{p_1} \right\rfloor (p_1 - e_1) \geq e_2 \vee p_2 \geq \left\lceil \frac{p_2}{p_1} \right\rceil e_1 + e_2 \tag{2.6}$$

$$e_2 \leq \left\lfloor \frac{p_2}{p_1} \right\rfloor (p_1 - e_1) + \max\left(0, p_2 - \left\lfloor \frac{p_2}{p_1} \right\rfloor p_1 - e_1\right) \tag{2.7}$$

It is matter of taste which form to prefer. These formulas can also be obtained or verified by the sufficient test DCT which is exact for $n = 2$, see 2.2.2 and [216]. A common drawback of equations (2.6) and (2.7) is their case-by-case structure.

By considering the gaps of length $p_1 - e_1$ in the schedule of the higher priority task one can obtain a WCRT term and finally criterion (2.8) by comparing this term with the relative deadline $p_i = D_i$.

$$e_2 \leq p_2 - \left\lceil \frac{e_2}{p_1 - e_1} \right\rceil e_1 \tag{2.8}$$

An advantage of (2.8) is its compact, easy-to-remember form. On the other hand, (2.8) is an implicit inequality rendering it unsuitable for a visualization

[4] In [52], there is also proposed a sufficient RMS schedulability test called δ-HET. It can be tuned by the parameter δ in order to find a balance between sensitivity and complexity.

of the schedulability region. Due to the ceiling operator, a separation of either e_1 or e_2 is impossible without a case-by-case analysis. With a distinction of 2 cases, condition (2.6) can be derived from (2.8).

The most interesting point in the schedulability region is the intersection of the two straight lines given in (2.6) since this point corresponds to a task set with u_{lub} which fully utilizes the processor, cf. Subsection 1.7.4. By equalizing the two parts in (2.6), we obtain (2.9) for this worst case.

$$\left(\left\lceil \frac{p_2}{p_1} \right\rceil - \left\lfloor \frac{p_2}{p_1} \right\rfloor\right) e_1 = p_2 - \left\lfloor \frac{p_2}{p_1} \right\rfloor p_1 \tag{2.9}$$

Interestingly, the prefactor on the LHS of (2.9) is typically 1 since the difference of the ceiled and the floored value should be 1. But there is an exception. The value rounded up and rounded down coincides for integer numbers. In such a case, we have a simply periodic task set, cf. 1.2.1. Clearly, for this special case, the two straight lines obtained from (2.6) coincide being the $u = 1$ line. Here, there is no unique solution for an execution time e_1 leading to a task set fully utilizing the processor of minimum utilization since *all* $u = 1$ task sets meet this condition, cf. (2.14). But for all task sets not being simply periodic, $e_1 = p_2 - \lfloor p_2/p_1 \rfloor p_1$ represents such a particular task set. Alternatively, this equation can also be obtained by considering the limit case of value 0 at the RHS of (2.7).

Here, the subtrahend at the RHS is the accelerated period p_2' under pivot period p_1 according to the DCT Algorithm 2.2. This shows how fundamental ASPTSs being discussed in 2.2.2 are. Next, for periods of the same (binary) order of magnitude, the condition simplifies to $e_1 = p_2 - p_1$ which is one of the basic ideas of the task set transformation in the Liu/Layland proof, see 2.2.2 and [189].

For a better understanding, the RMS schedulability region of an example task is shown in Figure 2.1. Further explanations can be found, e.g., in [216].

2.2.2 Sufficient Schedulability Tests

Liu/Layland Bound

The Liu/Layland (LL) bound [189] is one of the most famous results in real-time scheduling. It is the exact utilization bound of general implicit-deadline

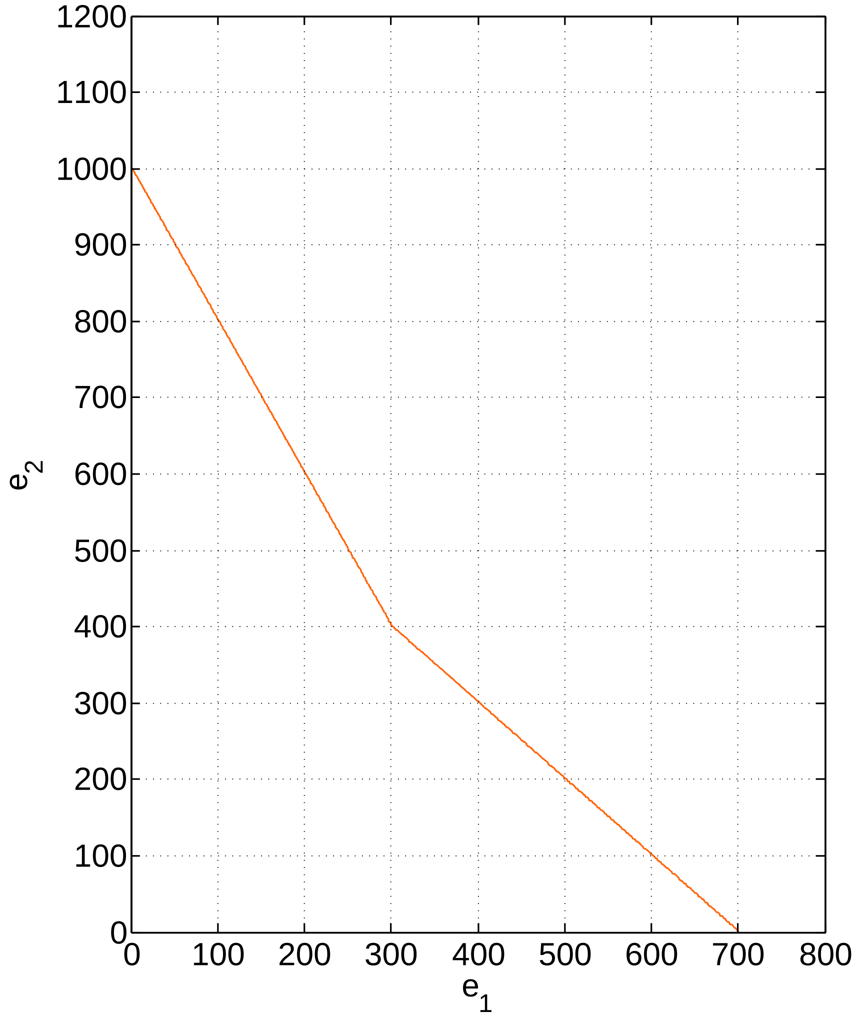

Figure 2.1: $e_1 e_2$ Plane With RMS Schedulability Region of the Task Set $\{(p_1 = 700, e_1), (p_2 = 1000, e_2)\}$ as a Contour Plot of the Boolean Schedulability Function (2.8); $(300, 400)$ is the point representing the Liu/Layland task set fully utilizing the processor of lowest total utilization according to (2.9).

preemptive task sets schedulable by RMS and can, thus, serve as a sufficient RMS schedulability test. The bound depends solely on the number of tasks n, see (2.10).

Another core result of Liu and Layland is that a *critical instant*, a point in time where a request for that task (a job) results in its largest response time, for fixed-priority scheduling (e.g. RMS) occurs when it is requested simultaneously with all higher-priority tasks (jobs) [189]. This justifies the consideration of a synchronous task set, see 1.2.1, as the worst-case configuration which is sufficient for obtaining a bound, and even allows to generalize the bound to the sporadic event model, cf. Subsection 1.2.2.

$$u \leq u_{LL}(n) = n(2^{1/n} - 1) \tag{2.10}$$

For the limiting process $n \to \infty$, the exact utilization bound for RMS independent of the number of tasks is obtained. This is correct since $u_{LL}(n)$ is strictly monotonic decreasing in n. The surprisingly simple non-parametric exact utilization bound obtained is (2.11). Note that (2.11) is strictly dominated by (2.10).

$$u \leq u_{LL}(\infty) = \ln 2 \tag{2.11}$$

The Hyperbolic Bound

The hyperbolic bound was published in 1995 by Oh and Son [233], but it became famous by the paper by Bini *et al.* [50] published six years later. The idea is to consider a product of utilization terms instead of a sum. The sufficient criterion is (2.12).

$$\prod_{i=1}^{n}(u_i + 1) \leq 2 \tag{2.12}$$

The hyperbolic bound (HB) strictly dominates the Liu/Layland bound (2.10) and, by transitivity, also (2.11). Since the proof that all task sets fulfilling (2.11) also meet the HB condition (2.12) is particularly elegant, it is given here.

$$\begin{aligned} u &\leq \ln 2 \\ \Leftrightarrow \sum_{i=1}^{n} u_i &\leq \ln 2 \\ \Rightarrow \sum_{i=1}^{n} \ln(u_i+1) &\leq \ln 2 \\ \Leftrightarrow \ln \prod_{i=1}^{n} (u_i+1) &\leq \ln 2 \\ \Leftrightarrow \prod_{i=1}^{n} (u_i+1) &\leq 2 \end{aligned}$$

The first step uses definition (1.12). Next the Taylor series at $(0,0)$ of $\ln(u_i+1)$ and its tangent equation is used for estimating the logarithmic term from above. Finally, the logarithm identity for products is applied and exponentiation to base e is performed.

Note also that in the special case of *all-equal* task utilizations in a task set, HB (2.12) reduces to LL (2.10) as shown below.

$$\begin{aligned} \prod_{i=1}^{n} (u_i+1) &\leq 2 && |u_i := u/n \\ \Rightarrow \left(\frac{u}{n}+1\right)^n &\leq 2 && |()^{1/n} \\ \Leftrightarrow \frac{u}{n}+1 &\leq 2^{1/n} && |-1 \\ \Leftrightarrow \frac{u}{n} &\leq 2^{1/n}-1 && |\cdot n \\ \Leftrightarrow u &\leq n\left(2^{1/n}-1\right) \end{aligned}$$

The homogeneity of the individual task utilizations is used in the first step. But this scenario is a worst case, the product is bounded from above under the constraint of constant total utilization u. The reason is the *Gaussian sum-factor rule* which says that a product is maximized for equal factors under the constraint of a constant sum of the factors[5]. So, we can reason from below upwards in the general case, fulfillment of the LL condition implies fulfillment of the HB condition, but not vice versa. This indicates that HB strictly dominates LL.

[5]The problem ist equivalent to maximizing the hypervolume of a hypercuboid subject to a constant sum of edge lengths.

The natural logarithm with its base e is not so popular in computer science and sometimes inconvenient. Thus, it is worth to try an alternative formulation with base 2, being much more common in computer science. By taking the binary logarithm of (2.12), we obtain equivalently (2.13).

$$\sum_{i=1}^{n} \mathrm{ld}(u_i + 1) \leq 1 \tag{2.13}$$

Binary logarithms of powers of 2 can be calculated mentally since they correspond just to their exponents. A simple example with two task of utilizations $u_1 = 1/2$ and $u_2 = 1/3$ shall show the usefulness of the approach. The LL criterion is not met since total utilization $u = u_1 + u_2 = 5/6 \approx 0.833$ is beyond the bound $u_{LL}(2) = 2(\sqrt{2} - 1) \approx 0.828$. But using (2.13) we obtain

$$\begin{aligned}
\mathrm{ld}(u_1 + 1) + \mathrm{ld}(u_2 + 1) &\leq 1 \\
\Leftrightarrow \mathrm{ld}(3/2) + \mathrm{ld}(4/3) &\leq 1 \\
\Leftrightarrow \mathrm{ld}(3) - \mathrm{ld}(2) + \mathrm{ld}(4) - \mathrm{ld}(3) &\leq 1 \\
\Leftrightarrow -1 + 2 &\leq 1 \\
\Leftrightarrow 1 &\leq 1
\end{aligned}$$

which is true. Hence, the task set is RMS-schedulable, and proving it can be done without a calculator. This shows clearly that a change of the logarithm base can be a powerful tool. We will come back to that later in Subsection 7.3.3.

LL-based Tests: Number of Chains or Roots

The basic idea of subsequent improvements of sufficient RMS schedulability tests in terms of increased sensitivity is to take subsets of tasks being simply periodic or close to simply periodic into account. In the extreme case, the entire task set is simply periodic. Then, a utilization of 1 can be achieved, cf. (2.14), and the bound u_{SPTS} for RMS of simply periodic task sets is 100%. The reason is that tasks in a simply periodic chain can be subsumed under a *pseudo task* without any waste.

$$u \leq u_{SPTS} = 1 \tag{2.14}$$

Periodic Chains as Pseudo Tasks. When having several simply periodic chains, each of them can be considered as a pseudo task. Hence, the number of tasks in (2.10) can be substituted by the number of such chains k, cf. (2.15), which equals the number of *fundamental frequencies.*

$$u \leq u_{LL}(k) = k(2^{1/k} - 1) \tag{2.15}$$

This can lead to an increased utilization bound since (2.10) is strictly monotonically decreasing in n, and $k \leq n$ holds in general. The periodic chains approach was first published in [171] by Kuo and Mok in 1997. Although it yields some improvement in particular cases, the discretization to LL bound values $(1, 0.828, 0.780, 0.757, \dots)$ remains a severe restriction.

Root-based Test. A next suggestion by Kuo *et al.* is the use of the minimum number of *roots* in the LL formula [170]. A *root period* is a task period with all longer task periods of tasks in the set *not* being an integer multiple of it. E.g., there are two simply periodic chains in Figure 1.8, but only one root. Each root belongs to a semi-harmonic subset. The minimum number of roots, thus, corresponds to the minimum number of semi-harmonic subsets covering the entire task set. An equivalent characterization of semi-harmonic task sets is that they have exactly one root.

A problem with this approach is that it has to be applied in an *iterative* way. It is necessary to ensure that the previous task set without the new task is RMS-schedulable[6]. This increases complexity of the test for an offline evaluation. On the other hand, iteration is well-suited for an online admission control. A shortcoming inherited from LL and the fundamental frequencies approach is the above mentioned discretization to LL bound values. These two drawbacks become apparent by regarding the task set $\{T_1(2,1), T_2(3,1), T_3(6,1)\}$. It is schedulable by RMS and has one root, the period 6. Thus, the utilization bound becomes 1 when ensuring that the subset $\{T_1(2,1), T_2(3,1)\}$ is RMS-schedulable. This subset has two roots and a utilization of $1/2 + 1/3 = 5/6 > u_{LL}(2) = 2(2^{1/2} - 1)$. Hence, only using Kuo's Root-based test, it is impossible to prove schedulability for the subset and,

[6]The reason is that the Root-based test implicitly assumes that always *only* the task with the longest period is prone to deadline misses. This assumption is wrong as shown in [237].

thus, for the entire task set. This example clearly reveals the shortcomings of the Root-based approach.

Burchard's Test

A breakthrough event in sufficient RMS schedulability tests was the publication of Burchard's uniprocessor test in 1995 [60]. Here, the *deviation* of task sets from the beneficial simply periodic structure was quantified and used for increasing the utilization bound. Task sets close to simply periodic have utilization bounds close to 100%, but such ones most distant from it conform to the worst-case situations described in the LL formula. The more data of the task set is taken into account, the more sensitive sufficient schedulability tests can be applied.

Let us consider first a set of $n = 2$ tasks. Further, let w.l.o.g. $p_2 \geq p_1$ hold. The period ratio $p = p_2/p_1$ is then lower-bounded by 1. Integer values of p correspond to a simply periodic task set. Then, the exact utilization bound is described by (2.16) as already published in [189]. The corresponding graph is given in Figure 2.2.

$$u_{lub}(p) = p + \lfloor p \rfloor \left(\frac{1 + \lfloor p \rfloor}{p} - 2 \right) \tag{2.16}$$

Note that this exact utilization bound reduces to (2.17) in the case of $\lfloor p \rfloor = 1$.

$$u_{lub}(p) = p + \frac{2}{p} - 2 \tag{2.17}$$

For a more comprehensive discussion of this exact bound and its extension to task sets with $n \geq 3$ by correlation diagrams, see [175].

Burchard's test uses so-called S values which quantify the distance of a period to a power of 2 from above, see (2.18).

$$S_i := \log_2 p_i - \lfloor \log_2 p_i \rfloor \tag{2.18}$$

A task's S_i value is the decimal fraction of the binary logarithm of its period. Then, similar S_i values imply a good fitting of periods. Note that the inverse is not true as shown by the period set examples $\{2, 6\}$ and $\{17, 31\}$.

Next, the spectrum of occurring S_i values is investigated. The statistical measure of (linear) range (2.19) is used.

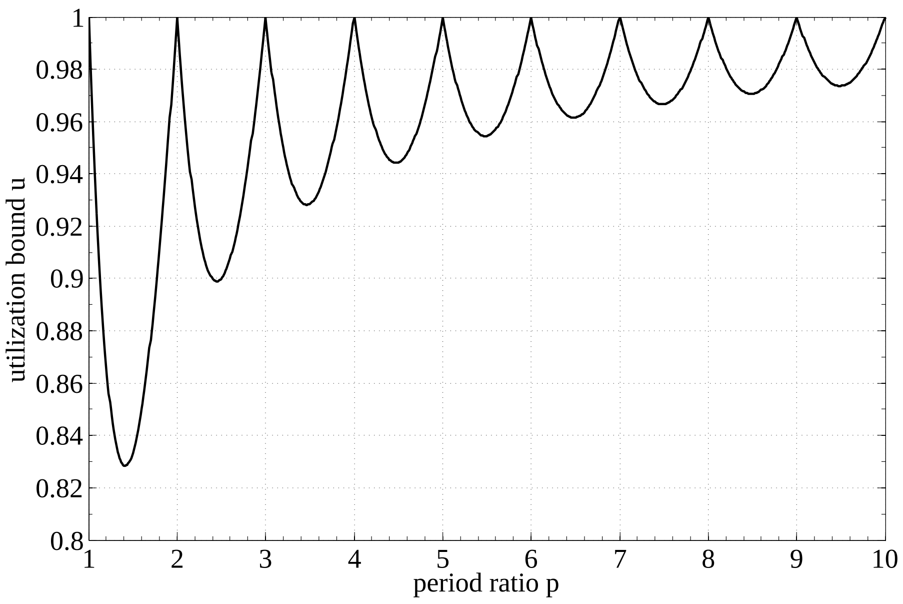

Figure 2.2: Exact Utilization Bound $u_{lub}(p)$ under RMS for 2 Tasks with Period Ratio p

$$\beta := \max_{1 \le i \le n} S_i - \min_{1 \le i \le n} S_i \tag{2.19}$$

Note that a sorting of the tasks according to S_i values is *not* necessary. Minimum and maximum value can be obtained by a linear search which is cheaper than an $\mathcal{O}(n \log n)$ sorting.

Finally, a case-by-case analysis is given by (2.20) where conditions for the RMS schedulability of the task set on a uniprocessor are specified.

$$u \le \begin{cases} (n-1)\left(2^{\frac{\beta}{n-1}} - 1\right) + 2^{1-\beta} - 1, & \text{if } \beta < 1 - \frac{1}{n} \qquad (a) \\ n(\sqrt[n]{2} - 1), & \text{if } \beta \ge 1 - \frac{1}{n} \qquad (b) \end{cases} \tag{2.20}$$

Note the equivalence of both terms (a) and (b) in (2.20) for the borderline case $\beta = 1 - \frac{1}{n}$ what makes the bound a continuous one. Interestingly, case (b) is a resort to the general but pessimistic Liu/Layland bound (2.10). Later in 7.4.1, we will come back to this unfavorable fact.

Burchard's uniprocessor test in its original form given in [60] uses a simple lower bound (2.21) of the more complicated bound (2.20). It is piecewise linear with two pieces. The first piece is the tangent equation at $\beta = 0$, the second one equals the Liu/Layland limit bound (2.11).

$$u \leq \max\left(1 - \beta \ln 2, \ln 2\right) \tag{2.21}$$

This test (2.21) is called simplified Burchard's test (sBu).

Later on we will improve on Bu by presenting an improved Burchard test in 7.4.1.

R-BOUND

Another improvement of the Liu/Layland bound taking period values into account was given by Lauzac *et al.* in 1998 [174]. Here, task periods are first transformed using *ScaleTaskSet* in order to obtain periods of the same binary order of magnitude (2.22).[7] Then, the eponymous constant r is defined as the ratio of maximum to minimum of the transformed periods (2.23). Its range is, thus, $r \in [1, 2)$.

$${p_i}' := p_i 2^{\lfloor \log_2 (p_{\max}/p_i) \rfloor} \tag{2.22}$$

$$r = \frac{\max\limits_{1 \leq i \leq n} {p_i}'}{\min\limits_{1 \leq i \leq n} {p_i}'} \tag{2.23}$$

Interestingly, sorting is not necessary for obtaining the required maximum and minimum value, a linear search is sufficient. The R-BOUND test itself finally uses this r value according to (2.24) given in [174].

$$u \leq (n-1)(r^{1/(n-1)} - 1) + \frac{2}{r} - 1 \tag{2.24}$$

Note that the r value is sometimes [137] defined as the inverse value $r' = 1/r$ with $r' \in (0.5, 1]$. Then, an equivalent formula can be obtained [137].

Relation to Burchard's test. The similarity between the bound (a) in (2.20) and the R-BOUND (2.24) is remarkable. The substitution $\beta = \log_2 r$ in (a) in (2.20) gives the same RHSs. As a consequence, very similar sensitivities of these two tests are reported, see [288] and [217]. But the improved Burchard test, see 7.4.1, correctly interpreting the fractional parts of the period logarithms as *circular* data will accomplish the two approaches.

[7]Note that there is an oversight mistake in [174], flooring and logarithmization have been swapped.

More Period Ratios and the T-BOUND. As a refinement, more period ratios can be taken into account. A criterion using *all* period ratios is the T-BOUND [174] which might be an exaggeration. More convenient is the use of the largest and the smallest period ratio in the transformed task set as suggested in [275].

Virtual Periods. Wei *et al.* [275] suggest also to use *virtual periods* as an improvement to the *ScaleTaskSet* transformation. A virtual period $v_{i,k}$ is the virtual period of task k with respect to task i defined as (2.25). So, the virtual period $v_{i,k}$ is the largest integer multiple of a period p_k not greater than the reference period p_i. According to [275], all tasks are transformed by using their virtual period with respect to the longest period.

$$v_{i,k} := \left\lfloor \frac{p_i}{p_k} \right\rfloor p_k \tag{2.25}$$

But the virtual periods transformation incorrectly assumes that always only the longest-period task is prone to deadline misses. A counter-example is given in [237]. So, the approach can only be used correctly in an *iterative* manner[8] which multiplies complexity by a factor of n.

Pillai/Shin Test

In the context of energy efficient computing, Pillai and Shin proposed their Pillai/Shin test (PS test) [241] in 2001. The number of a task's preemptions by a higher-priority task is upper-bounded by the period ratio of the two tasks. This corresponds to taking the upper limit of the interval $(0, p_i]$ in (2.3). The general approach for the construction of a sufficient RMS schedulability test taken here is the consideration of only a strict subset of the values in $(0, p_i]$, cf. Subsection 2.2.1.

$$\forall i : e_i + \sum_{Prio_j > Prio_i} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \leq D_i = p_i \tag{2.26}$$

Computational complexity is $\mathcal{O}(n^2)$. A problem of the PS test is that it is not dominating the LL test, see [216]. Hence, it can even fail at low ($u < \ln 2$) total utilizations.

[8]The same argument makes the Root-based test, see 2.2.2, inconvenient.

Response Time Sufficient Condition (RTSC)

Han *et al.* [137] suggest to search for an accelerated task set with virtual periods $v_{i,k}$ fulfilling (2.3). Then, another sufficient RMS schedulability condition is obtained (2.27).

$$\forall i : \exists k, 1 \leq k \leq i : e_i + \sum_{Prio_j > Prio_i} \left\lceil \frac{v_{i,k}}{p_j} \right\rceil e_j \leq v_{i,k} \tag{2.27}$$

This test is even exact [137] for task sets with a maximum period ratio $r < 3$, cf. [238], and for two-task task sets ($n = 2$)[9].

For the general case, RTSC is superior to all hitherto discussed sufficient schedulability tests [137] and close to exact tests. Computational complexity of RTSC is $\mathcal{O}(n^3)$ due to the search for a convenient shorter period compared to the Pillai/Shin test.

Tests Based on ASPTSs: Sr and DCT

Most of the sufficient schedulability tests given up to now are in closed form and based on utilization bounds. They share the construction principle of difficult-to-schedule task sets (which fully utilize the processor) of minimum total utilization presented in [189].

Now, we want to suggest alternative tests which are no longer given in closed form[10]. At first glance, they seem to be exotic, but they are promising and powerful [136].

ASPTSs. Accelerated Simply Periodic Task Sets (ASPTSs) are accelerated ones, see Subsection 1.8.2, and simply periodic ones, see 1.2.1, at the same time.

The acceleration operation is a non-decreasing one in terms of total utilization. Since simply periodic task sets reach the highest possible utilization

[9]The evaluation of the two-task set condition results in (2.6). So, it is only conceptually a different way of an exact characterization of $n = 2$ RMS schedulability and is not explicitly mentioned in 2.2.1.

[10]The above explained R-BOUND, see 2.2.2, is sometimes also considered as a non-closed-form algorithm [246] since it requires a transformation of the task set. But the utilization bound equation (2.24) is more formative, and so it is regarded as a closed-form approach here.

bound of 1, this combination turns out to be convenient. An acceleration operation transforming an orginal task set into a simply periodic one is also called *specialization* [190], p. 414f. The basic proof idea of RMS schedulability is summarized in Figure 2.3. Note that schedulability tests obtained using this approach are *sufficient* conditions for RMS schedulability.

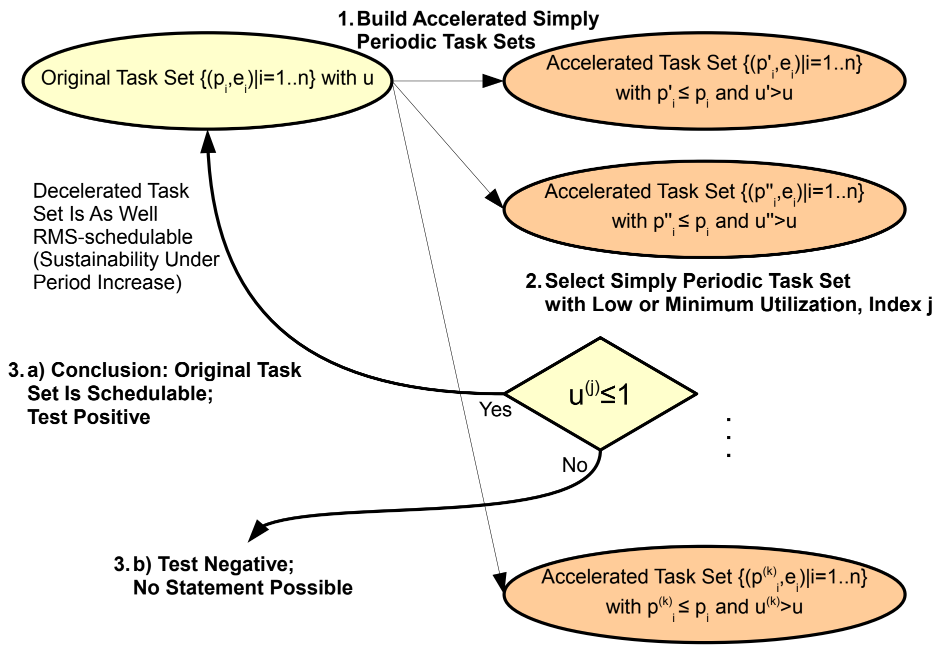
1. Build Accelerated Simply Periodic Task Sets
Original Task Set {(p_i,e_i)|i=1..n} with u
Accelerated Task Set {(p'_i,e_i)|i=1..n} with p'_i ≤ p_i and u'>u
Accelerated Task Set {(p''_i,e_i)|i=1..n} with p''_i ≤ p_i and u''>u
Decelerated Task Set Is As Well RMS-schedulable (Sustainability Under Period Increase)
2. Select Simply Periodic Task Set with Low or Minimum Utilization, Index j
3. a) Conclusion: Original Task Set Is Schedulable; Test Positive
u^(j)≤1
Yes
No
3. b) Test Negative; No Statement Possible
Accelerated Task Set {(p^(k)_i,e_i)|i=1..n} with p^(k)_i ≤ p_i and u^(k)>u

Figure 2.3: Sufficient RMS Schedulability Tests Using Accelerated Simply Periodic Task Sets (ASPTSs)

In step 2, we choose the minimum possible utilization among all transformed task sets. Step 3 branches based on the total utilization of this transformed task set. The condition $u^j \leq 1$ is an exact schedulability test for ASPTSs, see (2.14). If the accelerated task set is schedulable, then the original task set with relaxed constraints is also schedulable due to sustainability, see Subsection 1.8.2, with respect to increased periods and later deadlines, cf. [34]. If not, no statement is possible, the original task set might be schedulable or not. Step 2 can be canceled as soon as an ASPTS with $u^j \leq 1$ is found. For the proof of schedulability, it is not worth to search for further ASPTSs of even lower total utilization.

The difficult step is number 1, the search for ASPTSs under the constraint of a maximum utilization of 1. Two algorithms, *Specialization with respect to r*

(Sr) and *Distance-constrained Tasks* (DCT) have been developed and presented in [136] and [216]. While Sr restricts its search to power-of-two period ratios[11], DCT is a greedy algorithm using locally optimal period transformations.

Sr and DCT both originate from algorithms for the *pinwheel scheduling problem* which was posed in 1989 by Holte *et al.* [141]. Sr was proposed for the application of pinwheel scheduling to distance-constrained tasks [134] [142] and later adapted to (normal) RT scheduling [135].

Both algorithms take one of the n periods as a pivot, i.e., maintain its value. This is a necessary condition for a minimum utilization of the ASPTS [136] which maximizes the chances of obtaining a $u' \leq 1$ as required for passing the sufficient test. It can be shown by a *reductio ad absurdum*[12].

Assume that an ASPTS with a minimum utilization contains a transformed task with a periods *all* strictly less than its original period factors $0 < c_i < 1$. Then, a consistent period scaling of the task set by the factor $1/\max_i c_i > 1$ modifies[13] the transformed task set in such a way that the new total utilization $u'' = \frac{u'}{1/\max_i c_i} < u'$ is further dropped. This is a contradiction to the assumption that we had already an ASPTS of minimum utilization.

Hence, it is sufficient to consider the n cases of taking $p'_i = p_i$ as a pivot which translates into an outer *for* loop with i running from 1 to n. Computational complexity of Sr and DCT is, thus, at least linear, i.e., they are in $\Omega(n)$.

For a better understanding, we consider a *non-optimized* version of Sr (Algorithm 2.1) which is slower with its $\mathcal{O}(n^2)$ complexity. This results from the two nested loops both having n runs.

For Algorithm 2.2 (DCT), the starting point is always the current pivot period. Then, periods are transformed iteratively upwards and downwards according to (2.28) and (2.29). These iterative steps require a (pre-)sorting of the task set according to nondecreasing periods since the period transformations

[11]Sr constructs relaxed *binary task sets*, cf. 1.2.1, where the omission of periods in the binary period ratio chain is allowed. Another way to relate Sr-accelerated task sets to binary task sets, cf. 1.2.1, is to allow for dummy tasks with WCET $e_i = 0$ to model the above mentioned period omission.

[12]also known as *proof by contradiction* or *indirect proof*

[13]Taking the maximum of the scaling factors c_i ensures the retention of the task set property *accelerated.*

Algorithm 2.1 Specialization with respect to r (Sr) in a Better Understandable $\mathcal{O}(n^2)$ Version, adapted from Han&Tyan [136]

```
1: $p_{min} := p_1$ {first find minimum period}
2: **for** $i := 2$ **to** $n$ **do**
3:   **if** $p_i < p_{min}$ **then**
4:     $p_{min} := p_i$
5:   **end if**
6: **end for**
7: **for** $i := 1$ **to** $n$ **do** {main loop over all possible pivot indices}
8:   $b_i := p_i/(2^{\lceil \log_2(p_i/p_{min}) \rceil})$ {transformed period of $p_{min}$ if $p_i$ is pivot}
9:   $u' := 0$
10:  **for** $j := 1$ **to** $n$ **do** {loop through all tasks to be transformed}
11:    $p'_j := b_i \cdot 2^{\lfloor \log_2(p_j/b_i) \rfloor}$ { transformed period}
12:    $u' := u' + e_j/p'_j$ {sum up utilization of accelerated task set}
13:  **end for**
14:  **if** $u' \leq 1$ **then** {ASPTS is schedulable}
15:    {Sr test passed, schedulable}
16:    **exit**
17:  **end if**
18: **end for**
19: {Sr test failed, might be schedulable or not}
```

always require the recently transformed period. Otherwise, no period chain with integer ratios would be constructed, a characteristic property of simply periodic task sets, cf. 1.2.1.

$$p'_j := p'_{j-1} \cdot \left\lfloor \frac{p_j}{p'_{j-1}} \right\rfloor \tag{2.28}$$

$$p'_j := \frac{p'_{j+1}}{\left\lceil \frac{p'_{j+1}}{p_j} \right\rceil} \tag{2.29}$$

Equations (2.28) and (2.29) are elegant ones since they seem to cancel out p'_{j-1} respectively p'_{j+1} yielding equality $p'_j = p_j$ when ignoring the directional rounding (flooring and ceiling). The local DCT transformations are designed to stay as close as possible to the original period values while ensuring integer period ratios in the ordered list. Hence, DCT belongs to the class of *greedy* algorithms, see Chapter 16 in [82]. Integer ratios are obtained due to the flooring and ceiling operator.

Note that (2.28) and (2.25) are very similar. For both approaches, acceleration plays a central role. But the virtual periods method (2.25) accelerates one task with respect to all the other ones which makes a time-consuming iterative test necessary, cf. 2.2.2. More elegantly, DCT constructs using (2.28) and (2.25) a chain of integer-ratio periods and, thus, waives the iterative testing.

Besides the direct use for proving uniprocessor RMS schedulability, DCT has several applications. One can use it in partitioned multiprocessor scheduling, see Subsection 7.4.1. Also, its property of relating the general to the best case (simply periodic task sets) allows for a safe stochastic analysis of RMS, see [166].

The computational complexity of Sr in its optimized version, see [136] and [190], p. 415ff., is $\mathcal{O}(n \log n)$ or log-linear[14], that of DCT[15] $\mathcal{O}(n^2)$. On the average, DCT works better than Sr [136]. But there are some task sets which can be deemed schedulable by Sr, but not by DCT [216]. DCT turns out to be an exact test for task sets with $n = 2$ tasks [216].

The sensitivity of DCT is slightly better than that of Sr. Both tests are superior to other sufficient tests like LL, HB or Burchard's test [216]. But

[14]This comes from a presorting which is dominating.

[15]Again, this is due to two nested loops both of n runs. The sorting step is $\mathcal{O}(n \log n)$, cf. Part II in [82], and, thus, not dominating.

Algorithm 2.2 Distance-constrained Tasks (DCT), Han&Tyan [136]

```
1: {sort task set according to non-decreasing periods}
2: for $i := 1$ to $n$ do {main loop over all possible pivot indices}
3:     $p'_i := p_i$ {pivot period}
4:     for $j := i + 1$ to $n$ do {upward transformation for all longer periods}
5:         $p'_j = p'_{j-1} \cdot \left\lfloor \frac{p_j}{p'_{j-1}} \right\rfloor$
6:     end for
7:     for $j := i - 1$ downto 1 do {downward transformation for all shorter periods}
8:         $p'_j = \frac{p'_{j+1}}{\left\lceil \frac{p'_{j+1}}{p_j} \right\rceil}$
9:     end for
10:    $u' := 0$
11:    for $j := 1$ to $n$ do
12:        $u' := u' + e_j/p'_j$ {sum up utilization of accelerated task set}
13:    end for
14:    if $u' \leq 1$ then {ASPTS is schedulable}
15:        {DCT test passed, schedulable}
16:        exit
17:    end if
18: end for
19: {DCT test failed, might be schedulable or not}
```

there is a trade-off between sensitivity and computational complexity as can be seen in Figure 2.4.

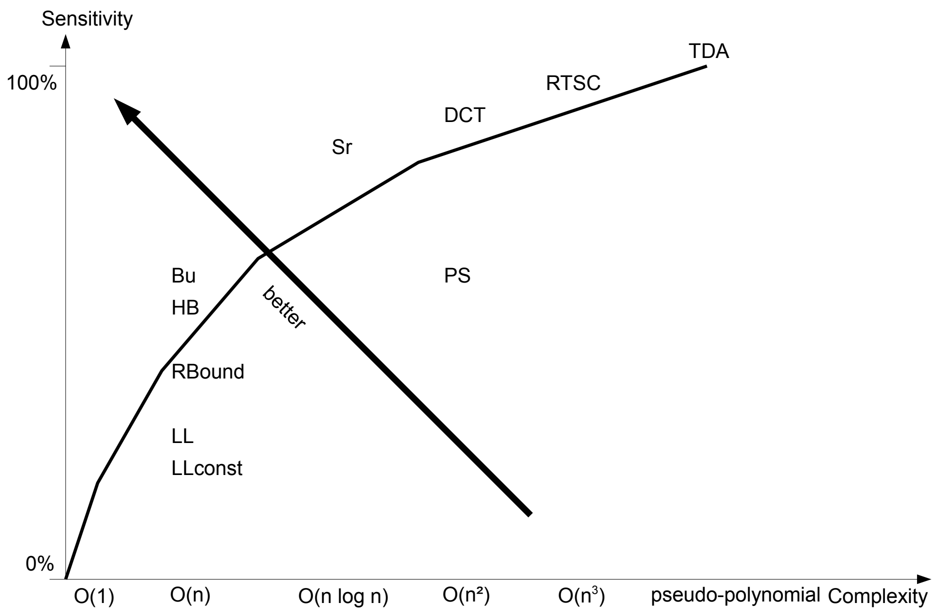

Figure 2.4: Sufficient RMS Schedulability Tests: Trade-off of Sensitivity and Complexity

DCT belongs to the best all-purpose sufficient RMS schedulability tests [246] [216] in terms of sensitivity.

RMS-schedulable Task Sets of 100% Utilization

It is an interesting problem, see [75], p. 118ff., to identify task sets of 100% utilization being schedulable by RMS. An obvious approach is to use simply periodic task sets and then to scale WCETs to obtain a total utilization of 1. They are then schedulable due to general utilization bound of 1 for simply periodic task sets (2.14).

But look at the task set $\{T_1(2,1), T_2(3,1), T_3(6,1)\}$ with $u = 1/2 + 1/3 + 1/6 = 1$. It is schedulable by RMS although it is not simply periodic. But it is semi-harmonic with a single root 6. This example clearly shows that not only simply periodic task sets can achieve full utilization under RMS.

In [75], p. 118f., Chen showed that being semi-harmonic is a necessary condition for the goal of both being RMS-schedulable and of total utilization 100% .

Then, it is proved that such a task set can be constructed based on *even perfect numbers*, cf. [75], p. 118ff. A perfect number is a positive integer which equals the sum of its proper[16] integer divisors. The longest period p_n and, thus, the root, is the perfect number itself. The proper divisors except for 1 are taken for shorter periods p_i. All WCETs are chosen to equal 1. Note that the example given above corresponds to the first even perfect number 6. The next utilization-one task set is then corresponding to 28: $\{T_1(2,1), T_2(4,1), T_3(7,1), T_4(14,1), T_5(28,1)\}$. Since there are 47 even perfect numbers known today in the year 2012 [140] we can construct 47 task sets of this kind. But the problem of even perfect numbers and *Mersenne primes*[17] is an open one. Thus, there may be even infinite many solutions.

But the three next perfect numbers are 496 and 8,128 and 33,550,336 indicating that this sequence grows rapidly. Also the number of proper divisors increases quickly and, thus, the number of tasks in the constructed task sets grows beyond a practically relevant level. Next, the restrictions that the WCETs have to be all the same and equal to 1 are severe ones. Hence, the importance of such task sets seems to be restricted to academia.

The relationships among semi-harmonic, simply periodic and perfect-number task sets are summarized in the Venn diagram in Figure 2.5. The result is that the property semi-harmonic can serve as a necessary and the use of periods based on perfect numbers as a sufficient criterion for task sets with a maximum RMS utilization of 1.

[16]Proper means that the divisor is different from the number itself.

[17]Mersenne prime numbers are prime numbers being predecessors of powers of 2, $M_p = 2^p - 1$. There is a one-to-one relationship between even perfect numbers and Mersenne primes: $M_p(M_p + 1)/2$. The both directions were proved by Euclid [105], Elements IX.36, and by Euler. For both proofs, see [69].

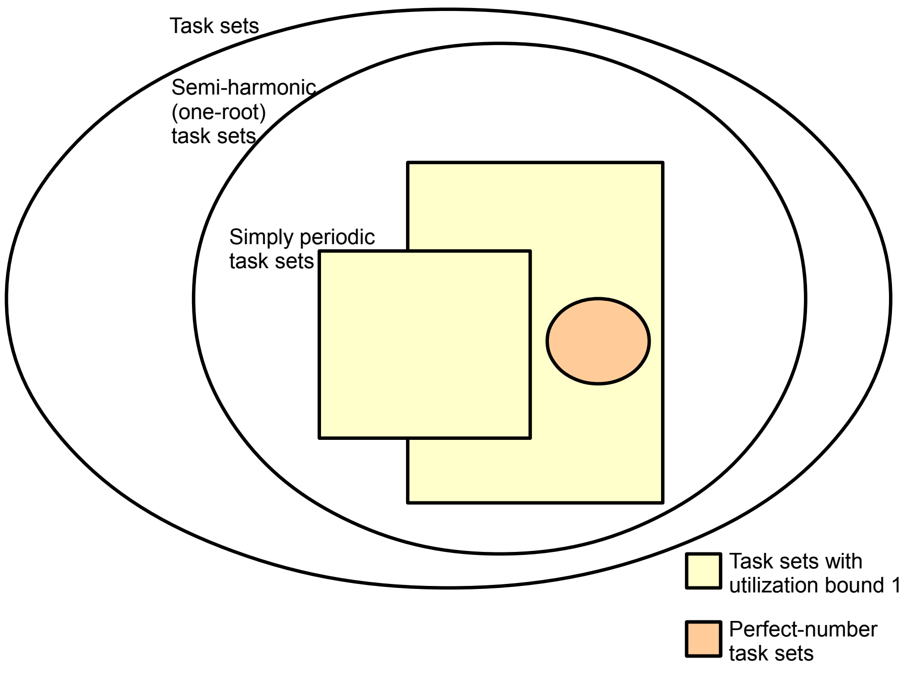
Task sets
Semi-harmonic (one-root) task sets
Simply periodic task sets
Task sets with utilization bound 1
Perfect-number task sets

Figure 2.5: Semi-harmonic, Simply Periodic and Perfect-number Task Sets Related to Task Sets of RMS Utilization Bound 1

2.3 Static Priorities and DMS for Constrained Deadlines

In the case of the more general model of constrained deadlines, schedulability is even harder to characterize. Deadline-monotonic scheduling (DMS) is the analogue to RMS using relative deadlines D_i instead of periods p_i. It is a priority-based scheduling policy with task priorities assigned the higher the shorter the relative deadline D_i of a task.

2.3.1 Exact Tests: TDA

Only the exact test TDA (2.5) can be reused in almost exactly the same manner. Since TDA refers already to deadlines, the procedure and properties like exactness and pseudo-polynomial complexity are kept. Since for constrained deadlines, the case of a response time larger than the period cannot coincide with meeting the deadline, a consideration of the first jobs of each task remains sufficient[18].

2.3.2 Sufficient Tests

For sufficient tests of polynomial complexity, some variants shall be discussed in the following. The simplest and straightforward approach is to substitute utilization terms by density terms. This is justified due to sustainability, see Section 1.8, of DMS when deadlines are extended. The approach works fine for LL (2.10) and HB (2.12). As a result, we obtain two sufficient conditions for DM schedulability (2.30) (2.31) which can be again generalized to the sporadic event model due to the critical instant of uniprocessor scheduling, cf. 2.2.2.

$$\delta = \sum_{i=1}^{n} \delta_i = \sum_{i=1}^{n} \frac{e_i}{D_i} \leq n(2^{1/n} - 1) \tag{2.30}$$

$$\prod_{i=1}^{n} \left(1 + \frac{e_i}{D_i}\right) \leq 2 \tag{2.31}$$

These two tests remain simple and of linear time complexity, but their sensitivity is low [204]. A task's density is considered an inappropriate indicator of its long-term processor usage since it serves only as an upper bound of its utilization for constrained deadlines. This motivates the need for better tests of higher sensitivity taking into account the long-term processor usage (utilization) and bounding the WCRT of each task by its (less-than-period) deadline.

Masrur *et al.* developed in 2010 the *load test* [204] which improves in terms of sensitivity compared to the hyperbolic density test (2.31). The sufficient DM schedulability condition of the load test is (2.32).

$$\sum_{i=1}^{n} \max\left(\delta_i, \frac{2e_i}{p_i + e_i}\right) = \sum_{i=1}^{n} \max\left(\frac{e_i}{D_i}, \frac{2e_i}{p_i + e_i}\right) \leq 1 \tag{2.32}$$

[18]Since the level-i busy window, see Subsection 1.2.4, can be greater than p_i, the entire level-i busy window has to be regarded for an arbitrary-deadline setting. E.g., for the task set $\{T_1(70, 26), T_2(100, 62)\}$, the response times of the first jobs of T_2 are $114, 102, 116, 104, 118, 106$, and 94. Hence, the third and the fifth job have longer response times than T_2's first job. This situation makes the scheduling of tasks with arbitrary deadlines much more complex. An appropriate algorithm is called *general TDA* which is given in [190], p. 141f.

A further improved test working on the more general model of sporadic task sets with arbitrary deadlines and an arbitrary but fixed static priority order was suggested by Bini *et al.* in 2010 [54]. The condition (2.33) calculates upper bounds on response times, bearing some pessimism in them. Hence, by comparing these upper bounds with the individual deadlines, a sufficient static-priority schedulability test is obtained.

$$WCRT_i \leq \frac{e_i + \sum\limits_{Prio_j > Prio_i} e_j(1-u_j)}{1 - \sum\limits_{Prio_j > Prio_i} u_j} \tag{2.33}$$

An obvious drawback of (2.33) is that it requires the tasks to be ordered by priority as opposed to (2.10), (2.12) and (2.32). Hence, a constant-time admission control is no longer possible.

For low-utilization task sets where $u \leq \ln 2$ holds, a further improvement was suggested recently by Baruah [32], valid for sporadic task sets with constrained deadlines. The WCRT can then be bounded according to (2.34).

$$WCRT_i \leq \frac{\exp e_{i,high,long}}{2 - \exp e_{i,high,long}} u_{i,high,short} \tag{2.34}$$

Here, $e_{i,high,long}$ “[..] denotes the sum of the WCETs of all higher-priority tasks with period[..]” [32] $\geq WCRT_i$ and $u_{i,high,short}$ “[..]denotes the sum of the utilizations of all higher-priority tasks with period[..]” [32] $< WCRT_i$.

2.4 Dynamic Priorities Fixed on Job Level and EDF

Allowing different priorities attached to different jobs of the same tasks, the optimal scheduling algorithm for sporadic task sets is EDF regardless of the type of deadlines [90].

EDF is a task-level dynamic and job-level static scheduling algorithm which selects always the job with the nearest absolute deadline d of a job[19].

For implicit deadlines, the exact schedulability test is the comparison of the total utilization u with the value 1. So, the exact criterion [189] is (2.35) with linear complexity.

[19]For periodic tasks, d_{ij} according to (1.11) has to be considered.

$$u = \sum_{i=1}^{n} u_i \leq 1 \tag{2.35}$$

This means that EDF is an optimal scheduling algorithm for implicit-deadline periodic task sets [189].

2.4.1 EDF with Constrained Deadlines

Already for the first generalization, for constrained-deadline task sets, the situation is far from being trivial [40].

Exact Tests

Exact tests have been given by Baruah [40], George *et al.* [120], Albers and Slomka [4] as well as by Ripoll [252]. They are based on the famous concept of a *demand-bound function* (DBF) introduced by Baruah *et al.* [40].

Demand-Bound Function. The DBF of a task T_i is defined as (2.36), cf. [40]. Baruah *et al.* have shown that the critical instant happens with a synchronous release of all first jobs and a release of subsequent jobs according to the periodic pattern which is a limit case of the allowed sporadic job releases [40].

$$DBF(T_i, t) := \max\left(0, \left\lfloor \frac{t - D_i}{p_i} \right\rfloor + 1\right) e_i \tag{2.36}$$

This is the maximum cumulative execution requirement of all jobs belonging to T_i with both their release times and their absolute deadlines in the interval $[0, t]$. Note that, compared to the request-bound function (RBF) (2.4) this definition of DBF adds the condition that the absolute deadlines of the jobs belong to the interval $[0, t]$.

The DBF is the RBF right-shifted by the relative deadline D_i. So, the RBF is an upper bound of the DBF. For implicit-deadline tasks, the DBF is the result of a one-period right shift of the RBF. But for constrained-deadline tasks, the RBF-to-DBF ratio is bounded from above by 2 for all $t \geq D_i$ as shown in [108]. An example of an RBF and a DBF of a task is given in Figure 2.6.

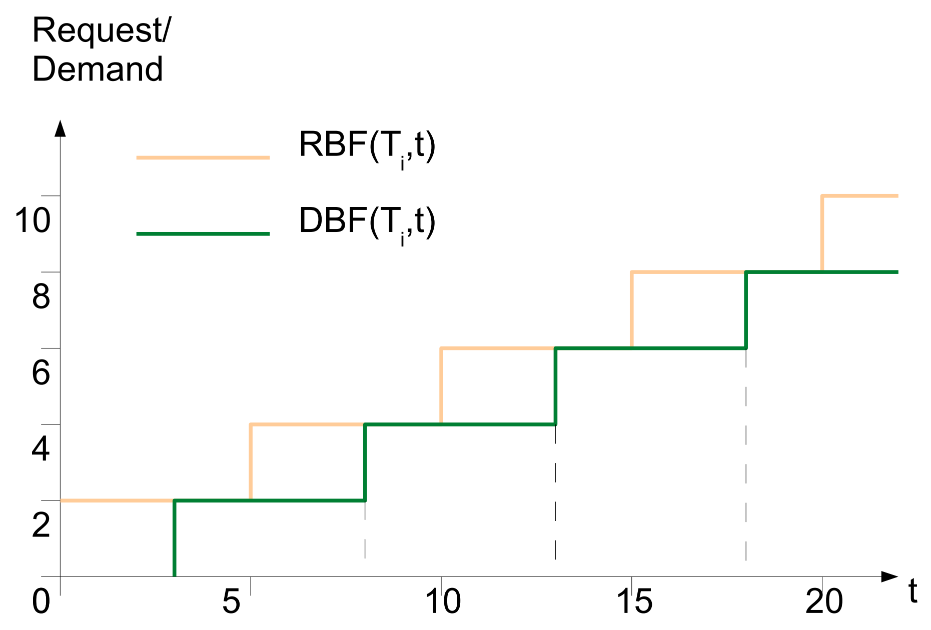

Figure 2.6: Request-Bound Function (RBF) and Demand-Bound Function (DBF) of the Constrained-Deadline Task $T_i(p_i = 5, e_i = 2, D_i = 3)$

Condition. The exact condition for task set feasibility using preemptive EDF (which is optimal) on a uniprocessor is then, according to [40], (2.37).

$$\forall t \geq 0 : \sum_{i=1}^{n} DBF(T_i, t) \leq t \tag{2.37}$$

This exact uniprocessor condition can be generalized to a necessary MP condition, see Subsection 3.1.3. For implicit-deadline task sets, the exact EDF schedulability condition (2.37) simplifies to (2.35) as shown in [145].

Comparison. Since full sensitivity is given for all these tests due to their exactness, the number of necessary steps correlating with the test's execution time remains as a reasonable quality criterion. Here, it turned out [4] that the test by Albers and Slomka outperforms both Baruah's and George's test since it switches to slow but good approximations or to the exact one only as necessary[20] [4].

[20]This general idea of an appropriate level of approximation is also used in a test schema in Subsection 8.7.6 where an exact schedulability test is based on sufficient tests.

Approximating the DBF. Since the exact feasibility test (2.37) is coNP-Hard [97] which can become intractable, there is a desire for good approximations of this test.

According to [71], p. 78ff., this can happen at two stages. First, the DBF itself can be approximated. The floor function in (2.36) makes the DBF a piecewise constant (step) function. So, it can be approximeted, e.g., by a piecewise linear function as suggested in [36]. A sufficient test is obtained using an upper-bound estimation of the DBF, and a necessary test can be based on a lower-bound estimation of the DBF.

Second, (2.37) contains a universal quantifier. So, thoroughly selected sample check times can both lead to necessary and sufficient tests when additionally adapting the RHS of (2.37), cf. [71], p. 81.

Sufficient Tests

The straightforward approach to a sufficient test for constrained deadlines is to substitute utilization in (2.35) by density. This is called *density test* and is given as (2.38), see also [190] , p. 219.

$$\delta = \sum_{i=1}^{n} \delta_i \leq 1 \tag{2.38}$$

The modified test keeps its linear computational complexity $\mathcal{O}(n)$.

Devi's test [91] as a special case of Real-Time Calculus [267] [4] is less pessimistic. But this comes at the price of a log-linear computational complexity $\mathcal{O}(n \log n)$ since tasks have to be presorted.

Masrur *et al.* [205] developed a test of linear complexity with a sensitivity between that of the *density test* and *Devi's test*. Its advantage is the only linear complexity which corresponds to a constant-time admission control in the online scenario.

Further sufficient tests based on an approximated DBF are by Chakraborty *et al.* [72] and by Albers and Slomka [3].

2.5 Comparisons of EDF and RMS

A comprehensive comparison between RMS and EDF can be found in [66]. Note that—in spite of theoretical inferiority—RMS is still being more widespread in practice due to a direct support by many operating systems.

More theoretical quantifications of the advantage in schedulability of EDF compared to RMS can be found in terms of *breakdown utilization* [181], *utilization upper bound* [236] and *numerical optimality degree* (NOD) [51] [53]. A common result is that RMS has a significant shortcoming in terms of schedulability compared to EDF on a uniprocessor. This gap increases under an increasing number of tasks n.

In addition to this concrete quantification question, there were heated debates on the pros and cons when it comes to a comparison of RMS and EDF. Some misconceptions and common beliefs were corrected and clarified by Buttazzo [66]. One of the most common misconceptions on RMS is that "[..]in the presence of transient overload conditions, deadlines are missed predictably, that is, the first tasks that fail are those with the longest period." [66]. This property often attributed to RMS is wrong as shown in [66] by a counter-example. Important consequences of this fact are the inadequacy of the virtual periods method for R-BOUND, cf. equation (2.25), and that the Root-based test can only be applied in an interative manner correctly, cf. 2.2.2. Other misconceptions clarified in [66] concern implementation complexity, runtime overhead, resulting jitter, and efficiency in handling aperiodic tasks.

By now, the researchers agree that RMS and EDF have relative merits to each other. RMS is simple and response-time predictable while EDF can maximize total utilization under real-time guarantee, rendering it an optimal uniprocessor scheduling algorithm. The straightforwardness of RMS and the direct support of priority levels by prevalent operating systems is still guaranteeing its predominance in uniprocessor scheduling. On the other hand, advanced technniques like a consideration of quantized relative deadlines [138] for EDF reduces the memory requirements and can achieve good approximations of EDF to be useful for soft RT systems, cf. 1.1.4.

2.6 Fully-dynamic Priorities and LLF

An alternative approach to EDF, on uniprocessor as well being optimal, is the Least Laxity First (LLF) scheduling algorithm suggested by Mok [212]. It schedules at time t the job i with the least laxity $l_i(t)$. This ranking is fully-dynamic, i.e., it can change even inside a job execution and not only at job releases or completions. Note further that the remaining execution of a job needs to be known in order to assign the correct current priority to a job. Often, only an upper bound of the execution time, the WCET, is available, cf. 1.1.1 and 1.1.2. A consequence of this fully-dynamic approach is LLF's liability to preemptions.

Frequent preemptions are particularly a problem when a *laxity tie* at time t occurs. Then, the job to run has to be selected arbitrarily but fixed. The most popular tie resolution is that by minimum index. Since all other before-tie jobs are then not running, their laxities decrease. Thus, already at time $t + \epsilon, \epsilon > 0, \epsilon \to 0$, the selected job is no longer at the head of the queue since its laxity has remained constant during execution. Thus, another job has to be selected until being overtaken in terms of minimum laxity by the then not-running jobs. This situation repeats until all but one involved jobs have been completed. Note that the situation in practice is slightly better since integer parameters guarantee laxity checks to be necessary only at integral instants of time.

A common extension of LLF to meliorate this problematic behavior is Modified Least Laxity First (MLLF) [232]. This algorithm defers a preemption in a situation described above until a zero-laxity situation occurs or the running job is completed. Note that MLLF inherits its optimality with a 100% utilization bound from LLF while the number of preemptions is reduced significantly.

In spite of these good theoretical properties, neither LLF nor MLLF are of importance on uniprocessor systems. The reason is that they are even harder to implement than the already optimal EDF.

For a short discussion on LLF on multiprocessors, see Subsection 3.5.1.

2.7 Dual Priority Scheduling

Dual Priority Scheduling (DPS) combines predictability of static priorities with a higher utilization bound than pure RMS by allowing for minimally dynamic priorities. Each task has two static priorities where the switching point (aka *promotion point*) between them is fixed in a period. It was introduced by Burns and Wellings in 1993 [64]. There, it was even conjectured that the utilization bound of DPS equals that of EDF, i.e., 100%. It is still an open problem since neither a proof nor a counter-example have been found [62] [207].

2.7.1 Relationships to Other Preemption Variants

If the conjecture turns out true, DPS will be the all-in-one for uniprocessor RT scheduling. It is the most promising approach among several preemption point variants for static priorities. An overview of some important approaches can be found in Figure 2.7.

The two extreme ones are preemptive and non-preemptive scheduling. By generalizing, scheduling points can be given *explicitly*, Fixed-priority Scheduling with Deferred Preemption (FPDS) [61] as a variant of *cooperative* scheduling, or *indirectly* by Fixed-priority Preemption Threshold Scheduling (FPTS) [272] or DPS. When using FPTS, each task is assigned a preemption threshold. Up to this priority level, premption of the respective task is disabled.

Note that for cooperative scheduling, the scheduling points can be specified either fully explicitly by the programmer[21] aka *fixed non-preemptive region scheduling* or they can be hinted by non-preemptive intervals aka *floating non-preemptive region scheduling* [203]. In the latter case, the determination of per-task longest non-preemptive intervals is a key problem. For FP scheduling, it has been first solved by Yao *et al.* [287]. Due to this dilution of the pure cooperative approach, the use of longest non-preemptive intervals can be coined a *semi-cooperative* scheduling.

[21]Then, this corresponds to task splitting.

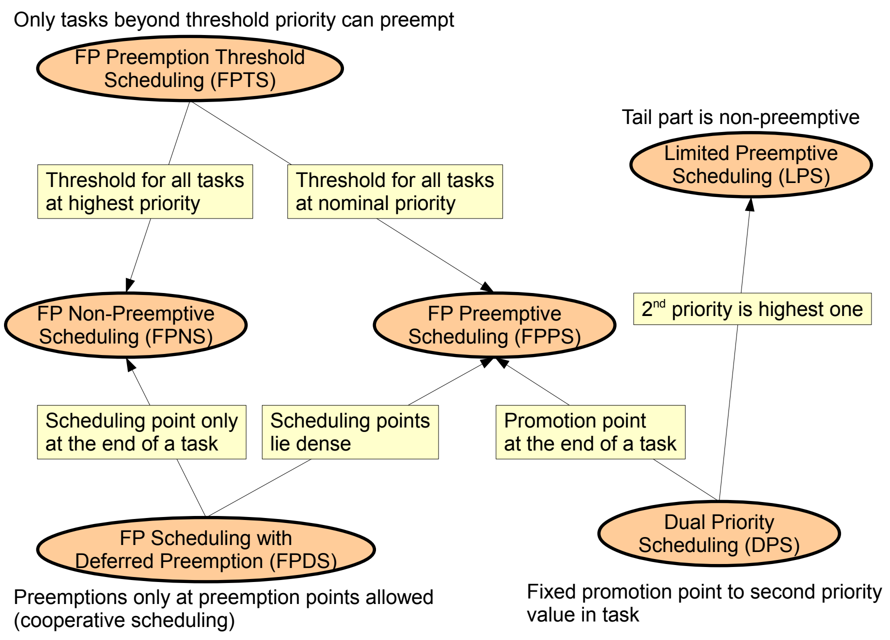
Only tasks beyond threshold priority can preempt
FP Preemption Threshold Scheduling (FPTS)
Threshold for all tasks at highest priority
Threshold for all tasks at nominal priority
Tail part is non-preemptive
Limited Preemptive Scheduling (LPS)
FP Non-Preemptive Scheduling (FPNS)
FP Preemptive Scheduling (FPPS)
2nd priority is highest one
Scheduling point only at the end of a task
Scheduling points lie dense
Promotion point at the end of a task
FP Scheduling with Deferred Preemption (FPDS)
Dual Priority Scheduling (DPS)
Preemptions only at preemption points allowed (cooperative scheduling)
Fixed promotion point to second priority value in task

Figure 2.7: Preemption Control with Static Priorities On Uniprocessor

2.7.2 Choice of Priorities and Promotion Points

The second-phase priorities can be arbitrarily chosen in the general case. But the separation into two non-overlapping *priority bands* for the first and the second phase is an attractive special case [64] [89]. Then, the second-phase band shall be of higher priority than the first-phase one. This fact can be interpreted as a simulation of LLF on a meta level.

Inside the lower-priority band, priority order can be chosen arbitrarily, but fixed. The straightforward approach is to use RM priority order [64] [207].

Inside the higher-priority band, a rate-monotonic priority order is useful [207] since it allows for an exact analysis according to (2.5) for the close-to-deadline task parts [89]. On the other hand, an *inverse* RM priority order in the higher-priority band allows for reduction in the number of priority levels to consider since, then, the first task does not need a second-phase priority [64].

Bertogna *et al.* [47] proposed another scheme called Limited Preemptive Scheduling (LPS). The tail part of each task is raised to the *highest* priority making it non-preemptive. This is considered to be a special variant of DPS.

An advancement of LPS suitable for multiprocessors is FPSL to be discussed in Subsection 3.6.4.

For the promotion times, the straightforward approach sets it to the task's worst-case response time (WCRT) according to (2.5) before its deadline [89]. This is attractive, but only successful for low-utilization task sets. If there is at least one task with a larger WCRT than its deadline, the task set not schedulable under DPS using these particular promotion points. In such a case, the approach can be though used as a starting point for an iterative solution, cf. [64] and [207].

2.7.3 DPS for $n = 2$ Tasks

For the simplest reasonable case of $n = 2$ tasks, Burns [62] gave a constructive proof that DPS is optimal. There, it is w.l.o.g. assumed that $p_2 \geq p_1$ and $u_1 + u_2 = u = 1$. The first condition is achieved by renaming, the second one by filling up execution times with of dummy subtasks to reach the worst-case utilization of 1.

Then, only the second task T_2 needs to be promoted to a higher priority than T_1 since all priority orders of 2 tasks are covered by such a priority inversion. The only remaining degree of freedom is the promotion time S_2 where the priority inversion occurs. It has to be set to a value in the interval[22] (2.39) according to [62].

$$p_2 - p_1 + e_1 \leq S_2 \leq p_2 - \frac{(p_1 - \gcd(p_1, p_2))\, e_2}{p_2} \tag{2.39}$$

Thus, the simplest choice of the promotion time S_2 is the minimum of the interval (2.39) in the general case.

Some special cases shall be discussed in the following. First, for simply periodic task sets, $\gcd(p_1, p_2)$ equals p_1. Then, it is easiest to set S_2 to p_2 meaning that there is no promotion at all, and we obtain FPPS with RM priority order. This reconfirms the RMS schedulability of a two-task simply periodic task set up to a utilization of 1., cf. (2.14).

Next, if the both periods are coprime, their gcd becomes 1. This simplifies (2.39) to (2.40), leaving a narrower valid range for S_2.

[22]The operator *gcd* denotes the *greatest common divisor.*

$$p_2 - p_1 + e_1 \leq S_2 \leq p_2 - \frac{(p_1 - 1)e_2}{p_2} \tag{2.40}$$

Non-integer, but rational periods can be equivalently mapped to an integer-period task set by scaling it with the lcm[23] of the two period denominators. Alternatively, the gcd of non-integer, rational numbers can be defined as the greatest number where all ratios to it are integers [276].

This is no more possible for an irrational period ratio. But the gcd's domain can be further extended to irrational numbers and setting it there to *zero* as suggested in [276]. Then, only a single promotion time remains possible since lower and upper bound in (2.39) then coincide. This becomes also visible in the proof of existence of a promotion point in [62] where a gcd of zero exactly describes the limit case. Note that an irrational period ratio means that *any* phasing except to equal deadlines can occur. This is only achieved in the limit $t \to \infty$ which can be interpreted as the lcm of the periods.

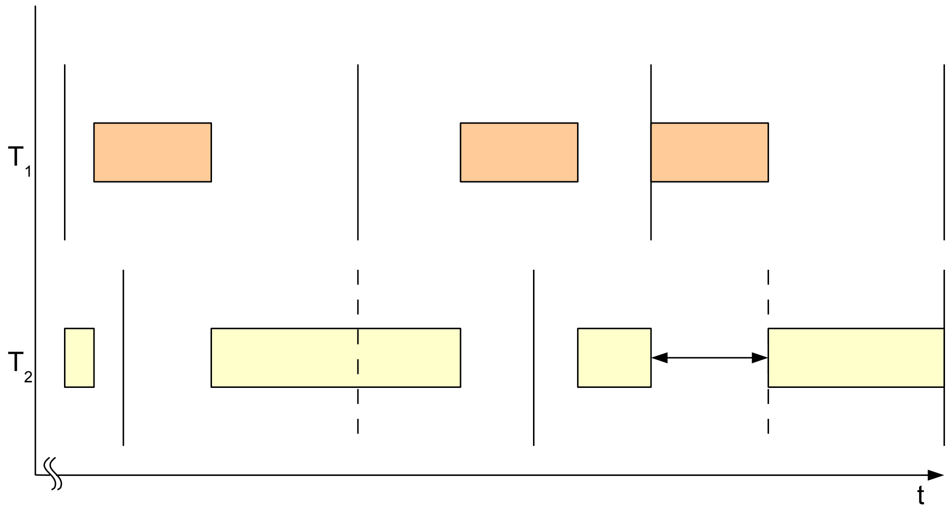

Figure 2.8: DPS Schedule of the Task Set $\{T_1(1, \sqrt{2} - 1), T_2(\sqrt{2}, 2\sqrt{2} - 2)\}$, Promotion Point at $S_2 = 2\sqrt{2} - 2$ Without any Margin of Movement; Deadlines as Solid Lines and Promotion Points as Dashed Lines

A DPS schedule of such a task set is given in Figure 2.8. Note that the worst-case phasing fot T_1 occurs at $t \to \infty$ where the absolute deadlines of T_1 and T_2 coincide. An earlier promotion lets T_1 miss its deadline.

[23]The operator *lcm* denotes the *least common multiple.*

On the other hand, for a large appropriate time t, a deadline of T_1 lies arbitrarily close to a deadline of T_2, but before it. Then, T_2 needs at least a utilization-proportional duration of priority inversion in the two last overlapping periods, i.e., $p_1 u_2$ as shown in [62]. Thus, any later promotion lets T_2 violate its deadline. Hence, any slight shift of the promotion point to the left or to the right yields deadline violation.

Summing up, DPS is optimal for two-task task sets. Only the longer-period task needs to be promoted which can be interpreted as a priority inversion. The greatest margin of promotion point placement occurs for simply periodic tasks with an integer period ratio. This margin decreases with decreasing harmonicity down to zero for irrational period ratios. A drawback of DPS could be the destroyed regularity of the highest-priority task execution. Such a zero start and completion time jitter can be a requirement in multimedia applications, cf. 1.1.4.

2.8 Summary

RMS and EDF can be regarded as the two most important RT uniprocessor scheduling algorithms. For implicit-deadline task sets, RMS is an optimal static priority scheduling algorithm, and EDF is an optimal uniprocessor scheduling algorithm using static priorities on job level. As pointed out in Section 2.5, both scheduling policies have relative merits to each other. Hence, there is no general preference.

The fully-dynamic algorithm LLF, see Section 2.6, is also optimal. But LLF has gained almost no importance on uniprocessor systems since it involves many preemptions and is harder to implement. Further, the exact execution times of jobs need to be known beforehand which is unrealistic[24].

Among a variety of sufficient schedulability tests for implicit-deadline task sets presented in Subsection 2.2.2, the DCT test shows an outstanding sensitivity related to its complexity, see 2.2.2. At the same time, DCT is simple since it uses the straightforward approach of accelerating task sets. The mighty con-

[24]The same holds for the non-RT scheduling algorithms *Shortest Job First* (SJF) and *Shortest Remaining Time First* (SRTF) which maximize throughput in the non-preemptive (SJF) and in the preemptive (SRTF) case.

cept of sustainability justifies such acceleration operations. These fundamental interrelationships make DCT suitable for educational purposes.

The approach of Dual Priority Scheduling, see Section 2.7, turned out to be a good synthesis of RMS and EDF. DPS combines predictability of RMS with a higher utilization bound. It is conjectured that this utilization bound is 1 as it is for EDF. But the choice of priorities and promotion points is still an open problem in DPS for the general case.

Part II

A Survey of the State of the Art in Multiprocessor Scheduling

This part gives a survey on real-time multiprocessor scheduling and is based on the article [221] published in Central European Journal of Computer Science (CEJCS) in 2011.

Chapter 3

Global Scheduling

The straightforward generalization of uniprocessor scheduling to multiprocessors is global scheduling where jobs are waiting for being processed in a per-system queue. But it will turn out that this simple generalization[1] is problematic since a great deal of theorems valid on a uniprocessor is no longer valid in the multiprocessor case.

One of these core issues is that a synchronous task set, see 1.2.1, is not the critical instant where WCRTs are reached as shown in [173]. The quest for such a critical instant in the multiprocessor case is still an open problem [87].

According to [10] and contrary to the uniprocessor scheduling, the response time of a task using static-priority preemptive global multiprocessor scheduling depends not only on the parameters of the higher-priority tasks but also on their internal order. This observation renders a combination of exact response times with Audsley's priority assignment OPA, cf. Section 2.2, inappropriate. However, Audsley's approach is still suitable when using certain sufficient RMS schedulability tests being OPA-compatible as shown in [84]. But there are explicit priority orders being almost as effective as OPA for this purpose, see Subsections 3.3.7 and 3.3.8.

An analysis of feasibility and schedulability of task sets conforming to more general task models like GMF, see Subsection 1.3.4, is given in [109]. There, a feasibility region in the two-dimensional space of maximum density (1.20)

[1]Later on, we will call this type of generalization a *vertical* one. The more auspicious alternative is a *horizontal* generalization, see Subsection 5.3.2.

and load, the “[..]maximum cumulative computational demand of any subset of the set of jobs[..]” [109], is described.

3.1 Necessary Schedulability Tests

3.1.1 Maximum Total Utilization

The first necessary condition is a generalization of the necessary uniprocessor condition given in Section 2.1. A feasible schedule is only possible if the total utilization, cf. 1.2.1, does not surmount m or, equivalently, the system utilization does not exceed 1, see (3.1). In other words, a linear speedup is the limit in our model.

$$u \leq m \Leftrightarrow u_{sys} \leq 1 \tag{3.1}$$

Note that there can be super-linear speedups in practice. This is no magic—as some computer science beginners believe—but mainly can have two reasons.

First, the memory access bottleneck, see also 1.1.2, can be widened by cache levels available to all processors. Then, more or even all of the working set can be kept in the cache reducing access times by orders of magnitude. E.g., in the context of GPUs such super-linear speedups have been observed [242].

Second, even on the algorithmic level, super-linear speedup is possible. It has been observed when applying parallel backtracking algorithms [258]. A possible explanation of this effect is based on stochastic properties of the solution distribution [123]. More advanced algorithms reuse results obtained at other processors for pruning branches or for *memoization*[2], see [133].

Since we—according to Subsection 1.9.1—ignore cache effects and do not restrict the type of tasks to be executed to backtracking algorithms, the given two issues of super-linear speedups do not affect our necessary schedulability criterion (3.1).

3.1.2 Maximum Individual Utilization

The second necessary condition is sometimes omitted or implicitly assumed. A task which is not allowed to be split (no task splitting, see also 1.6.7) must

[2]The concept of building lookup tables dynamically was suggested in 1968 by Mitchie [209]. He coined his approach memoization.

have a utilization u_i no greater than 1, $\forall 1 \leq i \leq n : u_i \leq 1$. Equivalently, we can write this condition as (3.2) when using a definition in 1.2.1.

$$u_{max} \leq 1 \tag{3.2}$$

3.1.3 Maximum Total Demand

Baruah and Fisher [36] generalized the uniprocessor condition (2.37) to a necessary condition on the total demand (3.3).

$$\forall t > 0 : \sum_{i=1}^{n} DBF(T_i, t) \leq mt \tag{3.3}$$

Interestingly, this is valid both for partitioned and global scheduling [36].

Baker and Cirinei [29] improved this condition further by taking task execution into account which needs to take place in the considered time interval even if release time or deadline is not actually within this interval.

3.1.4 Maximum Individual Demand

A fourth condition is a statement on the maximum individual demand [36]. It says that the maximum synthetic utilization (8.10) must not surmount one (3.4).

$$\max_i \frac{e_i}{D_i} \leq 1 \tag{3.4}$$

If a task violates (3.4), there is no chance for the task to complete within its deadline even if it runs exclusively on a processor. Clearly, for the here mainly considered implicit-deadline task sets, (3.4) collapses to (3.2).

3.2 Scheduling Anomalies

For the case of restricted migration, cf. 1.6.7, an execution-time reduction can lead to a response time increase as shown in [190], p. 72ff. This counterintuitive behavior is called a *scheduling anomaly.* For non-preemptive tasks, cf. Subsection 1.4.2, with precedence constraints, cf. Subsection 1.3.3, the addition of processors or the relaxation of dependencies can lead to deadline misses [190], p. 74. The last two types of scheduling anomalies are even more counterintuitive. Finally, the shortening of a critical section can also result in increased response times [190], p. 280f.

Even for the task and system model mainly used in this thesis, cf. Subsection 1.9.1, there are scheduling anomalies for global multiprocessor scheduling as shown first in [15]. An increase of the period can make a task set unschedulable. This means that any exact schedulablity test for a global scheduling policy will be unsustainable, cf. Section 1.8, with respect to increased periods. Hence, a possible way to circumvent this unwanted behavior is to apply a self-sustainable schedulability test, see Section 1.8. For global fixed-priority scheduling, such a test is discussed in Subsection 3.3.5.

3.3 Static Priorities

At first glance, the combination of a *global scheduling with migrations always possible* with *static priorities which never change during the entire lifetime of a task* seems to be strange. But it will turn out that these approaches are powerful when related to the necessary management overhead.

An overview of important global fixed-priority scheduling algorithms can be found in Figure 3.1. First, the integrative consideration of implicit and constrained deadlines is visible. Here, slack-monotonic (SM) scheduling plays a central role. The priorities are assigned according to the smallest initial laxities l_i. We will discuss the different approaches in historical order, but separate for implicit and constrained deadlines.

3.3.1 Rate-monotonic Priorities and Dhall's Effect

The first attempt using static priorities was to lift the uniprocessor-optimal approach of rate-monotonic priorities to the multiprocessor level. This is possible, but results in utilization bound of $u = 1 + \epsilon$ with an arbitrarily small positive ϵ which corresponds to a maximum system utilization u_{sys} approaching zero for $m \to \infty$ as was shown by Dhall and Liu in their seminal paper [93] published in 1978.

The scenario when this occurs is to have m tasks with short periods infinitesimally small utilizations, and one task with a longer period and a utilization that approaches 1 what became known under Dhall's effect. RM is no longer

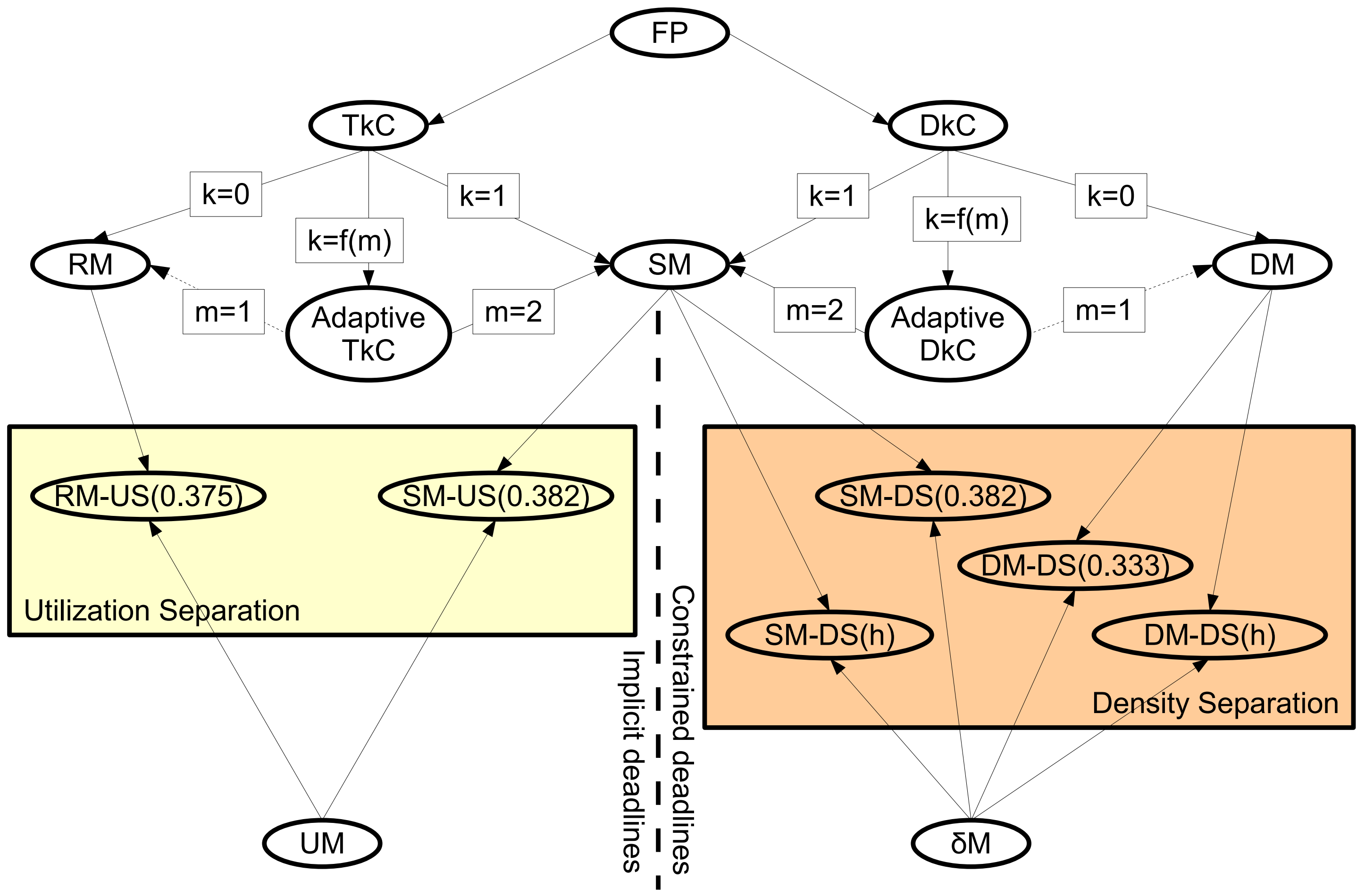

Figure 3.1: Global Scheduling with Static Priorities on Multiprocessors with Implicit and with Constrained Deadlines; Dotted Arrow Lines for $m = 1$ Since this Refers to Uniprocessor FP Scheduling

optimal for multiprocessors and, thus, leaves space for better approaches. The discovery of Dhall's effect lead to a concentration of research on partitioned multiprocessor scheduling for the following about 25 years.

When taking the maximum utilization u_{max} into account, the global RMS utilization bound improves to (3.5).

$$u \leq u_{max} + \frac{m}{2}(1 - u_{max}) \tag{3.5}$$

3.3.2 TkC as Linear Combination of Period and WCET

Andersson and Jonsson [15] achieved a breakthrough in fighting Dhall's effect in 2000. They could show that a priority order given by the shortest value of the linear combination of period p_i and (negatively weighted) WCET e_i according to $p_i - ke_i$ with the constant $k \in \mathbb{R}_{\geq 0}$ has the power of circumventing Dhall's effect. The optimal value of k is a function of the number of processors m, see (3.6). The scheduling algorithm using this formula is called *adaptive TkC*[3].

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m} \tag{3.6}$$

In the limit case of $m = 1$ we have a uniprocessor system with RM priority order, see Section 2.2. For $m \to \infty$, we get $k \to \frac{\sqrt{5}+1}{2} \approx 1.618$, the *golden ratio*. Applying TkC on a two-processor system ($m = 2$), it reduces to slack-monotonic scheduling, see 3.3.3.

It was conjectured that the system utilization bound using adaptive TkC is $u_{sys} \leq \frac{3-\sqrt{5}}{2} \approx 0.382$. This could be proved only for a specific class of task sets, but an empirical study suggests correctness of the conjecture. Additionally, the value bounds the utilization bound for general task sets from above.

Some years later, in 2003, the same authors [16] showed that the maximum utilization bound for fixed-priority global scheduling where priorities are defined solely as a scale-invariant function of p_i and e_i is (3.7).

$$u_{opt} \leq (\sqrt{2} - 1)m \approx 0.414m \tag{3.7}$$

Note that this equation acts on the meta level of utilization bounds. This theoretical result gave rise to the search for more advanced approaches narrowing the gap between $\frac{3-\sqrt{5}}{2} \approx 0.382$ and $\sqrt{2} - 1 \approx 0.414$ for the system utilization bound.

[3]The name TkC is based on the formula with symbol T for period and C for WCET.

3.3.3 Utilization Separation

An analysis of the task set characteristics in the scenario of Dhall's effect lead to the conjecture that the coexistence of low- and high-utilization tasks is a key factor for the bad performance of RM on it. So, a first-level prioritization according to decreasing utilization values seems to be a key to improvements.

Andersson *et al.* [13] found that a two-group prioritization with a threshold value dependent upon m is convenient. This concept is called *utilization separation* (US). The task with a utilization exceeding the threshold value (aka heavy task) run at highest priority, with ties broken arbitrarily, but fixed. All other tasks (aka light tasks) are assigned RM priorities. Lundberg [199] showed that the optimal threshold value for the class of RM-US[t] algorithms is the value of the transcendent equation (3.8) with a unique solution $y \in \mathbb{R}$.

$$\frac{1-y}{m(1+y)} + \ln(1+y) = \frac{1-y}{1+y} \tag{3.8}$$

Note that allowed values of m start with 2 due to the construction of extremal task sets [199]. Hence, the uniprocessor case is not covered by this formula. For the limit $m \to \infty$, (3.8) becomes (3.9).

$$\ln(1+y) = \frac{1-y}{1+y} \approx 0.375 \tag{3.9}$$

Hence, the optimal threshold for RM-US is approximately 0.375, giving the same system utilization bound of 0.375 [199].

Slack-monotonic scheduling. Slack-monotonic scheduling uses a priority order according to increasing slack times l_i. On a uniprocessor, the utilization bound is 0.5, cf. Section 2.2. Slack-monotonic scheduling is a special case of TkC with the choice $k = 1$, cf. Subsection 3.3.2. This is at the same time the optimal k value according to adaptive TkC for $m = 2$ processors. In [11], Andersson applied utilization separation to SM obtaining SM-US[t]. The optimal threshold value he obtained is (3.10) which turns out also to be the system utilization bound [11].

$$u_{sys} \leq t = \frac{2}{3+\sqrt{5}} = \frac{3-\sqrt{5}}{2} \approx 0.382 \tag{3.10}$$

Note that this bound is slightly better than that for RM-US[0.375] and equals the conjectured bound for TkC, see Subsection 3.3.2.

Summary. Summing up, utilization separation is a convenient approach to circumvent Dhall's effect. The discrimination into heavy and light tasks by setting a threshold value can result in a system utilization bound equal to this threshold. The basic proof idea is that the number of heavy tasks cannot exceed m due to the maximum total utilization. Then, the other tasks are all light ones and schedulable on the remaining processors [27].

The theoretical best possible system utilization bound is $\sqrt{2}-1 \approx 0.414$ for static-priority global scheduling of implicit-deadline sporadic task sets. The best value achieved constructively by an algorithm is about 0.382 by SM-US[0.382].

3.3.4 Deadline-monotonic Priorities

DM for constrained deadlines corresponds to RM for implicit deadlines. It was shown to be the optimal priority order for uniprocessors [185]. This is no longer true for multiprocessors. It suffers also from Dhall's effect, cf. Subsection 3.3.1.

3.3.5 TDA for Global Fixed-priority Scheduling as a Sufficient Schedulability Test

The mighty approach of TDA which gives exact tests in the uniprocessor case for implicit (cf. Subsection 2.2.1) and for constrained deadlines (cf. Section 2.3.1). Unfortunately, a generalization of the TDA to global RMS or global DMS does not give an exact test, but a sufficient one since the exact interference suffered by the analyzed task can only be bounded from above. The reason is that the critical instant in MP scheduling is unknown [87].

The work a task has to do in a level-i busy period, cf. Subsection 1.2.4, can be decomposed into three contributions:

body: from all jobs completely (i.e., both release time and deadline) within the level-i busy period;

carry-in: from a job with a release time before the level-i busy period and a deadline in it;

carry-out: from a job with a release time in the level-i busy period and a deadline after it.

Formula (3.11) gives an upper bound calculation of WCRTs for global fixed-priority scheduling with constrained deadlines. It was first proposed in [15] and is based on an idea in [198]. It describes a fixed-point iteration similar to (2.5) with the following two modifications in the interference term.

1. Compared to the uniprocessor case, one *extra instance* of each higher-priority task has to be taken into account in the interference analysis.

2. The interference term is *scaled by the factor* $1/m$ since the interference can be equally distributed to the m processors.

Interference can only occur for the $n - m$ lowest-priority tasks since the m highest-priority tasks can always run in parallel on m processors without any contention. This means in particular that their WCRTs correspond to their WCETs [198]. For the $(m + 1)$th highest priority task, the worst-case scenario is still the synchronous task release as it is the critical instant in the uniprocessor case. But for all other (lower-priority) tasks and, thus, in general, the synchronous case does not represent the critical instant [198] [173].

An upper bound of the worst case can be calculated when all higher-priority tasks completely overlap their execution. From this, we obtain the first modification that an extra instance of the higher-priority tasks needs to be considered.

The start value for the iteration is chosen as $WCRT^{UB}_{i,k+1} = e_i$ and *UB* stands for *upper bound* since the test is no longer exact but only sufficient.

$$WCRT^{UB}_{i,k+1} = e_i + \frac{1}{m} \sum_{Prio_j > Prio_i} \left(\left\lceil \frac{WCRT^{UB}_{i,k}}{p_j} \right\rceil e_j + e_j \right) \tag{3.11}$$

This sufficient test is self-sustainable, cf. Section 1.8, which makes it in this respect superior to any corresponding exact test since it circumvents the increased-periods scheduling anomaly, cf. Section 3.2, for global RMS described in [15]. On the other hand, it has a low sensitivity [46], p. 93, and is, thus, pessimistic since it assumes the entire WCETs of the respective tasks both for the carry-in and the carry-out.

The test can be improved by considering refined carry-in and carry-out estimations as well as by exploiting slack (cf. 1.1.2) values. For an overview on these methods and resulting tests, see, e.g., Chapter 5 in [46]. Further improvements and a generalization to the case of arbitrary deadlines are given in [130].

3.3.6 Density Separation

The analogous approach to utilization separation for constrained-deadline task sets is *density separation.* Bertogna *et al.* [48] suggested to use DM-DS[1/3] with a density bound of $\frac{m+1}{3}$.

3.3.7 Adaptive DkC

Davis and Burns [84] generalized the TkC approach to constrained deadlines obtaining DkC meaning “deadline minus k times execution time”. The formula for the optimal k remains (3.6). According to [84], “[..]DkC priority assignment policy is almost as effective as optimal priority assignment[..]” when used in combination with sufficient global FP schedulability tests.

3.3.8 Slack-monotonic without and with Density Separation

Also for the slack-monotonic approach, an improvement is expected when applying utilization separation to implicit deadlines and density separation to constrained deadlines. The simplest variant is to reuse optimal threshold values calculated for implicit deadlines [11] for a density separation threshold. This results in an SM-DS[$\frac{2}{3+\sqrt{5}}$] algorithm. But, according to [86], the scheduling performance is restricted. Probably, the assumed density threshold is not the optimal one.

The slack-monotonic approach is called *$D_i - C_i$ monotonic priority ordering*[4] (D-CMPO) in [84]. Davis and Burns argue that D-CMPO could be a useful heuristic. In their empirical investigation of sufficient global FP schedulability tests, they showed that D-CMPO is better than DM priority order but slightly inferior to a DkC priority assignment, see Subsection 3.3.7.

[4]The term *Deadline minus Computation time Monotonic Priority Ordering*, DCMPO, is used synonymously in the literature [88].

3.3.9 Simply Periodic Task Sets

For the special case of simply periodic task sets, cf. 1.2.1, a system utilization bound of 100% can be reached. Jung *et al.* [148] found a global scheduling algorithm that is based on task splitting. Note that the split-up of a task at an arbitrary position is required. This is not always possible due to semantic reasons and, thus, has to be seen as a restriction. This algorithm has been designed to work on the homogeneous multiprocessor model, see 1.6.2, allowing for different processor speeds in the set of processors belonging to the multiprocessor.

3.3.10 Release Time Jitter

For varying release times (called release jitter, see 1.1.4) of constrained-deadline periodic tasks, the DM priority order is no longer optimal. The optimal priority assignment is based on deadline minus jitter (D-J) as shown in [289]. The rationale behind it is that a job's release is postponed by its jitter in the worst case. So, the job's relative deadline is reduced by this amount.

3.4 Dynamic Priorities Fixed on Job Level and Global EDF

Global EDF scheduling suffers as well from Dhall's effect [93]. Hence, the system utilization u_{sys} can drop to zero, there is no useful utilization bound, see Subsection 3.3.1.

This makes it in its pure form inconvenient [33]. But for a utilization-bounded task set with u_{max}, see (1.14), a sometimes useful simple and tight bound for global EDF (3.12) was derived in [124].

$$u \leq u_{max} + m(1 - u_{max}) \tag{3.12}$$

Note that, compared to the global RMS utilization bound (3.5), the omission of the prefactor 1/2 in the second summand expresses the superiority of global RMS to global EDF in terms of utilization bound, in a similar way as we could see it on uniprocessor, cf. Section 2.5.

Due to Dhall's effect, it might be a good idea to apply utilization separation, cf. Subsection 3.3.3. As shown in [26], the algorithm EDF-US[1/2] is optimal in the sense that it maximizes the worst-case schedulable utilization to the theoretical bound of $(m+1)/2$, see Section 4.2. For task sets with constrained and arbitrary deadlines, the corresponding algorithm EDF-DS[1/2], an instance of *EDF with Density Separation*, is optimal in the same sense, [46], p. 51f.

3.4.1 Constrained and Arbitrary Deadlines

For the generalization to sporadic task sets with constrained or arbitrary deadlines, utilizations can be replaced by densities [33] giving (3.13).

$$\delta \leq \delta_{max} + m(1 - \delta_{max}) \tag{3.13}$$

A framework used for an entire class of sufficient global EDF schedulability tests for sporadic task sets was developed by Baker [25].

The basic steps are the following ones:

1. Assume that a taskset is not schedulable by global EDF and consider a concrete a legal set of job release (according to the sporadic task model) with one or more deadline violations.
2. Consider the earliest of these deadline violations at time t_d;
3. Choose a zero time $t_0 < t_d$ and consider in the following the *problem window* $[t_0, t_d)$;
4. Calculate an upper bound of the interference to be done in the problem window and denote it as I_{UB};
5. Establish a necessary unschedulability condition fot the job to miss its deadline t_d;
6. Combine the last two results giving a necessary unschedulability condition in the form of an inequality;
7. Finally, convert the obtained necessary unschedulability test into an sufficient schedulability test by negating the condition.

3.4.2 G-EDF-like Scheduling with Bounded Lateness and Tardiness for Soft RT Systems

For the design of soft RT systems like video processing, cf. 1.1.4, a bounded lateness/tardiness, cf. 1.1.2, allows for setting the jitter buffer size to an appropriate value [104]. In conventional global EDF, the absolute deadlines are the *priority points* (PPs). EDF picks always the job with the earliest PP. By adapting priority points, one obtains priority-based global scheduling algorithms similar to global EDF which will be called *G-EDF-like* [104].

Erickson and Anderson [104] propose an algorithm coined *global fair lateness* (G-FL) which modifies the priority points by bringing them forward by $\frac{m-1}{m}e_i$. This means that tasks with a longer WCET become more important. As an effect, maximum tardiness is improved and all tasks experience equal analytical tardiness bounds which is considered as a kind of fairness [104].

Interestingly, this prioritization on job level[5] is similar to the adaptive DkC approach, see Subsection 3.3.7, with per-task static priorities. This clearly shows the power of approaches with priorities based on linear combinations of deadlines and WCETs.

3.5 Fully-dynamic Priorities

3.5.1 LLF on MPs

In spite of its optimality on uniprocessors, LLF is not widely used there, see Section 2.6, due to the existence of the simpler optimal algorithm EDF. The situation is more promising for the upcoming multiprocessors.

In 1989, Leung [183] proposed the application of LLF on MPs. He found that global LLF is not optimal for systems of more than one processor, but better than global EDF. A further investigation of LLF on MPs including a schedulability test is given in [177]. Similar to the uniprocessor algorithm modification MLLF, see Section 2.6, the number of preemption and, thus, the overhead can be reduced by applying the so-called *Least Laxity-Group First* (LLGF) algorithm [178].

[5]In [104], this is called *job-level static-priority* (JLSP).

Though, there are task sets schedulable by global EDF, but not by global LLF. This means that EDF is not dominated by LLF [150].

3.5.2 Pfair Scheduling

Baruah *et al.* proposed proportionate fair (pfair) scheduling in 1996 [39]. The additional requirement of proportional fairness besides real-time led to an entire family of real-time scheduling algorithms. All of the algorithms in this family are optimal for periodic tasks, they reach 100% utilization. So, the necessary conditions (3.1) and (3.2) become also sufficient ones. Hence they are exact conditions characterizing feasibility and, at the same, time schedulability for this family of algorithms.

The two principles extensively used with pfair scheduling are *proportional fairness* and the simplicity of scheduling when deadlines are equal. The latter one originates from McNaughton's wrap-around approach [206].

A major drawback is, though, that these algorithms result in a large number of preemptions. Thus, they are more of theoretical than of practical relevance.

3.5.3 T-L-plane-based RT Scheduling

Another optimal scheduling approach uses the *time and local remaining execution-time plane* (T-L plane). The driving principle behind the LLREF algorithm [76] is that the largest local remaining execution time tasks are selected first for execution.

In 2009, Funk and Nadadur presented an improved TL algorithm and coined it LRE-TL [115] meaning local remaining execution-TL. They showed that their algorithm is optimal for sporadic implicit-deadline tasks on identical and homogeneous MPs. Additionally, the number of preemptions and migrations has been reduced. In spite of these important improvements, applicability in practice remains arguable.

3.5.4 EDF with Task Splitting and k Processors in a Group (EKG)

The EKG scheduling algorithm was presented by Andersson and Tovar in 2006 [17]. This algorithm has the parameter k to be set in the domain from 1 to m. A separation into light and heavy tasks is governed by a threshold value depending on k. The extreme case of $k = m$ gives another optimal scheduling algorithm. The number of preemptions is restricted to $2k$ per job.

3.5.5 DP-Wrap

A systematization on the principle of *deadline partitioning* (DP) combined with fairness was provided by Levin *et al.* [186] in 2010. A byproduct of this analysis is the DP-Wrap algorithm in the family of DP-Fair algorithms. It uses the idea of wrap-around, first proposed for MP scheduling by McNaughton as early as in 1959 [206].

Wrap-around is a task-to-processor scheme [114] which slices the stack of tasks with lengths corresponding to their densities δ_i into chunks of unit size. The total length of this stack is the task set's total density δ. It is bounded from above by the number of processors m. Otherwise, the necessary criterion (3.3) would be violated. Next, each chunk is assigned to its own processor. Tasks being sliced into two parts[6] migrate between the respective two processors [114].

3.6 Minimally-dynamic Priorities based on Zero Laxity

When the laxity of a job reaches zero, this job needs to be executed for the entire remaining time to its absolute deadline. Otherwise, a deadline violation would occur. Hence, the special treatment of zero-laxity situations is reasonable. A temporary promotion to the highest priority is recommended since

[6]Due to the necessary conditions (3.4) (3.2), the individual density δ_i is at most 1. Hence, a maximum of two parts can be the result of such a slicing. Cases with three or more parts are impossible.

it is the only scheduling strategy allowing for meeting the zero-laxity job's deadline.

3.6.1 Earliest Deadline first until Zero Laxity (EDZL)

The EDZL algorithm by Lee *et fal.* [179] is an extension of the EDF algorithm. EDF is strictly dominated by EDZL. EDZL is not optimal on MPs. In [274] and [73], it was shown that the system utilization bound of EDZL on an MP with $m \to \infty$ is in the interval (3.14). Note that the lower bound of the utilization bound of EDZL was shown by complete induction on m in [240].

$$0.5 \leq u_{sys,lub,\mathrm{EDZL}} \leq 1 - 1/e \approx 0.632 \tag{3.14}$$

3.6.2 Earliest Deadline first until Critical Laxity (EDCL)

Kato and Yamasaki [158] published a modification of EDZL in 2008. There, a calculated critical laxity value is used to promote a task even before a zero-laxity situation occurs. This allows for a relaxation in the laxity checks. Only at job releases and completions, laxity needs to be checked. With EDZL, laxity has to be monitored continuously. Schedulability is slightly reduced compared to EDZL, but still much better than with global EDF. Hence, EDCL is considered a good trade-off between schedulablity and implementation complexity of the scheduler.

3.6.3 Fixed-Priority until Zero Laxity (FPZL)

Zero-laxity situations in static-priority scheduling were considered by Davis and Burns [85]. Again, FPZL dominates global static priority scheduling. It achieves schedulability levels comparable to EDZL.

Rate-monotonic until Zero Laxity (RMZL)

The special case of rate-monotonic scheduling taking zero-laxity situations into account was published in [162], [94] and [230] in 2009. A further contribution explaining the approach is [156] as a translation of [265].

Rate-monotonic until Critical Laxity (RMCL)

Kato and Yamasaki [159] published a modified RMZL called RMCL in 2008. The idea is again to restrict the set of scheduling points to job releases and completions. The maximum number of context switches per job is two [88]. Potential negative laxities at scheduling points are detected as early as possible and, then, prevented by an immediate promotion of the respective job to the highest priority level. A generalization of RMCL allowing also for other than RM priority orders is *Fixed Priority until Critical Laxity* (FPCL) introduced in [88].T

3.6.4 Fixed Priority until Static Laxity (FPSL)

Davis and Kato introduced FPSL in 2012 [88]. Opposed to the dynamic detection of a critical-laxity situation with EDCL and RMCL, pre-computed static *laxity thresholds* X_i are used for obtaining promotion times [88]. An obvious upper bound (equality excluded) of a task's laxity threshold is (3.15) since otherwise, each job of such a task would enter "[..]the critical-laxity state as soon as it is released." [88] With the other extreme, a laxity threshold of zero for all tasks, FPSL turns into FPZL since then, priority promotion is restricted to zero-laxity situations [88].

$$X_i < D_i - e_i \tag{3.15}$$

Schedulability tests for FPSL can also be applied to FPCL [88].

Hence, FPSL is very general and can be seen as an advancement[7] of Dual Priority Scheduling (DPS) or Limited Preemptive Scheduling (LPS), cf. Section 2.7. The tail part in a LPS-scheduled task corresponds then to the critical-laxity part where the job's laxity falls below the task's laxity threshold. On the other hand, there is some offline knowledge needed, *laxity thresholds* X_i for FPSL and *promotion points* for LPS. This makes experimental evaluation more complicated since only feasible task sets can bet taken into account for FPSL evaluations [88].

[7]At first glance, one might expect that it is closer related to Fixed-priority Preemption Threshold Scheduling (FPTS) due to the naming. But this is not true since the threshold in FPTS refers to a priority level while the threshold in FPSL refers to a period of time.

3.7 Summary

Global scheduling using a straightforward generalization of EDF or RMS suffers from Dhall's effect, see Subsection 3.3.1, which reduces the system utilization bound to zero. By applying *utilization separation*, see Subsection 3.3.3, Dhall's effect can be circumvented

For global fixed-priority scheduling, the uniprocessor TDA can be generalized then becoming only a sufficient test. It uses a raw estimation of the maximally possible interference and is, thus, pessimistic.

Another option is to apply the fully dynamic algorithms LLF, pfair scheduling, or EDZL/RMZL. Here, the latter ones treating zero-laxity situations specifically seem to be promising. LLF and, in particular, pfair scheduling algorithms are prone to frequent preemptions.

Pfair scheduling, cf. Subsection 3.5.2, is over-specified by its fairness requirement which is not necessary to meet deadlines. Fairness will be relaxed in Section 5.3.

The emerging algorithm FPSL has promising properties due its combination of dynamic (laxity) and static (precalculated laxity thresholds) features.

Chapter 4

Partitioned Scheduling

Partitioned scheduling is based on a fixed task-processor relationship. Entire tasks are mapped to a processor. Thus, each subsystem of a single processor behaves like a uniprocessor. Since uniprocessor scheduling is widely solved, see Chapter 2, the mathematical principle of *reduction to the known* applies here. The remaining challenge is the proper distribution of the tasks to the processors. This problem is closely related to *bin packing* and, thus, NP-Hard in the strong sense[1] [119]. Due to this intractability, heuristics for finding suboptimal approximate solutions are widely used.

During the eighties and nineties of the 20th century, after Dhall's effect [93] was found, partitioned scheduling was the favorite approach for most of the research community. Later, it was shown that Dhall's effect becomes apparent only in exotic situations with a mix of high and low utilization tasks. Hence, global scheduling is a realistic option for practical scheduling. Research on global scheduling was intensified in the decade 2000-2009.

Meanwhile, the pendulum is swinging into the other direction again. A huge application area is car electronics. There, the classical approach of a one-to-one mapping between applications and *electronic control units* (ECUs) is reaching its limit. When the number of ECUs goes beyond 70 or 80, complexity, mass of the equipment and energy consumption[2] become issues. Hence, it is intended

[1]This means that there is no fully polynomial-time approximation scheme (FPTAS) for bin packing unless $\mathsf{P} = \mathsf{NP}$.

[2]Note that energy consumption is closely related to the mass of the car and its acceleration by Newton's second law. In a more accurate analysis, not only the car's acceleration, but also changes in the car's mass due to gas consumption should be considered. In its general

to place several applications on one and the same ECU. This is partitioned scheduling *par excellence*.

A recent article [235] tries to take cache effects into account which are not considered in our model, cf. Subsection 1.9.1. The approach is based on a generalization of the WCET to a so-called WCET-matrix, a collection of WCETs of a task depending on the concrete execution environment.

4.1 Necessary Schedulability Tests

Despite the big conceptual differences between global and partitioned scheduling, the necessary schedulability conditions remain with (3.1), (3.2), (3.3) and (3.4) the same.

At first glance, one could expect that the conditions become tighter for partitioned scheduling. But this does not happen since a perfect fit (no waste), see (3.1), of the individual utilizations u_i is possible with the partitioned approach, and a too heavy task, see (3.2), would require a task split in both global and partitioned approach.

Second, the demand needs to bounded from above (3.3) independent of the scheduling approach chosen since the deadlines are requirements, and the demand considers all portions of jobs which are released and need to be completed in a time window.

4.2 Utilization Bound

Partitioned Scheduling is characterized by the fact that a single task can block a processor when having a utilization of $0.5 + \varepsilon$ with $\varepsilon > 0, \varepsilon \to 0$ and all other tasks have the same utilization value. For blocking m processors, m such tasks are needed. Then, the $(m+1)$-th task cannot be assigned to one of these m processors. An infinitesimally small decrease in its u_i value by 2ε would render the task set schedulable since, then, it would exactly[3] fit to an arbitrary one

form, Newton's second law says that force is equal to the time rate of change of the linear momentum p, i.e., $F = \frac{\mathrm{d}(mv)}{\mathrm{d}t}$. On the other hand, the rate of change of mass in a car is quite low and often negligible.

[3]without any waste, i.e., idle processor time

of the m processors. So, the utilization bound for partitioned scheduling is the total utilization in this worst-case situation: $\frac{m+1}{2}$. Asymptotically, for $m \to \infty$, it becomes—normalized to system utilization—0.5, cf. Subsection 1.6.5. This utilization bound is valid for all fixed-priority multiprocessor scheduling algorithms, both partitioned and global ones [13].

Even for all eight remaining combinations of prioritization dynamics, see Subsection 1.4.1, with migration dynamics[4], see Subsection 1.6.7, except to fully-dynamic global scheduling approaches, cf. Section 3.5, the maximum utilization bound is $\frac{m+1}{2}$.

Only 50% of the potential processing capacity can be used for sure, see (4.1) and (4.2) formulæ for utilization bound and system utilization bound. This upper bound can serve as a sufficient schedulability test, but a *non-constructive* one. So, for constructing a schedule, we need to use the mapping of tasks to processors which is NP-Hard, see the introduction to this Chapter. Such mappings can be obtained using heuristics, see Sections 4.3 and 4.4, if the brute-force method of testing all task-to-processor mappings becomes intractable.

$$u_{lub,part} = \frac{m+1}{2} \tag{4.1}$$

$$u_{sys,lub,part} = \frac{m+1}{2m} \tag{4.2}$$

4.3 Partitioned EDF

4.3.1 Relationship to Bin Packing

Partitioned EDF with implicit deadlines is very closely related to bin packing. Each bin's capacity is set to the normalized processor speed which equals 1 on an identical multiprocessor system. Nevertheless, a subtle difference is that classical bin packing assumes a given set of items (n tasks) with a total utilization u and minimizes the necessary number of bins (processors), $m \to$ min, see (4.3). More typical for partitioned scheduling, we have a given hardware platform with a fixed number of processors m. Then, the number of tasks in a feasible task set is maximized or an upper utilization bound with $u \le u_b$, cf.

[4]This classification was presented in [70]. For further explanations, see also [221].

Subsection 1.7.4, guaranteeing schedulability is calculated. Achieving the best possible such bound corresponds to $u \to \max$, see (4.4).

$$m \to \min \qquad \text{subject to} \qquad u = \text{const.} \tag{4.3}$$

$$u \to \max \qquad \text{subject to} \qquad m = \text{const.} \tag{4.4}$$

4.3.2 Constrained Deadlines

When generalizing to constrained deadlines, partitioned EDF is no longer easy since it must be based on constrained-deadline EDF uniprocessor schedulability tests [40]. A constant-time admission control for partitioned EDF with constrained deadlines was presented by Masrur *et al.* in [205].

4.3.3 Arbitrary Deadlines

The scheduling of arbitrary-deadline sporadic task sets was studied using DBFs, see Subsection 2.4.1, and their approximations in [36] and [74].

Baruah and Fisher use an approximated DBF and the allocation heuristic First Fit (see Subsection 4.3.4) for tasks presorted (see Subsection 4.3.5) according to non-decreasing relative deadlines [36]. They quantify the trade-off of changing to more general models of constrained-deadline and arbitrary-deadline task sets.

Chen and Chakraborty present a new partitioning scheme based on vector scheduling [74]. The vector packing problem is a generalization of the bin packing problem to multiple dimensions. For this scheduling problem, there are two dimensions to consider: the task densities δ_i and the task utilizations u_i. The density test (2.38) is applied.

4.3.4 Allocation Heuristics

Popular allocation heuristics are Next Fit (NF), First Fit (FF) and Best Fit (BF). BF yields the best performance, but there is only a marginal difference to FF. Hence, FF is considered most suitable because of its lower average calculation effort [202]. Worst Fit (WF) as a fourth approach aims at load balancing and reduction of energy consumption. Balancing is reasonable since there is a supralinear influence of frequency on power consumption [22].

In [196], the concept of a *reasonable* allocation algorithm is introduced. There, we have:

> "By a reasonable allocation algorithm we means one which fails to allocate a task only when there is no processor in the multiprocessor with sufficient capacity to hold the task. For example, FF and the optimal allocation algorithm are reasonable.", [196]

According to this definition, all mentioned heuristics (NF, FF, BF, WF) are reasonable ones.

4.3.5 Presorting

Originally, these allocation heuristics are invented for online algorithms. When changing to offline algorithms, an additional degree of freedom is the sequence of the tasks presented to the allocation heuristics. This sequence can be modified by a presorting taking $\mathcal{O}(n \log n)$ time. A good presorting criterion for partitioned EDF is Decreasing Utilization (DU) as First Fit Decreasing (FFD) is well-suited for bin packing. Nevertheless, it is not optimal. But there is always at least one permutation of tasks yielding an optimal solution when using First Fit [187].

4.3.6 Utilization Bound

The utilization bound of partitioned EDF with implicit deadlines remains at $(m+1)/2$ as in the general case, see Section 4.2. This has been shown in [196] and [195] by López *et al.* The proof there is for any reasonable allocation strategy, see Subsection 4.3.4.

According to [196], the bound can be refined when restricting the maximum utilization of a task, see 1.2.1. The result obtained there is given in (4.5).

$$u \leq \frac{\lfloor 1/u_{max} \rfloor m + 1}{\lfloor 1/u_{max} \rfloor + 1} \tag{4.5}$$

Striking is the double appearance of the $\lfloor 1/u_{max} \rfloor$ term. It has a special interpretation, the minimum number of tasks with a u_{max}-limited utilization fitting onto one processor. Hence, each task set containing at most $\lfloor 1/u_{max} \rfloor m$ tasks is schedulable by partitioned EDF. Thus, the fulfillment of this condition or the validity of (4.5) is sufficient for partitioned EDF schedulability.

4.4 Partitioned RMS

Scheduling using partitioned RMS is more complicated than partitioned EDF since period values have to be taken into account. A simple estimation from below using the most general Liu/Layland bound of $\ln 2 \approx 0.69$ wastes more than 30% of potential CPU resources on the level of uniprocessor scheduling. Since widely applied mapping heuristics like First Fit generate additional waste, the net processor waste becomes unacceptable.

The separate tuning of uniprocessor schedulability test and allocation strategy is an option of minor success. The reason is that a presorting of tasks as an additional preprocessing step shall be used before allocation heuristics act. Such a presorting increases the chances for a fit in the uniprocessor test drastically.

4.4.1 Utilization Bound

The utilization bound of partitioned RMS was shown to be at least $(\sqrt{2} - 1)m \approx 0.414m$ by a constructive proof using First Fit [231]. By taking maximum task utilization u_{max} into account, López *et al.* later on [194] [197] improved this bound to (4.6) for partitioned RMS using FF[5].

$$u \leq (m-1)\left(2^{\frac{1}{k_{max}+1}} - 1\right) k_{max} + (n - k_{max}(m-1))\left(2^{\frac{1}{n-k_{max}(m-1)}} - 1\right) \quad (4.6)$$

Here, k_{max} is the minimum number of tasks with a u_{max}-limited utilization fitting onto one processor. It is calculated by (4.7), cf. [197].

$$k_{max} = \lfloor 1/\log_2(u_{max} + 1) \rfloor \quad (4.7)$$

Clearly, this function is bounded from above by $\lfloor 1/u_{max} \rfloor$ obtained for partitioned EDF, see Subsection 4.3.6, since the total utilization on one processor

[5]Applying the slightly better allocation strategy *Best Fit Decreasing* (BFD), the bound can be further improved to $(mk_{max} + 1)\left(2^{1/(k_{max}+1)} - 1\right)$ as shown in [194].

is limited by the LL bound. Analogously, a task set containing at most $k_{max}m$ tasks is schedulable by partitioned RMS. Thus, the fulfillment of this condition or the validity of (4.6) indicates partitioned RMS schedulability.

The utilization gain of the refined bound (4.6) compared to the simple estimation $(\sqrt{2}-1)m \approx 0.414m$ becomes significant for a low number of processors m and a low maximum utilization u_{max}. If the maximum utilization is small enough, $u_{max} \to 0$, the refined utilization bound becomes $m \ln 2$. This means, independent of the number of tasks, we can assign tasks up to a system utilization of $\ln 2$ as with RMS uniprocessor scheduling, see (2.11). Hence, a linear speedup[6] can be achieved in the special case of infinitely light tasks which is considered the optimum, see also Subsections 3.1.1 and 5.1.2 where the transfer of the LL bound to MPs is achieved for arbitrary task utilizations u_i but using *task splitting.*

Another interesting limit case of (4.6) being at the same time a safe bound is to make the bound independent of the number of tasks n by $n \to \infty$. The second summand in the sum then converges to $\ln 2$ which gives (4.8).

$$u \leq (m-1)\left(2^{\frac{1}{k_{max}+1}}-1\right)k_{max} + \ln 2 \tag{4.8}$$

R-BOUND-MP-NFR (Next Fit Ring) [16] which will be shortly described in 4.4.3 reaches the optimal system utilization bound of 50%, cf. Section 4.2. Despite this seminal result, it does not mean that R-BOUND-MP-NFR is an optimal partitioned approach in the sense that it can schedule any task set that is schedulable according to any other partitioned approach, cf. Subsection 1.7.5.

Next, the average-case performance can be more meaningful. Steps in this direction will be taken in Chapter 7.

4.4.2 Rate-Monotonic Small Tasks (RMST) and Variants

The first algorithm for partitioned RMS taking the tasks' individual periods for a presorting into account was Rate-Monotonic Small Tasks (RMST), see

[6]Unfortunately, the factor $\ln 2$ remains in the multiprocessor setting. But this is still giving a linear speedup since the $\ln 2$ factor is also present in the uniprocessor LL bound.

Algorithm 4.1 by Burchard *et al.* [60]. There, the fact that task sets being close to simply periodic in terms of their period value deviations from a simply periodic pattern yield a utilization bound close to that of simply periodic task sets, i.e., 1. Thus, a presorting according to increasing decimal fractions of the periods' logarithms is reasonable. These values are called S values and were already defined in (2.18) in the context of Burchard's uniprocessor schedulability test, see 2.2.2.

Algorithm 4.1 Rate-monotonic Small Tasks (RMST) by Burchard *et al.* [60]

```
1: {sort task set according to non-decreasing $S$ values defined in (2.18)}
2: $S_{n+1} := S_1 + 1$
3: $i := 1$ {task index}
4: $j := 0$ {processor index}
5: $j := j + 1$ {new empty processor}
6: $v_j := u_i$ {assign task $i$ to processor $j$; per-processor $u$ is $v$}
7: $S := S_i$
8: $\beta_j = 0$
9: $i := i + 1$ {process next task}
10: if $i > n$ then {all tasks assigned}
11:     $\beta_n := S_i - S$ {calculate $\beta$ value of current task}
12:     exit
13: end if
14: $\beta_j := S_i - S$
15: if $u_i + v_j \le \max(1 - \beta_j \ln 2, \ln 2)$ then {sBu, see (2.21)}
16:     $v_j := v_j + u_i$
17:     go to 9 {next task}
18: else
19:     go to 5 {next (new) processor}
20: end if
```

Interestingly, the logarithm base 2 was chosen in (2.18). This might be a good choice, but it is obviously not the only possible one. We will come back to this later in 7.3.3.

RMST uses the Next Fit heuristics, see Subsection 4.3.4, and the simplified Burchard's test, see (2.21) in 2.2.2, to allocate presorted tasks to the processors.

The asymptotic approximation ratio of this algorithm is $1/(1 - u_{max})$ depending on the maximum task utilization [60]. The utilization bound of RMST is (4.9), see [60], with a wide influence of the maximum utilization deteriorating performance. This calls for a special treatment of heavy tasks what will be done in RMGT.

$$u \leq (m-2)(1-u_{max}) + 1 - \ln 2 \tag{4.9}$$

Later on we will present a variety of modifications on RMST and discuss them in Section 7.4.

Rate-Monotonic General Tasks (RMGT)

Burchard *et al.* noted that high-utilization tasks are not well considered in the simple scheme RMST. They can block processors. RMST works only well if the utilization of all tasks is limited by 0.5 [60].

The idea of RMGT, see Algorithm 4.2, is to subdivide the task sets into two subsets, those with $u_i \leq 1/3$ (light) and the others with $u_i > 1/3$ (heavy). The light tasks are then allocated using RMST. Subsequently, heavy tasks are assigned using the First Fit heuristics and the exact two-task criterion, a special case of TDA, see (2.6) and (2.7) in Subsection 2.2.1.

The asymptotic approximation ratio of this algorithm is 1.75 according to [60]. The utilization bound of RMGT is (4.10), see [60].

$$u \leq \frac{1}{2}\left(m - \frac{5}{2}\ln 2 + \frac{1}{3}\right) \approx \frac{1}{2}(m - 1.400) \tag{4.10}$$

4.4.3 R-BOUND-MP

R-BOUND-MP was suggested by Lauzac *et al.* [174]. It rearranges tasks after a transformation of all task set periods to the same (binary) order of magnitude. This scaling operation called *ScaleTaskSet* uses the binary logarithm of the ratio between maximum and own period, see (2.22). The tasks are sorted according to increasing transformed periods $p_i{}'$. Thus, clusters of tasks with well-fitting periods are formed. Then, FF and the R-BOUND uniprocessor criterion based on the ratio between maximum and minimum period (2.23) is applied for the assignment of tasks to processors.

Algorithm 4.2 Rate-monotonic General Tasks (RMGT) by Burchard *et al.* [60]

```
1: {Partition the task set into two subsets $G_1$ of light tasks and $G_2$ of heavy tasks with threshold 1/3 and tasks with $u_i = 1/3$ allocated to $G_1$}
2: {Apply Algorithm 4.1 (RMST) for assigning tasks in $G_1$}
3: $i := 1$ {task index in $G_2$}
4: $j :=$ index of next empty processor {new empty processor}
5: $v_j := u_i$ {assign task $i$ to processor $j$; per-processor $u$ is $v$}
6: $i := i + 1$ {process next task}
7: if $i > |G_2|$ then {all tasks of $G_2$ assigned}
8:     exit
9: end if
10: if $\exists$ processor with only 1 task, say $T_k$, and
    $\left\lfloor \frac{p_k}{p_i} \right\rfloor (p_i - e_i) \geq e_k \vee p_k \geq \left\lceil \frac{p_k}{p_i} \right\rceil e_i + e_k$
    then {condition (2.6), w.l.o.g. $p_i \leq p_k$, First Fit}
11:     $v_j := v_j + u_i$ {mark processor $j$ as full}
12:     go to 6 {next task}
13: else
14:     go to 4 {next (new) processor}
15: end if
```

The (cyclic) presorting sequence with R-BOUND-MP is the same as with RMST [220]. This fact clearly shows the close relationship of the two independently found algorithms.

R-BOUND-MP-NFR

Interestingly, R-BOUND-MP served as a basis for the breakthrough algorithm R-BOUND-MP-NFR (Next Fit Ring) with 50% utilization bound [16]. There, FF is replaced by NFR. The difference compared to NF is the *second chance* of allocating a task to the first processor when not succeeding allocating it to the last available processor. Note that this additional policy violates the principle that closed bins are never touched again which is a characteristic of classical NF.

4.4.4 First Fit Matching Periods (FFMP)

The FFMP algorithm published by Karrenbauer and Rothvoß [153] is closely related to RMST. The only difference is that NF has been replaced by FF. Then, the average-case behavior is significantly improved. It is shown [153] that the expected approximation ratio is $1+\mathcal{O}(n^{-1/4}(\log n)^{3/8})$ tending to 1 for $n \to \infty$. This means that the solution is asymptotically optimal on average. The assumed utilization distribution is uniform.

4.4.5 k-Rate-Monotonic-Matching (k-RMM)

The k-RMM algorithm was suggested in 2011 by Karrenbauer and Rothvoß [154]. As an advancement of RMGT, it uses utilization bands and an exact scheduling criterion for two tasks, too. But here, even three groups of tasks are distinguished: light, medium and heavy. The two thresholds are 1/3 and approximately 1/2. The second threshold is defined as $\frac{1}{2} - \frac{1}{12k}$, making it dependent upon the parameter k. A reasonable choice for k is $\lfloor\sqrt{n}\rfloor$ according to [154]. The light tasks are further subdivided into k utilization bands. A greedy maximal matching[7] is calculated using special weights of nodes representing the tasks. There is a task-to-task edge iff the two considered tasks

[7]The idea of using matchings comes from a group of bin packing algorithms. For a survey on that, see [79].

are RMS-schedulable determined by an exact scheduling criterion. The weight of each node is the average number of processors necessary for scheduling a large number of (imagined) tasks having all the original task's utilization. This value is $u_i/(1-u_i)$ for light tasks, 1/2 for medium tasks and 1 for heavy tasks.

For all edges in the matching, a separate processor is opened. Finally, from heavy to light, tasks are assigned to processors using FFMP for all tasks in the respective utilization band.

The algorithm achieves an asymptotic approximation ratio of 1.5.

4.4.6 Algorithms Based on the Exact Uniprocessor Test TDA

The algorithms hitherto presented for partitioned RMS are all mainly based on *sufficient* tests on the uniprocessor level. But the exact test TDA, cf. Subsection 2.2.1, can also be employed as it was suggested for heavy tasks in RMGT, cf. 4.4.2.

An algorithm called FFD-DM[8] or synonymously RT-FFD was presented in [108] for sporadic task sets with arbitrary deadlines which is a generalization of the Liu/Layland task model in two dimensions. Note that FFD-DM remains a heuristic for the solution of the NP-Hard problem of the distribution of tasks to processors in spite of using the exact TDA on the uniprocessor level.

By using a linear approximation of the TDA, the algorithm FFB-FDD[9] was proposed as an relaxation of RT-FFD of with $\mathcal{O}(n^2)$ instead of pseudo-polynomial complexity[10]. [110].

[8]This means a variant of First Fit Decreasing, but now combined with a deadline-monotonic sequence which gives a list of tasks sorted by non-decreasing relative deadlines. The First Fit heuristic operates then on this list. So, the given name of the algorithm might be misleading.

[9] "FFB" origins from the initial letters of the authors' surnames Fisher, Baruah and Baker.

[10]FFD-DM has inherited pseudo-polynomial complexity from its building block TDA.

4.5 Comparison of Some Utilization Bounds For Common Parameter Settings

In order to get an impression of the utilization bounds given by equations in this Chapter, we present the concrete values for common parameter settings in Table 4.1.

m	$u_{max} = 1$	$u_{max} = 0.5$	$u_{max} = 0.25$	$u_{max} = 0.1$	$u_{max} \to 0$
2	15	15	15	15	15
	15				
	55	55	63	69	69
	75	83	90	95	100
5	6	36	51	60	66
	36				
	47	47	59	65	69
	60	73	84	93	100
10	3	43	63	75	83
	43				
	44	44	58	64	69
	55	70	82	92	100
50	1	49	73	87	97
	49				
	42	42	57	63	69
	51	67	80	91	100
$\to \infty$	31	50	75	90	100
	50				
	41	41	57	63	69
	50	67	80	91	100

Table 4.1: System Utilization Bounds (in %) of RMST (4.9), RMGT (4.10), partitioned RMS (4.8) and partitioned EDF (4.5) for Typical Values of m and u_{max} (Rounded)

In Table 4.1, system utilization bounds of three partitioned RMS approaches are compared to that for partitioned EDF based on typical settings of the number of processors m and the maximum utilization u_{max}. Note that the second value for RMGT is only printed for tasks with an arbitrary utilization ($u_{max} = 1$) since this algorithm was specially designed for such general tasks.

We see that RMGT outperforms RMST in the arbitrary-utilization ($u_{max} = 1$) case consistently. Also, partitioned EDF is throughout better than partitioned RMS which does not surprise due to the additional problem of period fitting with RMS. In general, for small tasks on a high number of processors, we get very competitive utilization bounds with RMST and partitioned EDF, recommending them to be applied. Nevertheless, RMST can be further improved especially in its average-case behavior as we will see in Chapter 7.

4.6 Summary

With partitioned approaches, there is no Dhall's effect. But the problem of a static mapping of tasks to processors is NP-Hard and can become intractable. Advanced algorithms like RMST or RMGT take period values into account to assign tasks to processors. For non-exotic task sets, it could be shown that utilization bounds are much greater than the theoretical bound of 50%, cf. Section 4.5.

Chapter 5

Hybrid Scheduling Approaches

As we could see in Chapters 3 and 4, the two extreme approaches of global and partitioned scheduling have some pros and cons. Thus, it is natural to combine them in order to profit from their respective advantages. It will turn out that some of the new approaches can even be regarded as generalizations of both partitioned and global scheduling.

At the end of this Chapter in Section 5.3, we will regard cutting-edge algorithms which relax the usually applied fairness condition to the benefit of a significantly reduced number of preemptions and migrations.

5.1 Semi-partitioned

With semi-partitioned scheduling suggested by Anderson *et al.* [9], some tasks are distributed to processors according to the partitioned approach. The rest of the tasks is split up into subtasks which are then assigned to processors with sufficient rest capacity. Note that the initial publication [9] describes a soft RT scheduling algorithm called EDF-fm (fixed or migrating) whose principles where used for the development of corresponding hard RT scheduling algorithms later on. For normal tasks, no migration at all is allowed while for split tasks, migration occurs in a regular pattern and becomes the rule.

5.1.1 EDF-based Algorithms

With EDF as a basis, there are three hard RT scheduling algorithms to mention: *EDF with Window-constraint Migration* (EDF-WM) [164], *Earli-*

est Deadline Deferrable Portion (EDDP) [160] and *Earliest Deadline First with the highest-priority deferrable portion-2 task - Sequential assignment in Increasing Period* (Ehd2-SIP) [157]. EDDP and Ehd2-SIP are both designed for task sets with implicit deadlines and have system utilization bounds of $4\sqrt{2} - 5 \approx 0.657$ and 0.5. EDF-WM works even on the general class of task sets with arbitrary deadlines.

5.1.2 RMS-based Algorithms

Combinations of partitioned and global RMS, coined *Rate Monotonic Deferrable Portion* (RMDP) [161] and *Deadline Monotonic with Priority Migration* (DM-PM) [163] have been suggested, too. Note that DM-PM specializes to RM-PM when considering implicit deadlines where deadlines equal periods. Another algorithm belonging to this group is *Partitioned Deadline-Monotonic scheduling with Highest Priority Task on each processing core is allowed to be Split* (PDMS-HPTS) given in [172].

The system utilization bounds of these fixed-priority semi-partitioned algorithms are 0.5 both for RMDP and DM-PM which equals the theoretical upper bound of partitioned scheduling algorithms. A first improvement to about 0.6547 was achieved by PDMS-HPTS, see [172]. PDMS-HPTS uses a threshold value of 0.25 to distinguish between heavy and light tasks, cf. [172]. It assigns the highest priority to split tasks and, thus, exploits the fact that highest priority tasks are executed completely undisturbed. Thus, it is closely related to the utilization separation approaches discussed in Subsection 3.3.3. The algorithm EKG-2, see [17], a variant of EKG (EDF with task splitting and k processors in a group), see Subsection 3.5.4 is a semi-partitioned algorithm with a system utilization bound of $2/3 \approx 0.667$ and a maximum of 4 preemptions per job on the hyperperiod average, cf. [17]. The practical relevance of semi-partitioned algorithms was recently confirmed in [42]. There, real-world overheads are taken into account.

Liu/Layland Bound on MPs

A breakthrough by Guan *et al.* was worked out in 2010. The famous Liu/Layland (limit) bound for uniprocessors ($\ln 2 \approx 0.693$) could be transferred to multiprocessors for the first time. Their algorithm *Semi-partitioned algorithm 2* (SPA2) is given in [131]. This seminal algorithm differs from the other ones in the allocation policy to the processor. While First Fit was the standard before, Guan *et al.* rely on the Worst Fit strategy. With it, the processors are filled in turn, improving on the chances for the split tasks to find an appropriate processor. Note that the core idea (and as well another application field) of the Worst Fit strategy is load balancing. Another clever move in SPA2 is the discrimination into heavy and light tasks known from the hybrid prioritization dynamics algorithms, see Subsection 3.3.3. The specialty here is that the threshold value is a function of n and not of m, see [131].

Beyond Liu/Layland Bound on MPs

Even higher system utilization bounds for semi-partitioned RMS-based algorithms haven been achieved by considering period compatibility.

In [151], R-BOUND, see 2.2.2, is used as uniprocessor utilization bound taking period compatibility into account. Their algorithm coined *Period-COMPatible-Allocation and Task-Splitting* (pCOMPATS) achieves a 0.72 system utilization bound. pCOMPATS combines ideas from R-BOUND-MP-NFR, see 4.4.3, and *Highest Priority Task Splitting* (HPTS) [172]. An excellent property of this approach is the convergence of schedulable system utilization to 1 as the number of cores tends to infinity. In the average case, the schedulable system utilization goes up to 0.92 for 4 cores and to 0.99 for 32 cores [151].

The algorithm RM-TS presented in [132] can even reach a system utilization bound[1] of $\frac{2\ln 2}{1+\ln 2} \approx 0.818$. It relies on the uplifting of *parametric utilization bounds* (PUBs) like Liu/Layland bound, see 2.2.2, or R-BOUND, see 2.2.2, from the uni- to the multiprocessor situation. Due to the use of exact response time analysis (TDA, see Subsection 2.2.1), average-case behavior is largely improved compared to previous approaches [132].

[1]Note that this utilization level is incidentally very close to the Liu/Layland bound for two-task task sets $u_{LL}(2) = 2(\sqrt{2}-1) \approx 0.828$. This is no typo, and there is no direct interrelationship as can be seen by the fairly different terms with different origins.

Simply Periodic Task Sets on MPs

For the special case of simply periodic task sets, see Part 1.2.1, an optimal algorithm with a system utilization bound of 1 is presented in [148]. This bound decreases when not all tasks can be split arbitrarily. Their approach is general enough to provide a solution even for the more general model of uniform MPs.

5.2 Cluster-based Approaches

Semi-partitioned scheduling appears to be a good compromise between partitioned and global scheduling. But there exist more general concepts which will be discussed in the following.

5.2.1 Physical Clusters

Recently, the concept of semi-partitioned scheduling was generalized to cluster-based multiprocessor scheduling, see [95]. Semi-partitioned scheduling partitions tasks into pinned and migrating ones, cf. [163]. Thus, some tasks are split into subtasks, making the migration more regular. In contrast to that, cluster-based scheduling tackles the problem from the resources side, the processors/cores. First, physical clusters with one-to-one mappings between their processors and the processors of the platform can be created. Inside each cluster, global scheduling is applied, whereas the cluster formation itself resembles the partitioned approach.

Hence, a single cluster of m processors corresponds to global scheduling. The other extreme of m clusters of size 1 matches partitioned scheduling.

Tests on a concrete platform [41] showed that global EDF behaves worse than partitioned EDF for hard RT systems. The new approach of cluster-based scheduling is typically found between these two extreme approaches when it comes to a comparison according to schedulability success ratios.

5.2.2 Virtual Clusters

While for physical clusters, cluster boundaries are restricted to be aligned with processor boundaries. For the generalization to virtual clusters, cluster boundaries can cross processors [95]. Hence, virtual clusters can share one or several processors. So, it turns out that semi-partitioned scheduling can be regarded as a special case of virtual-cluster scheduling since it uses virtual clusters of size 1 for the non-split tasks and virtual clusters containing at least 2 processors for the split tasks.

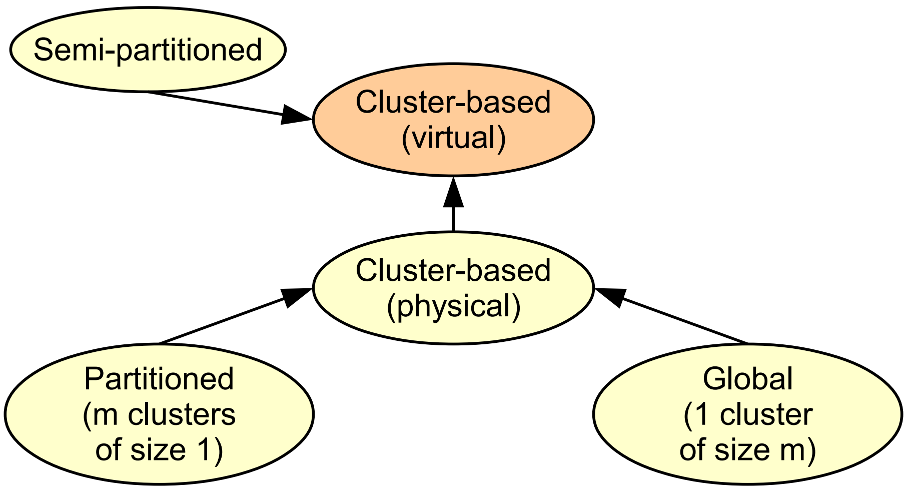

Figure 5.1: Cluster-based Scheduling as a General Concept on Migration Dynamics; The Directed Edges Denote the Direction of Generalization

Summing up, see Figure 5.1, the mighty concept of cluster-based scheduling is able to subsume both global and partitioned scheduling as well as the semi-partitioned approach. For partitioned scheduling, there are m clusters of size 1 each. The dual situation of just 1 cluster with m processors applies for global scheduling. By the parameter *cluster size* several intermediate levels combining partitioned and global scheduling can be achieved, cf. [41]. There, it is also shown that cluster size choices are often determined by the cache hierarchy. So, for a three-level cache-hierarchy, eligible choices of the cluster size are those on the L2 and L3 level[2] , see [41].

[2]Note that the L1 caches are typically private per-core caches. Hence, they would yield a cluster size of 1, resulting in partitioned scheduling.

5.3 Less Preemptions and Migrations by Less Fairness

The requirements of *hard real time* and *fairness* are orthogonal ones. Optimal global scheduling algorithms like pfair, LLREF, LRE-TL, EKG and DP-Wrap presented in Section 3.5 rely on this over-constraining and apply fairness which “forces tasks to march in step with their fluid rate curves more precisely than is theoretically necessary.” [186]

Hence, a reasonable approach is to abandon fairness. We expect that by concentrating on the hard real-time requirement, a lower overhead in terms of the expected number of migrations and preemptions can be achieved.

5.3.1 RUN: Reduction to a Series of Uniprocessor Problems

Very recently, Regnier *et al.* proposed a new approach which solves the multiprocessor scheduling problem by a reduction to several uniprocessor scheduling problems in their seminal paper [251]. They coined their algorithm *RUN (Reduction to UNiprocessor)*. RUN is an optimal scheduling algorithm with at most 3 preemptions per job observed [251].

It results in significantly less preemptions than typically obtained using pfair algorithms, see Subsection 3.5.2, since it uses only a weak version of *proportional fairness* which exploits duality and uses the concept of servers [251].

More exactly, according to [251], the average number of preemptions per job on an m-processor system is upper-bounded by $\mathcal{O}(\log m)$. In all simulations performed, there have been observed fewer than 3 preemptions per job [251].

Interestingly, RUN employs a semi-partitioned approach and reduces to partitioned EDF, see Section 4.3, for moderate system utilizations since a first-stage number of $\leq m$ servers can be physically provided by the hardware using dedicated processors and does not need to be simulated [251]. Servers are virtual tasks with a utilization set to the total utilization of the task set involved. Such servers enforce weak proportional fairness [251].

5.3.2 U-EDF: Unfair Scheduling Algorithm based on EDF

In 2011, another unfair scheduling algorithm called U-EDF was published [226]. Its basic idea is to fill the processors in a priority-driven manner with highest-priority tasks on the first processor. Since U-EDF is based on EDF, the earliest absolute deadline of a job gives it highest priority. U-EDF is an optimal scheduling algorithm with less than one preemption and one migration per job [226].

It can be interpreted as a generalization of uniprocessor EDF in a *horizontal* manner executing "[..] as much as possible on as few processors as possible." [227] Opposed to that, global EDF has extended uniprocessor EDF in a vertical manner which maintains the principle of EDF that jobs are executed as soon as possible. Note that the horizontal generalization is less *greedy* than the vertical one. Thus, we can expect that, in the long run, U-EDF leads to less Dhall's-effect-like blocking of urgent tasks, cf. Subsection 3.3.1.

U-EDF reduces to partitioned EDF for system utilizations up to 50%. According to [226], it uses the same basic clustering strategy as EKG, see Subsection 3.5.4.

Both RUN and U-EDF showed significantly less preemptions and less migrations in simulated schedules compared to EKG, see Subsection 3.5.4, and DP-Wrap, see Subsection 3.5.5, for periodic tasks [227].

5.4 Summary

Hybrid approaches are very good choices, both in terms of abstraction and generalization on global and partitioned scheduling and in terms of utilization bounds they reach. *Task splitting* is a typical procedure involved in hybrid scheduling. It breaks the basic model of a job as an execution unit and might be semantically dangerous.

Recent new algorithms presented in Section 5.3 are even optimal and reduce the number of preemptions and migrations significantly compared to, e.g., LL-REF, DP-Wrap or EKG presented in Section 3.5. The natural reduction of these two algorithm to partitioned EDF for task sets of a lower total utilization makes them attractive from a theoretical point of view. Their low overheads make them suitable for implementation on real multiprocessor platforms.

Chapter 6

Generating Synthetic Task Sets

For the evaluation of scheduling algorithms and schedulability tests, the value *sensitivity* or *success ratio* introduced in Subsection 1.7.3 is very common. The goal behind that is to estimate the *average-case* behavior of the different approaches both in relative and absolute manner. The method of choice is to generate randomly synthetic task sets. Then, a success ratio of the number of favorable cases to the number of possible cases gives an estimate of the sensitivity of the test or the schedulability of the scheduling algorithm.

Contrary to worst-case and best-case analyses, which are well defined already by their wording itself, the situation is much more complicated for average-case analyses. The reason is that they always have to refer to assumed parameter distributions. Among all possible distributions, the *uniform* distribution is the simplest one. Hence, in situations without any additional background knowledge, a uniform distribution should be assumed due to the philosophical principle of *Occam's razor*. But sometimes, there is some previous knowledge available which justifies the change to other distributions, see Section 6.5. This has to be checked step-by-step for all relevant parameters. In Section 6.1, it will turn out that even the insurance of a uniform distribution is paved with pitfalls.

Important parameters characterizing a synchronous implicit-deadline periodic task set are the utilizations u_i and the periods p_i. Note that all other parameters can be obtained[1] from them: $e_i = u_i p_i$, $D_i = p_i$ and $\varphi_i = 0$. Hence, it is sufficient to focus on utilization and period values.

The utilization distribution is considered to be more important and will be discussed in Sections 6.1 to 6.4. There, the algorithmic implementation of the favored uniform utilization distribution is a major concern. For evaluating scheduling algorithms and schedulability algorithms, often a total utilization–success rate curve is taken into account. Hence, we need task sets with a uniform distribution of the individual utilizations u_i under the constraint of a constant total utilization u. The main focus in Sections 6.1 to 6.4 is, thus, the presentation of correct and practical (fast) algorithms producing such task sets.

Finally, period distributions and their generation algorithms will be a topic in Section 6.5. The choice of periods plays a significant role in RM scheduling.

6.1 Naïve Incorrect and Impractical Approaches for Uniprocessor Task Sets

It is widely accepted that a *uniform* utilization distribution is the most appropriate assumption when generating synthetic task sets. Cirinei and Baker [78] have performed experiments with other distributions like exponential and bimodal ones[2] But their results are very similar to the ones obtained assuming a uniform distribution [78]. Hence, it seems to be appropriate to restrict investigations to uniform utilization distributions.

In the following, we will see that the correct and fast generation of such task sets is paved with some pitfalls, some intuitive approaches are incorrect or very slow and, thus, impractical.

[1]The phase values φ_i are by default set to zero hence assuming an synchronous model. Thus, more precisely, the phases are chosen independent of period and utilization.

[2]In the PhD thesis of Emberson [98], a *beta* distribution is used to model utilization. This is motivated by a good fitting of a *beta* distribution to case-study data taken from [43] and [268], both aircraft engine control systems. However, the uniform utilization distribution model is still widely used. This can be justified in spite of the use of beta and triangular distributions in task duration modeling [250]. There, the actual execution time $e_{i,act}$ as opposed to the WCET e_i, see 1.1.2, is considered, resolving this apparent conflict.

6.1.1 *UScaling*

The *UScaling* algorithm [51] [53] with the parameters n and u first generates u_i values in the interval $[0, u]$. Next, these values are scaled by $u / \sum_{i=1}^{n} u_i$ with the result that the new u_i meet the constraint of a total utilization of u. But the problem of *UScaling* is that task sets with similar u_i values being close to the expected value u/n are favored [53]. This is clearly visible in Figure 6.1 which is a 3d scatter plot. Mathematically, this means that the mean utilization $1/n$ is guaranteed while the variance is rendered too small. Hence, no uniform distribution of task set points in the $n-1$ dimensional hyperplane is provided, the algorithm is *incorrect* for the posed problem. Governed by its main loop, *UScaling* has linear computational complexity $\mathcal{O}(n)$.

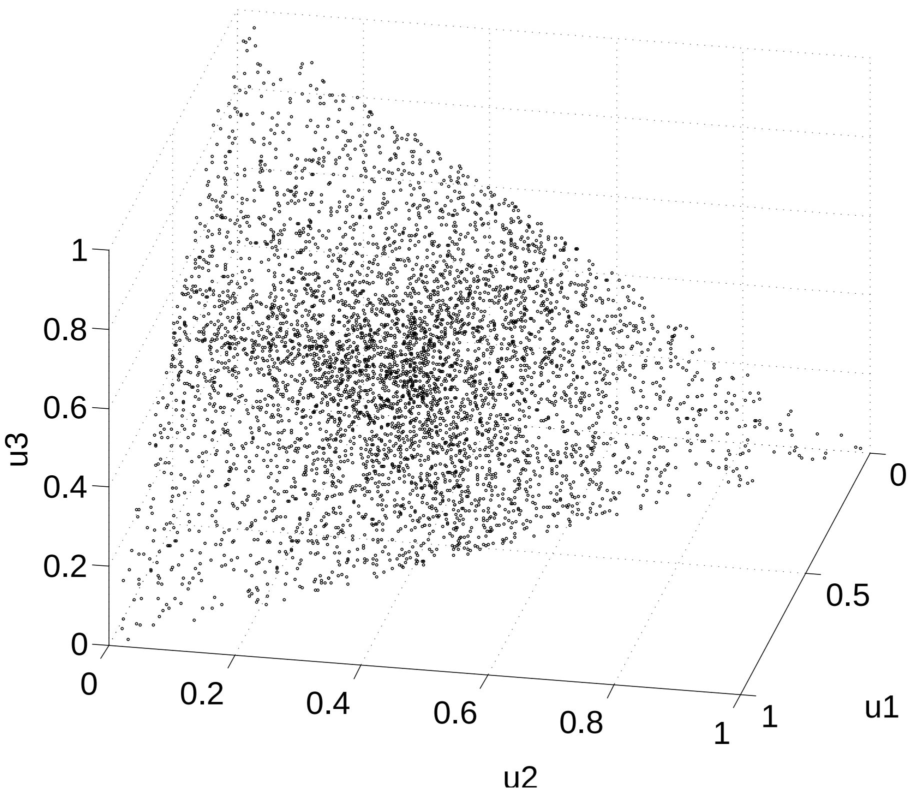

Figure 6.1: 5000 Task Sets Generated with *UScaling*($n = 3$,$u = 1$)

6.1.2 *UFitting*

The *UFitting* algorithm [51] [53] is a second intuitive approach of synthetic task set generation which turns out to be incorrect as well. First, u_1 is made uniform in $[0, u]$. Next, u_2 is chosen as a uniform sample in the remaining interval $[0, u - u_1]$. Subsequently, u_3 is made uniform in $[0, u - u_1 - u_2]$ etc., until u_n is set to $u - \sum_{i=1}^{n-1} u_i$. This algorithm ensures meeting the requirement of a constant total utilization u which can be easily verified by the last construction step. But the result is asymmetrical and non-uniform, see the scatter plot in Figure 6.2.

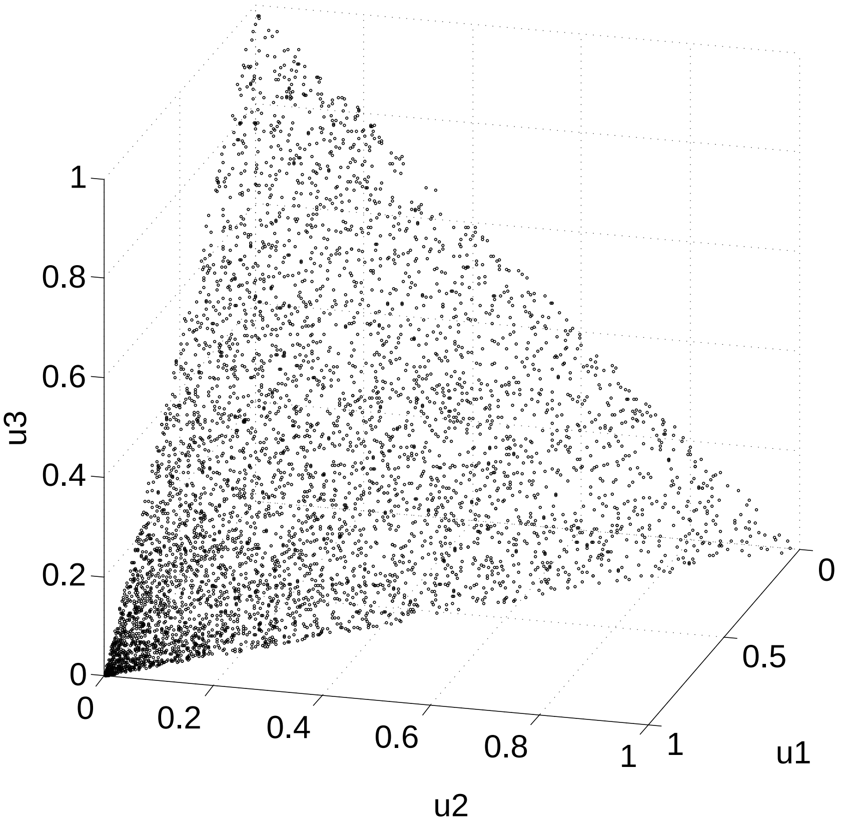

Figure 6.2: 5000 Task Sets Generated with *UFitting*$(n = 3, u = 1)$

From a mathematical point of view, we see that even the mean utilization condition $\bar{u_i} = 1/n$ is not met. The mean values are decreasing with the task index: $\bar{u_i} = 1/2^i$ except for the last task with $\bar{u_n} = 1/2^{n-1}$. This clearly shows the asymmetric result of the *UFitting* algorithm. Due to its main loop, *UFitting* has linear computational complexity $\mathcal{O}(n)$.

6.1.3 *UUniform*

The *UUniform* algorithm [53] is the first correct version solving the problem. It sums up $n-1$ uniformly distributed utilizations and checks then whether this sum is less than or equal to the total utilization u. This check is called the *boundary condition*. As soon as it is met, the last utilization value is set to the rest, ensuring a total utilization u as required.

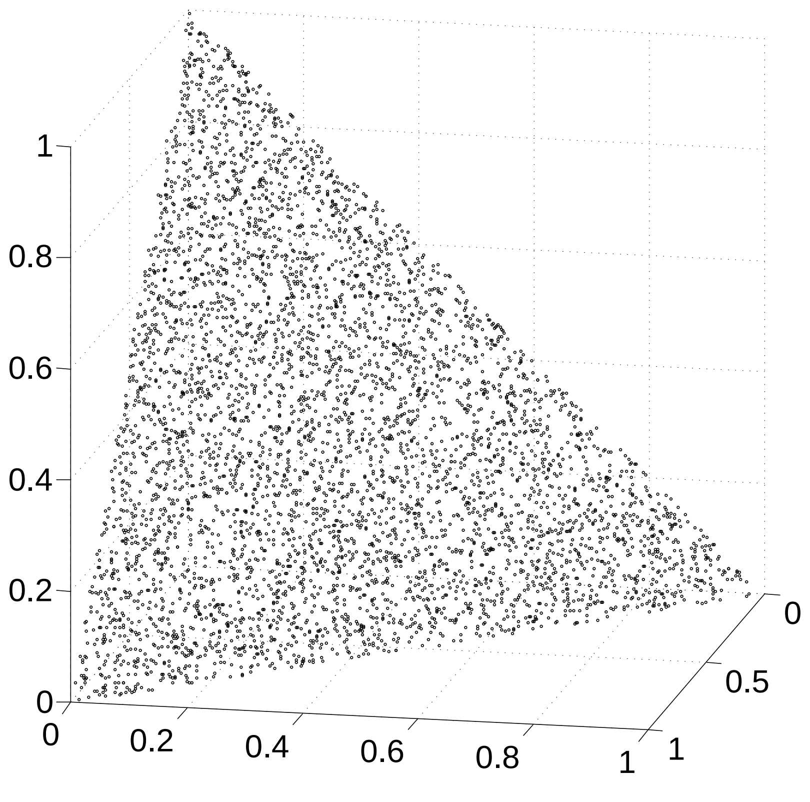

Figure 6.3: 5000 Task Sets Generated with *UUniform*$(n = 3,u = 1)$

UUniform has at least factorial complexity $\mathcal{O}(n!)$ since the probability of satisfying the boundary condition is $1/(n-1)!$ as shown in [49]. Thus, the expected number of necessary steps is $(n-1)!$, rendering the use of this algorithm impractical.

The points representing task sets really follow a uniform distribution in the hyperplane of a constant total utilization as can be seen for the case of $n = 3$ tasks and $u = 1$ in Figure 6.3.

6.2 A Correct and Fast Approach For Uniprocessor: *UUniFast*

We have seen that some straightforward approaches of task set generation are very slow or even incorrect. The *UUniFast* algorithm both ensures correctness and provides by its $\mathcal{O}(n)$ complexity quick results [51] [53].

The algorithm calculates a single utilization as the difference of two neighboring partial sums of i and $i-1$ uniformly distributed variables. The pdf of the sum of independent variables is the *convolution* of the individual pdf's.

6.2.1 An Elegant but Slow Variant For Uniprocessor: *UUniSort*

An even better understandable variant of this approach is the *UUniSort* algorithm [53]. It generates $n-1$ uniformly distributed values in the interval $[0, u]$. Next, the elements 0 and u are added to this array. Then, the array is sorted according to non-decreasing values. In this list of $n+1$ elements, all n differences between two adjacent values are taken as the individual utilization values. The computational complexity of the *UUniSort* algorithm is due to the sorting $\mathcal{O}(n \log n)$. This fact renders it slower and less practical than *UUniFast* which is considered the perfect approach to uniprocessor task set generation.

6.3 A Correct but Sometimes Slow Extension to MP: *UUniFast-Discard*

A straightforward correct extension of the *UUniFast* algorithm to the multiprocessor case was given by Davis and Burns [84]. It was coined *UUniFast-*

Discard. The idea is to run *UUniFast* repeatedly with the target utilization u. In order to guarantee the second necessary schedulability condition (3.2), no tasks of a utilization greater than one are allowed. Thus, all task sets having a task with $u_i > 1$ are rejected, and *UUniFast* is restarted.

When the target utilization increases towards $n/2$, the probability of obtaining a valid task set with $\forall i : u_i \leq 1$ decreases quickly. For $n = 3$ and $u = 1.5$, already one third of all samples are rejected. Due to the *curse of dimensionality*, the acceptance probability decreases dramatically for increasing task set size n.

For $u > n/2$, a valid task set becomes even more unlikely. Thus, it is an improvement to run the algorithm with the complementary target utilization $u' = n - u$. This gives complementary task utilizations u'_i which are finally corrected by $u_i = 1 - u'_i$. The overhead for forward and backward transformation is small. Even with this technique, a disadvantageous interval $u \in [n/2 - \varepsilon n, n/2 + \varepsilon n]$ remains. When setting a timeout to a pragmatic level of, e.g., $1,000$ loop cycles in the *UUniFast-Discard* main loop, the ε band starts with a width of 0 and widens when n increases. At $n = 50$, task sets with $u \in [20, 30]$ are hard to generate [84]. Thus, $\varepsilon \approx 0.1$ for $n = 20$, blocking $2\varepsilon \approx 0.2$ what is one fifth of the reasonable total utilization values. This is considered to be a major restriction in theory, but might be acceptable in practice since "[..] the vast majority of commercial real-time systems using multiprocessors will have significantly more tasks than processors" [84], rendering the ratio number of tasks to target utilization being close to 2 an unlikely scenario.

6.4 A Correct and Fast Approach For MP: *RandFixedSum*

This seminal algorithm called *RandFixedSum* was first published by Stafford in 2006 [260] already as a Matlab implementation. Its application in the context of real-time scheduling synthetic task sets was discussed in [99]. *RandFixedSum* is considered a breakthrough since it provides a correct and fast approach for the entire parameter space.

The main difference to *UUniFast-Discard* is that no rejection of generated task sets is necessary. Instead, the target hyperface is subdivided into simplexes of dimension $n-1$. A particular simplex is then selected with a probability proportional to its hyperarea. Then, points are evenly distributed inside each simplex. Finally, the order of dimensions within each point is randomly permuted in order to get coverage of the whole valid region.

A simple, non-optimized *C* implementation of the algorithm including a minimum main function to test the function itself is given in Listing 6.1. This implementation is based on Stafford's Matlab implementation [260]. Besides a matrix containing the utilization vectors in the columns, a value describing the hypervolume of the $(n-1)$-dimensional target hyperface containing all valid (with the target total utilization) task set representations is returned. It can be used for an additional check. The integral over all s from a to b scaled by $\sqrt{n}$ shall give the hypervolume of the hypercube $(b-a)^n$ which is just 1 for the most typical case of $a=0$ and $b=1$ in our scheduling context.

Listing 6.1: Simple *C* Implementation of the *RandFixedSum* Algorithm

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <time.h>
#define MAX_TASKS 1000
#define MAX_SAMPLES 100

 int randfixedsum(float x[MAX_TASKS+1][MAX_SAMPLES+1],
   float *v, int n, int m, float s, float a, float b)
   // return values are (n x m)-matrix x and volume v
   // meaning m vectors of dimension n with min. a, max. b
   // and constant sum s of vector elements
 { // v volume of the subset as a check
   int i, j;
   int k;
   float tiny = FLT_MIN;     // smallest positive number
   float s1[2*n], s2[2*n];
   float w[n+1][n+2];
   float t[n][n+1];
```

```
  float tmp1[n+2], tmp2[n+2], tmp3[n+2], tmp4[n+2];
  int sv[m+1], jv[m+1];
  float sm[m+1], pr[m+1], sx[m+1];
  int e[m+1];
  float rt[n][m+1], rs[n][m+1];
  int p[n+1][m+1];
  float rp[n+1][m+1];
  float ftemp;
  int itemp, ii;
  float xx[n+1];
  if (m<0 || n<1)    // otherwise no sense
    return 1;
  if (s<n*a || s>n*b || a>=b) // constraints conflicting
    return 2;

  // Rescale to a unit cube: 0 <= x(i) <= 1
  s = (s-n*a)/(b-a);
  // Construct the transition probability table, t.
  // t(i,j) will be used only in the region where j<=i+1.
  k = (int)fmax(fmin(floor(s),n-1),0); // Need 0<=k<=n-1
  s = fmax(fmin(s,k+1),k);   // Must be k<=s<=k+1
  for (i=k;i>=k-n+1;i--)  // s1 & s2 will never be neg.
    s1[k-i+1] = s - i;
  for (i=k+n;i>=k+1;i--)
    s2[k+n-i+1] = i - s;
  for (i=1;i<=n;i++)
    for (j=1;j<=n+1;j++)
      w[i][j]=0;
  w[1][2] = FLT_MAX;      // Scale for full 'float' range
  for (i=1;i<=n-1;i++)
    for (j=1;j<=n;j++)
      t[i][j]=0;

  for (i=2;i<=n;i++)
```

```
{
  for (j=1;j<=i;j++)
    tmp1[j] = w[i-1][j+1] * s1[j] / i;
  for (j=1;j<=i;j++)
    tmp2[j] = w[i-1][j] * s2[j+n-i] / i;
  for (j=2;j<=i+1;j++)
    w[i][j]=tmp1[j-1]+tmp2[j-1];
  for (j=1;j<=i;j++)
    tmp3[j] = w[i][j+1] + tiny;
  for (j=1;j<=i;j++)
    tmp4[j] = s2[n-i+j] > s1[j];
  for (j=1;j<=i;j++)
    t[i-1][j] = (tmp2[j] / tmp3[j]) * tmp4[j]
        + (1- tmp1[j]/tmp3[j])*(1-tmp4[j]);
}

// Derive the polytope volume v from the appropriate
// element in the bottom row of w.
*v = pow(n,3.0/2.0)*(w[n][k+2]/FLT_MAX)*pow(b-a,n-1);

// Now compute the matrix x.
for (i=1;i<=n;i++)
  for (j=1;j<=m;j++)
    x[i][j]=0;
if (m==0)    // If m is zero, quit with x = []
  return 0;

for (i=1;i<=n-1;i++)
  for (j=1;j<=m;j++)
  {
    rt[i][j] = (float)rand()/(float)RAND_MAX;
    // For random selection of simplex type
    rs[i][j] = (float)rand()/(float)RAND_MAX;
    // For random location within a simplex
    }

  for (i=1;i<=m;i++)
  {
    sv[i]=s;
    jv[i]=k+1;  // For indexing in the t table
    sm[i]=0;  // Start with sum zero & product 1
    pr[i]=1;
  }

  for (i=n-1;i>=1;i--)// Work backwards in the t table
  {
    for (j=1;j<=m;j++)
    {
      e[j] = rt[n-i][j]<=t[i][jv[j]];
      // Use rt to choose a transition
      sx[j] = pow(rs[n-i][j],1.0/(float)i);
      // Use rs to compute next simplex coord.
    }
    for (j=1;j<=m;j++)
    {
      sm[j] = sm[j] + (float)((1.0-sx[j])*pr[j]*sv[j])
            /((float)(i+1.0)); // Update sum
      pr[j]=sx[j]*pr[j]; // Update product
    }
    for (j=1;j<=m;j++)
    {
      x[n-i][j] = sm[j] + pr[j] * e[j];
      // Calculate x using simplex coords.
      sv[j] = sv[j] - e[j]; jv[j] = jv[j] - e[j];
      // Transition adjustment
    }
  }
  for (j=1;j<=m;j++)
```

```
    x[n][j] = sm[j] + pr[j] * sv[j]; //Compute the last x

  // Randomly permute in the columns of x and rescale
  for (i=1;i<=n;i++)
    for (j=1;j<=m;j++)
      rp[i][j] = (float)rand()/(float)RAND_MAX;
      // Use rp to carry out a matrix 'randperm'
  for (j=1;j<=m;j++)
    for (i=1;i<=n;i++)
      p[i][j]=i;  // index init (prep. for permutation)

  for (j=1;j<=m;j++)
    for (i=1;i<=n;i++)  // Bubble Sort on each column
      for (ii=1;ii<=n-1;ii++)
        if (rp[ii][j]>rp[ii+1][j])
        {
          ftemp=rp[ii][j]; rp[ii][j]=rp[ii+1][j];
            rp[ii+1][j]=ftemp; // value exchange
          itemp=p[ii][j]; p[ii][j]=p[ii+1][j];
            p[ii+1][j]=itemp;  // index exchange
        }

  for (j=1;j<=m;j++)          // loop over all samples
  {
    for (i=1;i<=n;i++)        // loop over all elements
      xx[i] = (b-a) * x[p[i][j]][j]+a;
      //need 2nd array for permutations inside one sample
    for (i=1;i<=n;i++)
      x[i][j]=xx[i]; //save copy from 2nd array for result
  }
  return 0;
}
int main()
{
  int i, j;
  float v;
  int n=8;  // #tasks per task set
  int m=10; // #samples
  float u_ges=n/2.0; //total u set to most difficult case
  float x[MAX_TASKS+1][MAX_SAMPLES+1];
  float sum;
  srand(time(NULL));
  randfixedsum(x, &v, n, m, u_ges, 0, 1);
  for (j=1;j<=m;j++)
  {
    sum=0;
    for (i=1;i<=n;i++)
    {
      printf("%4.3f ", x[i][j]);
      sum += x[i][j];
    }
    printf("| sum=%4.3f\r\n", sum);
  }
  return 0;
}
```

6.5 Period Distribution: Uniform vs. Log-uniform

For the distribution of period values, based on the principle *Occam's razor*, uniformity seems to be appropriate. But there is a subtle difference to the parameter set u_i. The periods of real-time tasks are typically grouped in different *time bands*, see [63]. E.g., temperature control loops sample at periods of seconds or minutes while rotational speed sensors in a car need to sample in the millisecond band.

Hence, uniformity remains an important principle, but it should be applied at the level of bands. Each level of magnitude of a period is represented by the same number of tasks. Mathematically, such a distribution is called *log-uniform*. There, the logarithms of the values follow a uniform distribution.

Experiments in [99] indicate that the change from a uniform to a log-uniform period distribution is not only a cosmetic surgery. Its impact on schedulability is significant. Log-uniform period task sets are easier to schedule by RMS, and there is a better orthogonality to the parameter number of tasks n than with uniformly distributed periods.

6.6 Summary

Based on the previous discussions, we have obtained the following recommendations for the task set parameter generation in synthetic task sets.

6.6.1 Utilizations

Utilizations based on WCETs shall be chosen according to a uniform distribution although those based on actual execution times are better modeled by a beta distribution.

A typical constraint is a constant total utilization. On a uniprocessor, a good algorithm quickly giving unbiased results under this constraint is *UUniFast*. A straightforward extension to it, *UUniFast-Discard*, works as well on multiprocessor, but only for a proper subset of the parameter space. An algorithm working on the entire parameter space is *RandFixedSum*.

6.6.2 Periods

Periods can be chosen uniformly distributed, but a log-uniform distribution is more appropriate due to the different time bands typical for real-time systems.

6.6.3 WCETs

Once having generated utilization and period values, WCETs are easily obtained by $e_i = u_i p_i$.

6.6.4 Deadlines

For implicit-deadline task sets, deadlines are already fixed by $D_i = p_i$. With constrained deadlines, the obvious choice is to distribute them uniformly between the WCET e_i and the period p_i. For arbitrary deadlines, there is no obvious choice of the deadline distribution. Reasonable values start at e_i and go (potentially) to infinity.

Part III

New Results in Multiprocessor Scheduling

This part contains new results related to partitioned RMS and partitioned EDF scheduling.

Chapter 7

Exploiting Period Compatibility for Improved Partitioned RMS Schedulability Tests Based on RMST

This Chapter is based on the publications [216] at ERTS^2 2010, [219] at RTSOPS 2010, [220] at RTSOPS 2011, [222] at RTNS 2011 [217] at RTCSA 2012, and [224] in IEEE Transactions on Computers.

7.1 Introduction

A key cue for the understanding of rate-monotonic scheduling is the concept of *period compatibility*. The more the periods are compatible to each other, the less idle time is in the RM schedule and, thus, the greater is the utilization bound. This realization is basic for most of the algorithmic modifications and improvements to be given in this Chapter.

The best case in terms of period compatibility is for simply periodic task sets, cf. 1.2.1. Here, periods are integer multiples of each other for *all* possible pairings. As a consequence of such a setting, there is no waste (idle time) when combining any two periods, they fit perfectly. Hence, the RM utilization bound of simply periodic task sets reaches the maximum 1.

The worst case was studied in the seminal paper of Liu and Layland [189]. The famous utilization bound (2.10) is valid for all task sets since it covers the worst case possible. As expected, the most disadvantageous configuration is with uneven period ratios. But, for the simple case of two tasks, what concrete ratio is worst? Is it 1.5? Or is it the golden ratio $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$? The analysis in (2.10) has shown that a ratio of $\sqrt{2}$ is the worst for a two-task system. Note that this equals the *geometric mean* of the basic simply periodic configuration with periods $p_1 = 1$ and $p_2 = 2$: $p = \sqrt{p_1 p_2} = \sqrt{2}$. The fact that the geometric mean is the appropriate metric is not surprising since it is designed for calculating means of rates and ratios. In order to be able to handle ratios like absolute values, a logarithmic transformation[1] is appropriate. In the transformed space, the arithmetic mean is calculated. Finally, a back transformation gives the geometric mean: $\exp \frac{\ln p_1 + \ln p_2}{2} = \sqrt{p_1 p_2}$.

Neither the best case nor the worst case are common in practice. Hence, the obvious challenge is to exploit special properties of in-practice common intermediate configurations by relating them to the best case. This is one of the basic ideas of our approach, see Subsection 7.3.2.

The famous algorithm RMST already presented in Subsection 4.4.2 serves as a base for modifications and improvements later to be given in Section 7.4 and then evaluated in Section 7.6. The concept of presorting and some principles for the RMST improvements are presented in Sections 7.2 and 7.3. Additionally, a clear terminology is introduced in Section 7.5. Finally, Section 7.7 contains a summary.

7.2 Presorting

As already pointed out in Section 4.4, a presorting of tasks is a viable way of reducing waste when relying on allocation heuristics like First Fit or Best Fit, cf. [202]. The amount of possible savings depends on the maximum utilization of an individual task. The higher this maximum utilization $u_{\max}$ becomes the more impact has the order in which tasks are presented to the online scheduler. According to Kato, "[..] sorting the tasks does not dominate

[1]This fact is the basis for the predecessor of the pocket calculator, the *slide rule*.

a schedulable utilization very much for the case in which the utilization of every individual task is less than 50%." [155]

The main drawback of these heuristics is that they take only a minor subset of task combinations for processor allocation into account. Hence, they can lead to suboptimal solutions. The obvious challenge is to narrow the gap between suboptimal and optimal results as far as possible both in worst and in average case. Clearly, in the best case, First Fit is able to achieve the optimal solution as shown in [187].

But the sequence of tasks to be presented to First Fit for obtaining this optimal solution might be hard to describe by a generic approach in the general case. Hence, sorting according to an easily accessible criterion is much more common in practice.

The best known presorting order in the context of the closely related problem of bin packing is *decreasing utilization* (DU). It is well-working there, but too simplistic for partitioned RMS where period values need to be taken into account.

Conceptually, the adding of a presorting goes along with a change from online to offline scheduling. In the offline situation, the additional degree of freedom is the sequence of the tasks they are presented to the online scheduler. The typical workflow also used in the RMST algorithm is presented in Figure 7.1.

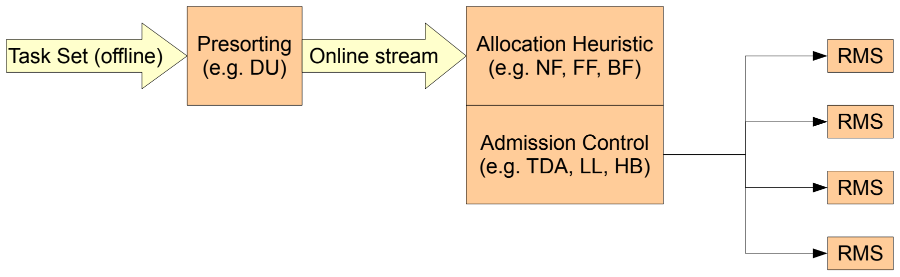
Task Set (offline)
Presorting (e.g. DU)
Online stream
Allocation Heuristic (e.g. NF, FF, BF)
Admission Control (e.g. TDA, LL, HB)
RMS
RMS
RMS
RMS

Figure 7.1: Workflow when Using Online Heuristics for the Offline Scheduling with Partitioned RMS

The computational effort of such a sorting is $\mathcal{O}(n \log n)$. In a discussion, this additional effort has to be weighed against its gain.

Since common heuristics are all based on sequential processing of tasks to be allocated, neighboring tasks in the presorted list should fit together. The required property is *period compatibility.*

7.2.1 Sorting for Period Compatibility

This plausible intermediate result describing local (neighboring) properties of the required list has to be refined to a global sorting order. As first regarded in the RMST algorithm, cf. Subsection 4.4.2, a sorting according to increasing decimal fractions of the periods' logarithms is a good starting point. Mathematically, this is a sorting with increasing S values, see (2.18). Based on these S values with the domain $\forall i : S_i \in [0, 1)$., a representation of tasks on a *ring* in clockwise direction is appropriate. This is a hint for a *circular* similarity measure for the simplicity of periods discussed in [222]. A second issue to consider here is the logarithm base set to 2 by default. But this is not the only reasonable choice although binary approaches are predominant in computer science, see [222].

A more detailed discussion on sorting orders as preprocessing for partitioned scheduling can be found in [220].

7.3 Principles for Modifications of the Workflow Stages

In this Section, we will give four important principles to be applied later on: a *circular*[2] similarity measure in the period space, a relation to the best case of simply periodic task sets, the choice of the logarithm base for period similarity, and the allocation strategy itself.

7.3.1 Circular Similarity Measure in the Period Space

The semantics of the S values defined in the RMST algorithm is a similarity to a power of 2 from above. At first glance, this seems to be a good and

[2]A generalization of circular data is *directional* data which includes, e.g., also spherical data. A directional data set consists of unit vectors in $\mathbb{R}^n$ with $n = 2$ for circular and $n = 3$ for spherical data.

perfectly appropriate approach for describing period similarity by a difference of S values. When two periods are beyond a power of 2 to the same degree, they fit perfectly since their relative distance among each other becomes 0. Note that *differences* in S values become *ratios* of periods due to the logarithm operation involved in (2.18). The ratio of these two periods p_1/p_2 is then a power of two, cf. [222].

In a more general case, the slight (multiplicative) increase of one of two perfectly fitting periods leads to an S value difference slightly above zero[3]. Thus, the schedulability depending on the period ratio as given in Figure 2.2 is well reflected for a slight *increase* of periods, see (7.1).

$$\begin{aligned}
S_1 - S_2 &= \log_2 p_1 - \lfloor \log_2 p_1 \rfloor - (\log_2 p_2 - \lfloor \log_2 p_2 \rfloor) \\
&= \log_2 (ap_2) - \lfloor \log_2 (ap_2) \rfloor - \log_2 p_2 + \lfloor \log_2 p_2 \rfloor \\
&= \log_2 a + \log_2 p_2 - \log_2 p_2 - \lfloor \log_2 a + \log_2 p_2 \rfloor + \lfloor \log_2 p_2 \rfloor \\
&= \log_2 a \qquad \text{for a} \to 1+0 \\
&\to 0 \qquad \text{for a} \to 1
\end{aligned} \tag{7.1}$$

Note that in (7.1), $\to 1 + 0$ signifies the limit process to 1 from above. For this (still special) case of a slightly increased period, the continuous graph in Figure 2.2 is well depicted by the absolute difference of S values $|S_1 - S_2|$.

A fundamental observation when regarding S values defined according to the RMST algorithm is that "[d]issimilarity to a power of 2 from above turns into similarity to a power of 2 from below at the threshold value of 0.5." [222] Related to this, if a in (7.1) becomes larger, ap_2 once surpasses the next power of two. Then, the ordinal relation between S_1 and S_2 changes at this discontinuity which is no longer appropriate to the intended semantics of period similarity.

The reason is that the absolute difference $|S_1 - S_2|$ describes an inappropriate *linear* dissimilarity[4] measure. So, we need to correct this deficiency by the change to a *circular* dissimilarity measure. That fact that fractional parts of numbers give circular data can be used for time series generation methods

[3]W.l.o.g., we can assume $S_1 > S_2$. Only the absolute difference of S values is relevant.

[4]Dissimilarity is expressed by distance in the feature space. The complementary concept is similarity.

based on *wrapping* as suggested in [83], p. 10f. Note that one of the first publications on circular statistics is by von Mises [211] from 1918. He used the fractional parts of relative atomic masses as data set. It is interesting that he studied fractional parts as we do it for the logarithms of periods giving the S values (2.18).

But the distribution of the fractional parts of atomic masses has physical reasons (mixtures of different isotopes in nature giving a weighted-average atomic mass[5] and the relativistic mass defect, cf. [262]) while the distribution of S values is synthetic and not based on observation[6].

Another, more recent application field of circular statistics is geography with the paradigm *wind direction*, cf. [117] and [111]. Here, the naïve application of linear statistics like *linear mean* easily leads to paradox results. E.g., two wind direction samples with the same deviation from East can both average to West (incorrectly) and East (correctly), depending upon the choice of zero direction[7].

Returning to our circularly interpreted S values, $0° \mathrel{\widehat{=}} S = 0$, $90° \mathrel{\widehat{=}} S = 0.25$, $180° \mathrel{\widehat{=}} S = 0.5$, $270° \mathrel{\widehat{=}} S = 0.75$, and $\rightarrow 360° - 0° \mathrel{\widehat{=}} S \rightarrow 1 - 0$. This circular data set consists of n unit vectors. The direction θ of each vector is obtained by wrapping and scaling according to $\theta_i = 2\pi\,(S_i - \lfloor S_i \rfloor)$, cf. [143], p. 31 and [83], p. 10f.

As a summary, Figure 7.2 shows the bijective mapping of directions to S values.

A Circular Similarity Measure in Everyday Life: Roundabouts

From everyday life, a roundabout gives a perfect example where a (directed[8]) circular similarity measure can be explained. Entering a traffic circle at one

[5]A third effect here subsumed under the first one is the slight mass deviation of a neutron from a proton/electron pair.

[6]On the other hand, the choice of synthetic distributions is sometimes based on observed task sets, see Chapter 6.

[7]The second undesirable dependency when using linear statistics is upon *sense of rotation* only becoming effective in more complex settings.

[8]In right-hand traffic countries like France, the orientation is counter-clockwise. Contrary to this, left-hand traffic countries like UK are characterized by a clockwise roundabout direction. Since all traffic in the Channel Tunnel connecting UK with France is railway-based, there is no need to switch the driving side of insular UK. But between left-hand traffic

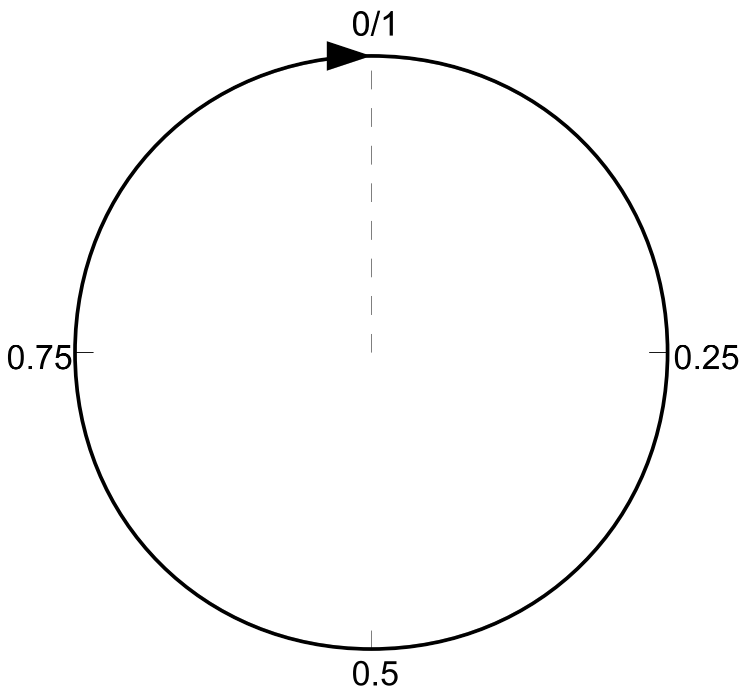

Figure 7.2: Circular Representation of S Values, North as Zero, Clockwise

position of and leaving it at another one requires to drive in the roundabout for a time proportional to the directed circular distance between entry and exit point. There is no outstanding point on a circle. Hence, any linear measure with the outstanding point zero is not appropriate here. In the following, we will consider undirected circular similarity which considers the shorter of the clockwise and the counter-clockwise circular distance.

Summing up, undirected circular distance and similarity are invariant with respect to the choice of a zero direction and sense of rotation, cf. [143], p. 3. We will come back to this invariances in 7.4.1.

Circular Similarity of S Values

A formula reflecting a circular similarity measure normalized to the interval $\bar{d}_n(S_i, S_j) \in [0, 1]$ is given in (7.2) and was introduced in [222]. Here, a

Macao and right-hand traffic mainland China, there is the Lotus Bridge with a sophisticated changeover system [282].

$\bar{d}_n(S_i, S_j) = 1$ means equality (maximum similarity) and $\bar{d}_n(S_i, S_j) = 0$ means complete dissimilarity.

$$\bar{d}_n(S_i, S_j) := \Big|2\,|S_i - S_j| - 1\Big| \tag{7.2}$$

Note that this complementary distance measure calculates the circular similarity between two tasks T_i and T_j based on their S values according to (2.18), complements it to the maximum value 0.5 and finally scales it to the interval $[0, 1]$. Let us discuss some limit cases. First, if the two tasks have the same S value, their distance is 0, they are maximally similar denoted by a $\bar{d}_n = 1$ similarity value. Next, if the S value difference approaches 1 (from below)[9], their circular distance approaches 0, they are only minimally different from being simply periodic denoted by a $\bar{d}_n \to 1$ similarity.

Furthermore, two tasks are maximally non-simply-periodic, if their period ratio equals $\sqrt{2}$ as already shown in [189] and [65], p. 85-87. W.l.o.g., we can choose $p_1 = 1$ and $p_2 = \sqrt{2}$. Then, $S_1 = 0$ and $S_2 = 0.5$. The similarity $\bar{d}_n(S_1, S_2)$ reaches then its minimum 0 what is perfectly in accordance with established RM scheduling theory. The distance values between these extremes are linearly interpolated [222]. This is motivated by *Occam's razor* since the straight line is the shortest and simplest connection between two points in a plane[10]. The new circular similarity measure $\bar{d}_n$ is visualized in 2D, Figure 7.3, and 3D, Figure 7.4.

Historically, already Figure 1 in [60], p. 1437, suggests such a circular similarity. But the fact has not been elaborated in the text or in the formulas. Next, Rothvoß became aware of the problem [253], p. 23f., but did not tackle it.

7.3.2 Relation to the Best Case

As mentioned in the introduction in Section 7.1, the appropriate relation of the general case to the best case in terms of schedulability is a challenge. But the in Subsection 1.8.2 defined acceleration operation is an appropriate means of providing such a relationship since simply periodic task sets, see

[9]A difference of exactly 1 cannot occur as explained in detail in [222].

[10]This is true for the assumed case of Euclidean geometry.

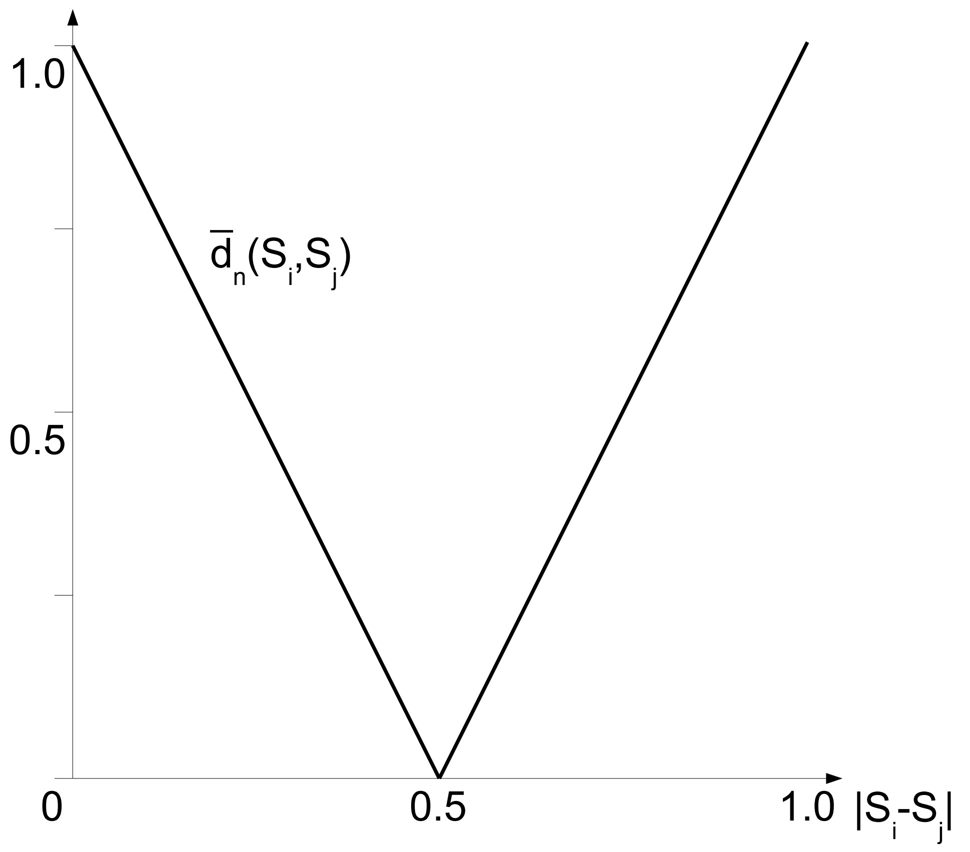

Figure 7.3: Normalized Circular Similarity as a Function of the Absolute S Value Difference $|S_i - S_j|$

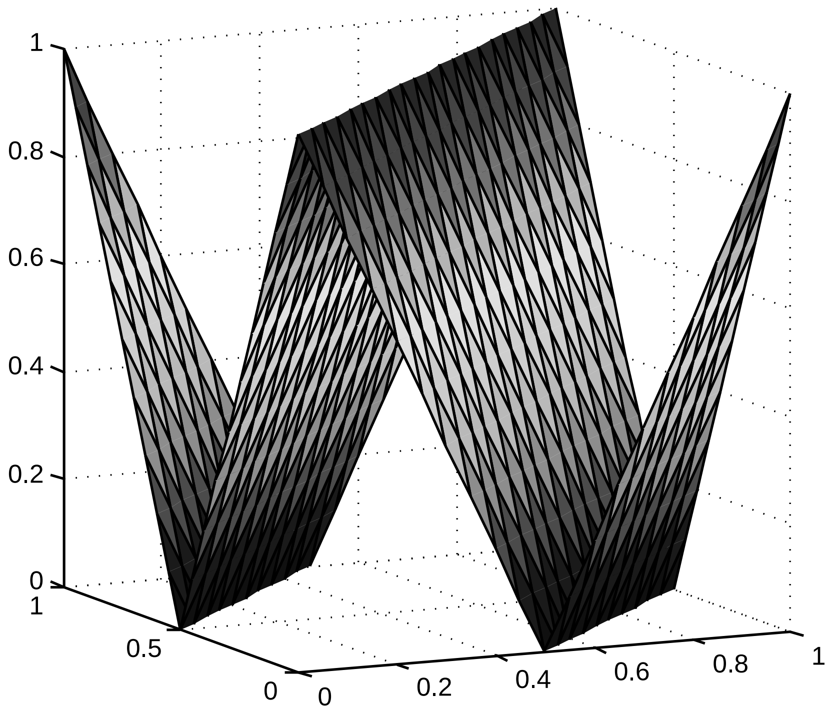

Figure 7.4: Normalized Circular Similarity as a Function of the Values S_i and S_j

1.2.1, represent the best case with a maximum utilization bound of 1 and, by acceleration, every task set can be transformed into an ASPTS.

Hence, the uniprocessor tests Sr and DCT [216] discussed in 2.2.2 are a good choice for implementing the principle of exploiting a relation to the best case.

Harmonic Fit

An alternative scheme, integrating DCT uniprocessor test and allocation strategy, is proposed in [106]. The approach is called *Harmonic-Fit Partitioned Scheduling* (HFPS) and exploits the fact that DCT does not only yield a binary decision whether the task set is schedulable or might be not schedulable, but also gives a quantification of the margin to a barely schedulable task set. Such a metric is based on the utilization of the ASPTS. In the conventional DCT approach, only a utilization $u' \leq 1$ is searched. Hence, the algorithm can be stopped as soon such a u' is found, see Algorithm 2.2. Harmonic Fit uses the global minimum of all u' values. Then $u - u'$ indicates the distance of a task set to an ASPTS[11].

7.3.3 Logarithm Base for Period Similarity

As explained in Subsection 7.3.1, the considerations done so far are only valid for power-of-two period ratios. The reason is the logarithm base 2 in the respective formulas.

Since this base is a variable in a more general analysis, the question for justification of such a particular choice arises. Clearly, 2 is not the only possible prime factor in an integer period ratio giving a simply periodic task set, see 1.2.1. E.g., Periods 2 and 6 with a ratio 3 make up a simply periodic task set. But 2 is a power of two and 6 is fairly remote from the closest powers of two 4 and 8. Hence, the standard base-2 approach taken from RMST would hint at tasks with such periods not fitting together. This is an example of a false hint based on S values.

[11]Note that this margin can also be used to save energy by slowing down processors accordingly. A DCT-based slowdown fits well in the framework of speed-assignment techniques with Dynamic Voltage Scaling (DVS) proposed, e.g., in [5]. There, the Pillai/Shin test, cf. 2.2.2, shall be replaced by DCT.

7.3.4 Allocation Strategy

As allocation strategy, originally Next Fit was suggested with RMST. But Next Fit wastes processor capacity in that it keeps once “closed” processors[12] closed for the rest of the allocation phase. Thus, First Fit, Best Fit and Worst Fit shall be considered as alternatives to the simple and quick Next Fit.

7.4 Modifications of RMST

This section applies the three principles presented in the previous Section 7.3 to the RMST algorithm, cf. Subsection 4.4.2.

We will consider four modifications [222] of the RMST algorithm: uniprocessor schedulability test, allocation strategy, index offset for start of X Fit[13], and logarithm base for period similarity. For the uniprocessor schedulability test, several variants are discussed. The three principles given in the last Section 7.3 are mapped to the more concrete modifications by Figure 7.5.

7.4.1 Uniprocessor Schedulability Test

RMST uses sBu, see (2.21) in 2.2.2, as uniprocessor schedulability test. This test is quick but has a low sensitivity which is even lower than that of Bu due to its enveloping and, thus, approximative character related to Bu, see also Figure 2.4. Hence, it is a good idea to replace sBu by a more sensitive uniprocessor schedulability test. Candidates to be discussed here are Sr/DCT and the Improved Burchard Test.

Sr and DCT

Two sufficient RMS schedulability tests of high sensitivity have already been given in 2.2.2. They are both based on the principle of exploiting a relation to the best case, see Subsection 7.3.2. This best case for uniprocessor RMS is achieved by simply periodic task sets (ASPTSs) with a utilization bound of 1, see 2.2.2, but is not restricted to them as discussed in 2.2.2.

[12]This wording is taken from the bin packing community where bins are said to be opened and closed.

[13]As often done, X is a wildcard and can be refined to Next, First, Best or Worst.

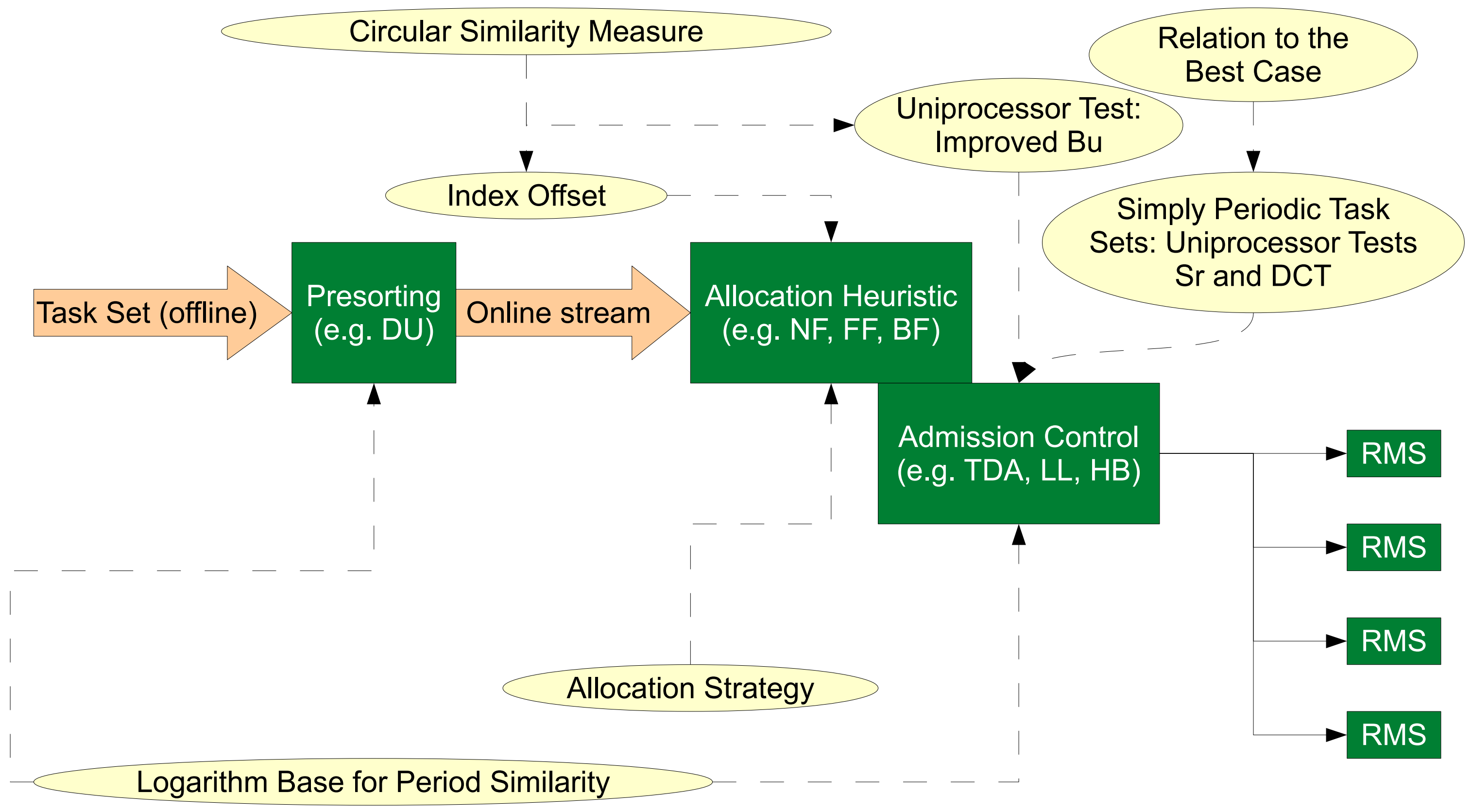
Circular Similarity Measure
Relation to the Best Case
Uniprocessor Test: Improved Bu
Index Offset
Simply Periodic Task Sets: Uniprocessor Tests Sr and DCT
Task Set (offline)
Presorting (e.g. DU)
Online stream
Allocation Heuristic (e.g. NF, FF, BF)
Admission Control (e.g. TDA, LL, HB)
RMS
RMS
RMS
RMS
Allocation Strategy
Logarithm Base for Period Similarity

Figure 7.5: Mapping of Principles to Modifications in the Workflow of Partitioned RMS based on RMST Characterized by Dashed Line Arrows

Improved Burchard Test

As pointed out in Subsection 7.3.1, the S values constitute a circular (distance) measure. Burchard's Test, see 2.2.2, always bases its schedulability formula (2.20) on a β value obtained as a *linear* range given in (2.19). This approach is inconsistent with the elaborated concept of circular similarity.

Circular Range. Hence, the changeover to a circular range [143] measure is appropriate[14]. Circular range is the shortest length of the (undirected) arc covering all data points on the circle [143], p. 162. For just two data points, we can define it as "[..]the smaller of the two arclengths between the points along the circumference[..]" [143], p. 15. Since the supremum S value is 1 and not 2π as for radian measure, we obtain the circular range of two S values as (7.3). The difference of S values has to be taken modulo one[15] since the chosen sense of rotation (here clockwise) has to be maintained.

$$d(S_i, S_j) := \min\left((S_i - S_j) \bmod 1, (S_j - S_i) \bmod 1\right) \tag{7.3}$$

Formula (7.3) can be rewritten using absolute values as (7.4).

$$d(S_i, S_j) = \frac{1}{2} - \left|\frac{1}{2} - |S_i - S_j|\right| \tag{7.4}$$

This turns out to be a complementary (to value 1) and scaled version (factor 1/2) of (7.2). The range of $d(S_i, S_j)$ is the interval $[0, 0.5]$.

A low value of a circular range indicates a clustered data cloud [143], p. 162. In our application, clustered S values indicate a good fitting of periods to each other, tasks are close to being simply periodic. Mathematically, the circular range β' of the S_i values of the n tasks is one minus the largest distance (arc) between two neighboring-task S_i values (7.5) where $S_0 := S_n$ is set[16] in

[14]The use of approximate linear equivalents for circular statistics measures is in general not recommended [143], p. 21.

[15]This corresponds to the fractional part $\{x\}$ of x defined as $\{x\} = x - \lfloor x \rfloor$. Due to the danger of confusion with set delimiters, this notation will be rarely used. The modulo-1 notation is preferred.

[16]Technically, this is done using the inner modulo operator in (7.5). Note further that in (7.5), usual operator precedence ("Multiplicative operations take precedence over additive operations.") with modulo as an multiplicative operation is assumed. This corresponds to *C/C++* as well as to *Ada* operator precedence.

order to represent index circularity. Formula (7.5) works on a sorted list of non-decreasing S values.

$$\beta' := 1 - \max_{1 \le i \le n} \left(\left(S_i - S_{(i-2) \bmod n+1} \right) \bmod 1 \right) \tag{7.5}$$

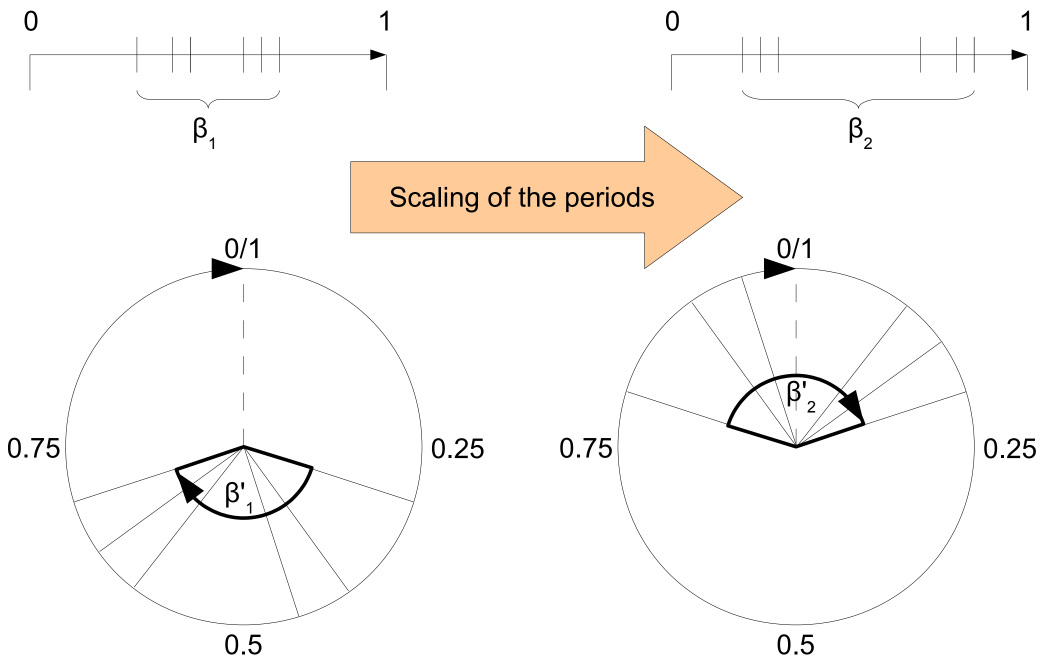

Figure 7.6: Scale Invariance of Circular Range of the Fractional Parts of Binary Period Logarithms aka S Values (below), $\beta'_1 = \beta'_2$, Opposed to Scale Dependency, $\beta_1 < \beta_2$, of the Linear Range (above)

Alternatively, we can define arc distances between neighboring-task S_i values by (7.6) which includes an exception for the first difference which straddles the origin [143], p. 162.

$$\begin{aligned} S_0 &:= S_n - 1 \\ \Delta_i &:= S_i - S_{i-1} \end{aligned} \tag{7.6}$$

The outer modulo-one operator becomes effective only for the term $S_1 - S_n$ when the loop variable i equals 1 where it gives the value $1 - (S_n - S_1)$ for the first and last value that straddle the origin.

Then, we have $\Delta_i \geq 0$ and $\sum_{i=1}^{n} \Delta_i = 1$ being in conformance with intuition. Finally, the circular range (7.5) can now be given equivalently as (7.7).

$$\beta' = 1 - \max_{1 \leq i \leq n} \Delta_i \tag{7.7}$$

Invariances of Circular Range. The circular range measure (7.5) is rotation-invariant, the choice of zero point has no impact on the circular range. The reason is that it is based on arc lenghts between neighboring directions (7.6). This set of differences is the maximal invariant in our context and can, thus, be used for invariance tests for uniformity like *Rao's Spacings test* [247], p. 63ff. and [143], p. 164.

For the period values themselves, rotation invariance of S values corresponds to *scale invariance*, see Figure 7.6, since the logarithm operation with the fraction operation (2.18) acts as a homomorphism translating multiplication (scaling) into addition modulo the logarithm base (rotation). Such a rotation invariance[17] is appropriate for a statistical measure of dispersion, see also [143], p. 271. The period scale invariance of circular range is justified since only the period *ratios* determine the degree of harmonicity of a task set. Any (non-zero) scaling factor cancels out in a period ratio. So, the replacement of the linear range by the circular range makes the quantification of the harmonicity of a task set much more sound.

For a more detailed discussion on period scale invariance of circular range and schedulability, see [217].

The second invariance is *orientation invariance*. This means that the value of the circular range is independent of choosing *clockwise* or *counter-clockwise* sense of rotation when measuring angles[18]. For our S values representing fractional parts, complementary fractional parts $S_c = 1 - S$ could be used as data points. This gives the same circular range since in a calculation restricted to *differences* of S values as in (7.5), the largest arc keeps its magnitude independent of its orientation (sign), see Figure 7.7.

[17]For the more common case of *linear* data, this corresponds to location invariance.

[18]With hindsight, the choice of a clockwise orientation in Subsection 7.2.1 and in Figure 7.6 being in conflict with the mathematically positive counter-clockwise orientation is justified by this fact.

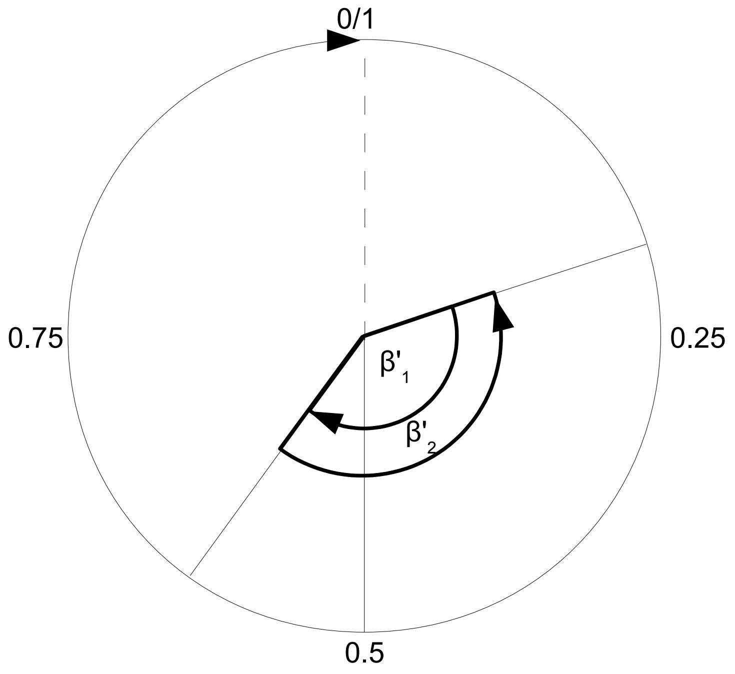

Figure 7.7: Orientation (Sense of Rotation) Invariance of Circular Range, $\beta_1' = \beta_2'$,

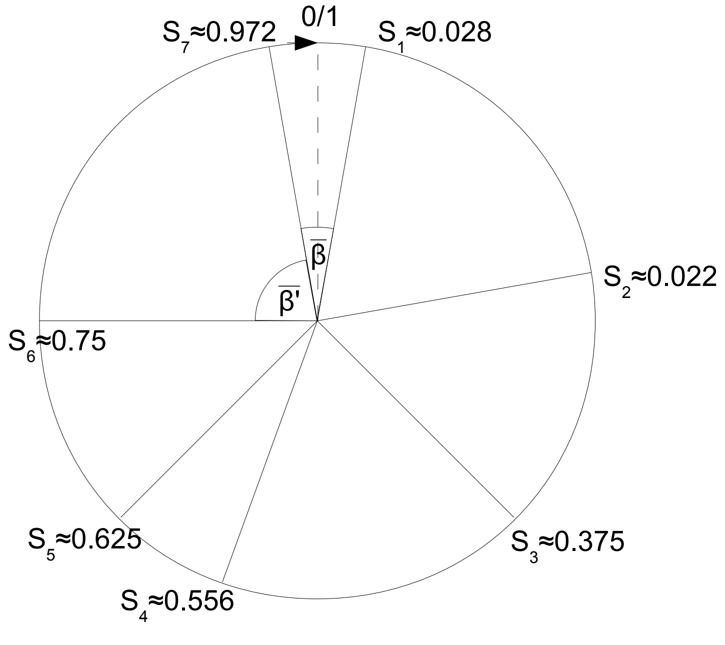

Figure 7.8: Task Set with 7 Tasks: Linear Range $\beta \approx 0.944$, but Circular Range $\beta' \approx 0.778$; Complementary Gaps $\overline{\beta}$ and $\overline{\beta'}$ are Given in the Diagram; Note that Orientation is Clockwise

Limit Cases. It is always a good idea to consider limit cases. The lowest circular range β' is achieved when all S values are the same. Then, the task set is simply periodic.

Slightly less obvious is the situation for a maximum circular range. Here, the limit case is a uniform circular distribution, cf. [143], p. 33, of the tasks by their S values. Due to the uniformity, all distances between neighboring tasks are $1/n$. Hence, the maximum circular range of S values determined by the periods of a task set is $1 - 1/n$, see (7.8), cf. [143], p. 162. This maximum range corresponds to a maximum dispersion which is characterized by a compensation of all unit vector directions to the null vector $\overrightarrow{0}$, cf. [143], p. 18.

$$\beta' \leq 1 - \frac{1}{n} \tag{7.8}$$

The New Test. The circular range β' is always less than or equal to the linear range β of one and the same task set. The reason is that for the linear range, the gap $\overline{\beta}$ is tied to the $S = 0$ point while the gap $\overline{\beta'}$ can be anywhere for the circular range measure. There is no outstanding point on a circular line, but there is the origin as an outstanding point on a straight line in a coordinate system. Second, this gap being dual to the range is maximized for the circular range. Clearly, we obtain the linear range as an upper bound of the circular range. An illustrative example of this relationship is shown in Figure 7.8.

Next, we can easily use the circular range β' instead of the linear range β in (2.20). A surprising spin-off of this modification is that case (b) in (2.20) can no longer happen since the maximum β' is exactly the limit case[19]. Hence, equation (7.9) remains as the Improved Burchard Test (impBu).

$$u \leq (n-1)\left(2^{\frac{\beta'}{n-1}} - 1\right) + 2^{1-\beta'} - 1 \tag{7.9}$$

Note that this utilization bound is now based on the circular S_i range β' defined in (7.5).

[19]The case of equality to $1 - 1/n$ is originally considered as belonging to case (b). But substituting β by $1 - 1/n$ shows that there is no discontinuity at the switching point from case (a) to (b). Hence, the threshold value can be assigned as well to case (a) without making a difference.

Due to the sorting necessary for obtaining the circular range β', the complexity of impBu is $\mathcal{O}(n \log n)$ compared to $\mathcal{O}(n)$ for Bu, cf. 2.2.2. Based on this result, an updated version of Figure 2.4 is presented of Figure 7.9. The lower sensitivity of impBu compared to Sr is due to the fact that impBu (as Bu) takes only two period values into account[20] while Sr uses *all* of them. First experiments have confirmed this claim on a qualitative base, and Figure 7.9 can only be interpreted qualitatevely for sensitivities.

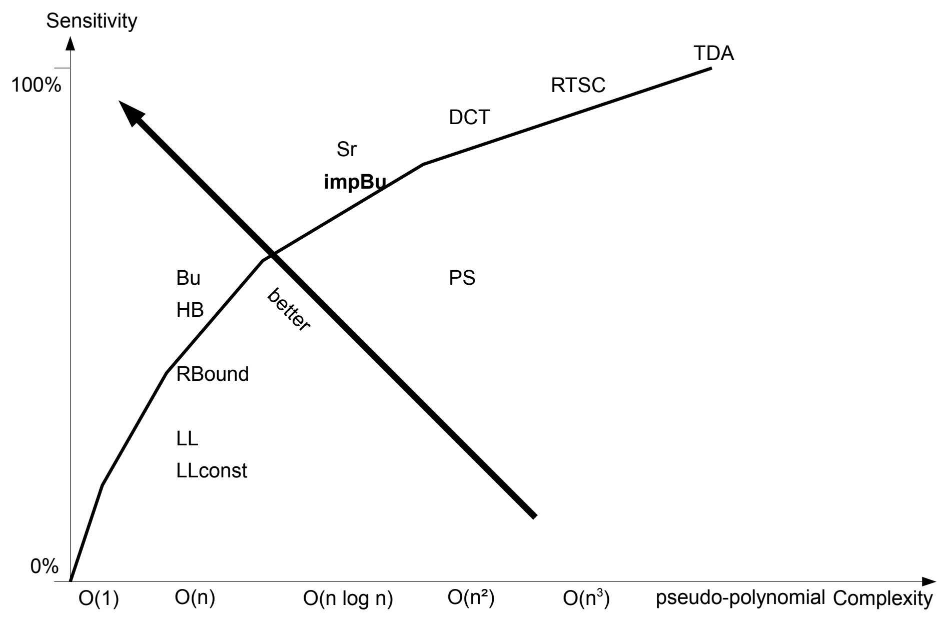

Figure 7.9: Sufficient RMS Schedulability Tests: Trade-off of Sensitivity and Complexity; Placement of the New Test impBu

The Case of $n = 2$ Tasks. For the simple situation of $n = 2$ tasks with two periods p_i of the same order of (binary) magnitude, the Improved Burchard Test turns out to be a test with a tight utilization bound. This will be shown in the following.

Putting $n = 2$ into case (a) of (2.20) gives (7.10).

[20]Since sorting is used as preprocessing for impBu, potentially all values can have an impact on the circular range. More illustratively, a rotation of some period vectors (in the circular representation of S values) can lead to a greater maximum gap between neighboring tasks.

$$u \leq 2^{\beta} + 2^{1-\beta} - 2 \tag{7.10}$$

Further assuming $p_1 \leq p_2 < 2p_1$ which is $\lfloor p \rfloor = \lfloor p_2/p_1 \rfloor = 1$ gives (7.11) using (2.18) and the logarithmic quotient identity.

$$\beta = S_1 - S_2 = \log_2 p_2 - \log_2 p_1 = \log_2 \frac{p_2}{p_1} \tag{7.11}$$

Next, we can improve to $\beta' = 1-\beta$ if $\beta > 1 - 1/n = 0.5$. Replacing β by $1-\beta$ in (7.10) gives the same inequality due to the formula's symmetry. But this reflection about the line $\beta = 0.5$ renders β small enough to qualify for case (a) in (2.20). Thus, (7.10) applies for all possible β values. By putting (7.11) into (7.10), we obtain (7.12) which is the exact utilization bound (2.17) under the above mentioned restrictions $n = 2$ tasks and the same binary order of period magnitude.

$$u \leq 2^{\log_2 \frac{p_2}{p_1}} + 2^{1-\log_2 \frac{p_2}{p_1}} - 2 = \frac{p_2}{p_1} + \frac{2}{\frac{p_2}{p_1}} - 2 = p + \frac{2}{p} - 2 \tag{7.12}$$

Hence, the Improved Burchard Test uses an exact utilization bound in this special case.

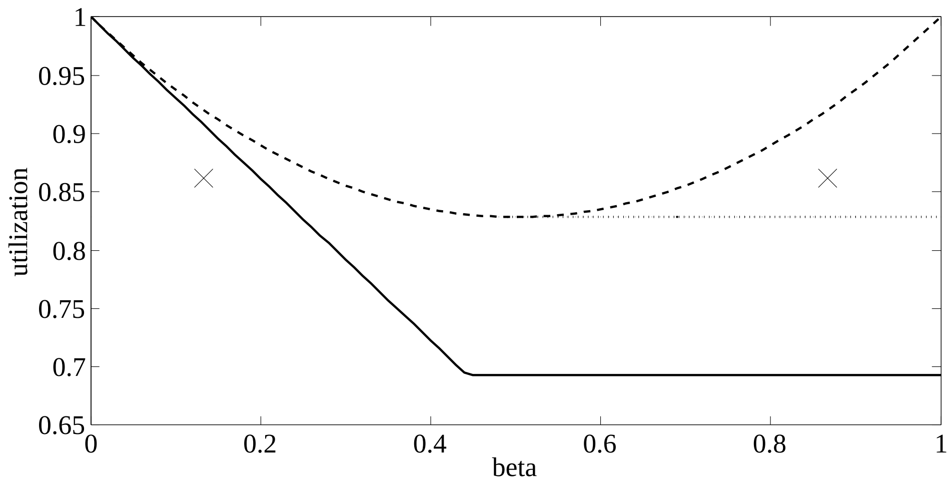

Figure 7.10: Utilization Beta Diagram for $n = 2$: Improved Burchard Bound (7.10) Which is Exact (dashed), Burchard Bound (2.20) (dotted) and Simplified Burchard Bound (2.21) (solid); Sample task set $\{(17, 13), (31, 3)\}$ is transformed from the right to the left marker by a reflection about the $\beta = 0.5$ line, correctly rendering it schedulable.

In Figure 7.10, the bounds impBu (7.10), Bu (2.20) and sBu (2.21) are given. By a sample task set, the reflection operation is shown as successful by a mapping to the provably schedulable region. The two horizontal line sections in Figure 7.10 are at levels $u_{LL}(n = 2) = 2\left(\sqrt{2} - 1\right) \approx 0.828$ and $u_{LL}(\infty) = \ln 2 \approx 0.693$, cf. (2.10) and (2.11).

As a spin-off of these considerations, we obtain a new way of proving the LL bound (2.10) in the special case of $n = 2$. The approach is to minimize (7.10) for $\beta \in [0, 1)$. The two limit cases $\beta = 0$ and $\beta \to 1$ give, since corresponding to simply periodic task sets, the global maximum instead of the minimum. Since (7.10) is continuous in its domain, the computation of local minima is appropriate. By the well-known necessary condition $u'(\beta_0) = 0$, we obtain easily (7.13) where β_0 denotes the argument for the minimum utilization value.

$$\ln 2 \cdot 2^{\beta_0} - \ln 2 \cdot 2^{1-\beta_0} = 0 \tag{7.13}$$

Solving (7.13) for β_0 gives $\beta_0 = 1/2$. Finally, this value put into (7.10) confirms the already known LL bound (7.14).

$$u_{LL}(2) = 2^{1/2} + 2^{1-1/2} - 2 = 2\left(\sqrt{2} - 1\right) \approx 0.828 \tag{7.14}$$

These theoretical arguments suggest that the Improved Burchard Test is more powerful in terms of higher sensitivity and simpler at the same time [224].

7.4.2 Allocation Strategy

The simple and, with $\mathcal{O}(n)$, quick allocation strategy Next Fit originally proposed in RMST is inferior to First Fit and Best Fit, both $\mathcal{O}(n \log n)$, in terms of minimizing CPU time waste [202] [44].

Since the additional gain which can be obtained by changing from First Fit to Best Fit is very low, the higher effort needed for Best Fit is hard to justify, see [202] and [44].

Hence, the use of First Fit is recommended since it gives a good trade-off of effort against savings in processor time.

7.4.3 Index Offset for Start of X Fit

The *circular* nature of the similarity measure elaborated in Subsection 7.3.1 shall be used on the level of task allocation to the processors. The classical allocation strategies NF, FF, BF and WF assume a linear similarity measure.

On a list of n tasks, there are $n!$ (linear) permutations possible. Contrary to that, on a ring with n arranged tasks, there are only $(n-1)!$ (circular) permutations [270]. So, at first glance, the degree of freedom seems even to be reduced with the circular approach. This fact is demonstrated by the consolidation of the Lauzac [174] and the Burchard [60] sorting sequence, cf. [220].

But, besides the permutation itself, the initial task when it comes to an allocation by X Fit is another degree of freedom. On a ring of tasks, there is no outstanding point. Hence, all n possibilities of a task to start with are equally possible and shall be checked as done in [222]. The motivation for a variation in the initial task is that, without it, a processor border would be tied to run always between the first and the last task in the list. Such a situation is obviously too restrictive.

Note that the total number of initial (pre-X-Fit) task arrangements becomes $n \cdot (n-1)! = n!$ and is, thus, equal to the number of linear permutations. This is not surprising since the first element in a list corresponds to an initial index element in a ring. In spite of this coincidence in cardinality, the new approach with a ring and a varying initial index is the only appropriate one. For more details on circular statistics, see [143].

Complete Index Offset Search

As a first try, *all* possible index offsets from 0 to $n-1$ shall be considered. This multiplies the effort by the factor n. It might be a reasonable compromise since an improvement is likely while an exhaustive search as a complete case-by-case analysis of all task permutations as inputs for an X Fit strategy is much more complex since the number of permutations of n tasks grows with $n!$.

Start After Largest S Value Gap

Nevertheless, for some situations, the complete index offset search ca be too complex. Then, the question for a more sophisticated choice of a start index compared to the RMST choice of the lowest-S-value task arises.

Based on the considerations on impBu in 7.4.1, a start directly after the *largest S value gap* becomes a reasonable choice. By doing this, the maximum gap is kept out of the required coverage by mappings of task sets to processors, see Algorithm 7.1.

Algorithm 7.1 Modified RMST with Start of X Fit After Largest S Value Gap, based on Burchard *et al.* [60]

```
1: {Order task set in nondecreasing order of $S$ values according to (2.18)}
2: {Find largest gap in the sorted list and set index behind this gap}
3: {Start allocation with Next Fit and the impBu presented in 7.4.1 with task
   after this gap; Apply modulo arithmetics for cycling through all tasks}
```

The gain of it will be highest in situations with a very small S value gap around the zero axis, i.e., task sets with periods both closely below and above powers of 2 as, e.g., valid for the sample task set given in Figure 7.10. See also [224] for further explanations.

7.4.4 Logarithm Base for Period Similarity

Based on the considerations and the example in Subsection 7.3.3, we generalize (2.18) to a parametrized S_i^b value with $b \in \mathbb{N} \setminus \{0, 1\}$ and obtain (7.15).

$$S_i^b := \log_b p_i - \lfloor \log_b p_i \rfloor \tag{7.15}$$

Note that (7.15) includes (2.18) by setting $b = 2$, and so $S_i^2 = S_i$ holds.

The question for convenient choices for the base b remains. Due to the fact that integer powers of the base ratio are implicitly considered, a sieve algorithm similar to the *Sieve of Eratosthenes* [234] seems to be a good choice for selecting useful bases. Each natural number has a unique prime factorization. Thus, its logarithm can be written as the sum of the logarithms of its prime factors since logarithmization is a homomorphism translating multiplication into addition and potentiation into multiplication by the exponent. Hence, an obvious choice for good bases is the set of prime numbers. So, one should try bases $\{2, 3, 5, 7, 11, ...\}$.

On the other hand, the greater base b is chosen the larger the gaps between subsequent integer powers of this base b. But simply periodic task sets (see 1.2.1) with a utilization bound of 1 allow for *every* integer period ratio. Thus, base 2 remains the most important base choice.

But the classic base-2 approach does *not* dominate approaches with other bases as shown by the following counter-example. Let $\{T_1(5, 2), T_2(15, 9)\}$ be a given task set to be RM scheduled. Its total utilization is $u = 1$. With $15 = 3{\cdot}5$, it is simply periodic. Hence, it is RMS-schedulable on a uniprocessor. The S values are $S_1 \approx 0.322$ and $S_2 \approx 0.907$, but the S^3 values are $S_1^3 = S_2^3 \approx 0.465$. While a presorting according to S values tends to separate T_1 from T_2, an allocation of them to one and the same processor is much more likely when using the identical S^3 value. Hence, for this example, base 3 is much more appropriate than base 2.

7.5 A Systematic Terminology for RMST-related Algorithms

The variety of modifications proposed in Subsections 7.4.1 to 7.4.4 calls for a consistent naming scheme. Based on the four modifications suggested, a convenient scheme [222] is presented in Table 7.1.

NF	sBu	noOffset	Base2
FF	Bu	Offset	Base3
	DCT		
	TDA		

Table 7.1: Terminology for RMST-related Scheduling Algorithms

Note that RMST itself is NF-sBu-noOffset-Base2 in this terminology. The first part of the naming is the allocation strategy, the second one describes the uniprocessor schedulability test, and the third one is a binary switch signifying whether the index offset is completely checked or just the task with the lowest S value is taken first. Finally, the last column specifies the logarithm base.

The sample assignments of the four columns in Table 7.1 are not exhaustive. This is quite obvious for columns 1, 2 and 4. Other possible allocation strategies are BF and WF. Further possible uniprocessor schedulablility tests are, e.g., HB and LL, see Section 2.2.2. As discussed in Subsection 7.4.4, other bases than 2 and 3 might be reasonable choices.

But even the binary switch in column 3 of Table 7.1 can be refined. There are compromises between the two extremes *noOffset* meaning to take the minimum-S-value task as start and to test for all tasks as start (*Offset*). As an idea, it is possible to start always behind the largest S value gap on the circle of S values, see 7.4.3, similar to the idea of the Improved Burchard Test, see 7.4.1. Furthermore, one could vary the start index offset in a window of reasonable size. A possible cue to this window size is the expected number of tasks per processor n/m since a processor boundary is likely to be hit when varying the start task index this way.

7.6 Evaluation of the Proposed RMST Modifications

The purpose of this Section is the evaluation of proposed modifications of the RMST algorithm. This is an important step after all the theoretical considerations. Here, simulations with synthetic task sets based on the results from Chapter 6 have been performed.

In this Subsection, the terminology introduced in Section 7.5 is consistently used for reference to the RMST modifications.

7.6.1 Uniprocessor Schedulability Test

Sr and DCT

Both Sr and DCT provide a high sensitivity under a reasonable complexity which is $\mathcal{O}(n \log n)$ for Sr and $\mathcal{O}(n^2)$ for DCT, see Figure 2.4 and [216]. There, their superiority in terms of sensitivity compared to other sufficient RMS schedulability tests up to quadratic complexity has been verified.

Improved Burchard Test

We have run experiments to validate our approaches. For the impBu evaluation, $10,000$ synthetic task sets of $n = 10$ tasks each have been generated for each total utilization level from 0.7 to 0.9 in steps of 0.01. Periods follow a uniform distribution. Utilizations are uniformly distributed, too, and generated by *UUniFast* given in [51], see Subsection 6.2. In Figure 7.11, it is clearly visible that the improved Burchard test (7.5) (7.9) is superior to Burchard's original criterion (2.19) (2.20).

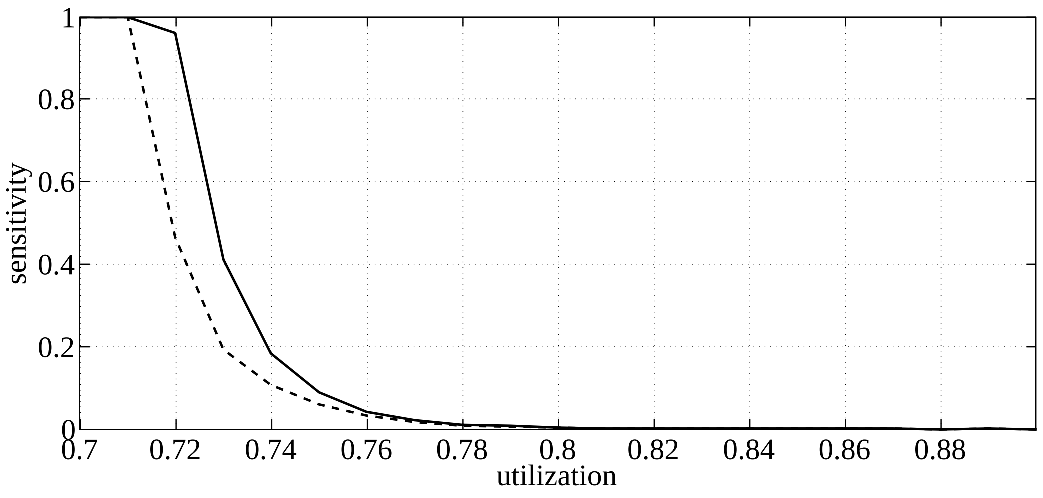

Figure 7.11: Sensitivity Utilization Diagram for Synthetic Task Sets of $n = 10$ Tasks: Burchard's Criterion (dashed) and Suggested New Criterion (solid)

Figure 7.11 considers only the total utilization interval $[0.7, 0.9]$ since below and above, no significant differences between impBu and Bu were detected. From theory [60] we know that Bu is at least as good as LL. By transitivity, this holds also for impBu which is an improvement of Bu. LL has a sensitivity of 100% (exact test) for utilizations up to $\ln 2 \approx 0.693$. Hence, Bu and impBu are as well exact tests, and there is no sensitivity ranking between them for utilizations up to (rounded) 0.7.

Beyond utilizations of 0.9, both Bu and impBu have a sensitivity close to zero. This means that both are no longer suited for the detection of RMS-schedulable task sets. In such high-total-utilization situations, the use of

ASPTS-based sufficient tests, see 2.2.2 and 7.4.1, or even of an exact test like TDA, see 2.2.1, are reasonable options.

7.6.2 Allocation Strategy

As already given in 4.3.4, First Fit is considered the most reasonable allocation strategy. It belongs among NF, FF, BF and WF to the two best ones in terms of minimizing the waste, but BF is only slightly better than FF and needs more calculation, see [202] and [44].

7.6.3 Index Offset for Start of X Fit

Complete Index Offset Search

A *complete* index offset search as suggested in 7.4.3 might be a case-by-case analysis in abundance. The odds for obtaining a valid task partition of a small size are not uniform. Hence, the choice of the index offsets with the highest odds (see next part) or in a restricted window due to the repetitive trend pattern of odds seems to be more reasonable. A processor boundary is likely to be hit when varying the start task index offset from 0 to $\lceil n/m \rceil$, the expected number of tasks per processor rounded up as already alluded in Section 7.5.

Start After Largest S Value Gap

For testing the behavior of RMST with a start of the allocation algorithm after the largest S value gap, $10,000$ synthetic task sets consisting of $n = 10$ tasks each have been generated with a total utilization of 3.5 each. Periods follow a uniform distribution. Utilizations are uniformly distributed, too, and generated by *UUniFast-Discard*, see Section 6.3.

Figure 7.12 demonstrates that the improved algorithm reduces the average number of necessary processors for scheduling the task sets.

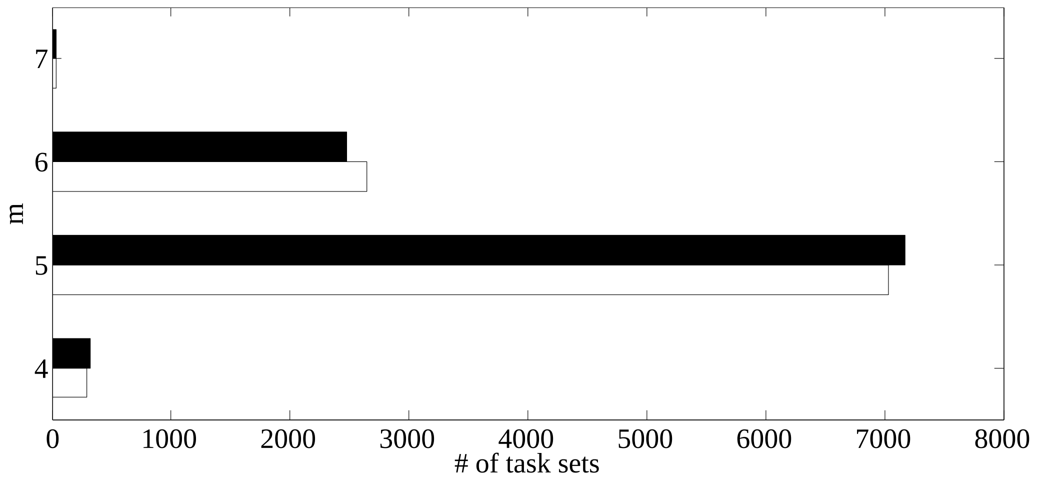

Figure 7.12: Improved RMST (black) vs. RMST (white) : Number of Processors m Required to Schedule Synthetic Task Sets; Decreased Average Visible

7.6.4 Logarithm Base for Period Similarity

The use of the logarithm base 2 can be inferior to base 3 as shown in Subsection 7.4.4. Thus, base-2 approaches are not dominating approaches with other bases. Hence, other bases might be useful.

But in the average case, base 2 turns out to be the best choice which will be confirmed in Subsection 7.6.5.

7.6.5 Combinations

$100,000$ task sets of exactly 10 tasks each are investigated. The target *total* utilization is 2.5. Task utilization values follow a uniform distribution and are obtained using the *UUniFast-Discard* algorithm, see Section 6.3, which is a multiprocessor extension of the *UUniFast* algorithm, see Subsection 6.2. Period values follow an integer (i.e., granularity equals one) log-uniform distribution in the interval $[10, 10^5]$ which is considered to be more appropriate than a uniform distribution, see Section 6.5. WCETs are fully determined by utilization and period values and can be calculated easily by multiplying these two values.

The results of obtained percentages of m values are given in Figure 7.13. Note that m values of 4 and 3 are dominating. The rare m value 5 occurs at a

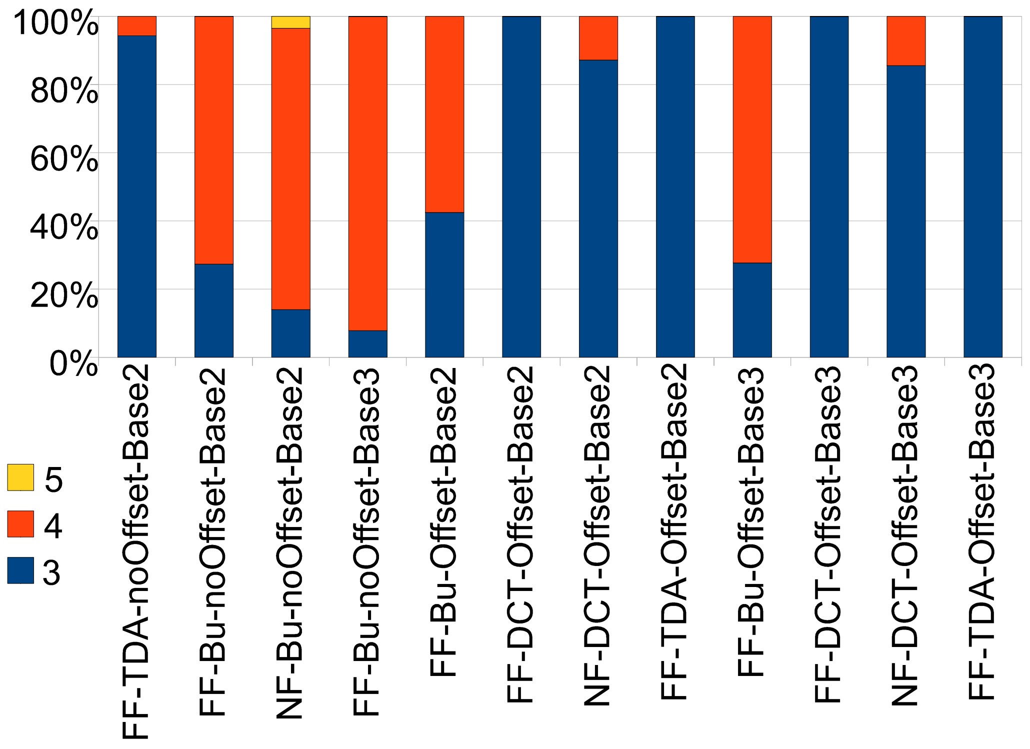

Figure 7.13: Performance of Suggested Algorithms on Synthetic Task Sets: Number m of Processors Required for Partitioned Scheduling, Percentages of Synthetic Task Sets; $n = 10$, $u = 2.5$, $u_{max} = 1$

significant portion only with the algorithm NF-Bu-noOffset-Base2. Since Figure 7.13 shows no clear difference between the four algorithms FF-DCT-Offset-Base2, FF-DCT-Offset-Base3, FF-TDA-Offset-Base2 and FF-TDA-Offset-Base3, the results for them are given as numbers in Table 7.2.

Algorithm	m=3	m=4	m=5
FF-DCT-Offset-Base2	99,908	92	0
FF-DCT-Offset-Base3	99,910	90	0
FF-TDA-Offset-Base2	99,918	82	0
FF-TDA-Offset-Base3	99,917	83	0

Table 7.2: Performance of RMST Modifications on Synthetic Task Sets: Number m of Processors Required for Partitioned Scheduling, Percentages of Synthetic Task Sets; $n = 10$, $u = 2.5$, $u_{max} = 1$; Zoom Into the Best Algorithms

The most promising combination in terms of average number-of-processors minimization performance is FF-DCT-Offset-Base2. Since all four dimensions can be combined freely, there are 32 combinations. In the simulations given here, only 12 of them have been listed since FF-X-noOffset-Base2 algorithms[21] have already been discussed in [216], and trends of the usefulness of each modification can be concluded from single-dimension modifications. The exact test TDA has been added to the uniprocessor scheduling criteria in order to have a reference. Note that a promising combination like FF-TDA-Offset-Base2 is not optimal for the minimization of m, it is just a very good heuristic. Just one optimal stage does not make the entire procedure optimal. Only an exhaustive search would find the global optimum.

7.7 Summary

Summing up, the algorithm with the best performance in terms of minimizing the expected number of processors necessary is FF-TDA-Offset-Base2. But FF-TDA-Offset-Base3, FF-DCT-Offset-Base3 and FF-DCT-Offset-Base3 show results only slightly worse. The differences between FF-DCT-Offset-X and FF-TDA-Offset-X are so small that the polynomial complexity of FF-DCT-Offset-X compared to the pseudo-polynomial one of FF-TDA-Offset-X makes

[21]X is a wildcard.

FF-DCT-Offset-Base2 to the overall winner taking both performance and complexity into account. Similar results have been obtained under variation of the

Algorithm	m=6	m=7	m=8	m=9
FF-TDA-noOffset-Base2	99,584	416	0	0
FF-Bu-noOffset-Base2	64,409	35,387	4	0
NF-Bu-noOffset-Base2	14,480	78,672	6,843	5
FF-Bu-noOffset-Base3	1,804	96,672	1,523	1
FF-Bu-Offset-Base2	87,475	12,525	0	0
FF-DCT-Offset-Base2	100,000	0	0	0
NF-DCT-Offset-Base2	79,653	20,335	12	0
FF-TDA-Offset-Base2	100,000	0	0	0
FF-Bu-Offset-Base3	15,230	84,770	0	0
FF-DCT-Offset-Base3	100,000	0	0	0
NF-DCT-Offset-Base3	73,461	26,512	27	0
FF-TDA-Offset-Base3	100,000	0	0	0

Table 7.3: Performance of RMST Modifications on Synthetic Task Sets: Number m of Processors Required for Partitioned Scheduling, $n = 20$, $u = 5$, $u_{max} = 0.5$

number of tasks $n = 5, 10, 20, 40$, of the total utilization $u = 1, 2, 2.5, 5, 10$, and of the maximum utilization $u_{max} = 0.5, 1$. The advantage of base-2 approaches compared to base-3 approaches increases with an increasing number of tasks, see Table 7.4. With a restricted maximum utilization $u_{max} = 0.5$, FF-DCT-Offset-Base2 (and its TDA and base-3 relatives) can sometimes reach the optimal number of processors even for all synthetic task sets, see Table 7.3. These results suggest that the favored algorithm FF-DCT-Offset-Base2 even increases its winning margin when it comes to more realistic scenarios with more tasks ($n > 10$) and a restricted maximum utilization $u_{max} < 1$.

Algorithm	m=11	m=12	m=13	m=14	m=15	m=16	m=17	m=18
FF-TDA-noOffset-Base2	15,430	67,550	16,102	902	16	0	0	0
FF-Bu-noOffset-Base2	303	43,296	49,502	6,608	289	2	0	0
NF-Bu-noOffset-Base2	9	2,645	27,116	47,523	20,395	2,256	55	1
FF-Bu-noOffset-Base3	21	23,126	61,692	14,369	771	21	0	0
FF-Bu-Offset-Base2	445	59,741	36,725	2,974	113	2	0	0
FF-DCT-Offset-Base2	46,384	49,432	3,978	204	2	0	0	0
NF-DCT-Offset-Base2	101	8,214	43,107	39,987	8,250	338	3	0
FF-TDA-Offset-Base2	48,287	47,573	3,935	203	2	0	0	0
FF-Bu-Offset-Base3	152	55,252	41,299	3,184	111	2	0	0
FF-DCT-Offset-Base3	45,248	50,574	3,970	206	2	0	0	0
NF-DCT-Offset-Base3	58	7,716	42,305	40,648	8,896	374	3	0
FF-TDA-Offset-Base3	47,441	48,417	3,935	205	2	0	0	0

Table 7.4: Performance of RMST Modifications on Synthetic Task Sets: Number m of Processors Required for Partitioned Scheduling, $n = 20$, $u = 10$, $u_{max} = 1$

Chapter 8

Total Utilization Thresholds for Partitioned EDF

This Chapter is based on the publication [223] at RTAS 2011.

8.1 Introduction

As explained in Subsection 1.7.4, utilization bounds are a very convenient means in constructing sufficient schedulability tests for scheduling problems. But utilization bounds in their typical application in RT scheduling theory are sufficient and no exact schedulability tests, especially for RM scheduling on uniprocessor and partitioned multiprocessor scheduling. This means that there are always schedulable task sets with a utilization beyond a utilization bound as shown in [53]. In many situations, especially for soft real-time systems, it might be too pessimistic to ignore these high-utilization task sets completely. The measure Numerical Optimality Degree (NOD) [51] takes them into account. Comparisons there show that the increase of schedulability metrics by applying NOD is significant.

The probability of schedulability shall be calculated depending on the total utilization of a task set. Knowing this function or at least some nodes of this curve allows us then to infer the probability of schedulability purely based on the total utilization value by a simple comparison with important data points. Obviously, due to the use of probabilities, the approach is mainly suitable for soft real-time systems. But we will also try to profit from it by a hard RT test schema for reducing the average computation effort of schedulability tests, see Subsection 8.7.6.

This general procedure shall unleash the potential of task sets of medium utilization, see Figure 1.15, for obtaining less pessimistic schedulability results.

8.2 Thresholds and Sharpness

8.2.1 Phase Transitions in Physics

Phase transitions are abrupt changes between states of matter[1] and are very common in nature. An outstanding example are the transitions between solid, liquid, and gaseous phases[2]. The phase depends upon the physical variables temperature and pressure. This functional relationship can be visualized in a phase diagram. Since boiling and freezing goods belongs to our everyday experience, phase transitions are easily accessible by regarding this example.

An example apart from everyday life is the disappearance of the electrical resistance of some materials when cooled down below a critical temperature, aka *superconductivity*. Such a temperature is then a *sharp threshold* value.

Third, the magnetic properties of some materials when heating them change at the *Curie temperature* from ferromagnetic to paramagnetic. Again, this temperature is a sharp threshold.

All three examples have in common that macroscopic physical properties like volume, electrical resistance or magnetic properties change abruptly [3]. Such an abrupt change is a hallmark of phase transitions and would be very useful in scheduling theory. We will use *per-task-set* properties like total utilization

[1]In the more abstract view of system theory, phase transitions in the state space from non-chaotic to chaotic behavior can be predicted by the *Lyapunov Projection Method* [286]. Chaotic systems are deterministic but sensitive to initial conditions. The extent of this sensitivity to initial conditions is described by the Lyapunov exponent.

[2]Sometimes, the plasma phase is included as a fourth, and the Bose-Einstein condensate (BEC) as a fifth state of matter.

[3]A second commonality of them is that the temperature serves as control parameter. Since temperature is a macroscopic intensive variable with a microscopic explanation by statistical mechanics and thermodynamics, there is always an indirect impact of a macroscopic variable (temperature) on other ones with explanations on the microscopic level.

analogously to temperature. The goal is to control schedulability by total utilization.

But to enter the field of real-time schedulability analysis, we make a detour via graph theory which is powerful enough to build a bridge from physics to the computer-science field of scheduling. Only by a modeling of scheduling problems in terms of graph theory, the existence of sharp total utilization thresholds can be proved using Friedgut's Theorem, cf. Subsection 8.3.5. All three fields, physics, graph theory and scheduling have in common that reasoning on macroscopic (per-graph, per-task-set) properties happens using statistics on microscopic (per-vertex/per-node, per-task) properties.

8.2.2 Thresholds and Sharp Thresholds

The following two definitions are based on the ones given in Section 4.3 in [125]. A *threshold* function is a function that imitates the perfect case of a step function with the lower level for very small and the upper level for very large argument values[4]. This needs only to be valid in an approximate manner, i.e., for large numbers $n = |A|$ of elements in a set A. We want to make statements on the property $Q(A)$ based on the test function $t(A)$. Further, let $Q(A)$ be a monotone increasing property. The function $x_T(n)$ is a threshold function if (8.1) holds for any $x(n)$.

$$\lim_{n\to+\infty} \Pr\left(Q(A) \mid t(A) = x(n)\right) = \begin{cases} 0 & \text{if } x(n) \ll x_T(n) \\ 1 & \text{if } x(n) \gg x_T(n) \end{cases} \tag{8.1}$$

In (8.1), $f(n) \ll g(n)$ means that $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$.

Often, this threshold condition can be ensured by simple necessary and sufficient conditions. So, a stronger concept is required to obtain more useful results. A threshold function $x_T(n)$ is called *sharp* if and only if for any $x(n)$ condition (8.2) holds.

$$\forall\epsilon > 0 : \lim_{n\to+\infty} \Pr\left(Q(A) \mid t(A) = x(n)\right) = \begin{cases} 0 & \text{if } x(n) < (1-\epsilon)x_T(n) \\ 1 & \text{if } x(n) > (1+\epsilon)x_T(n) \end{cases} \tag{8.2}$$

[4]The symmetric case of a *downward* step can be easily simulated by a negation of the original function and is, thus, w.l.o.g. covered.

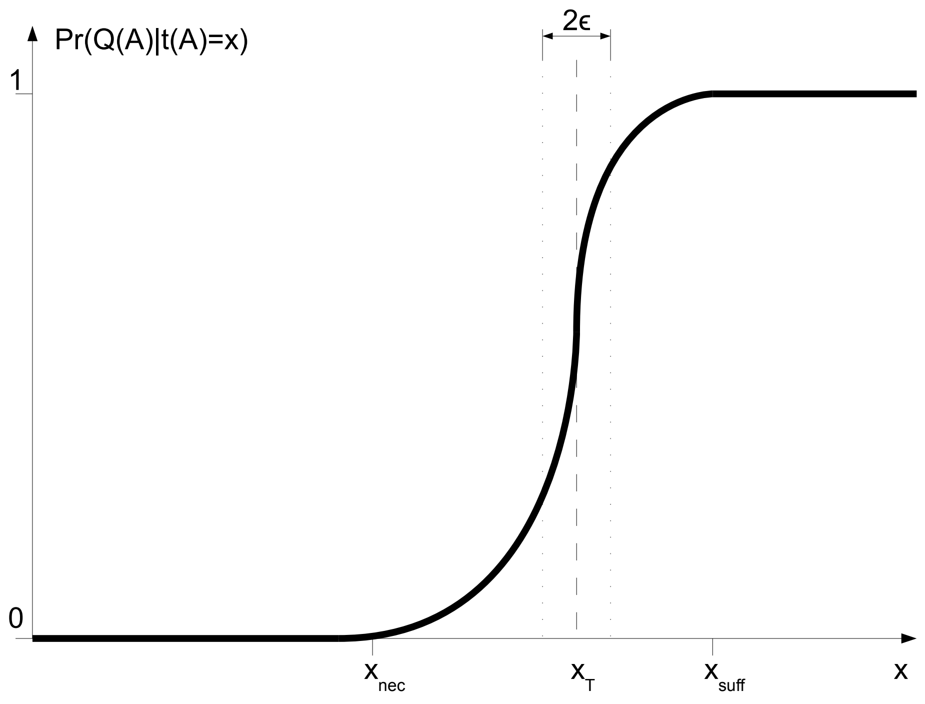

Figure 8.1: A Sharp Increasing Threshold x_T of $\Pr(Q(A) \mid t(A) = x)$ Localized Between the Value of the Necessary Condition x_{nec} and the Value of the Sufficient Condition x_{suff} With Control Parameter x

The interval of width 2ϵ over which the probability of property Q increases from 0 to 1 is called the *threshold interval.* As $n \to +\infty$, the threshold interval becomes arbitrarily narrow, indicating a sharp threshold. This expresses that the target step function can be arbitrarily well approximated by the threshold function when using a sufficiently large n. A threshold not being sharp is called a *coarse* threshold.

The basic idea of a sharp threshold function with a threshold interval is summarized in Figure 8.1.

8.3 Sharp Thresholds of Graph Properties

As we could see from the physics examples in Subsection 8.2.1, adjusting a macroscopic parameter which describes microscopic properties can lead to the abrupt emergence of a new phase with modified macroscopic properties. Before studying phase transitions in graphs, some terms and concepts will be introduced.

8.3.1 Graphs and Hypergraphs

An (undirected) *graph* is an ordered pair $G = (V, E)$ with a set V of vertices and a set E of edges which are 2-element subsets[5] of V. The number of edges meeting at vertex i is called the *degree of the vertex* k_i.

For the generalization to a (undirected) *hypergraph*, the restriction of the cardinality of the elements of V to 2 is omitted, all non-empty subsets of V are allowed. Formally, for a hypergraph $H = (V, E)$ we have $E \subseteq \mathcal{P}(V) \setminus \emptyset$, i.e., the set of hyperedges is a subset of the power set of the set of vertices without the empty set.

A *graph property* is a predicate "[..] of graphs which depends only on their isomorphism class." [101][6] A graph property is called *monotone* if the addition of edges to a graph satisfying this property cannot invalidate this property. Well-known examples of such monotone graph properties are: connectivity, existence of a Hamiltonian cycle, existence of a clique (complete subgraph) of size t, planarity, the clique number of G is larger than that of its complement, the diameter of G is upper-bounded by s, etc. [101]

8.3.2 Random Graphs

The key concept for studying sharp thresholds of monotone graph properties is that of *random graphs* which was independently proposed by Erdős and Rényi [101] and Gilbert [121] in 1959.

[5]This implies that neither multiple edges nor loops (aka slings) are allowed. For clarity, we give this explicitly.

[6]This means that this predicate depends only on the graph's abstract structure but not on its representations such as particular labellings or drawings.

Assuming a graph $G = (V, E)$ with $|V| = n$ vertices[7] , $G(n, M)$ shall have M edges. The range of M is then $0 \leq M \leq N = \binom{n}{2}$ since an empty graph has 0 edges and a complete graph possesses $\binom{n}{2} = \frac{n(n-1)}{2}$ edges.

Erdős/Rényi Model

According to Erdős and Rényi [102], a random graph $G(n, M)$, aka *uniform* random graph, is chosen with equal probability from all graphs with exactly n vertices and exactly M edges. Hence, the probability of a particular such graph under these constraints is $Pr(G(n, M)) = 1/\binom{\binom{n}{2}}{M}$. This corresponds to a snapshot of a stochastic process of a step-by-step inclusion of edges after M steps, starting with an empty graph[8]. At each (time) step, the stochastic process adds one new edge chosen equiprobably from the set of missing edges.

Gilbert Model

In a slightly different approach, in a Gilbert random graph $G(n, p)$, aka *binomial* random graph, [121] every possible edge is set independently with a probability of p. The number of edges in a $G(n, p)$ graph follows a binomial distribution with parameters $N = \frac{n(n-1)}{2}$ and p.

Relationship and Unification

Using the conversion $M \sim pN = p\frac{n(n-1)}{2}$, both models can almost be used interchangeably [56] [102] for large graphs[9].

Dependency upon Number of Vertices

It is important to mention that typically both $M = M(n)$ and $p = p(n)$ are functions of the number of vertices n. The probability of having a graph

[7]Although variable n is already occupied by the meaning *number of tasks*, cf. Section 1.2, we use it as well for the number of vertices in a graph. Later on, it will turn out that these two semantics can be merged without any danger of ambiguity since a task will be represented by a vertex.

[8]An empty graph has no edges and, thus, n isolated vertices.

[9]In [280], it is demonstrated by an estimate using binomial distribution converging to Poisson distribution and Kolmogorov's inequality that the difference in the number of edges assumed by the two models is of order $1/n$, i.e., negligible for large n.

property goes then to zero for a parameter below the threshold value and goes to one for it beyond the threshold value when $n \to +\infty$.

Some Other Random Graph Models

The *world wide web* (WWW)[10] is not well described by the Erdős/Rényi model. The reason is that pages (vertices) with a number of links (edges) show a higher probability of receiving new links, an effect aka *preferential attachment*. This leads to the creation of hubs by a self-energizing accumulation of links. The *Barabási-Albert model* [30] uses no longer a constant edge probability p. Instead, the chance for the presence of an edge is chosen proportionally to the percentage of already existing edges to this node (8.3).

$$p_i := \Pr(\text{edge from new vertex } a \text{ to vertex } i) = \frac{k_i}{\sum_{j \le a} k_j} \tag{8.3}$$

Note that k_i in (8.3) means the degree of vertex i and the summation is performed over all preexisting vertices j. Then, a degree distribution is scale-free and follows a power-law distribution of the form $\Pr(k) = \frac{2m^2}{k^3}$, see [30].

In the *Watts/Strogatz model* [273], the high level of clustering observed in real-world networks is better modeled. The model uses three parameters, the number of vertices n, the average vertex degree[11] $k = \left(\sum_{i=1}^{n} k_i\right)/n$ and a randomness degree $\beta \in [0,1]$ giving a regular lattice for $\beta = 0$ and a Gilbert graph $G(n,p)$ with $p = M/N = nk/\left(2\binom{n}{2}\right)$ for $\beta = 1$.

8.3.3 Connectivity in Random Graphs

A graph is said to be connected if there is a path from every vertex in the graph to every other one. Simple considerations show that there are at least $n-1$ edges needed to build a minimum tree, a spanning tree for obtaining a connected graph.

The value of $M(n) = \frac{n}{2} \ln n$ edges is a sharp threshold for the connectivity property of a graph. This corresponds to a Gilbert probability threshold of $p(n) = \ln n/n$. More precisely, in a narrow band (threshold interval)

[10]Earlier studies used citation networks of papers [244] where the same effects occur.

[11]Variable k is assumed to be an *even* integer since otherwise the number of edges $M = \frac{nk}{2}$ could be non-integer.

around the threshold value, there is a probability of $e^{-e^{-2c(n)}}$ of connectivity for $M(n) = \frac{n}{2} \ln n + c(n)n$. This is only a generalization of the first mentioned more instructive cases where this term tends to 0 or 1. It is sometimes called the *double-exponential theorem*.

Erdős and Rényi [102] observed that something magical happens when passing the value of $M(n) = n/2$ edges in a graph. A giant connected component of a complex structure appears. The discovery of this phenomenon opened the door to an entire new field of sharp thresholds of graph properties which enables to predict graph properties of large graphs (in terms of the number of vertices) solely based on the number of edges.

In a similar way, we would like to predict the schedulability property of given task sets solely based on their total utilization u.

8.3.4 More Graph Properties with Sharp Thresholds

The success of the sharp threshold approach on random graphs concerning the connectedness property has not been ephemeral. For various graph properties, sharp thresholds were found. Some examples are given subsequently.

The *expected number of isolated points* in a graph converges to e^{-2c} for $M(n) = \frac{n}{2} \ln n + c(n)n$ and $n \to +\infty$ since the limit distribution is of Poisson type [102].

The minimum vertex degree $\delta(G)$, the minimum number of edges incident to a vertex, tends to a Poisson distribution for $n \to +\infty$ [103].

The property of the existence of a Hamiltonian cycle[12] is characterized by a sharp threshold at $\frac{n}{2} \ln n + \frac{n}{2} \ln \ln n$ as proved in [168].

These examples shall suffice to illustrate the wide applicability of the probabilistic sharp threshold approach in graph theory.

8.3.5 Friedgut's Theorem

In 1996, Friedgut proved his seminal theorem that every monotone graph property has a sharp threshold [113]. In [112], his theory was applied to various graph properties and to the k-SAT problem to be discussed in Section 8.4.

[12]A Hamiltonian cycle is a closed loop through a graph that visits each vertex exactly once.

A step towards application of sharp threshold theory to graph property testing was achieved by Alon and Shapira [6] as they showed that every monotone graph property is testable, i.e., it can be tested by a tester with *one-sided error*[13], and with query complexity depending only on ϵ but not on the size of the graph n.

8.4 Satisfiability Problems

8.4.1 Basic Problem and NP-Completeness

Boolean logics is a powerful means of abstraction. The two-valued approach restricts the domain of n Boolean variables $x_1, x_2, ..., x_n$ to *true* (1) and *false* (0) each. Then, we have 2^n possible assignments. Every Boolean expression can be transformed into a *conjunctive normal form* (CNF). There, disjunctions of negated and non-negated variables[14] are combined by conjunctions. According to the generalized form of De Morgan's Laws, each k-clause with a maximum of $2 < k \leq n$ literals can be interpreted as a constraint excluding exactly one of the 2^k possible assignments in the clause[15]

Satisfiability (SAT) is now a binary decision problem asking for the existence of a variable assignment satisfying all constraints imposed by such a CNF. If the maximum number of literals in a clause of the CNF is restricted by k, the problem specializes to k-SAT.

It is well-known that the problem can be solved in exponential time by the brute-force approach of examining all possible 2^n assignments which quickly becomes intractable for larger values of n. Now, the obvious question is whether there is a polynomial-time algorithm for solving k-SAT. The answer is yes for $k = 2$, 2SAT is in P. As shown by Cook [81] and Karp [152], k-SAT for $k \geq 3$ is NP-complete and 21 infamous combinatorial and graph theoretical problems

[13]This says that whenever the predicate $Q(G)$ is satisfied, the tester declares this with probability 1 [6]. In our terminology, this corresponds to a sufficient test, see Subsection 1.7.1.

[14]This is also called a *literal*.

[15]The restriction to a CNF is essential since a Boolean formula in a disjunctive normal form (DNF), a disjunction of conjunctions, can easily be solved in polynomial time. One has only to check whether there is a conjunction which does not contain a variable in both negated and non-negated form.

are computationally as complicated as SAT. Thus, unless $\mathsf{P} = \mathsf{NP}$, there is no polynomial-time algorithm for solving SAT or the special case of 3SAT.

8.4.2 Sharp Thresholds of SAT when Controlled by Constraint Density

Constraint density of a SAT problem is the ratio of the number of clauses in the CNF to the number of variables n. Controlled by this parameter, the probability of the existence of a solution in a SAT instance exhibits a sharp threshold behavior where it drops quickly from 1 to 0 [112]. This statement of existence of sharp thresholds is theoretically important, but without application.

So, we are searching for the concrete (downward) threshold of the probability (8.4).

$$\Pr_k(c) := \Pr(\text{a random } k - \mathsf{SAT} \text{ formula with } cn \text{ clauses is satisfiable}) \quad (8.4)$$

This was solved exactly for $k = 2$ by Goerdt [122] where the threshold is at $c = 1$. By statistical methods, bounds of the k-SAT problem thresholds were calculated as $2^k \ln 2 - k$ and $2^k \ln 2$ in [2]. Using a method from statistical physics approximate values for $k = 3, 4, 5$ were obtained as 4.267, 9.931 and 21.117 in [208].

An application of the sharp threshold values method to the *traveling salesman problem* (TSP) and to *anytime algorithms* is discussed in [18].

In [290], it is doubted whether SAT really exhibits a phase transition. The authors argue that SAT thresholds could be simple sharp threshold phenomena since "[..] there is no interaction of the elements of a Boolean instance, i.e., clauses, variables, or solutions, that leads to this phenomenon." [290] The synthesis of these two points of view is that both are correct, the microscopic view of a huge number of particles turns into a phase transition as the macroscopic manifestation of it.

8.5 Percolation Theory

Percolation is a noun originating from the latin verb *percōlāre* meaning to filter, to trickle through. It is a question practically relevant whether a fluid can move through a porous material. The leakage of a fluid through such a material is called percolation.

The border between the phases of non-percolation and percolation is called the *percolation threshold*. More precisely, given a random process with an occupancy probability p, the percolation threshold is the critical value of this probability such that infinite connectivity (percolation) first occurs.

This general concept can be applied to lattices which is the most typical case, but also to random graphs as we will see in Subsection 8.5.2.

Besides the fact that percolation theory has some direct application in geology, physics and epidemiology, a second advantage is that percolation is visually accessible [57].

8.5.1 Percolation Thresholds of Lattices

An n-dimensional lattice is a graph $G(\mathbb{Z}^n, E)$ on the n-dimensional grid $\mathbb{Z}^n$. The simplest non-trivial case is in $n = 2$ dimensions with edges restricted to take course to lattice sites in the 4-neighborhood[16], aka *square lattice in 2D*, see Figure 8.2. This is called *bond percolation*[17] since occupance is regarded according to edges.

As in the Gilbert model, see 8.3.2, each edge may exist with probability p. Then, a percolation threshold exists [58]. The threshold value is at 0.5 as shown in [165].

A second type of percolation is *site percolation* where lattice sites instead of edges (bonds) are randomly occupied. Here, the exact threshold is known for a triangular lattice which is also 0.5, see [277]. For almost all other lattices, no closed-form results of thresholds have been found up to now [277].

8.5.2 Percolation Thresholds of Graphs

Percolation theory can also be applied to graphs without a metric determined by the lattice. First, assuming a regular graph where all vertices have the same vertex degree k, the percolation threshold is $1/k$. For Erdős/Rényi random

[16]This is aka Von Neumann neighborhood. It is determined by all sites having a *Manhattan distance* of 1 to the original site.

[17]Bond is used synonymously for edge.

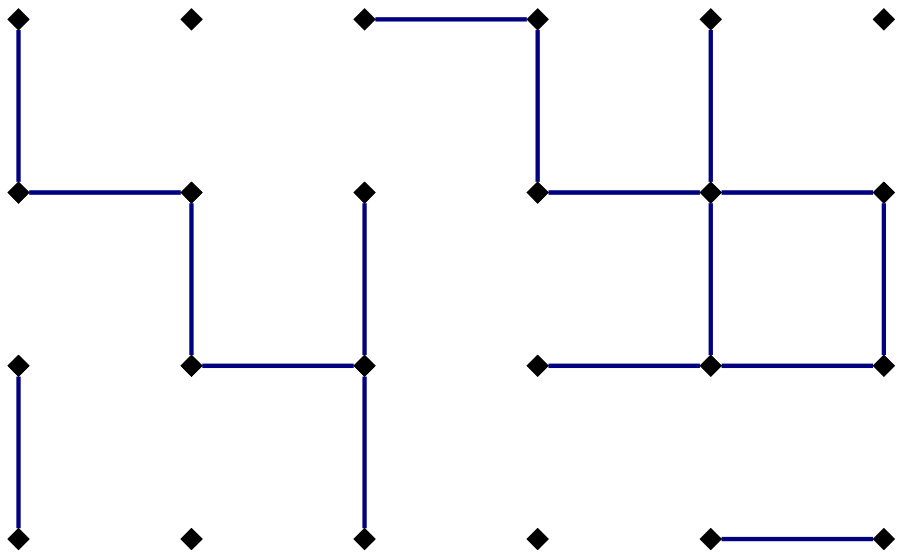

Figure 8.2: Example of Square Lattice Bond Percolation in Two Dimensions With Percolation Threshold 1/2

graphs with a Poissonian vertex degree distribution, the percolation threshold is $1/\mathbb{E}[k] = n/(2M) = 1/(p(n-1))$ where $\mathbb{E}[k]$ is the *expectation* of the vertex degree [80].

8.6 Thresholds for Uniprocessor RMS

In 2007, Gopalakrishnan and Caccamo [128] showed that task set[18] property RMS schedulability on a uniprocessor has a sharp threshold when controlled by the parameter total utilization u. But this is a non-constructive proof of existence of such an RMS threshold. Hence, the problem of determining the threshold value remains open.

The sharp threshold behavior remains also when considering constrained subsets like assigning all tasks the same utilization $u_i = 1/u$ or fixing the period p_i to a constant period vector [126].

A closed-form solution for the RMS threshold is elusive and still an open problem [126] [127]. Hence, the empirical determination of the threshold was tried. First, it is lower-bounded by the LL bound (2.10) as a sufficient test

[18]Assuming the Liu/Layland task model, see Subsection 1.9.1

and upper-bounded by the necessary test (2.1), see Figure 8.1. Next, the results of the experiments performed in [126] suggest that the uniprocessor RMS threshold is situated between 0.8 and 0.9, giving 0.85 as a good estimator. Third, the experiments show that the total utilization threshold decreases when increasing the number of tasks n which indicates an analogous behavior to the LL bound (2.10). Even a convergence is expected due to Friedgut's theorem, see Subsection 8.3.5, and a graph model of scheduling [126].

The real value might even be higher since a uniform period distribution was assumed in [126] which is not appropriate and penalizes RM, cf. [99]. Note that the threshold value then seems to come very close to the famous classical result of a breakdown utilization of 0.88, see [181], in spite of its weaknesses uncovered in [51] and [53].

8.6.1 Simply Periodic Task Sets

For simply periodic task sets, see 1.2.1, we know that the utilization bound (2.14) and the utilization of the necessary schedulability condition (2.1) coincide. This leaves no space for any variation of a sharp threshold. Hence, the sharp threshold in this simple case is at $u = 1$. It is not only a sharp threshold but a step function, a binary switch.

8.7 Thresholds for Partitioned EDF

Since uniprocessor EDF for implicit-deadline tasks has the simple exact schedulability condition $u \leq 1$, see (2.35), the sharp threshold value there is located at 1, too. Hence, the next challenging problem is to determine the total utilization threshold for partitioned MP scheduling, and first for partitioned EDF as in partitioned RMS, period values have to be taken into account which leads to a further complication, see also Section 4.4.

This is a reasonable problem since the existence of a sharp total utilization threshold was proved by Gopalakrishnan in 2006 [125], chapter 6.3 using matchings in hypergraphs. There, the vertices represent the tasks and a hyperedge[19] is set if and only if the adjacent tasks can be mapped to the same processor. To have a matching with at most m hyperedges represents then exactly the existence of a feasible partitioning, the property we want to characterize. It is a global and monotone hypergraph property. Hence, Friedgut's theorem can be applied. The only missing link is that between the hyperedge probability and our control parameter, the total utilization u of the task set. The lower the total utilization, the higher the expected number of hyperedges and, thus, the hyperedge probability [125].

Here, we accept this result and focus on the determination of partitioned EDF thresholds based on [223].

8.7.1 Recapitulating Lower and Upper Bound

From previous chapters, we know that partitioned EDF system utilization threshold is lower-bounded according to (4.1) (4.2) and upper-bounded by (3.1). Integrating both inequalities gives (8.5). The basic idea of the threshold is to remove the middle part in Figure 1.15 by splitting it and relocating it to the two fractions to the left and to the right respectively, or, at least, to reduce its width significantly to a tiny threshold interval.

$$\frac{m+1}{2} \le u_T \le m \Leftrightarrow \frac{m+1}{2m} \le u_{T,sys} \le 1 \tag{8.5}$$

8.7.2 Basic Approach of Estimating Probability of Schedulability

According to Laplace and the entire school of frequentists, probability is defined as the ratio of the number of *favorable* to the number of *possible* cases on the long-run average.

Our task set to be scheduled using partitioned EDF is fully characterized by the n-dimensional vector of task utilizations $(u_1, u_2, ..., u_n)$. Typically, we consider the influence of the total utilization u. Hence, $u = \text{const}$ is a constraint for the possible cases. Then, taking a measure of the region of favorable cases (schedulable) and a measure of the region of possible cases, both under the

[19]A hyperedge can connect any number of vertices ≥ 1. Here, contrary to Subsection 8.3.1, we do not exclude loops. This allows us to model the fact that every task fits on its own on a processor, cf. the necessary condition (3.2), by such a loop.

$u = \text{const}$ constraint, in the Laplacian quotient gives a reasonable approach for the calculation of the probability of schedulability depending on the total utilization u.

Here, we choose the standard volume based on the Euclidean norm as a measure of the regions. The subsequent methods in Subsections 8.7.3, 8.7.4 and 8.7.5 differ in the integration methods and in the application of sufficient or exact schedulability tests.

8.7.3 Monte Carlo Simulation using FFD

In [125], the sufficient constructive test First Fit Decreasing (FFD), see Subsection 4.3.5, on randomly chosen task sets was applied. In effect, this is Monte Carlo integration combined with a sufficient schedulability test.

Gopalakrishnan's study [125] considers $m = 2; 4; 8; 16; 32; 64$ processors and $n = 2^i m + 1; i = 0; 1; 2; 3; 4$ tasks respectively except for $m = 64$ where only the three cases $n = 65; 129; 257$ are regarded. In all cases, a sharp threshold for larger values of n—as predicted by theory—was observed. These thresholds are located at about 0.95 in terms of system utilization. There is a huge difference to the lower bound in (8.5) which quickly drops from 0.75 for $m = 2$ to about 0.51 for $m = 64$, indicating its convergence to 0.5. Thus, the sharp threshold approach succeeds empirically for the property schedulability with partitioned EDF scheduling.

Such a Monte Carlo simulation is absolutely reasonable, it is an approved method of numerical integration. Nevertheless, Gopalakrishnan's approach bears two shortcomings. First, the FFD heuristics is only an approximation from below. Task sets not being schedulable by FFD might be schedulable by the brute-force method of complete search. Second, the Monte Carlo method applied here comes at the risk of *biases* which are sometimes hard to discover, cf. Chapter 6. Hence, it is worth to look for other methods to be discussed in Subsections 8.7.4 and 8.7.5.

8.7.4 Numerical Integration With Constraint Evaluation on a Lattice

The sufficient test using FFD is replaced by an exact test based on constraint evaluation. Next, non-randomized numerical integration methods are applied.

For determining the schedulability probability, we first use a generalized midpoint rule for numerical integration [223]. An equidistant ($\mathit{step} > 0$) lattice of u_i values is generated in the space $(0, 1]^n$. There are $(1/\mathit{step})^n$ vertices in the lattice. Then, the ratio of instances meeting the RMS schedulability constraints to the number of them meeting the total-utilization condition is an estimator of the probability. The number of instances under investigation is controlled by *step*. According to frequentists' probability, the ratio converges to $\Pr(u, n, m)$ for $\mathit{step} \to 0$.

First, the simplest reasonable case of $n = 3$ tasks on $m = 2$ processors is considered. The interval of interest is, from (8.5), $[1.5, 2.0]$. Here, exact constraints for schedulability can be derived when considering the EDF criterion $u \leq 1$ and that at least two of the three tasks must be assigned to one and the same processor. Thus, the schedulability constraint is (8.6).

$$u_1 + u_2 \leq 1 \vee u_1 + u_3 \leq 1 \vee u_2 + u_3 \leq 1 \tag{8.6}$$

The total-utilization condition can be obtained when taking into account that there are only two degrees of freedom when u is fixed. Thus, the third utilization value is just $u_3 = u - u_1 - u_2$. Then, in the u_1u_2 square, it has to be checked whether a $0 < u_3 \leq 1$ is obtained. Using a *C* program, see Listing 8.1, our numerical integration method is just counting the points meeting the constraints and then to print out the ratio for the respective u value.

Listing 8.1: *C* Code for Estimating the Probability of EDF Schedulability of 3 Tasks on 2 Processors Controlled by Their Total Utilization u

```
#include<stdio.h>
int main()
{
  double a, b, c, u;      // task utilizations and total u
  double u_step = 1e-2; // step for utilization loop
  int s;                  // # of schedulable task sets
  int t;                    // total number of task sets
  for (u=1.5; u<=2.0+u_step/2.0; u+=u_step)
  {
    t=0;  // initialization (reset) of
    s=0;  //  both counters
    for (a=0; a<=1+u_step/2.0; a+=u_step)
      for (b=0; b<=1+u_step/2.0; b+=u_step)
      {
        c = u-a-b;        // remaining utilization
        if (c<0 || c>1) // no useful utilization value
          continue;
        t++;
        if (a+b<=1 || a+c<=1 || b+c<=1)
          s++;
      }
    printf("%4.2f  %4.2f  %4.2f\r\n",
      u,
      (double)s/t,                    // success ratio
      3.0+3.0/(2.0*u*u-6.0*u+3.0)); // closed form
  }
  return 0;
}
```

In an analogous way, the problem of assigning 4, 5 or 6 tasks to 2 processors has been analyzed using this integration method. The results are plotted in Figure 8.3. Greater n values have not been considered yet since the complexity of the constraint system (number of inequalities) grows exponentially[20] with n which becomes quickly intractable[21].

[20]The overall complexity is driven by two additional effects. First, the complexity of the individual inequalities grows in parallel. The number of summands increases linearly with n. Second, the number of lattice vertices grows exponentionally with the dimension n.

[21]Using 100 lattice vertices in each dimensions leads to a complexity factor 100 when incrementing n. Calculations for $n = 5$ took about one minute, and more than one hour for $n = 6$. Hence, we estimate the running time for $n = 7$ in the order of days, and for $n = 8$ in the order of years.

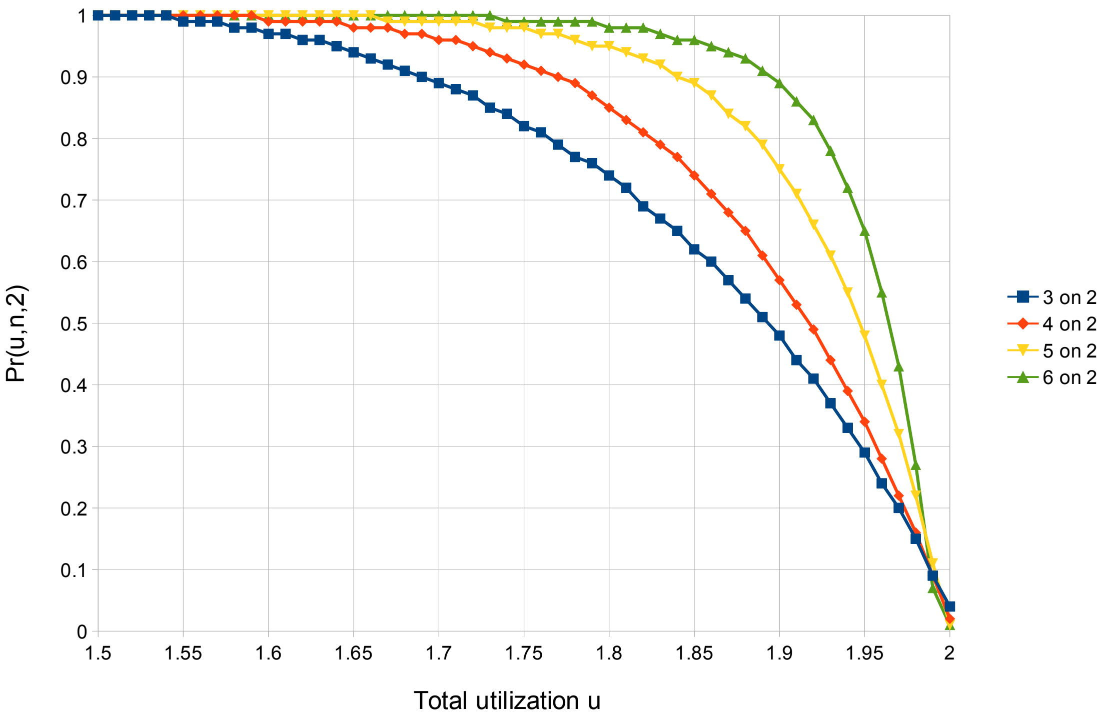

Figure 8.3: Estimated Probabilities of Schedulability for Partitioned EDF by Numerical Integration Based on a Lattice of 100^n Vertices and Constraint Evaluation

Unfortunately, the probability curves in Figure 8.3 do not exhibit a sharp threshold as idealized in Figure 8.1. But the trend of convergence to such a sharp threshold—as predicted by Friedgut's theorem, see Subsection 8.3.5—could be confirmed. For 3, 4, 5 and 6 tasks on 2 processors, the *pseudo-threshold*[22] is at total utilizations of about 1.89, 1.92, 1.95 and 1.965. Since these pseudo-thresholds are coarse[23] ones, it might be more reasonable to recede to the 5% percentile of residual risk of unschedulability. This gives utiliza-

[22]For small values of n, we are not yet in the convergence region. So, we introduce the term pseudo-threshold for a control parameter value (here: total utilization) where the odds for schedulability are fifty-fifty. For $n \to \infty$, the pseudo-threshold u_{PT} tends to the threshold value u_T.

[23]The threshold itself is sharp, but this does not yet become manifest since the n values are too small. For $n \to \infty$, the threshold's sharpness also becomes visible.

tions of about 1.64, 1.72, 1.79 and 1.86, which is in any case still significantly better than applying the guaranteed utilization of 1.5, especially for soft real-time systems where 5% miss rates might be tolerable.

8.7.5 Towards a Partitioned-EDF Threshold Formula

As proposed in [223], the indication of a closed-form solution of $\Pr(u, n, m)$ would be the all-in-one for the problem of thresholds of partitioned EDF. Having such a general formula would make it easy to obtain threshold values by analytical methods like inflection point calculation or the above mentioned pseudo-threshold approach.

In the following, a formula for the special case of $\Pr(u, 3, 2)$ is derived. It is based on the Laplace probability definition and the method of geometrical probabilities. The regions of favorable and possible cases are fully characterized by the respective constraints.

The approach is to calculate the areas of the appropriate shapes in the u_1u_2 plane, see Figure 8.4. The ratio of these areas is the exact value of probability $Pr(u, 3, 2)$, derived using the geometrical method. The first domain condition $u_3 \leq 1$ cuts the lower left triangle from the u_1u_2 square since it becomes $u - u_1 - u_2 \leq 1 \Leftrightarrow u_1 + u_2 \geq u - 1$. Next, the second domain condition $u_3 > 0$ cuts the upper right triangle due to its equivalence to $u_1 + u_2 < u$. Thus, the residual shape representing all task sets meeting the total-utilization condition is a hexagon plotted with a thick line. Finally, the schedulability constraint (8.6) cuts an inner triangle (filled with solid black) from this hexagon. Hence, the ratio we are looking for is the area of the hatched region to the area of the hexagon. This evaluates to (8.7).

$$\begin{aligned}\Pr(u, 3, 2) &= 1 - \frac{\frac{(2(u-1.5))^2}{2}}{1 - \frac{(u-1)^2+(2-u)^2}{2}} \\ &= 3 + \frac{3}{2u^2 - 6u + 3}; \qquad 1.5 \leq u \leq 2\end{aligned} \tag{8.7}$$

Both the lattice-based (for *step* = 0.01) and the analytical result are given in Figure 8.5 for comparison. The two curves lie close to each other confirming the analytical result. As expected, the probabilities are slightly greater than

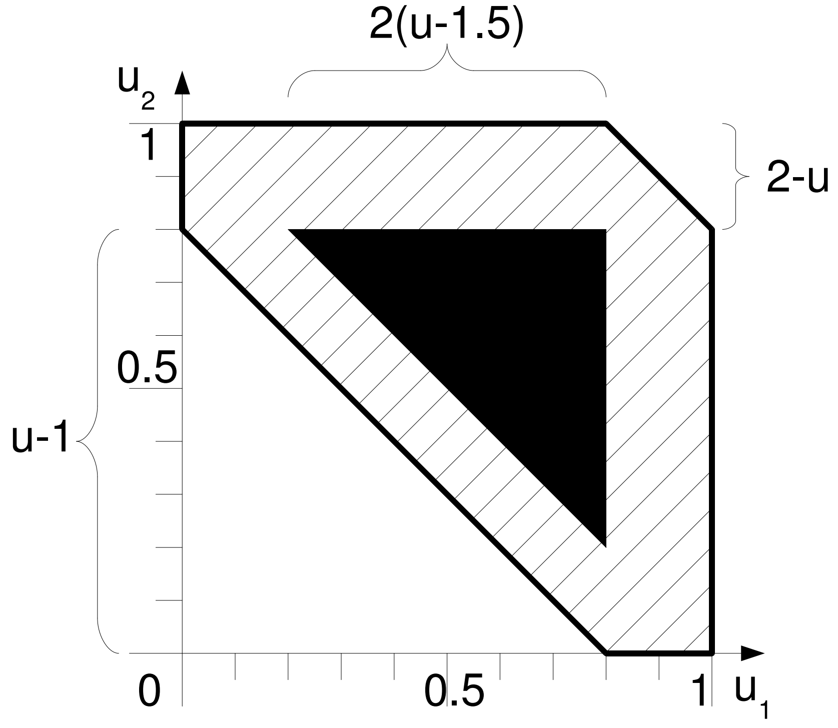

Figure 8.4: Derivation of the Closed-form Formula for $\Pr(u, 3, 2)$ Using the Example of $u = 1.8$

the ones obtained by the Monte-Carlo-FFD method, cf. Subsection 8.7.3, due to the inherent pessimism in the FFD approximation.

Finally, the pseudo-threshold u_{PT} at fifty-fifty odds is found analytically by solving (8.7) equated with 0.5 for u_{PT}. This gives (in the u_{PT} domain $[1.5, 2]$ according to (8.5)) the partitioned-EDF threshold for scheduling 3 tasks on 2 processors (8.8).

$$\Pr(u_{PT}, 3, 2) = 3 + \frac{3}{2u_{PT}^2 - 6u_{PT} + 3} = 0.5 \Leftrightarrow u_{PT} = 1.5 + \frac{\sqrt{15}}{10} \approx 1.887 \quad (8.8)$$

The value obtained in (8.8) is in accordance with the estimated pseudo-threshold value of 1.89 from Subsection 8.7.4[24]. Interestingly, the first summand in (8.8) is a constant and coincides with the lower EDF threshold bound,

[24]Additionally, the 5% percentile of residual risk of unschedulability found to be about 1.64 is confirmed by solving (8.7) equated with 0.95 for u_T. We obtain $u_{5\%} = 1.5 + \sqrt{123}/82 \approx 1.635$.

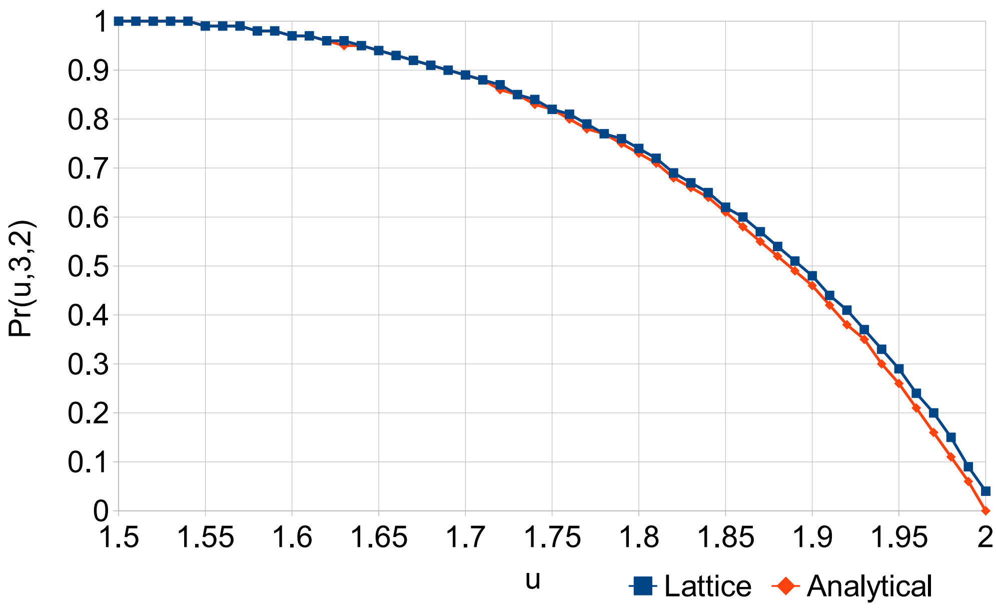

Figure 8.5: Comparison of Utilization-dependent Probability of Schedulability for Partitioned EDF of 3 Tasks on 2 Processors Using a) Numerical Integration Based on a Lattice and b) a Formula (Analytical); In both cases, an exact test using constraints is applied.

cf. Subsection 8.7.1 for this special case of $m = 2$. This term structure suggests that there might exist a more general formula for the pseudo-threshold of partitioned EDF on 2 processors. Using the values for $n = 4$ and $n = 5$ obtained in Subsection 8.7.4, (8.9) is found as a first approximation.

$$u_{PT}(n) \approx 1.5 + \frac{\sqrt{25 - \frac{30}{n}}}{10} \tag{8.9}$$

Note that (8.9) might not be the correct pseudo-threshold formula since its convergence to the threshold would mean a threshold of $u_T = 2$. This is not excluded by Friedgut's theorem, cf. Subsection 8.3.5, since it gives only a sharp threshold *existence* statement. But it seems more reasonable to expect a threshold slightly below 2. On the other hand, $m = 2$ may be a limit case with a threshold 2 equal to the necessary condition. Results for $m = 2$ processors in [125] enable such an interpretation. Even with the estimate from below there, the function series $f_n(u) = \Pr(u, n, 2)$ seems to converge towards a step function with a step at $u = 2$. The question whether there is a qualitative or just a quantitative difference between the case $m = 2$ and the cases $m \geq 3$ cannot be answered here.

8.7.6 A New Partitioned-EDF Test Schema For Hard Real Time

Figure 8.6 presents a newly developed test schema for partitioned EDF scheduling. It uses the threshold method with a threshold and a pseudo-threshold classification elaborated in the previous Subsections. Note that these two classification steps enable for shortcuts circumventing the computationally expensive advanced sufficient test which is constructive and based on the FFD heuristics as discussed in Subsection 8.7.3.

Unfortunately, a detour of the expensive exact test is impossible (unless $\mathsf{P} = \mathsf{NP}$) since the problem of partitioned EDF is NP-Hard, cf. Chapter 4. Nevertheless, a bypass of the constructive sufficient $\mathcal{O}(n \log n)$ test is enabled by a negative threshold or pseudo-threshold classification.

The constant-time complexities $\mathcal{O}(1)$ of the Simple Sufficient Test, the Threshold Classification and the Pseudo-Threshold Classification are based on a reuse of the total utilization u of the task set which has already been calculated (with linear complexity $\mathcal{O}(n)$) when the Necessary Test was performed. This is possible since the Necessary Test is mandatory as the initial test.

8.8 Thresholds for Further Scheduling Problems

There are a few other real-time scheduling problems with a sharp threshold behavior. For some of them, the sharp threshold has been proved while other ones are just conjectured to have a sharp threshold. There are also settings where experimental data suggests that there is no sharp threshold. We will regard examples for each of these three groups in this sequence.

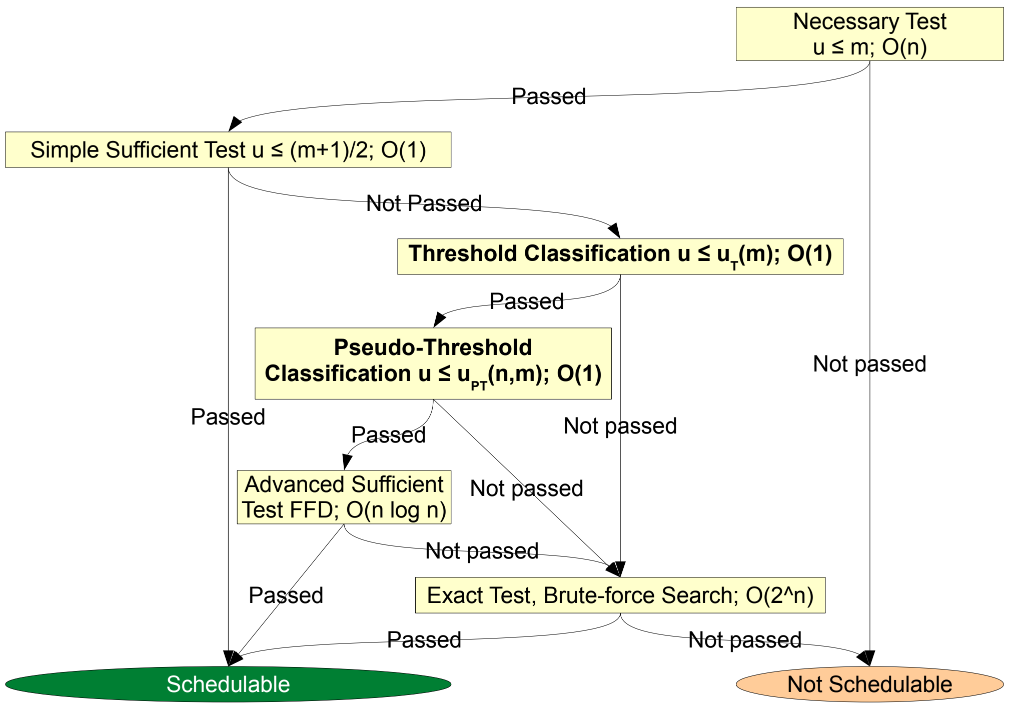
Necessary Test
u ≤ m; O(n)
Passed
Simple Sufficient Test u ≤ (m+1)/2; O(1)
Not Passed
Threshold Classification u ≤ u_T(m); O(1)
Passed
Pseudo-Threshold Classification u ≤ u_PT(n,m); O(1)
Not passed
Passed
Passed
Not passed
Advanced Sufficient Test FFD; O(n log n)
Not passed
Not passed
Passed
Exact Test, Brute-force Search; O(2^n)
Passed
Not passed
Schedulable
Not Schedulable

Figure 8.6: A Partitioned-EDF Test Schema For Hard Real-Time Systems Using Threshold Methods (in **bold** letters) Including Computational Complexities

8.8.1 Non-preemptive uniprocessor scheduling

The property of non-preemptive uniprocessor schedulability of recurring[25] tasks with distance constraints[26] has a sharp total utilization threshold [129]. In their proof, they use the fact that schedulability "[..] of recurring tasks is determined by the existence of a directed cycle in the scheduling graph." [129] Since this graph property is a monotone one, Friedgut's theorem, see Subsection 8.3.5, applies.

8.8.2 Aperiodic Workload and Web Server QoS

The *aperiodic* task model is closely related to the non-cyclic RRT, aka *acyclic* task model, see Subsection 1.3.4. In the acyclic task model, the absolute deadline of each job is the arrival time of the subsequent job, a quite natural assumption [1]. There is an equivalence relationship between acyclic and aperiodic task invocation patterns. The transformation from aperiodic to acyclic task invocations is based on the insertion of zero-execution-time dummy jobs which reset the total synthetic utilization, see below, to zero [1].

By applying the concept of *synthetic utilization* to aperiodic workload as suggested in [1], results on utilization bounds and thresholds could be established. Synthetic utilization[27] is the analogon to classical utilization, see 1.2.1, where the period p_i is replaced by the relative deadline D_i, see (8.10). A second difference is that the summation for obtaining the total synthetic utilization at time t is performed only over the set of *current* jobs at this time t, see (8.10). The current jobs are a subset of all jobs "[..]that have arrived but whose deadlines have not expired[..]" [1], i.e., $r_i \leq t < r_i + D_i = d_i$. The additional

[25]See Subsection 1.3.4 for relationships of the recurring RT model to other task models.

[26]The non-preemptive application chosen is radar (radio detection and ranging) dwell scheduling [129]. The temporal distance between the release times of two subsequent jobs is bounded from above, cf. [129], in order to enable a reliable tracking of targets. Note that this interarrival time is already bounded from below by the sporadic model which the recurring RT model is based on, see Subsection 1.3.4. The rationale for the minimum interarrival time here is the necessary cooling down of the equipment in order to protect it from damage.

[27]The concept of synthetic utilization was further generalized to the concept of *abstract load* in [192]. This allows for the determination of feasibility regions of aperiodic jobs under arbitrary fixed-priority (i.e., independent-of-arrival-time, which is fixed-on-job-level in the context of aperiodic jobs) policies like *Shortest Job First* (SJF) or DMS, cf. Section 2.3.

condition for current jobs is that they "[..]arrive after the beginning of the last schedule gap." [1] A schedule gap means "[..]a period of idle CPU time." [1]

$$u_{synth,i} = \frac{e_i}{D_i} \tag{8.10}$$

$$u_{synth}(t) = \sum_{\text{job } i \text{ is a current job}} u_{synth,i} \tag{8.11}$$

Deadline-monotonic (DM, see Section 2.2) scheduling of such aperiodic tasks is optimal [1] and has a sharp total synthetic utilization threshold [126]. The synthetic utilization bound for a uniprocessor[28] system is $2 - \sqrt{2} \approx 0.586$ as shown in [1]. This result can be used for the implementation of an admission control in online scheduling. But experimental results in [126] suggest that the threshold value of synthetic utilization is close to 0.85, and the 5% percentile of residual risk of unschedulability at about 0.8. Both values are substantially higher than the bound.

This synthetic utilization gain can be exploited to reduce the energy consumption of web servers or to extend the workload capabilities using existing infrastructure without significant loss in temporal guarantees [126].

8.8.3 Global MP Scheduling

In Chapter 6.6 in [125], it is conjectured that global MP scheduling has a sharp total utilization threshold, too. Given that global scheduling is affected by Dhall's effect, cf. Subsection 3.3.1, which can reduce the schedulable system utilization (1.27) down to zero, this seems to be debatable. On the other hand, if applying mechanisms to circumvent Dhall's effect like utilization separation, cf. Subsection 3.3.3, the conjecture might turn out true.

[28]For multiprocessor systems, the synthetic utilization bound is $(3 - \sqrt{7})m \approx 0.354m$ for the DM algorithm [200]. Utilization separation into light and heavy jobs, cf. Subsection 3.3.3, is applied [200].

An even better bound of $\frac{3-\sqrt{5}}{2}m \approx 0.382m$ can be obtained using an SM-US[t] algorithm, cf. Section 2.2 with this threshold value [201]. Note that for the slack-monotonic approach, the bound for the aperiodic (with synthetic utilization) and periodic (with utilization) model are the same (3.10).

This huge discrepancy in the synthetic utilization bound again indicates that MP scheduling is much more complicated than uniprocessor scheduling.

8.8.4 Two-type Heterogeneous MP Scheduling

Recently, the question for existence of a sharp utilization threshold in two-type heterogeneous multiprocessor scheduling was posed in [248]. But experimental data published there suggests that there is *no* sharp threshold behavior.

8.9 Summary

In this Chapter, we could see that the concept of phase transitions and sharp thresholds originating from physics and graph theory can be applied to schedulability problems. Friedgut's theorem is well-suited to prove the existence of sharp thresholds of graph properties. The translation of scheduling problems into random graph problems is often possible.

The determination of concrete threshold values as necessary for practically using this approach is more complicated. Utilization thresholds can fill the gap between necessary and sufficient total utilization conditions. Since sharp thresholds only give statements on the property of schedulability with high probability (w.h.p.) in the limit case of $n \to \infty$ tasks, it turned out useful to switch to pseudo-thresholds for more realistic task set sizes n. They are defined by the piercing point of the probability-utilization curve and the 50% level. This is based on the common-sense threshold of fifty-fifty odds.

For the problem of partitioned EDF scheduling, approximate pseudo-thresholds have been obtained for the special case of task-to-processor assignments on a 2-processor system. Further specialized to the case of assigning $n = 3$ tasks to $m = 2$ processors, a closed-form formula of the probability of schedulability dependent upon total utilization u was derived using geometrical probabilities.

The probabilistic nature of the statements suggests their application only to soft real-time systems. But a new test concept for hard real-time systems was developed. Threshold-based classifications are capable of indicating a likely result and thus, to circumvent complex tests according to the indication given.

These results show that the sharp threshold approach is auspicious in the field of real-time scheduling.

Part IV

Discussion and Outlook

Chapter 9

Discussion

Despite an extensive treatment of partitioned RT multiprocessor scheduling in this thesis, finding the optimal solution of task-to-processor allocation both for partitioned EDF and partitioned RMS remains NP-*Hard in the strong sense* rendering it intractable for larger task and processor sets, cf. Chapter 4. Nevertheless, this work delivers substantial contribution both to an overview of state-of-the-art real-time multiprocessor scheduling techniques (Part II) and an elaboration on sophisticated methods for improving heuristics for partitioned MP scheduling (Part III). The goal of increasing the sensitivity of schedulability tests has been achieved.

9.1 (Meta-)Principles Applied

9.1.1 Partitioned RMS, Accelerated Simply Periodic Task Sets, and Circular Period Similarity

Powerful RMS scheduling approaches both for uni- and multiprocesssor systems need to take *period compatibility* into account. The idea behind this is to bridge the gap between the two extremes of simply periodic task sets with a 100% utilization bound on uniprocessor as the best case and the subsequent period ratios of $\sqrt[n]{2}$ set by the famous Liu/Layland proof [189] as the worst case.

As building blocks of such a bridging, we have considered the *relation to the best case* by exploiting *Accelerated Simply Periodic Task Sets* (ASPTSs), see Subsection 7.3.2, and the use of a *circular similarity measure for periods*, cf. Subsection 7.3.1. The ASPTS-based schedulability tests Sr and DCT overcome the classical approach of utilization bounds and show a very good sensitivity-to-complexity ratio. To the best of the author's knowledge, principles of circular statistics with almost 100 years old origins in atomic research [211] have been applied to real-time scheduling theory for the first time. This lead to a significant simplification of Burchard's sufficient schedulability test undern an increase in its sensitivity.

These two basic ideas entail several modifications to the allocation of RT tasks to processors under partitioned RMS, see Figure 7.5.

9.1.2 Partitioned EDF and Sharp Thresholds

For several scheduling disciplines, *sharp thresholds* indicating phase transitions can be observed. The typical control parameter in scheduling is the total utilization u of a task set. For task sets with a small number of tasks, we can use pseudo-thresholds whose sequence tends towards the threshold for $n \to \infty$, cf. Subsection 8.7.4.

(Pseudo-)thresholds are convenient to approximate complex schedulability by a simple comparison of the total utilization u with the (pseudo-)threshold utilization u_T or rather u_{PT}. In conventional theory, there is a not-so-narrow utilization interval from the sufficient level to the necessary level. Up to the sufficient level of total utilization, all task sets are schedulable, and beyond the necessary total utilization level, no task set is schedulable. This "gray zone" represents all task sets which might be schedulable or not and collapses ideally to a point and practically to a much narrower interval called threshold interval, cf. Subsection 8.7.1 and Figure 1.15.

While for soft real-time systems, a direct application of thresholds and pseudo-thresholds is possible, hard real-time systems can profit from the approach by using sophisticated test schemes as shown for partitioned EDF in Figure 8.6.

Utilization thresholds can be seen as an extension or generalization of utilization bounds. Hence, the threshold approach is innovative from a conceptual point of view and can contribute to a better understanding of an approximation of the predicate *schedulability*.

9.2 Evaluation of New Approaches by Synthetic Task Sets

For a fair evaluation and comparison of scheduling algorithms and schedulability tests, an appropriate generation of synthetic task sets is fundamental. Since the total utilization u is the most important control parameter, task set ensembles of constant total utilization need to be created. Typically, a uniform distribution of the individual utilizations is assumed. Algorithms and programs solving this problem have been presented in Chapter 6 and then applied in Section 7.6.

The probability distribution of the periods in the task set ensemble has also some impact on RMS-schedulability. This problem has been discussed in Section 6.5.

9.3 Applicability

Sometimes people come up with the preconception that this is just another ramification of real-time scheduling theory. In the following, I would like to refute them.

ASPTSs and the RMS schedulability tests Sr and DCT based on it are ingeniously simple and—at the same time—powerful tools for the determination of schedulability under RMS. The exactness of DCT for task set sizes of 2 makes it a handy instrument for testing schedulability by heart. Thus, Sr and/or DCT shall be included in the curricula of real-time scheduling courses at university level.

Circular statistics increases the success ratio of task allocation to a given number of processors. Hence, evaluation becomes less pessimistic compared to quick sufficient tests. The gap to often intractable brute-force tests is narrowed. Here, the everyday example of roundabouts, see 7.3.1, can contribute to a better understanding of the concept.

Sharp threshold behavior of schedulability functions depending upon the total utilization can be exploited to obtain probability statements on the (non-)schedulability of task sets. For soft real-time systems, this is a viable approach. For hard RT systems, a proposed test schema can contribute to speed-up schedulability tests.

Sufficient schedulability tests may be useful for quick estimates of task set schedulability in automatic system-synthesis tools. There, the system designer expects an interactive behavior with both a low average and worst-case timing. So, fast approximate schedulability tests are still being relevant in spite of the ever-increasing computing power of computers which renders an exact offline schedulability test also of larger task sets more and more viable.

9.4 Societal Relevance

Real-time scheduling is a fundamental approach applied in many embedded systems. Embedded systems range from tiny cardiac pacemakers over car electronics up to airplanes and spacecrafts. So, they are part of our everyday lives. By a better packing of real-time tasks onto processors as suggested in this thesis, the system utilization can be increased towards its optimum value.

This helps to safe resources both in production and operation of embedded systems since more tasks can be accomplished with the same amount of hardware, or—vice versa—less hardware is required for a particular application. Note that energy can be saved during operation of a slim system which uses the available hardware in a more efficient way.

Such a reduction in resource consumption can help to bound the ongoing climate change and its negative effects like floodings and drought or the global sea-level rise.

Hence, sophisticated scheduling approaches can contribute to a better future of the humankind on the planet Earth.

Chapter 10

Outlook

Very naturally, my thesis could not cover all aspects of partitioned RT scheduling. On the other hand, Parts I and II give a comprehensive introduction to and an overview on real-time scheduling on uni- and multiprocessors.

In Part III, the principles of *Accelerated Simply Periodic Task Sets* (ASPTSs), *circular period similarity* and *sharp thresholds* including first applications to the development of more sensitive schedulability tests have been presented. I think that these three principles are powerful and bear the potential to obtain further and refined results.

As given in Subsection 1.9.1, the thesis concentrates on the Liu/Layland task model. But there is a great deal of variants and generalizations as shown in Sections 1.2 and 1.3, especially in Subsection 1.3.4. Some of them are practice-oriented and, thus, not only of academic concern. Hence, it is a natural task for the future to do research on the applicability of the three above mentioned principles. I would like to counter people objecting that this should have been done from the beginning of this research by stating that research has always to be done in a careful step-by-step manner. A rush always bears the danger of losing sight of important facts and interrelationships in it.

Multicore technologies are entering the embedded world due to better computational capacities and less cabling and energy consumption. But the closer integration of proccessing units in a multicore system with some levels in the memory hierarchy shared among cores makes the determination of timing properties even more complicated than for multiprocessor systems, cf. [8].

Another factor not examined in this thesis is energy consumption which becomes more and more important since the pollution-caused climate change shall be bounded. Second, energy saving extends the operating times of battery-driven mobile devices and, to a lower degree, the range of electric vehicles.

I hope that this thesis can contribute to a better understanding of multiprocessor real-time scheduling with a focus on partitioned scheduling and schedulability tests. This happens in full acknowledgment of the deep truth that we are like dwarfs standing "[..]on the shoulders of giants." Bernard of Chartres formulated this in the 12th century, but it was written down by John of Salisbury in his *Metalogicon* [266] and later on famously uttered by Isaac Newton.

Appendix A

A Brief Introduction to Circular Statistics

A.1 Motivation

Time is often modeled as a linear and dense phenomenon which can be described by a time bar with real numbers. In order to make time easier to access for human beings and their everyday experiences, time units like day, week, month and year have been introduced. In most cases, they are closely related to astronomic events of the two easiest-to-access celestial objects Sun and Moon. So, the time flow can be described by a *periodic* repetition of values, e.g., time of day and day of the week[1]. Such a single period then constitutes a time unit.

This periodic description leads to *discontinuities* at the start of each new period. The days of the week from Monday to Sunday, see Figure A.1, can be coded by numbers $1, 2, \ldots, 7$. When Monday starts, the numerical value of the day of the week jumps from 7 to 1 which does not appropriately reflect the neighboring of Sunday and Monday in time. Hence, for any statistical analysis based on the days of the week, a circular data arrangement by wrapping a

[1]A month is based on the duration of a Moon revolution about the Earth. Since there is no integer ratio to the duration of the Earth's revolution about the Sun, months of different lengths from 28 to 31 days have been defined in order to achieve an integer number of 12 months.

week to a circle is to prefer over a conventional linear data arrangement. For the application of circular statistics to analyze time patterns, see, e.g., [59].

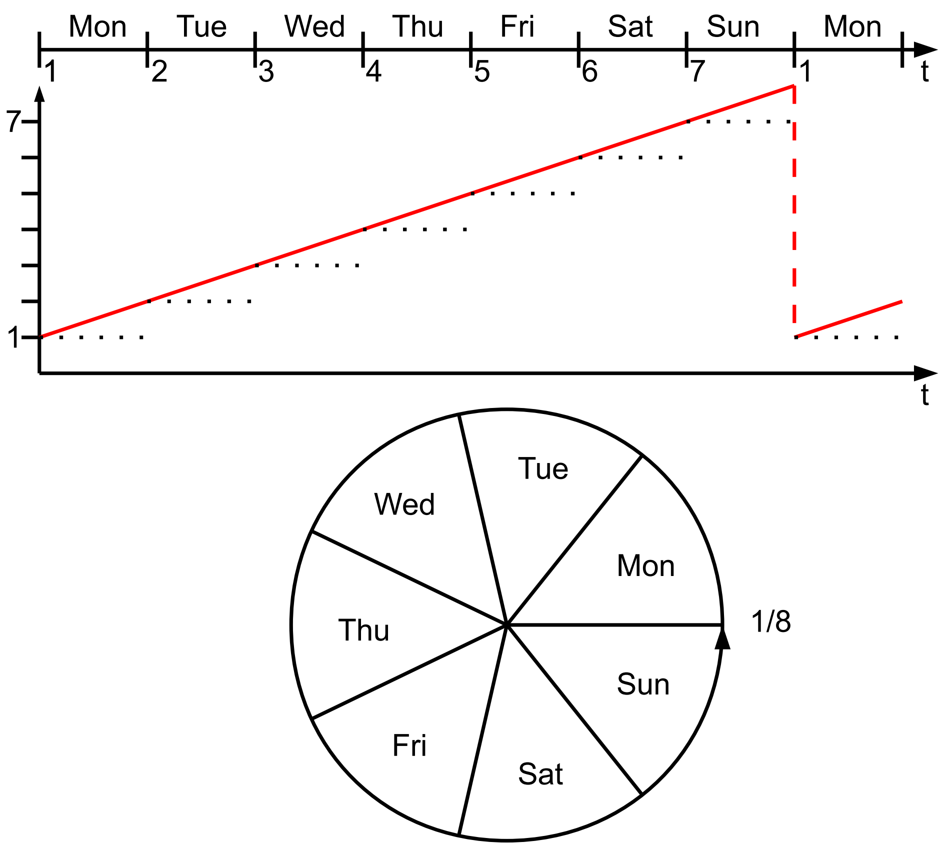

Figure A.1: The Days of the Week in a Linear (Upper Part) and a Circular (Lower Part) Arrangement; Note that the integer part of time t (dotted) actually gives the ordinal number of the day of the week. Further, the zero direction is chosen as East, and the sense of rotation as counter-clockwise what corresponds to mathematical convention, see Subsection A.1.1.

This example clearly shows the necessity of circular statistics for the appropriate statistical description of data based on a ring structure. A typical application area is temporal data like time of day or day of the week. Further fields of application are directional geographical data, see Subsection A.1.2, or the fractional parts of atomic masses which were first investigated by von Mises [211]. Note that this last example served as a starting point for the application of circular statistics to S values in this thesis, see Chapter 7.

Circular statistics has also applications in cutting-edge technologies like Global Navigation Satellite Systems (GNSSs). For high-precision (order of magnitude of centimeters) navigation and geodesy, carrier phase measurements are evaluated. Conventional approaches incorrectly assume a *Gaussian normal distribution* of phase data. But phases are directional data and follow a *von Mises normal distribution* [211]. For more details on this interesting application of circular statistics, see [67]. Even circular filters and circular fusion operators can be constructed in order to track phase data of GNSS signals [263].

A third application of circular statistics is image processing. Edges in color images can be detected using circular statistics of hue[2] values [229]. As for the GNSS application, circular filters are proposed in [229].

A.1.1 Direction of Vectors

Circular statistics investigates *unit vector* data instead of scalar data. A vector is fully characterized by its magnitude and its direction. Since by normalization to a unit vector, the magnitude is abstracted away, only the direction remains. This direction can be characterized by a unit vector in $\mathbb{R}^n$ with $n-1$ degrees of freedom. Due to the impact of the directions, the term *directional statistics* is the general one.

On a straight line ($\mathbb{R}^1$), there is no freedom to choose a direction[3].

On a plane ($\mathbb{R}^2$), there is one degree of freedom to determine the direction. For two-dimensional directions, directional statistics is refined to *circular* statistics. The data point specifying a two-dimensional direction is then given by a point on the circumference of a unit circle or by an angle. Note that this angle needs to chosen with respect to a fixed *zero direction* and a *sense of rotation*. The zero direction specifies the starting point, and the sense of rotation has the two options clockwise and counter-clockwise. Invariance of measures in terms of the choice of these two references is considered to be a quality criterion. Circular directions can be obtained by wrapping linear data.

[2]For an introduction to hue-based color spaces, see, e.g., [215], p. 89f.

[3]At first glance, there are two directions possible. But this kind of variation is already covered by the choice of the sign of the scalar value.

In space ($\mathbb{R}^3$), the directional two degrees of freedom are usually specified by two angles (e.g., longitude[4] and latitude as geographical coordinates). An alternative is the description of a spherical direction by a point on a unit sphere. Such spherical data can be analyzed by *spherical* statistics. Note that the simple approach of wrapping does not work for three-dimensional directional data, cf. [83], p. 6.

For any choice of zero direction and sense of rotation, the beginning coincides with the end, i.e., $359° + 1° \equiv 0°$. The values are taken modulo 360°. This means that $\forall k \in \mathbb{Z} : \theta \equiv \theta + k \cdot 360°$ holds. Thus, special care has to be taken for the measurement of distances between two data points as we will see in Subsection A.4.1. Since there is no natural ordering for circular data, rank-based methods known from linear statistics like *Spearman's rank correlation coefficient* [257] are not applicable.

A.1.2 Case Study: Wind Direction

Wind is an easy-to-access everyday phenomenon which can be described by a time-varying vector field on the earth surface. The magnitude of the wind vector is its velocity, the wind direction [111] is usually reported by the direction from which it originates. As explained in Subsection A.1.1, wind velocity is abstracted away in our model. Further, we assume the four cardinal directions North, East, South and West with a subsequent right angle difference[5]. An example of a wind direction distribution is given in Figure A.2.

From this example, we see that we need to agree on zero direction and sense of rotation as explained in Subsection A.1.1 in order to allow for a quantitative analysis. Geologists usually use North as zero and a clockwise orientation. But mathematicians define East as zero and prefer a counter-clockwise orientation. Hence, a data exchange between geologists and mathematicians might be ambiguous. Only the transmission of invariant values not depending upon zero and orientation choice comes without such a danger of misinterpretation and is, thus, preferable.

[4]The selection of a zero meridian was disputed. Finally, the meridian passing through Greenwich in London was selected in 1884.

[5]This is problematic at least directly on the poles since on the North pole only South winds are possible, and on the South pole, there are only North winds.

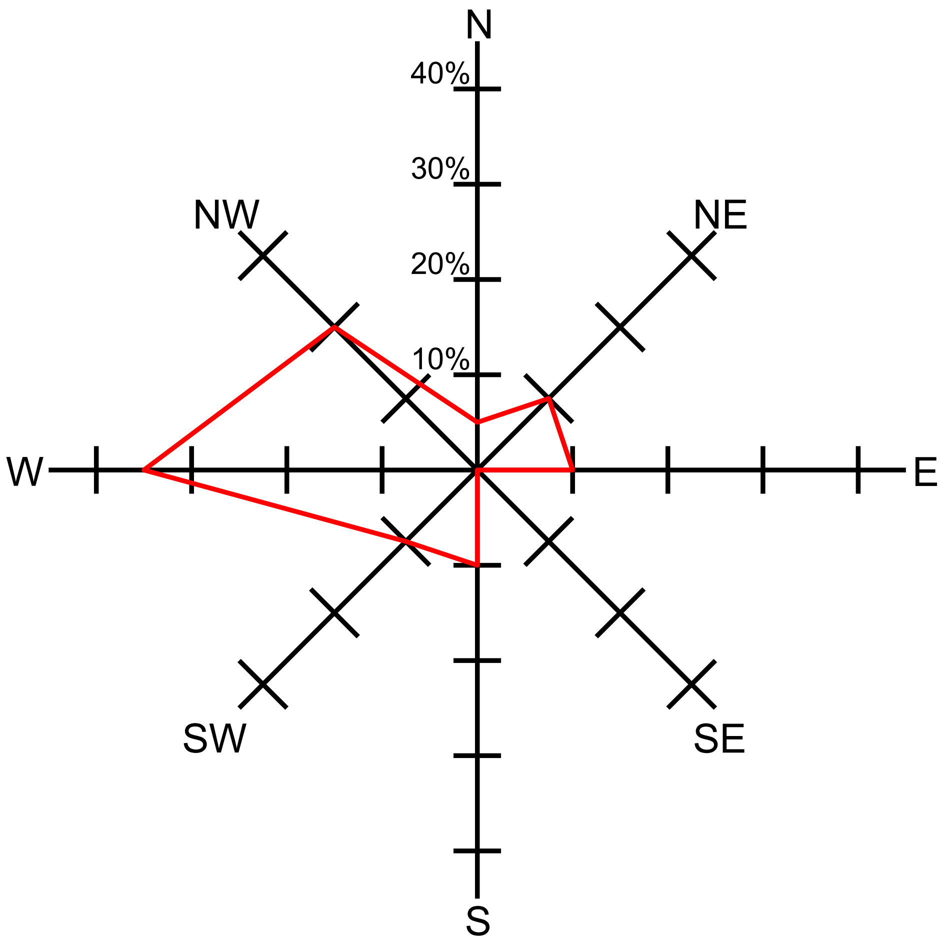

Figure A.2: Example of a Wind Direction Distribution Where Conventional Statistics is no More Appropriate and Should Be Replaced by Circular Statistics

A.2 First Order Moment: Mean Direction

A naïve method of estimating an average or expected direction might be summing up all angle values and scaling this sum by $1/n$ as known from linear statistics. But, due to the circular data space structure, such a simple approach has major deficiencies as shown in the following.

Suppose two wind directions deviating from East by 10° to the left and 20° to the right. Mathematically coded, this is 10° and 340° which averages to 175°, a a direction very close to West. This is remote from being consistent with our common-sense expectation to get an average direction close to East. Geologically coded, the two directions correspond to 80° and 110° which gives a mean value of 95°. This value really is close to East. The example shows that averaging the angles as commonly used for linear data is not appropriate for circular data. The result depends upon the choice of zero direction and sense of rotation, a fatal situation.

Since the directions are unit vectors, an appropriate measure of the mean direction is the direction of the resultant vector, see Figure A.3. It can be calculated via a decomposition into x and y components. Then, these components are summed up. The mean angle value is finally obtained by a quadrant-correct two-argument *arc tangent* function[6]. For a data set $\alpha_1, \alpha_2, \ldots, \alpha_n$ of directions, the mean direction $\overline{\alpha}$ can, thus, be calculated as (A.1). Since we are only interested in the direction of the resultant vector, a normalization to magnitude one under maintenance of the direction is not necessary.

$$\overline{\alpha} = \operatorname{atan2}\left(\sum_{i=1}^{n} \sin \alpha_i, \sum_{i=1}^{n} \cos \alpha_i\right) \bmod 2\pi \tag{A.1}$$

Note that the representation of directions by *unit* vectors ensures that the directions are equally weighted. Further, the periodic nature of angles is correctly depicted by the trigonometric functions used.

If both horizontal and vertical vector components sum up to zero, the resultant vector degenerates to the zero vector for which no meaningful direction

[6]The `atan2` function belongs, e.g., to the `math.h` standard library in the *C* language. There, the result is given in $[-\pi, \pi]$ which makes a modulo operation in (A.1) necessary in order to obtain values in $[0, 2\pi)$.

can be determined[7]. The meaninglessness is well reflected by raising an error in this case. This is another difference to the linear average where a reasonable mean value always exists[8].

The mean direction (A.1) is *rotationally equivariant,* a shift of each data point by an additive constant results in the same shift of the mean direction. For a proof, see [143], p. 14.

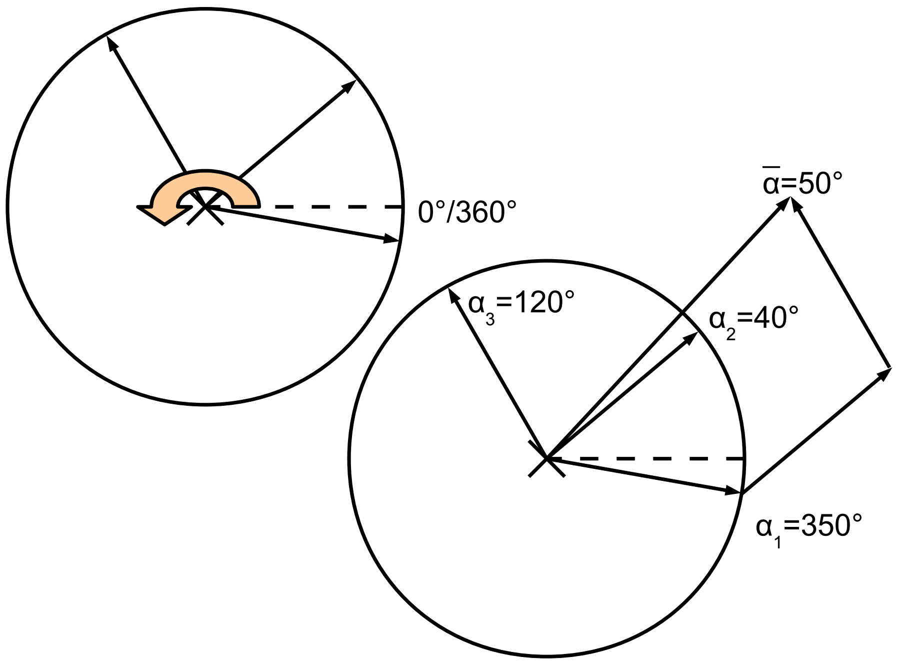

Figure A.3: Mean Direction $\overline{\alpha} = 50°$ of Directions $\alpha_1 = 350°$, $\alpha_1 = 40°$ and $\alpha_1 = 120°$ as the Direction of the Vector Resultant Found by Vector Addition

[7]For n observations, this occurs, e.g., if the n directions correspond to the n-th roots of unity defined by solutions of the equation $z^n = 1, z \in \mathbb{C}$. We call a data set with such equi-spaced distributed directions from here on a *roots-of-unity configuration.*

[8]If allowing for infinite numbers ($+\infty$ and $-\infty$), an indeterminate form like $\infty - \infty$ can be obtained. But usually, infinte numbers are not permitted.

A.3 Second Order Central Moment: Circular Dispersion, Variance and Standard Deviation

The two most popular measures of spread in linear statistics are *variance* and *standard deviation*. Variance is the average squared deviation from the mean. Using the mean as a reference makes it a *central* moment. Standard deviation is the dimension-corrected variance, i.e., the square root of the variance in order to compensate for the squaring performed before summation.

Unfortunately, this approach does not work for angular data sets. The simple reason is that, even if the angular mean is taken correctly according to (A.1), the deviation being a distance cannot be computed in the linear manner as an absolute difference. Circular distance is discussed in Subsection A.4.1.

In Section A.2, the magnitude of the resultant vector was disregarded. According to the Theorem of Pythagoras and (A.1), the magnitude is the square root of the squared sum of the sines and the squared sum of the cosines of the angles in the data set. The greater this magnitude the more concentrated is the data set around the mean direction. In the extreme case of all directions α_i being equal, the magnitude is n.

On the other hand, if both sum of sines and sum of cosines are zero, the resultant vector is the null vector with magnitude 0. A maximally heterogeneous data set with a uniform distribution around the circle[9], cf. the discussion in Section A.2 on this case, results always in such a zero resultant vector[10].

These considerations of the two limit cases suggests to define the *circular central dispersion* $D_{\overline{\alpha}}$ as the difference of the sample size n and the magnitude of the resultant vector, see (A.2). For a derivation of this formula, see [143], p. 16ff.

$$D_{\overline{\alpha}} := n - \sqrt{\left(\sum_{i=1}^{n} \sin \alpha_i\right)^2 + \left(\sum_{i=1}^{n} \cos \alpha_i\right)^2} \tag{A.2}$$

[9]From another point of view, we can identify this situation as maximally *homogeneous* since the directions follow a uniform distribution.

[10]Note that we cannot reason in the other direction. A zero resultant vector does not imply such a roots-of-unity configuration, cf. [45].

Note that the circular central dispersion varies between 0 and n, and, thus, depends on the sample size which might be undesirable. By normalization, a scaling with $1/n$, we obtain the *circular variance* $V_{\overline{\alpha}}$ with a range of $[0, 1]$, cf. [45].

$$V_{\overline{\alpha}} := \frac{D_{\overline{\alpha}}}{n} \tag{A.3}$$

It is remarkable that circular variance is bounded both from below and above. In linear statistics, variance ranges from 0 to ∞. Even the *coefficient of variation* known from linear statistics, the standard deviation normalized with respect to the mean value, is not bounded from above. But the *circular standard deviation* s can be defined as (A.4) with values in $[0, \infty)$ as we know it from linear statistics. In the case of a wrapped normal distribution, this s value can serve as an estimator for the linear standard deviation of the underlying Gaussian normal distribution [111].

$$s := \sqrt{-2\ln(1 - V_{\overline{\alpha}})} \tag{A.4}$$

The closely related measure *angular deviation* (A.5) (denoted s_{ad}) tries to simulate the linear approach of average squared deviation applied to angles. The square root operation in (A.5) indicates a dimension correction as we know it from the linear standard deviation. Indeed, formula (A.5) is regarded as the favored circular standard deviation formula for concentrated unimodal distributions (with a small variance) [111]. Its range is $[0, \sqrt{2}]$. As shown in [143], p. 21, $s \approx \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\alpha_i - \overline{\alpha})^2}$ holds using the second-order *Taylor expansion* $\cos\theta \approx 1 - \theta^2/2$ for the approximation. A problem of this approach is that the parabolic approximation of $\cos\theta$ is adequate[11] only for small θ values, i.e., $|\theta| \leq \pi/2$. But circular distances to the mean value can occur up to π, cf. Subsection A.4.1. The angular deviation (A.5) is an approximation[12] of the circular standard deviation (A.4) with an error of less than 5% for $V_{\overline{\alpha}} < 0.18$. For more details, see [45], [143], p. 21, and [111].

$$s_{ad} := \sqrt{2V_{\overline{\alpha}}} \tag{A.5}$$

[11]Note that $\cos\theta$ has an inflection point at $\theta = \pi/2$.

[12]Formula (A.5) can also be obtained by a first-order *Taylor expansion* of the *radicand* $-2\ln(1 - V_{\overline{\alpha}})$ in (A.4) about the point $V_{\overline{\alpha}} = 0$.

Remarkably, all here presented measures are *rotationally invariant* which is considered a natural requirement for reasonable circular measures of spread. This property corresponds to shift invariance in linear statistics.

A.4 Circular Distance and Circular Range

Besides variance, range is another popular measure of spread. It only takes minimum and maximum of the data into account which keeps its robustness against outliers at a low level[13]. The distance between maximum and minimum value is calculated.

A.4.1 Circular Distance

For linear data sets, the distance between two data points is the absolute value of their difference. This approach does not work analogously for circular data since the distance could be taken in clockwise or in counter-clockwise direction. Both distance angles add up to 2π or 360°, a perigon. The *circular distance* between two points is now defined as "[..]the smaller of the two arclengths between the points along the circumference[..]" [143], p. 15, mathematically given as (A.6).

$$d(\alpha_1, \alpha_2) := \min(\alpha_1 - \alpha_2, 2\pi - (\alpha_1 - \alpha_2)) \tag{A.6}$$

This can be expressed as (A.7) where the minimum is simulated by absolute values.

$$d(\alpha_1, \alpha_2) = \pi - |\pi - |\alpha_1 - \alpha_2|| \tag{A.7}$$

A second possibility is to consider the length of the scalar projection of one vector onto the other as a similarity measure. This is mathematically the scalar product of the two vectors which is commutative. Since both vectors are unit vectors, their scalar product is just the cosine of the angle between them. Hence, we can also reasonably define (A.8) since the distance is complementary to the similarity, and the minimum reasonable distance is zero.

[13]Note that the finite domain $[0, 2\pi)$ of circular data might reduce the impact of outliers. On the other hand, the detection of outliers in circular statistics is a relevant problem, cf. chapter 10 in [143].

$$d_{\text{proj}}(\alpha_1, \alpha_2) := 1 - \cos(\alpha_1 - \alpha_2) \tag{A.8}$$

As a third variant, we can take the length of the chord belonging to the central angle spanned by the two vectors as a measure of their distance. Using the *law of cosines* and taking into account to we have unit vectors, (A.9) is easily obtained. Note that this distance measure corresponds to the Euclidean distance between the two data points on the unit circle as suggested in [143], p. 275, and [139].

$$d_{\text{chord}}(\alpha_1, \alpha_2) := \sqrt{2\,(1 - \cos(\alpha_1 - \alpha_2))} \tag{A.9}$$

For the reader's convenience, two three hitherto presented circular distance measures and their geometric origin are compared in Figure A.4.

Finally, an approximation of (A.8) by a second-order *Taylor expansion* is possible. This gives (A.10).

$$d_{\text{approx}}(\alpha_1, \alpha_2) := \frac{(\alpha_1 - \alpha_2)^2}{2} \tag{A.10}$$

A summary of all four given distance measures can be found in Figure A.5. It is clearly visible that both scalar projection (A.8) and chord length (A.9) are good approximations of the arclength approach (A.6). But (A.10) is only appropriate in the interval $[-\pi/2, \pi/2]$. This makes (A.5) arguable since it is based on this approximation.

A.4.2 Circular Range

Linear range is just the difference of maximum and minimum value (A.11). So, in a non-decreasing sorted list of data, it is the last data value minus the first one. Note that this sorting with $\mathcal{O}(n \log n)$ complexity works, but is not required. A linear search of $\mathcal{O}(n)$ for minimum and maximum value is appropriate.

$$R_{lin} := \max_{1 \le i \le n} \alpha_i - \min_{1 \le i \le n} \alpha_i \tag{A.11}$$

To copy this approach to the circular data case requires some considerations. From Section A.1, we know that there is no natural ordering in circular data sets. So, the determination of a minimum and a maximum angle might be

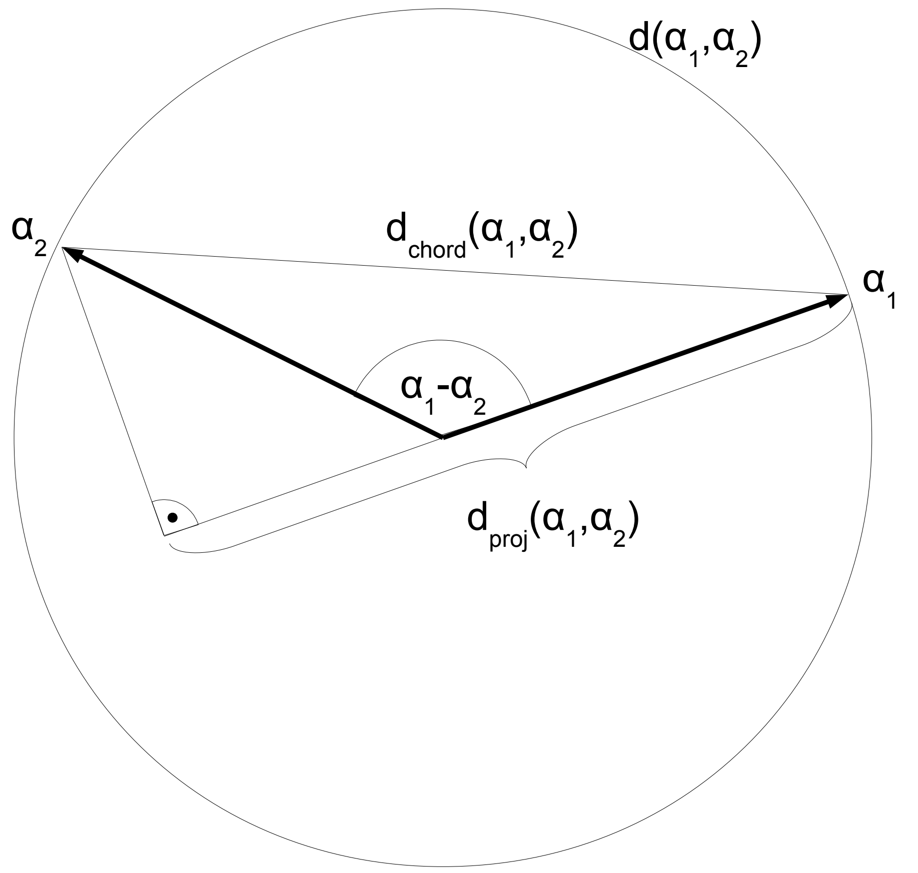

Figure A.4: Circular Distance Between Two Angles α_1 and α_2 Depending on Their Difference $\alpha_1 - \alpha_2$ According to Shortest Arclength (A.6), Scalar Projection (A.8) and Length of Chord (A.9)

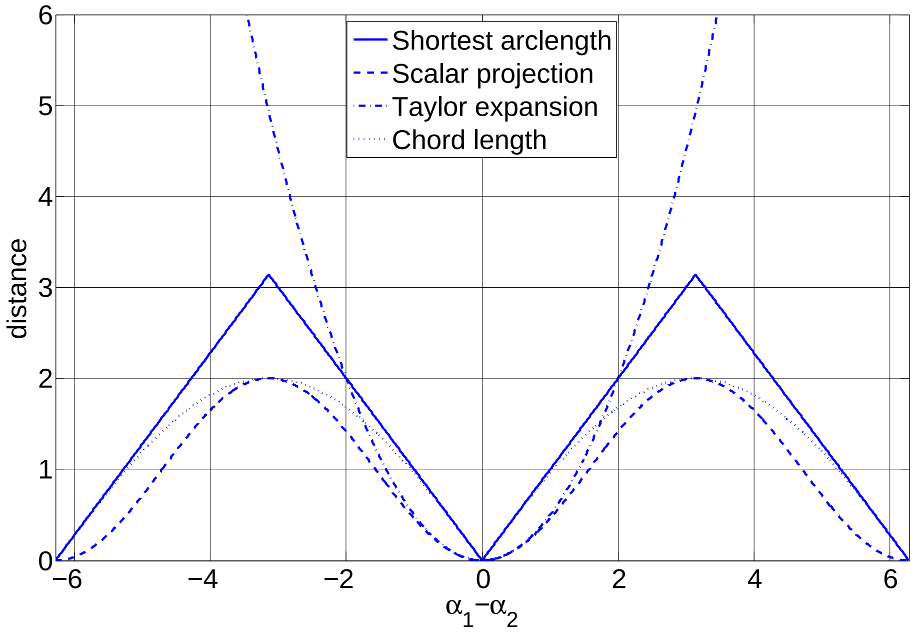

Figure A.5: Circular Distance Between Two Angles α_1 and α_2 Depending on Their Difference $\alpha_1 - \alpha_2$ According to Shortest Arclength (A.6) (solid), Scalar Projection (A.8) (dashed), Taylor Expansion (A.10) (dash-dot), and Chord Length (A.9) (dotted)

difficult. One could take the maximum and the minimum angle value. But then, the resulting range is dependent upon the choice of zero direction which is considered a bad solution, cf. Subsection A.1.1.

So, what is a reasonable definition of a circular range? The most meaningful placement of the zero direction is in the largest gap between two neighboring data points. Then, the so obtained range reaches its minimum. We can define circular range as "[..]the length of the shortest (undirected) arc which includes all observations." [217]

In order to the value of the circular range, we first need to sort all angular data in non-decreasing order. Then, the gaps between neighboring directions are given by $\alpha_i - \alpha_{i-1}$. For the first gap between the last and the first value in the list, $\alpha_0 := \alpha_n - 2\pi$ shall be set. It is easy to see that then all gaps are non-negative and sum up to a full circle of 360°, as expected. So, we can calculate the circular range as the arclength of the full circle minus the largest gap, see (A.12).

$$R_{circ} := 2\pi - \max_{1 \leq i \leq n} (\alpha_i - \alpha_{i-i}) \tag{A.12}$$

This is "[..]the shortest arc on the circumference which contains all the n observations[..]" [143], p. 162. Due to the bridging of the origin direction by $\alpha_0 := \alpha_n - 2\pi$, the so-defined circular range has the beneficial property of independence of zero direction choice, cf. Subsection A.1.1. Both linear and circular range are independent of the choice of sense of rotation which can be easily shown by the transformation $\alpha' = 360° - \alpha$.

The domain of the circular range is $[0, R_{circ,max} = 2\pi(1 - 1/n)]$ since the maximum gap is bounded from below by $2\pi/n$ which corresponds to the extreme case of uniformly distributed directions. Any tilting of any direction yields a greater maximum gap. In other words, a set of n directions can cover at most $1 - 1/n$ of a full circle which converges to a full circle for $n \to \infty$.

With the linear range approach, the first and the last direction are grouped around the zero direction which has to be chosen first. The skipped gap between last and first data value varies for one and the same data set depending upon the choice of zero direction. On the other hand, circular range chooses always the zero direction in a way maximizing this gap which in turn minimizes the arclength covering all data points. Hence, circular range corresponds to a special choice in the family of linear ranges. So, the circular range is a tight lower bound of the linear range of planar directional data, see (A.13).

$$R_{lin} \geq R_{circ} \tag{A.13}$$

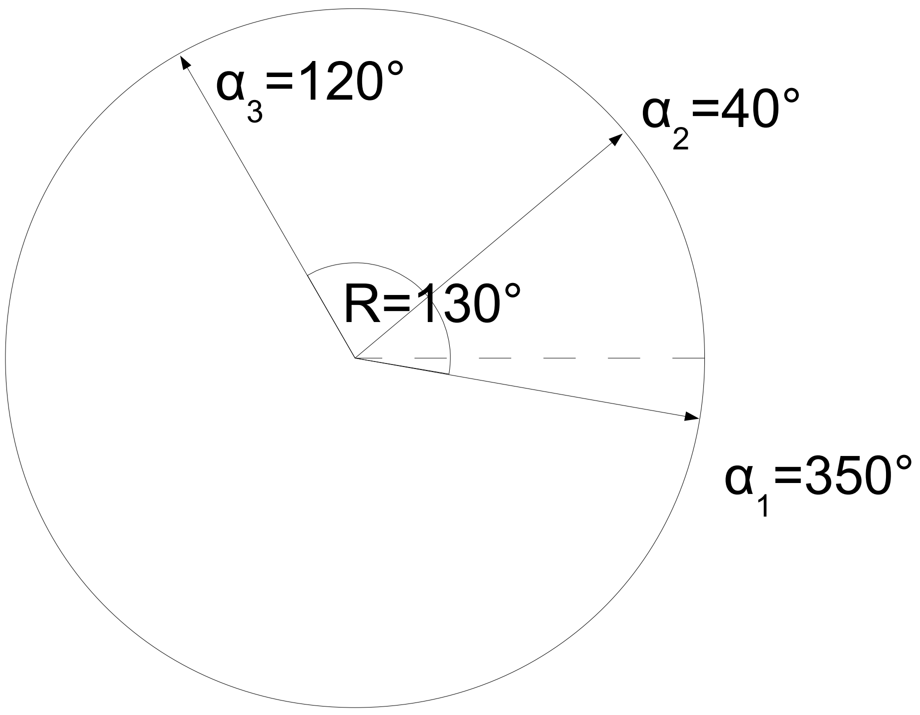

Figure A.6: Example of the Circular Range of Three Vectors With Gaps of 50°, 80° and 230° Giving $R_{circ} = 360° - 230° = 130°$; The linear range for this choice of zero direction (dotted) is $R_{lin} = \alpha_1 - \alpha_2 = 310°$ and the normalized circular range is $\hat{R}_{circ} = 130°/240° \approx 0.54$. Both linear and circular range are independent of the choice of sense of rotation whose specification is, thus, omitted here.

Summing up, circular range is rotationally invariant in contrast to linear range applied to a circular data set. This property makes circular range superior to linear range as rotational invariance is considered a basic requirement for all reasonable non-location circular statistical measures. Circular range is

"[..]the correct analogue of the linear range to the circular situation[..]" [247], p. 80.[14]

Normalized Circular Range

Since the so-defined circular range R_{circ} is still dependent upon the sample size, a normalization to the normalized circular range $\hat{R}_{circ}$ is suggested additionally (A.14).

$$\hat{R}_{circ} := \frac{R_{circ}}{R_{circ,max}} = \frac{R_{circ}}{2\pi\left(1 - \frac{1}{n}\right)} \tag{A.14}$$

A low (normalized) circular range indicates clustering. In the extreme case of $R = \hat{R}_{circ} = 0$, all directions α_i are equal. The other extreme of $\hat{R}_{circ} = 1$ corresponds to $R_{circ} = 2\pi(1 - 1/n)$ and to a roots-of-unity configuration. Compared to the circular variance $V_{\overline{\alpha}}$, cf. Section A.3, the normalized circular range $\hat{R}_{circ}$ has the advantage that there is an *equivalence* between the $\hat{R}_{circ} = 1$ condition and a roots-of-unity configuration.

An illustrative example of the circular range of three directions is given in Figure A.6.

A.5 Summary

Circular statistics can be regarded as directional statistics in a plane. It differs in many respects from classical linear statistics as it studies unit *vectors* instead of scalars with applications in, e.g., geography, physics, medicine, GNSS-based navigation, and color image processing.

Some basic measures of location (mean direction) and dispersion (circular central dispersion, circular variance, circular range) have been presented and discussed. The fundamental account of an appropriate circular distance measure has been highlighted. All these basic results belong to univariate analysis in descriptive statistics. For more advanced topics in circular statistics like probability distributions, tests, regression and correlation, the interested

[14]Note that the first author of [143] S. Rao Jammalamadaka is the same person as Jammalamadaka S. Rao, the author of [247]. This can well be interpreted as an attempt to illustrate circularity even in his name.

reader is referred to, e.g., [143]. For an understanding of the statistical tools applied in Chapter 7, this brief introduction is ample enough.

The author hopes that circular—or more general, directional—statistics will be applied in further areas where it is appropriate. As shown in Chapter 7 for the first application of circular statistics in real-time scheduling, such a seemingly small change can have surprising and significant effects, giving real improvements. Often it is just the force of habit which prevents us from applying directional statistics in situations where it is appropriate.

Bibliography

[1] Tarek F. Abdelzaher, Vivek Sharma, and Chenyang Lu. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Trans. Comput.*, 53(3):334–350, March 2004.

[2] Dimitris Achlioptas, Assaf Naor, and Yuval Peres. Rigorous location of phase transitions in hard optimization problems. *Nature*, 435(7043):759–764, 2005.

[3] Karsten Albers and Frank Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, ECRTS '04, pages 187–195, Washington, DC, USA, 2004. IEEE Computer Society.

[4] Karsten Albers and Frank Slomka. Efficient feasibility analysis for real-time systems with EDF scheduling. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '05, pages 492–497, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Tarek A. AlEnawy and Hakan Aydin. Energy-aware task allocation for rate monotonic scheduling. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 213–223, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Noga Alon and Asaf Shapira. Every monotone graph property is testable. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 128–137, New York, NY, USA, 2005. ACM.

[7] Rajeev Alur and David Dill. The theory of timed automata. In J. de Bakker, C. Huizing, W. de Roever, and G. Rozenberg, edi-

tors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer Berlin / Heidelberg, 1992. 10.1007/BFb0031987.

[8] James Anderson and Nathan Fisher. Guest editorial. *Real-Time Systems*, 48(6):635–637, November 2012. 10.1007/s11241-012-9163-z.

[9] James H. Anderson, Vasile Bud, and UmaMaheswari C. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Björn Andersson. *Static-priority scheduling on multiprocessors.* PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 2003.

[11] Björn Andersson. Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In *OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems*, pages 73–88, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] Björn Andersson. The utilization bound of uniprocessor preemptive slack-monotonic scheduling is 50%. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 281–283, New York, NY, USA, 2008. ACM.

[13] Björn Andersson, Sanjoy Baruah, and Jan Jonsson. Static-priority scheduling on multiprocessors. In *RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 193–202, Washington, DC, USA, December 2001. IEEE Computer Society.

[14] Björn Andersson, Konstantinos Bletsas, and Sanjoy Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *Real-Time Systems Symposium, 2008*, pages 385–394, December 2008.

[15] Björn Andersson and Jan Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proc. of the IEEE Real-Time Systems Symposium, Work-in-Progress Session*, pages 53 – 56, 2000.

[16] Björn Andersson and Jan Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 33–40, July 2003.

[17] Björn Andersson and Eduardo Tovar. Multiprocessor scheduling with few preemptions. In *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, Sydney, Australia, 2006. IEEE Computer Society.

[18] Balakrishnan Ashok and T. K. Patra. Locating phase transitions in computationally hard problems. *Pramana*, 75:549–563, 2010. 10.1007/s12043-010-0138-0.

[19] Neil C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, University of York, York, England, UK, 1991.

[20] Neil C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[21] Neil C. Audsley, Alan Burns, Mike F. Richardson, and Andrew J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems*, May 1991.

[22] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *Computers, IEEE Transactions on*, 53(5):584 – 600, May 2004.

[23] François Baccelli, Guy Cohen, Geert Jan Oslder, and Jean-Pierre Quadrat. *Synchronization and Linearity an Algebra for Discrete Event Systems.* Wiley, 1992.

[24] Theodore P. Baker. Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99, April 1991.

[25] Theodore P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 120–129, December 2003.

[26] Theodore P. Baker. An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):760–768, August 2005.

[27] Theodore P. Baker. An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Syst.*, 32:49–71, February 2006.

[28] Theodore P. Baker and Sanjoy K. Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *ECRTS*, pages 141–150. IEEE Computer Society, 2009.

[29] Theodore P. Baker and Michele Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 178–190, Washington, DC, USA, December 2006. IEEE Computer Society.

[30] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[31] Sanjoy Baruah. The non-cyclic recurring real-time task model. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, RTSS '10, pages 173–182, Washington, DC, USA, December 2010. IEEE Computer Society.

[32] Sanjoy Baruah. Efficient computation of response time bounds for preemptive uniprocessor deadline monotonic scheduling. *Real-Time Syst.*, 47:517–533, December 2011.

[33] Sanjoy Baruah and Theodore Baker. Schedulability analysis of global EDF. *Real-Time Syst.*, 38:223–235, April 2008.

[34] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*,

pages 159–168, Washington, DC, USA, December 2006. IEEE Computer Society.

[35] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Syst.*, 17(1):5–22, July 1999.

[36] Sanjoy Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pages 321–329, December 2005.

[37] Sanjoy K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Syst.*, 24(1):93–128, January 2003.

[38] Sanjoy K. Baruah and John Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *ECRTS*, pages 195–202. IEEE Computer Society, July 2003.

[39] Sanjoy K. Baruah, N. K. Cohen, C. Greg Plaxton, and Donald A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[40] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, December 1990.

[41] Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In *RTSS*, pages 14–24. IEEE Computer Society, December 2010.

[42] Andrea Bastoni, Björn B. Brandenburg, and James H. Anderson. Is semi-partitioned scheduling practical? *Real-Time Systems, Euromicro Conference on*, 0:125–135, July 2011.

[43] Iain John Bate. *Scheduling and Timing Analysis for Safety Critical Real-Time Systems*. PhD thesis, University of York, York, England, UK, 1998.

[44] Carter Bays. A comparison of Next-fit, First-fit, and Best-fit. *Commun. ACM*, 20(3):191–192, 1977.

[45] Philipp Berens. CircStat: A MATLAB Toolbox for Circular Statistics. *Journal of Statistical Software*, 31(10):1–21, 9 2009.

[46] Marko Bertogna. *Real-Time Scheduling Analysis for Multiprocessor Platforms.* PhD thesis, Scuola Superiore Sant'Anna, Pisa, Italy, 2007.

[47] Marko Bertogna, Giorgio Buttazzo, and Gang Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *RTSS '11: Proceedings of the 32nd IEEE Real-Time Systems Symposium*, Washington, DC, USA, December 2011. IEEE Computer Society.

[48] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, *OPODIS*, volume 3974 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2005.

[49] Enrico Bini, G. C. Buttazzo, and G. M. Buttazzo. Rate monotonic analysis: the hyperbolic bound. *Computers, IEEE Transactions on*, 52(7):933–942, July 2003.

[50] Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 59–66, Washington, DC, USA, 2001. IEEE Computer Society.

[51] Enrico Bini and Giorgio C. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 196–203, Washington, DC, USA, 2004. IEEE Computer Society.

[52] Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput.*, 53(11):1462–1473, November 2004.

[53] Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst.*, 30:129–154, May 2005.

[54] Enrico Bini, Thi Huyen Châu Nguyen, Pascal Richard, and Sanjoy K. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Trans. Comput.*, 58:279–286, February 2009.

[55] Jacek Błażewicz. Selected topics in scheduling theory. In Silvano Martello, Gilbert Laporte, Michel Minoux, and Celso Ribeiro, editors, *Surveys in Combinatorial Optimization*, volume 132 of *North-Holland Mathematics Studies*, pages 1–59. North-Holland, 1987.

[56] Béla Bollobás and Oliver Riordan. *Handbook of Graphs and Networks*, chapter Mathematical Results on Scale-Free Random Graphs, pages 1–37. Wiley–VCH, 1st edition, 2003.

[57] Béla Bollobás and Oliver Riordan. *Percolation*. Cambridge University Press, 2006.

[58] Simon R. Broadbent and John Michael Hammersley. Percolation processes. *Mathematical Proceedings of the Cambridge Philosophical Society*, 53(03):629–641, 1957.

[59] Chris Brunsdon and Jon Corcoran. Using circular statistics to analyse time patterns in crime incidence. *Computers, Environment and Urban Systems*, 30(3):300–319, 2006.

[60] Almut Burchard, Jörg Liebeherr, Yingfeng Oh, and Sang H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *Computers, IEEE Transactions on*, 44(12):1429–1442, December 1995.

[61] Alan Burns. Preemptive priority-based scheduling: an appropriate engineering approach. In Sang H. Son, editor, *Advances in real-time systems*, pages 225–248. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[62] Alan Burns. Dual priority scheduling: Is the processor utilisation bound 100%? In R. I. Davis and N. Fisher, editors, *Proceedings of 1st International Real-Time Scheduling Open Problems Seminar, RTSOPS 2010*, pages 3–4, 2010.

[63] Alan Burns and Gordon Baxter. Time bands in systems structure. In Denis Besnard, Cristina Gacek, and Cliff B. Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pages 74–88. Springer London, 2006. 10.1007/1-84628-111-3_4.

[64] Alan Burns and Andy J. Wellings. Dual priority assignment: A practical method for increasing processor utilisation. In *in Proceedings of 5th Euromicro Workshop on Real-Time Systems, Oulu, IEEE Computer Soc*, pages 48–55. Computer Society Press, 1993.

[65] Giorgio C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications.* Kluwer Academic Publishers, 1997.

[66] Giorgio C. Buttazzo. Rate monotonic vs. EDF: judgment day. *Real-Time Syst.*, 29:5–26, January 2005.

[67] Jianqing Cai, Erik W. Grafarend, and Congwei Hu. The Statistical Property of the GNSS Carrier Phase Observations and its Effects on the Hypothesis Testing of the Related Estimators. In *Proceedings of the ION GNSS 20th Intemational Technical Meeting of the Satellite Division, Fort Worth*, pages 331–338, September 2007.

[68] Yang Cai and M. C. Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica*, 15:572–599, 1996. 10.1007/BF01940882.

[69] Chris K. Caldwell. A proof that all even perfect numbers are a power of two times a Mersenne prime. [Online], November 2011. `http://primes.utm.edu/notes/proofs/EvenPerfect.html`.

[70] John Carpenter, Shelby Funk, Philip Holman, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*, pages 30–1–30–19. Chapman Hall/CRC, Boca, 2004.

[71] Samarjit Chakraborty. *System-Level Timing Analysis and Scheduling for Embedded Packet Processors*. PhD thesis, Swiss Federal Institute of Technology Zurich, April 2003.

[72] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. Approximate schedulability analysis. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, RTSS '02, pages 159–168, Washington, DC, USA, December 2002. IEEE Computer Society.

[73] Yi-Hsiung Chao, Shun-Shii Lin, and Kwei-Jay Lin. Schedulability issues for EDZL scheduling on real-time multiprocessor systems. *Inf. Process. Lett.*, 107:158–164, August 2008.

[74] Jian-Jia Chen and Samarjit Chakraborty. Partitioned packing and scheduling for sporadic real-time tasks in identical multiprocessor systems. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 24–33, July 2012.

[75] Jiongxiong Chen. *Extensions to fixed priority with preemption threshold and reservation-based scheduling*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2005. AAINR03007.

[76] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. *Real-Time Systems Symposium, IEEE International*, 0:101–110, December 2006.

[77] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, USA, 1976.

[78] Michele Cirinei and Theodore P. Baker. EDZL scheduling analysis. In *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 9–18, Washington, DC, USA, 2007. IEEE Computer Society.

[79] Edward G. Coffman, Jr., Michael Randolph Garey, and David S. Johnson. *Approximation Algorithms for NP-Hard Problems*, chapter Approx-

imation algorithms for bin packing: a survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1996.

[80] Reuven Cohen and Shlomo Havlin. *Complex Networks: Structure, Robustness and Function.* Cambridge University Press, 2010.

[81] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[82] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* MIT Press, 3rd edition, 2009.

[83] Peter S. Craig. *Time Series Analysis for Directional Data.* PhD thesis, Trinity College Dublin, Department of Statistics, Dublin, Ireland, September 1988.

[84] Robert I. Davis and Alan Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In Theodore P. Baker, editor, *IEEE Real-Time Systems Symposium*, pages 398–409, Washington, DC, USA, December 2009. IEEE Computer Society.

[85] Robert I. Davis and Alan Burns. FPZL schedulability analysis. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 245–256, April 2011.

[86] Robert I. Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Syst.*, 47:1–40, January 2011.

[87] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4):35:1–35:44, October 2011.

[88] Robert I. Davis and Shinpei Kato. FPSL, FPCL and FPZL schedulability analysis. *Real-Time Systems*, 48(6):750–788, November 2012.

[89] Robert I. Davis and Andy J. Wellings. Dual priority scheduling. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1995.

[90] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[91] UmaMaheswari C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 23–30, July 2003.

[92] Sudarshan K. Dhall. Approximation algorithms for scheduling time-critical jobs on multiprocessor systems. In *Handbook on Scheduling Algorithms, Methods, and Models*, pages 32–1–32–30. Chapman Hall/CRC, Boca, 2004.

[93] Sudarshan K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[94] Jiwen Dong and Yang Zhang. A modified rate-monotonic algorithm for scheduling periodic tasks with different importance in embedded system. In *ICEMI 2009 - Proc. of 9th International Conference on Electronic Measurement and Instruments*, pages 4606–4609, 2009.

[95] Arvind Easwaran, Insik Shin, and Insup Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Syst.*, 43(1):25–59, 2009.

[96] Friedrich Eisenbrand and Thomas Rothvoß. Static-priority real-time scheduling: Response time computation is np-hard. In *IEEE Real-Time Systems Symposium*, pages 397–406. IEEE Computer Society, December 2008.

[97] Friedrich Eisenbrand and Thomas Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1029–1034, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[98] Paul Emberson. *Searching For Flexible Solutions To Task Allocation Problems.* PhD thesis, University of York, York, England, UK, 2009.

[99] Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In Giuseppe Lipari and Tommaso

Cucinotta, editors, *Proc. of the 1st Int'l Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) In conjunction with the 22nd Euromicro Conference on Real-Time Systems (ECRTS10)*, pages 6–11, Brussels, Belgium, July 2010.

[100] Edward S. Epstein. Stochastic dynamic prediction. *Tellus*, 21(6):739–759, 1969.

[101] Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[102] Paul Erdős and Alfréd Rényi. On the Evolution of Random Graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[103] Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12:261–267, 1961. 10.1007/BF02066689.

[104] Jeremy P. Erickson and James H. Anderson. Fair Lateness Scheduling: Reducing Maximum Lateness in G-EDF-Like Scheduling. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 3–12, July 2012.

[105] Euklid. *Die Elemente. Buch I - XIII.* Deutsch (Harri), 2003. Original from about 300 B.C.E.

[106] Ming Fan and Gang Quan. Harmonic-fit partitioned scheduling for fixed-priority real-time tasks on the multiprocessor platform. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pages 27–32, October 2011.

[107] Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, August 2007.

[108] Nathan Fisher and Sanjoy Baruah. The partitioned, static-priority scheduling of sporadic real-time tasks with constrained deadlines on multiprocessor platforms. In *Proceedings of the 9th international conference*

on Principles of Distributed Systems, OPODIS'05, pages 291–305, Berlin, Heidelberg, 2006. Springer-Verlag.

[109] Nathan Fisher and Sanjoy Baruah. The Global Feasibility and Schedulability of General Task Models on Multiprocessor Platforms. In *Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on*, pages 51–60, July 2007.

[110] Nathan Fisher, Sanjoy Baruah, and Theodore P. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, ECRTS '06, pages 118–127, Washington, DC, USA, 2006. IEEE Computer Society.

[111] Nicholas I. Fisher. Problems with the Current Definitions of the Standard Deviation of Wind Direction. *Journal of Applied Meteorology*, 26:1522–1529, November 1987.

[112] Ehud Friedgut and Jean Bourgain. Sharp thresholds of graph properties, and the k-SAT problem. *Journal of the American Mathematical Society*, 12(4):1017–1054, 1999.

[113] Ehud Friedgut and Gil Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American mathematical Society*, 124(10):2993–3002, 1996.

[114] Shelby Funk, Greg Levin, Caitlin Sadowski, Ian Pye, and Scott Brandt. DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling. *Real-Time Systems*, 47(5):389–429, 2011.

[115] Shelby H. Funk and Vijaykant Nadadur. LRE-TL: An optimal multiprocessor scheduling algorithm for sporadic task sets. In *Proceedings of the 17th Real-Time and Network Systems (RTNS)*, pages 159–168, 2009.

[116] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. Modeling time in computing: A taxonomy and a comparative survey. *ACM Comput. Surv.*, 42(2):6:1–6:59, March 2010.

[117] Gary L. Gaile and James E. Burt. *Directional Statistics.* Concepts and Techniques in Modern Geography. Geo Abstracts, 1980.

[118] Henry L. Gantt. *Work, wages, and profits: their influence on the cost of living*. Works management library. The Engineering magazine, 1910.

[119] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[120] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research Report RR-2966, INRIA, Rocquencourt, France, 1996. Projet REFLECS.

[121] Edgar N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, December 1959.

[122] Andreas Goerdt. A threshold for unsatisfiability. *Journal of Computer and System Sciences*, 53(3):469–486, 1996.

[123] Andreas Goerdt and Udo Kamps. On the reasons for average superlinear speedup in parallel backtrack search. In Egon Börger, Yuri Gurevich, and Karl Meinke, editors, *Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 106–127. Springer Berlin / Heidelberg, 1994. 10.1007/BFb0049327.

[124] Joël Goossens, Shelby Funk, and Sanjoy Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Syst.*, 25:187–205, September 2003.

[125] Sathish Gopalakrishnan. *Resource Management for Real Time Environments*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, USA, 2006.

[126] Sathish Gopalakrishnan. Sharp utilization thresholds for some real-time scheduling problems. *CoRR*, abs/0912.3852, 2009.

[127] Sathish Gopalakrishnan. A sharp threshold for rate monotonic schedulability of real-time tasks. In *Proc. of the 1st Int'l Real-Time Scheduling Open Problems Seminar (RTSOPS) at 22nd Euromicro Int'l Conference on Real-Time Systems (ECRTS)*, page 23, July 2010.

[128] Sathish Gopalakrishnan and Marco Caccamo. Rate monotonic scheduling: sharp thresholds and applications to soft real-time systems. In *13th EEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Work-in-Progress Session*, pages 1–4, April 2007.

[129] Sathish Gopalakrishnan, Marco Caccamo, and Lui Sha. Sharp thresholds for scheduling recurring tasks with distance constraints. *Computers, IEEE Transactions on*, 57(3):344–358, March 2008.

[130] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, RTSS '09, pages 387–397, Washington, DC, USA, 2009. IEEE Computer Society.

[131] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. In *RTAS '10: Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 165–174, Washington, DC, USA, April 2010. IEEE Computer Society.

[132] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Parametric utilization bounds for fixed-priority multiprocessor scheduling. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 261–272, May 2012.

[133] Pablo Chico de Guzmán, Amadeo Casas, Manuel Carro, and Manuel V. Hermenegildo. Parallel backtracking with answer memoing for independent and-parallelism. *CoRR*, abs/1107.4724, July 2011. [Online], `http://arxiv.org/pdf/1107.4724.pdf`.

[134] Ching-Chih Han and Kwei-Jay Lin. Scheduling distance-constrained real-time tasks. In *Real-Time Systems Symposium, 1992*, pages 300–308, December 1992.

[135] Ching-Chih Han, Kwei-Jay Lin, and Chao-Ju Hou. Distance-constrained scheduling and its applications to real-time systems. *Computers, IEEE Transactions on*, 45(7):814–826, July 1996.

[136] Ching-Chih Han and Hung-Ying Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, RTSS '97, pages 36–45, Washington, DC, USA, 1997. IEEE Computer Society.

[137] Sangchul Han, Moonju Park, and Minkyu Park. A sufficient condition for rate monotonic schedulability based on response time analysis. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 1751–1757, Washington, DC, USA, 2010. IEEE Computer Society.

[138] Jeffery P. Hansen, John P. Lehoczky, Haifeng Zhu, and Ragunathan Rajkumar. Quantized EDF Scheduling in a Stochastic Environment. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 279–285, Washington, DC, USA, 2002. IEEE Computer Society.

[139] David Harrison, Gopal K. Kanji, and R. J. Gadsden. Analysis of variance for circular data. *Journal of Applied Statistics*, 13(2):123–138, 1986.

[140] Steffen Haugk. Die Deutsche GIMPS Homepage. [Online], June 2012. `http://haugk.co.uk/category/gimps/`, (in German).

[141] Robert Holte, Aloysius Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel. The pinwheel: a real-time scheduling problem. In *System Sciences, 1989. Vol.II: Software Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on*, volume 2, pages 693 –702 vol.2, January 1989.

[142] Chih-Wen Hsueh and Kwei-Jay Lin. An optimal pinwheel scheduler using the single-number reduction technique. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, RTSS '96, pages 196–205, Washington, DC, USA, December 1996. IEEE Computer Society.

[143] S. Rao Jammalamadaka and A. Sengupta. *Topics in Circular Statistics*. World Scientific Pub Co Inc, Singapore, har/dskt edition, 2001.

[144] Kevin Jeffay. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Real-Time Systems Symposium, 1992*, pages 89–99, December 1992.

[145] Kevin Jeffay and Donald L. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 212–221, December 1993.

[146] E. Douglas Jensen, C. Douglas Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, pages 112–122. IEEE Computer Society, 1985.

[147] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[148] Myoung-Jo Jung, Yeong Rak Seong, and Cheol-Hoon Lee. Optimal RM scheduling for simply periodic tasks on uniform multiprocessors. In *ICHIT '09: Proceedings of the 2009 International Conference on Hybrid Information Technology*, pages 383–389, New York, NY, USA, 2009. ACM.

[149] Rudolf Emil Kálmán. A new approach to linear filtering and prediction problems. *Trans. ASME, D*, 82:35–44, 1960.

[150] Bala Kalyanasundaram, Kirk R. Pruhs, and Eric Torng. Errata: A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 28:269–270, 2000. 10.1007/s004530010048.

[151] Arvind Kandhalu, Karthik Lakshmanan, Junsung Kim, and Ragunathan (Raj) Rajkumar. pCOMPATS: Period-Compatible Task Allocation and Splitting on Multi-core Processors. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, pages 307–316. IEEE Computer Society, April 2012.

[152] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

[153] Andreas Karrenbauer and Thomas Rothvoß. An Average-Case Analysis for Rate-Monotonic Multiprocessor Real-time Scheduling. In *17th Annual European Symposium on Algorithms (ESA'09)*, Lecture Notes in Computer Science, Berlin, 2009. Springer.

[154] Andreas Karrenbauer and Thomas Rothvoß. A 3/2-approximation algorithm for rate-monotonic multiprocessor scheduling of implicit-deadline tasks. In *Proceedings of the 8th international conference on Approximation and online algorithms*, WAOA'10, pages 166–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[155] Shinpei Kato. *Real-Time Scheduling of Periodic and Aperiodic Tasks on Multiprocessor Systems.* PhD thesis, Keio University, Tokyo, Japan, 2008.

[156] Shinpei Kato, Akira Takeda, and Nobuyuki Yamasaki. Global rate-monotonic scheduling with priority promotion. Technical Report CMU-ECE-TR10-05, Carnegie Mellon University, Pittsburgh, PA, USA, May 2010.

[157] Shinpei Kato and Nobuyuki Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 441–450, Washington, DC, USA, 2007. IEEE Computer Society.

[158] Shinpei Kato and Nobuyuki Yamasaki. Global EDF-based scheduling with efficient priority promotion. In *RTCSA '08: Proceedings of the 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 197–206, Kaohsiung, Taiwan, 2008. IEEE Computer Society.

[159] Shinpei Kato and Nobuyuki Yamasaki. Modular real-time linux. In *Proc. of the 10th Real-Time Linux Workshop (RTLWS10)*, pages 97–102, 2008.

[160] Shinpei Kato and Nobuyuki Yamasaki. Portioned EDF-based scheduling on multiprocessors. In *EMSOFT '08: Proceedings of the 8th ACM in-*

ternational conference on Embedded software, pages 139–148, New York, NY, USA, 2008. ACM.

[161] Shinpei Kato and Nobuyuki Yamasaki. Portioned static-priority scheduling on multiprocessors. In *IPDPS*, pages 1–12. IEEE, 2008.

[162] Shinpei Kato and Nobuyuki Yamasaki. Real-time scheduling module for linux kernel. In *IPSJ Transactions on Advanced Computing Systems*, volume 2, pages 75–86, 2009.

[163] Shinpei Kato and Nobuyuki Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *RTAS '09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 23–32, Washington, DC, USA, 2009. IEEE Computer Society.

[164] Shinpei Kato, Nobuyuki Yamasaki, and Yutaka Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 249–258, Dublin, Ireland, 2009. IEEE Computer Society.

[165] Harry Kesten. The critical probability of bond percolation on the square lattice equals 1/2. *Communications in Mathematical Physics*, 74:41–59, 1980. 10.1007/BF01197577.

[166] Kanghee Kim and Chang-Gun Lee. A safe stochastic analysis with relaxed limitations on the periodic task model. *Computers, IEEE Transactions on*, 58(5):634 –647, May 2009.

[167] Steffen Kollmann, Victor Pollex, and Frank Slomka. Reducing response times by competition based dependencies. In *Proceedings of the 14th Workshop of Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, February 2011.

[168] János Komlós and Endre Szemerédi. Limit distribution for the existence of hamiltonian cycles in a random graph. *Discrete Mathematics*, 43(1):55–63, 1983.

[169] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Springer Publishing Company, Incorporated, 2nd edition, 2011.

[170] Tei-Wei Kuo, Li-Pin Chang, Yu-Hua Liu, and Kwei-Jay Lin. Efficient online schedulability tests for real-time systems. *IEEE Trans. Softw. Eng.*, 29(8):734–751, 2003.

[171] Tei-Wei Kuo and Aloysius K. Mok. Load adjustment in adaptive real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 160 –170, December. 1991.

[172] Karthik Lakshmanan, Ragunathan Rajkumar, and John Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 239–248, Washington, DC, USA, 2009. IEEE Computer Society.

[173] Sylvain Lauzac, Rami Melhem, and Daniel Mossé. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 188 –195, June 1998.

[174] Sylvain Lauzac, Rami Melhem, and Daniel Mossé. An efficient RMS admission control and its application to multiprocessor scheduling. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proc. of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 511 –518, April 1998.

[175] Chang-Gun Lee, Lui Sha, and Avinash Peddi. Enhanced utilization bounds for QoS management. *Computers, IEEE Transactions on*, 53(2):187–200, February 2004.

[176] Edward A. Lee. Computing needs time. *Commun. ACM*, 52(5):70–79, May 2009.

[177] Jinkyu Lee, Arvind Easwaran, and Insik Shin. LLF schedulability analysis on multiprocessor platforms. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 25–36, December 2010.

[178] Jinkyu Lee, Arvind Easwaran, and Insik Shin. Laxity dynamics and LLF schedulability analysis on multiprocessor platforms. *Real-Time Systems*, 48(6):716–749, November 2012.

[179] Suk Kyoon Lee. On-line multiprocessor scheduling algorithms for real-time tasks. In *TENCON '94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, pages 607–611 vol.2, August 1994.

[180] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, December 1990.

[181] John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS*, pages 166–171, December 1989.

[182] Hennadiy Leontyev, Samarjit Chakraborty, and James H. Anderson. Multiprocessor extensions to real-time calculus. *Real-Time Systems*, 47(6):562–617, 2011.

[183] Joseph Y.-T. Leung. A new algorithm for scheduling periodic real-time tasks. *Algorithmica*, 4(2):209–219, 1989.

[184] Joseph Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Inf. Process. Lett.*, 11(3):115–118, 1980.

[185] Joseph Y. T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[186] Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye, and Scott A. Brandt. DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. In *ECRTS*, pages 3–13. IEEE Computer Society, 2010.

[187] Rhyd Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Comput. Oper. Res.*, 36:2295–2310, July 2009.

[188] Shuhui Li, Shangping Ren, Yue Yu, Xing Wang, Li Wang, and Gang Quan. Profit and penalty aware scheduling for real-time online services. *Industrial Informatics, IEEE Transactions on*, 8(1):78 –89, February 2012.

[189] Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[190] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[191] Shuo Liu, Gang Quan, and Shangping Ren. On-line scheduling of real-time services with profit and penalty. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1476–1481, New York, NY, USA, 2011. ACM.

[192] Xue Liu and Tarek Abdelzaher. Nonutilization bounds and feasible regions for arbitrary fixed-priority policies. *ACM Trans. Embed. Comput. Syst.*, 10(3):31:1–31:25, May 2011.

[193] V. Lomné, A. Dehaboui, P. Maurine, L. Torres, and M. Robert. *Security Trends for FPGAS: From Secured to Secure Reconfigurable Systems*, chapter Side Channel Attacks, pages 47–66. Springer, 2011.

[194] José María López, José Luis Díaz, and Daniel F. García. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *ECRTS*, pages 67–75, 2001.

[195] José María López, José Luis Díaz, and Daniel F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Syst.*, 28:39–68, October 2004.

[196] José María López, M. García, José Luis Díaz, and Daniel F. García. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, pages 25–33, 2000.

[197] José María López, M. García, José Luis Díaz, and Daniel F. García. Utilization bounds for multiprocessor rate-monotonic scheduling. *Real-Time Syst.*, 24:5–28, January 2003.

[198] Lars Lundberg. Multiprocessor scheduling of age constraint processes. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, RTCSA '98, pages 42–47, Washington, DC, USA, 1998. IEEE Computer Society.

[199] Lars Lundberg. Analyzing fixed-priority global multiprocessor scheduling. In *RTAS '02: Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, pages 145–153, Washington, DC, USA, 2002. IEEE Computer Society.

[200] Lars Lundberg and Håkan Lennerstad. Global multiprocessor scheduling of aperiodic tasks using time-independent priorities. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '03, pages 170–180, Washington, DC, USA, 2003. IEEE Computer Society.

[201] Lars Lundberg and Håkan Lennerstad. Slack-based global multiprocessor scheduling of aperiodic tasks in parallel embedded real-time systems. In *Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA '08, pages 465–472, Washington, DC, USA, 2008. IEEE Computer Society.

[202] Irina Lupu, Pierre Courbin, Laurent George, and Joël Goossens. Multi-criteria evaluation of partitioning schemes for real-time systems. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8, September 2010.

[203] José Manuel Marinho, Vincent Nélis, Stefan M. Petters, and Isabelle Puaut. Preemption delay analysis for floating non-preemptive region scheduling. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 497–502, March 2012.

[204] Alejandro Masrur, Samarjit Chakraborty, and Georg Färber. Constant-time admission control for deadline monotonic tasks. In *Proceedings of*

the Conference on Design, Automation and Test in Europe, DATE '10, pages 220–225, 3001 Leuven, Belgium, July 2010. European Design and Automation Association.

[205] Alejandro Masrur, Samarjit Chakraborty, and Georg Färber. Constant-time admission control for partitioned EDF. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, ECRTS '10, pages 34–43, Washington, DC, USA, 2010. IEEE Computer Society.

[206] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[207] Gilbert Mensah. *K-Priority Scheduling of Hard Real-Time Implicit-Deadline Periodic Task Systems on Uniprocessor*. PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 2011.

[208] Stephan Mertens, Marc Mézard, and Riccardo Zecchina. Threshold values of random K-SAT from the cavity method. *Random Structures & Algorithms*, 28(3):340–373, 2006.

[209] Donald Michie. "Memo" Functions and Machine Learning. *Nature*, 218:19–22, April 1968.

[210] Nasro Min-Allah, Samee Ullah Khan, Nasir Ghani, Juan Li, Lizhe Wang, and Pascal Bouvry. A comparative study of rate monotonic schedulability tests. *J. Supercomput.*, 59(3):1419–1430, March 2012.

[211] Richard von Mises. Über die "Ganzzahligkeit" der Atomgewichte und verwandte Fragen. *Physikalische Zeitschrift*, 19:490–500, 1918. (in German).

[212] Aloysius Ka-Lau Mok. *Fundamental Design Problems of Distributed Systems for the Hard-real-time Environment*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[213] Peter W. G. Morris. *The Management of Projects*. Thomas Telford Ltd, new edition, 1997.

[214] Noël Tchidjo Moyo, Eric Nicollet, Frédéric Lafaye, and Christophe Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, ECRTS '10, pages 271–278, Washington, DC, USA, 2010. IEEE Computer Society.

[215] Dirk Müller. *Subpixel-Filterung für eine autostereoskopische Multiperspektiven-3-D-Darstellung hoher Qualität.* PhD thesis, Universität Kassel, Kassel, Hesse, Germany, April 2006. (in German).

[216] Dirk Müller. Accelerated Simply Periodic Task Sets for RM Scheduling. In *Proc. of Embedded Real Time Software and Systems (ERTS^2)*, page 46, Toulouse, France, May 2010.

[217] Dirk Müller. Period Fitting for Rate-monotonic Scheduling Using a Circular Similarity Measure. In *Proc. of the 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012), WiP Session*, pages 427–430, Seoul, Korea, August 2012.

[218] Dirk Müller and Matthias Werner. Communication Protocols for Real-Time Systems: Time vs. Events Revisited. In *Proc. of the 30th IFAC Workshop on Real-Time Programming and 4th Int'l Workshop on Real-Time Software (WRTP/RTS'09)*, pages 143–148, Mrągowo, Poland, October 2009.

[219] Dirk Müller and Matthias Werner. Efficient RMS schedulability tests. In *Proc. of the 1st Int'l Real-Time Scheduling Open Problems Seminar (RTSOPS) at 22nd Euromicro Int'l Conference on Real-Time Systems (ECRTS)*, pages 16–17, Brussels, Belgium, July 2010.

[220] Dirk Müller and Matthias Werner. A Generalized View on Beneficial Task Sortings for Partitioned RMS Task Allocation on Multiprocessors. In *Proc. of the 2nd Int'l Real-Time*

***Scheduling Open Problems Seminar (RTSOPS) at 23rd Euromicro Int'l Conference on Real-Time Systems (ECRTS)*, pages 7–8, Porto, Portugal, July 2011.**

[221] Dirk Müller and Matthias Werner. Genealogy of Hard Real-Time Pre-emptive Scheduling Algorithms for Identical Multiprocessors. *Central European Journal of Computer Science (CEJCS)*, 1(3):253–265, September 2011.

[222] Dirk Müller and Matthias Werner. Improved Heuristics for Partitioned Multiprocessor Scheduling Based on Rate-Monotonic Small-Tasks. In *Proc. of the 19th International Conference on Real-Time and Network Systems (RTNS)*, pages 211–220, Nantes, France, September 2011.

[223] Dirk Müller and Matthias Werner. Towards Exact Thresholds for Scheduling n Tasks on m Processors Based on Partitioned EDF. In *Proc. of the17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011), WiP Session*, pages 25–28, Chicago, USA, April 2011.

[224] Dirk Müller and Matthias Werner. A Note on "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems". *Computers, IEEE Transactions on*, 62(9):1904–1905, 2012.

[225] B. Neelima and Prakash S. Raghavendra. Recent trends in software and hardware for GPGPU computing: A comprehensive survey. In *Industrial and Information Systems (ICIIS), 2010 International Conference on*, pages 319–324, August 2010.

[226] Geoffrey Nelissen, Vendy Berten, Joël Goossens, and Dragomir Milojevic. Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 15–24, August 2011.

[227] Geoffrey Nelissen, Vendy Berten, Vincent Nélis, Joël Goossens, and Dragomir Milojevic. U-EDF: An Unfair but Optimal Multiproces-

sor Scheduling Algorithm for Sporadic Tasks. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 13–23, July 2012.

[228] John von Neumann. First draft of a report on the EDVAC. *Annals of the History of Computing, IEEE*, 15(4):27–75, 1993. Original from 1945.

[229] Nikos Nikolaidis and Iannis Pitas. Nonlinear processing and analysis of angular signals. *Signal Processing, IEEE Transactions on*, 46(12):3181–3194, December 1998.

[230] Dionisio de Niz, Karthik Lakshmanan, and Ragunathan Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *RTSS '09: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, pages 291–300, Washington, DC, USA, December 2009. IEEE Computer Society.

[231] Dong-Ik Oh and Theodore P. Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2):183–192, 1998.

[232] Sung-Heun Oh and Seung-Min Yang. A Modified Least-Laxity-First scheduling algorithm for real-time tasks. In *Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on*, pages 31–36, October 1998.

[233] Yingfeng Oh and Sang H. Son. Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Syst.*, 9:207–239, November 1995.

[234] Melissa E. O'Neill. The Genuine Sieve of Eratosthenes. *J. Funct. Program.*, 19:95–106, January 2009.

[235] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Robert I. Davis, and Mateo Valero. IA^3: An Interference Aware Allocation Algorithm for Multicore Hard Real-Time Systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 280–290, April 2011.

[236] Dong-Won Park, S. Natarajan, A. Kanevsky, and Myung Jun Kim. A generalized utilization bound test for fixed-priority real-time scheduling.

Real-Time Computing Systems and Applications, International Workshop on, 0:73, 1995.

[237] Moonju Park. Comments on "Generalized rate monotonic schedulability bounds using relative period ratios". *Inf. Process. Lett.*, 111(7):334–337, March 2011.

[238] Moonju Park and Heemin Park. An efficient test method for rate monotonic schedulability. *IEEE Transactions on Computers*, 99(PrePrints):1, 2012.

[239] Daniel Parthey and Robert Baumgartl. Analyzing access timing of removable flash media. In *RTCSA*, pages 510–515. IEEE Computer Society, August 2007.

[240] Xuefeng Piao, Sangchul Han, Heeheon Kim, Minkyu Park, Yookun Cho, and Seongje Cho. Predictability of Earliest Deadline Zero Laxity Algorithm for Multiprocessor Real-Time Systems. In *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, ISORC '06, pages 359–364, Washington, DC, USA, 2006. IEEE Computer Society.

[241] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35(5):89–102, 2001.

[242] Daniel P. Playne and Ken A. Hawick. Comparison of GPU architectures for asynchronous communication with finite-differencing applications. *Concurrency and Computation: Practice and Experience*, 24(1):73–83, 2012.

[243] Victor Pollex, Steffen Kollmann, and Frank Slomka. Generalizing response-time analysis. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, pages 203–211, August 2010.

[244] Derek de Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American Society for Information Science*, 27(5):292–306, 1976.

[245] Amitava Raha, Nicholas Malcolm, and Wei Zhao. Hard real-time communications with weighted round robin service in atm local area networks. In *Engineering of Complex Computer Systems, 1995. Held jointly with 5th CSESAW, 3rd IEEE RTAW and 20th IFAC/IFIP WRTP, Proceedings., First IEEE International Conference on*, pages 96–103, November 1995.

[246] Arnoldo Ramírez and Pedro Mejía. Comprehensive Comparison of Schedulability Tests for Uniprocessor Rate-Monotonic Scheduling. *Journal on Applied Research and Technology*, 2012. to appear.

[247] Jammalamadaka S. Rao. *Some contributions to the analysis of circular data*. PhD thesis, Indian Statistical Institute, Calcutta, India, 1969.

[248] Gurulingesh Raravi, Björn Andersson, and Konstantinos Bletsas. Two-type heterogeneous multiprocessor scheduling: Is there a phase transition? In *Proc. of the 2nd Int'l Real-Time Scheduling Open Problems Seminar (RTSOPS) at 23rd Euromicro Int'l Conference on Real-Time Systems (ECRTS)*, pages 19–20, July 2011.

[249] Glenn E. Reeves. What really happened on mars ? [Online], December 1997. `http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Authoritative_Account.html`.

[250] Eva Regnier. Activity Completion Times in PERT and Scheduling Network Simulation, Part II. *DRMI Newsletter*, 12:1–9, April 2005.

[251] Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 104–115, December 2011.

[252] Ismael Ripoll, Alfons Crespo, and Aloysius K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Syst.*, 11:19–39, July 1996.

[253] Thomas Rothvoß. *On the computational complexity of periodic scheduling*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, 2009.

[254] Manas Saksena and Yun Wang. Scalable real-time system design using preemption thresholds. In *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, pages 25–34, 2000.

[255] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175–1185, September 1990.

[256] Chi-Sheng Shih, Sathish Gopalakrishnan, Phanindra Ganti, Marco Caccamo, and Lui Sha. Scheduling real-time dwells using tasks with synthetic periods. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 210–219, Washington, DC, USA, December 2003. IEEE Computer Society.

[257] Charles Edward Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101, January 1904.

[258] Ewald Speckenmeyer, Burkhard Monien, and Oliver Vornberger. Superlinear speedup for parallel backtracking. In E. Houstis, T. Papatheodorou, and C. Polychronopoulos, editors, *Supercomputing*, volume 297 of *Lecture Notes in Computer Science*, pages 985–993. Springer Berlin / Heidelberg, 1988.

[259] Brinkley Sprunt. *Aperiodic task scheduling for real-time systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, August 1990. AAI9107570.

[260] Roger Stafford. Random vectors with fixed sum. [Online], January 2006. `http://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum`.

[261] John A. Stankovic and Krithi Ramamritham. *Tutorial on Hard Real-Time Systems.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1987.

[262] Otto Stern and Max Volmer. Sind die Abweichungen der Atomgewichte von der Ganzzahligkeit durch Isotopie erklärbar? *Annalen der Physik*, 364(11):225–238, 1919. (in German).

[263] Georges Stienne, Serge Reboul, Jean-Bernhard Choquel, and Mohammed Benjelloun. Circular data processing tools applied to a phase open loop architecture for multi-channels signals tracking. In *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pages 633–642, April 2012.

[264] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The Digraph Real-Time Task Model. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 71–80, April 2011.

[265] Akira Takeda, Shinpei Kato, and Nobuyuki Yamasaki. Real-time scheduling based on rate monotonic for multiprocessors. *IPSJ Transactions on Advanced Computing Systems*, 2(1):64–74, March 2009. (in Japanese).

[266] Henry Osborn Taylor. *The Mediaeval Mind (V. 2); A History of the Development of Thought and Emotion in the Middle Ages.* General Books, 2009.

[267] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104 vol.4, 2000.

[268] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40(2-3):117–134, April 1994.

[269] Alan M. Turing. On computable number with an application to the *Entscheidungsproblem. Proc. Amer. Math. Soc.*, 42(2):230–265, 1936-7.

[270] TutorVista.com. Circular permutations. [Online], 2010. `http://math.tutorvista.com/statistics/permutation.html#circular-permutations`.

[271] Marcus Völp, Claude-Joachim Hamann, and Hermann Härtig. Avoiding timing channels in fixed-priority schedulers. In *Proceedings of the*

2008 ACM symposium on Information, computer and communications security, ASIACCS '08, pages 44–55, New York, NY, USA, 2008. ACM.

[272] Yun Wang and Manas Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, pages 328–335, 1999.

[273] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[274] Hsin-Wen Wei, Yi-Hsiung Chao, Shun-Shii Lin, Kwei-Jay Lin, and Wei-Kuan Shih. Current Results on EDZL Scheduling for Multiprocessor Real-Time Systems. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '07, pages 120–130, Washington, DC, USA, 2007. IEEE Computer Society.

[275] Hsin-Wen Wei, Kwei-Jay Lin, Wan-Chen Lu, and Wei-Kuan Shih. Generalized rate monotonic schedulability bounds using relative period ratios. *Inf. Process. Lett.*, 107(5):142–148, August 2008.

[276] Eric W. Weisstein. Greatest common divisor. [Online], November 2011. From MathWorld–A Wolfram Web Resource `http://mathworld.wolfram.com/GreatestCommonDivisor.html`.

[277] Eric W. Weisstein. Percolation threshold. [Online], April 2012. From MathWorld–A Wolfram Web Resource `http://mathworld.wolfram.com/PercolationThreshold.html`.

[278] Matthias Werner. *Responsivität – ein konsensbasierter Ansatz.* WBI GmbH, 2000. (in German).

[279] Matthias Werner, Michael A. Jaeger, and Helge Parzyjegla. An application of the (max, +) algebra to information flow security. In *Networking, 2008. ICN 2008. Seventh International Conference on*, pages 262–266, April 2008.

[280] Wikipédia. Graphe aléatoire — wikipédia, l'encyclopédie libre. [Online], October 2011. `http://fr.wikipedia.org/w/index.php?title=Graphe_al%C3%A9atoire&oldid=73602579`, (in French).

[281] Wikipedia. Jitter — wikipedia, the free encyclopedia. [Online], August 2012. `http://en.wikipedia.org/w/index.php?title=Jitter&oldid=505918724`.

[282] Wikipedia. Lotus bridge — wikipedia, the free encyclopedia. [Online], January 2012. `http://en.wikipedia.org/w/index.php?title=Lotus_Bridge&oldid=469695406`.

[283] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Müller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7:36:1–36:53, May 2008.

[284] Jianjia Wu, Jyh-Charn Liu, and Wei Zhao. A general framework for parameterized schedulability bound analysis of real-time systems. *Computers, IEEE Transactions on*, 59(6):776–783, June 2010.

[285] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995.

[286] Hong-liu Yang and Günter Radons. Geometry of Inertial Manifolds Probed via a Lyapunov Projection Method. *Phys. Rev. Lett.*, 108:154101, April 2012.

[287] Gang Yao, Giorgio C. Buttazzo, and Marko Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *RTCSA*, pages 351–360. IEEE Computer Society, 2009.

[288] Omar U. Pereira Zapata, Pedro Mejía-Alvarez, and Luis E. Leyva del Foyo. Comparative analysis of real-time scheduling algorithms on one processor under rate monotonic. Technical report, Departamento de Computación, CINVESTAV-IPN, Guadalajara, Mexico, 2005.

[289] Areej Zuhily and Alan Burns. Optimal (D-J)-monotonic priority assignment. *Inf. Process. Lett.*, 103(6):247–250, 2007.

[290] Katharina A. Zweig, Gergely Palla, and Tamás Vicsek. What makes a phase transition? Analysis of the random satisfiability problem. *Physica A: Statistical Mechanics and its Applications*, 389(8):1501–1511, 2010.