A Gravitational Task Model for Target Sensitive Real-Time Applications

vom
Fachbereich Elektrotechnik und Informationstechnik
der Technische Universität Kaiserslautern
zur Verleihung des akademischen Grades eines

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von
Raphael Pereira de Oliveira Guerra

D 386

Eingereicht am:	03.05.2011
Tag der mündlichen Prüfung:	10.06.2011
Dekan des Fachbereichs:	Prof. Dr.-Ing. Norbert Wehn
Promotionskomission	
Vorsitzender:	Prof. Dr.-Ing. Steven Liu
Berichterstattende:	Prof. Dipl.-Ing. Dr. Gerhard Fohler
	Prof. Sanjoy Baruah
	Prof. Julius Leite

Abstract

For many years real-time task models have focused the timing constraints on execution windows defined by earliest start times and deadlines for feasibility. However, the utility of some application may vary among scenarios which yield correct behavior, and maximizing this utility improves the resource utilization. For example, target sensitive applications have a target point where execution results in maximized utility, and an execution window for feasibility. Execution around this point and within the execution window is allowed, albeit at lower utility. The intensity of the utility decay accounts for the importance of the application. Examples of such applications include multimedia and control; multimedia application are very popular nowadays and control applications are present in every automated system.

In this thesis, we present a novel real-time task model which provides for easy abstractions to express the timing constraints of target sensitive real-time (RT) applications: the gravitational task model. This model uses a simple gravity pendulum (or bob pendulum) system as a visualization model for trade-offs among target sensitive RT applications. We consider jobs as objects in a pendulum system, and the target points as the central point. Then, the equilibrium state of the physical problem is equivalent to the best compromise among jobs with conflicting targets. Analogies with well-known systems are helpful to fill in the gap between application requirements and theoretical abstractions used in task models. For instance, the so-called nature algorithms use key elements of physical processes to form the basis of an optimization algorithm [Carnahan 01]. Examples include the knapsack problem, traveling salesman problem, ant colony optimization, and simulated annealing.

We also present a few scheduling algorithms designed for the gravitational task model which fulfill the requirements for on-line adaptivity. The scheduling of target sensitive RT applications must account for timing constraints, and the trade-off among tasks with conflicting targets. Our proposed scheduling algorithms use the equilibrium state concept to order the execution sequence of jobs, and compute the deviation of jobs from their target points for increased system utility. The execution sequence of jobs in the schedule has a significant impact on the equilibrium of jobs, and dominates the complexity of the problem — the optimum solution is Non-deterministic Polynomial-time hard (NP-hard) [Chen 96]. We show the efficacy of our approach through simulations results and 3 target sensitive RT applications enhanced with the gravitational task model.

"Rapadura é doce, mas não é mole não!"
English translation: Rapadura[1] is sweet, but isn't soft!

— Brazilian Proverb

[1]Rapadura is a form of dried sugarcane juice used in some brazilian regions as sweetener or as candy.

Preface

These are one of my first lines of text in this thesis, and ironically, the ones that I write last. Actually, I believe most students do the same. We students make so much effort to put our work together into a thesis, that, in the end, the preface seems the least important. Yet, I am sure that many people just open a thesis to read the preface, and then complain "Why didn't you mention me?!". The pressure on me as I write these words is comparable to the pressure of writing the whole thesis. However, before the acknowledgments, I will spend a few words telling about my experience doing my Ph.D. at Technische Universität Kaiserslautern.

I started my Ph.D. at the age of 23, and until then, I had never lived away from my grandparents, who raised me up and deserve all my gratitude. So, you might imagine that the beginning was very tough, as I had to learn to clean, to cook, to do laundry, etc. Even more importantly, I had to be responsible by myself: wake up early in the morning, keep track of appointments, be on time to take the trains and flights (these are tough ones!!!), and many more. These were the very first things I had to learn as a Ph.D. student, and surprisingly, when I had to struggle the most in order to succeed.

The experience of living alone and in Europe contributed a lot to my personal development. I could enjoy traveling around, making new friends, learning a new language (german) and improving my english and spanish (not to mention all the funny stories that go beyond the scope of this preface). I am particularly thankful for having the opportunity to interact with diverse cultures in the international environment that the university offers.

Professionally, I had the opportunity to work in big research projects which involved several industrial and academic partners from the whole Europe. I also had the chance to meet renowned researchers, and to attend and present my work in premier international conferences. Most importantly, the guidance of my supervisor Prof. Gerhard Fohler was crucial for my professional success. I am thankful for his effort to explore my potential to the limit, and turning me into a good researcher. Notably, our best paper award in ECRTS'08 (a premier conference in real-time systems) is a priceless achievement.

Of course, my professional success also depends on the quality of my private life. Therefore, I am thankful to all my friends who were always there to cheer me up during hard times, like depressive winters (remember that I am a tropical being) and lonely holiday. I want to thank Christoph Ruppert and his family for welcoming me in their house for Christmas, Gerrit Kehr for helping me in my social integration in Kaiserslautern,

and Leon Thurner for doing his best to improve my german — learning german was an important step for me to exploit the experience of living in Germany to the fullest. I want to thank my american circle of friends (notably Frank, Al, Lili, Nathan, Kip, and T.J.) for the nice moments they shared with me hanging out, partying, and traveling. Without them living in Kaiserslautern would have been unendurable.

The experience of living abroad and working in an international university also allowed me to meet many people, which are constantly entering and leaving such a dynamic environment. I am glad that I have met Florian Uhl, Jose Visquert and the rest of the spanish gang, the brazilian Gaúchos with whom I shared many trips and laughs, the couchsurfers that very kindly welcomed me in their homes during my trips, and many other people I had the pleasure to share a short moment of my life with. This experience also made me recognize the value of friendship that prevails over the distance. For that, I am grateful to Kadu, Leonardo Melo, Luan Chagas, Felipe Carone, Diana, and my family for supporting me on hard times despite of the distance.

Writing this thesis was a hard task. Besides my supervisor Prof. Gerhard Fohler, I thank my work colleagues (Ramon Oliver, Anand Kotra, Jens Theis, Stefan Schorr, Rodrigo Coelho, Cuong Ngo, and Alex Neundorf) and conference reviewers for their comments and reviews. I also thank Stephanie Jung for her constant help with the german bureaucracy, and Markus Müller for the technical support. Specially, I thank Julius Leite and Sanjoy Baruah for being in my defense committee, their valuable input, and assessment of my work.

Last, but not least, I want to thank my girlfriend Maria Evotschko for her support (by not disturbing me while I was writing this thesis). And by the way, I thank you as well for taking your time to read this preface, but only if you at least make it to the table of contents and check the number of pages!

Raphael Guerra
Kaiserslautern, 03.05.2011

The work presented in this thesis has been partially supported by the European Commission under the European IST-FP6 project *Framework for Real-time Embedded Systems based on COntRacts* (FRESCOR, FP6/2005/IST/5-034026), and the European ICT-FP7 project *Adaptivity and Control of Resources in Embedded Systems* (ACTORS, FP7/2007/ICT/216586).

Publications

I have authored or co-authored the following publications:

Journal papers

1. Enrico Bini, G. Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Vanessa Romero, Claudio Scordino, Karl-Erik Arzen, Resource Management on Multi-core Systems: the ACTORS approach, IEEE Micro, December 2010.

2. Raphael Guerra, Gerhard Fohler, A Gravitational Task Model with Arbitrary Anchor Points for Target Sensitive Real-Time Applications, Real-Time Systems Journal, September 2009.

Conference and refereed workshop papers

3. Raphael Guerra, Gerhard Fohler, Handling overload of target sensitive real-time applications for increased system utility and improved resource usage, SAC'11: Proceedings of 26th ACM Symposium on Applied Computing, ACM Publisher, Taiwan, March 2011.

4. Raphael Guerra, Gerhard Fohler, An optimum generalized equilibrium solution for the gravitational task model, RTNS'10: Proceedings of the 18^{th} International Conference on Real Time and Network Systems, IRIT lab, Toulouse, France, November 2010.

5. Raphael Guerra, Stefan Schorr, Gerhard Fohler, Adaptive Resource Management for Mobile Terminals - The ACTORS approach, WARM'10: Proceedings of the 1^{st} Workshop on Adaptive Resource Management, Stockholm, Sweden, April 2010.

6. Raphael Guerra, Gerhard Fohler, On-line Scheduling Algorithm for the Gravitational Task Model, ECRTS'09 - Proceedings of the 21^{st} Euromicro Conference on Real-Time Systems, IEEE Computer Society, Dublin, Ireland, July 2009.

7. Raphael Guerra, Gerhard Fohler, Video decoding for reduced jitter and improved resource usage, WRT'09: XI Workshop de Tempo Real, Recife, PE, Brazil, May 2009.

8. Raphael Guerra, Gerhard Fohler, A Gravitational Task Model for Target Sensitive Real-Time Applications (best paper award), ECRTS08 - Proceedings of the 20^{th} Euromicro Conference on Real-Time Systems, IEEE Computer Society, vol. 1, pag. 309 – 317, Prague, Czech Republic, July 2008.

9. Raphael Guerra, J.C.B. Leite, Gerhard Fohler, Attaining soft real-time constraint and energy-efficiency in web servers, ACM Symposium on Applied Computing, pag. 2085-2089, Fortaleza, Brazil, March 2008.

10. Raphael Guerra, Gerhard Fohler, A gravitational task model for target sensitive real-time applications, RTSS'07: Work-in-Progress Proceedings of the 28^{th} IEEE Real-Time Systems Symposium, IEEE Computer Society, pag. 49-52, Tucson, AZ, USA, December 2007.

11. Raphael Guerra, Luciano Bertini, J.C.B. Leite, Managing Energy and Quality of Service in Heterogeneous Server Clusters, CLEI'06 - Proceedings of the 32^nd Latin-American Conference on Informatics, Santiago, Chile, August 2006.

12. Raphael Guerra, Luciano Bertini, J.C.B. Leite, Improving Response Time and Energy Efficiency in Server Clusters, WRT'06: VIII Workshop de Tempo Real, Curitiba, PR, Brazil, May 2006.

Contents

List of Figures

List of Tables

Chapter I

Introduction

For many years real-time task models have focused the timing constraints on execution windows defined by earliest start times and deadlines to guarantee correct system behavior. However, the utility of some application may vary among scenarios which yield correct behavior, and maximizing this utility ensures that resources are used in the best way. For example, target sensitive applications have a target point where execution results in maximized utility, and an execution window which ensures correct system behavior. Execution around this point and within the execution window is allowed, albeit at lower utility. The intensity of the utility decay depends on the importance of the application. Examples of such applications include multimedia and control; multimedia application are very popular nowadays and control applications are present in every automated system.

In this thesis, we present a novel real-time task model which provides for easy abstractions to express the timing constraints of target sensitive real-time applications: the gravitational task model. This model uses an analogy with pendulum systems to ease the understanding of its temporal abstractions by application developers, who not always have deep knowledge of real-time scheduling theory. In other words, this model fills in the gap between application requirements and theoretical abstractions used in task models. We also present a few scheduling algorithms designed for the gravitational task model which fulfill the requirements for on-line adaptivity. These algorithms also exploit the analogy with pendulum system, which provides for clarity. Finally, we present a few applications enhanced with the gravitational task model, and highlight the improvement in the results.

In this chapter, we briefly describe the background of real-time scheduling and adaptive real-time systems in sections I.1 and I.2, respectively. Then, we informally describe the problem that we deal with in this thesis in section I.3, followed by the related work in this area in section I.4, and a summary of our main contributions to the state-of-the-art of real-time scheduling theory in section I.5. Finally, we present the outline of the rest of the thesis in section I.6.

I.1 Real-time scheduling background

Basic concepts

Real-time (RT) applications have timing constraints additionally to their logical output for functionally correct system behavior. Control applications, which act on a physical plant to make it behave according to a prescribed reference, are the source of the RT constraints. These systems include safety critical, mission critical, and business critical control applications, which have stringent timing constraints — the violation of a single deadline can jeopardize the entire system behavior, and even cause catastrophic consequences [Bouyssounouse 05].

Consider the air-bag system of a car, which works as a cushion that prevents the body of the passengers from hitting the dashboard upon a crash. The cushion effect of the air-bag lasts for a very short period of time. Therefore, early or late activation of the air-bag does not prevent the body of the passengers from hitting the dashboard. In order to guarantee the safety of the passengers, the air-bag activation must happen within a precise interval of time.

Some people erroneously associate RT systems with *fast* systems. However, the key concept in a RT system is *predictability*; the design of these systems consider worst-case assumptions, and execute in predictable kernel mechanisms to meet the required performance in all anticipated scenarios. Achieving predictability results in system design different of "conventional" computing systems. For example, approaches like Direct Memory Access (DMA), cache, and memory management provide for improved average performance, but insert unpredictability in the system behavior.

Real-Time Systems (RTS) commonly consist of multiple tasks that concurrently compete for system resources. Therefore, RT applications demand special scheduling algorithms which account for timing constraints, the so-called *RT scheduling algorithms*. These algorithms define a set of rules to schedule the execution of tasks at system runtime in order to preserve timing constraint. For example, some tasks have earliest start time and deadline constraints, which define the *execution window* of a task. This window is the interval of time where inputs become available and the logical output results in correct system behavior. The scheduler must guarantee that all tasks entirely execute within their respective execution windows.

Task model

The task model is an abstraction that allows applications to express their temporal attributes to the system scheduler. As a result, schedulers can provide for feasibility tests, i.e. analyses to guarantee the timing constraints of the applications, and runtime execution. These tests are typically provided for a particular scheduling scheme and task model. Many applications have temporal characteristics that can be represented by simple generic timing constraints. The most common temporal attributes are classified by the regularity of activations, computation times, and execution windows.

Regularity of activations. Task can be defined as *periodic*, *sporadic*, or *aperiodic*. "Periodic tasks consist of an infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate. The activation time of the first periodic instance is called phase, and the time elapsed between two consecutive activations is the period"[Buttazzo 04]. Examples of periodic tasks include the frame display of video applications. A video consists of a sequence of pictures displayed at a certain rate which provides for the perception of a moving picture. Such applications have strictly constant inter-display time for maximum user perceived quality of video (PQV).

Sporadic tasks also consist of an infinite sequence of identical activities, but have an irregular inter-activation time which is bounded by a minimum value. Consider the tracking system of an avionic system. The tracking task is only active if there is an object on the screen, and hence, is not active all the time. Upon activation, the inter-activation time is bounded to the refresh rate of the screen.

Finally, aperiodic tasks consist of an infinite sequence of identical activities with an irregular inter-activation time which is not bounded to a minimum value. Aperiodic tasks are usually generated by external events and activated by interrupts, e.g. coming from sensory acquisition boards, as in an alarm system [Bouyssounouse 05].

Computation times. The design of an RTS must assume worst case scenarios, and hence, tasks must inform their worst case execution time (WCET)) to the system for feasibility analysis. Therefore, a lot of effort has been spent in analyses that determine the worst case execution time at design time [Li 95, Puschner 00].

Some RTSs, e.g. multimedia systems, have highly variable execution times where the worst case rarely happens. Designing such systems for the worst case scenario leads to extreme resource under-utilization, and hence, high production cost. Therefore, some task models allow tasks also to express stochastic properties of the execution times, e.g. average, probabilistic distribution, etc. Such models clearly demand new forms of feasibility analysis which provide stochastic guarantees of the temporal constraints.

Execution windows. The execution window is an interval of time bounded by an earliest start time and a deadline. Completely executing a task within this window results in correct system behavior; a schedule is *feasible* if all tasks execute within their execution windows. The execution window is a powerful abstraction that provides for the expression of many temporal attributes. For example, the execution window can enforce precedence constraint among tasks — task A must execute before task B, and thus, the execution window of A is before the execution window of B. Defining execution windows to enforce precedence constraints at runtime demands special analysis, as proposed in [Orozco 97], in order to provide for feasibility. The execution window bounds the variations in inter-completion times, i.e. the length of the interval of time between the completion of two consecutive instances of a task. These variations are also known as *completion jitter*. Other kinds of jitter, e.g. start time jitter (variations in the length of the time interval between the start of execution of two consecutive instances of a task), can be also bounded using execution window constraints — jitters in general are a very

important temporal constraint in control loop applications for example.

System model

RTSs are classified according to a few criteria, being the most common the type of temporal constraints, and of system architecture. Regarding the temporal constraints, the classification of RTSs include the distinction between Hard Real-Time Systems (HRTS) and Soft Real-Time Systems (SRTS).

Hard. HRTSs must preserve temporal and functional feasibility in every possible scenario, hence focusing on the worst case. The schedulability or feasibility test must guarantee that a set of tasks with parameters describing their temporal behavior will meet their temporal constraints if executed at run-time according to the rules of the scheduling algorithm. "The result of such a test is typically a "yes" or "no" answer indicating whether feasibility will be met in the worst case or not. These schemes and tests demand precise assumptions about task properties, which hold for the entire system lifetime. Examples of HRTSs include digital controllers for aircraft, nuclear power plants, missiles, and high-performance production lines."[Bouyssounouse 05]

Soft. SRTSs manages Quality of Service (QoS). Often, task parameters and constraints are known only partially at system design time, or can change during system runtime, or the necessary worst case assumptions may be too costly to apply. In these cases, standard feasibility based on "yes" or "no" types of answers are not appropriate. Rather, SRTSs need a quantitative analysis of the performance of the scheduling algorithm for the given task set, i.e. the QoS. This performance analysis is strictly application dependent and not always trivial to derive from the schedule, which increases the challenge for QoS management. Examples of SRTSs include multimedia computing, video games, virtual reality, and robotic systems.

Regarding the system architecture, there are 2 relevant classifications: uniprocessor vs. multiprocessor architectures, and time-triggered vs. event triggered architectures.

Uniprocessor vs. multiprocessor. In uniprocessor systems tasks compete for a single Central Processing Unit (CPU), while multiprocessor systems offer more CPUs. This distinction has a big impact on the scheduling algorithm. Scheduling for uniprocessor systems is simpler, and the intensive research focus on these systems in the past decades have led to a mature development. As physical limitations hinder the further performance increase necessary to meet the market demand [Fuller 11], computer architectures exploit the deployment of multiple CPUs to further increase the system performance. However, multiprocessor systems lack scheduling algorithms that are capable of fully exploiting the resources' potential, hence limiting the performance.

Time-triggered vs. event-triggered. "In a time-triggered architecture all application tasks, distributed functions and system behavior are defined at design time,

and dependent only on the progression of time. Such an architecture provides for the predictability that HRTS platforms for dependable and safety-critical applications demand. The system behavior becomes easier to certificate, upgrade and modernize at manageable effort."[1] However, these features come at the expense of reduced scheduling flexibility and adaptivity, which may lead to resource under-utilization depending on the applications' properties. Event-triggered architecture, as opposed to time-triggered architectures, provide for system reaction on event occurrence, hence providing for flexibility and adaptivity. However, the incapability of defining those events at design time reduces the systems' predictability.

Scheduling paradigms

Most scheduling algorithms have been developed around one of three basic schemes: table driven, fixed priority, or dynamic priority. Depending on whether a majority of scheduling issues are resolved before or during system run-time, they are classified as *off-line*, or *on-line*.

Off-line scheduling. These scheduling algorithms construct a table determining which task executes at which point in time at runtime [Ramamritham 95]. Thus, feasibility is proven constructively, i.e., in the table, and the runtime rules are very simple, i.e., table lookup. Therefore, off-line methods are capable of managing distributed applications with complex constraints, such as precedence, jitter, and end-to-end deadlines with very low run-time overhead. On the other hand, the a priori knowledge about all system activities and events may be hard or impossible to obtain. Moreover, its rigidity enables deterministic behavior, but limits flexibility drastically. This approach is usually associated with time-triggered architectures.

On-line scheduling. These methods offer the necessary flexibility to handle partially or non-specified activities. A large number of schemes have been described in the literature. These scheduling methods can efficiently reclaim any spare time coming from early completions and allow handling overload situations according to actual workload conditions. On-line schedulers assign priority to tasks, and schedule the task with the highest priority to execute from the set of ready tasks at runtime. Therefore, the main difference among such schedulers is the priority assignment rule. On-line scheduling algorithms for RTSs can be further distinguished into two main classes according to the priority assignment rule: fixed-priority and dynamic-priority algorithms.

- Fixed priority scheduling [Liu 73] assigns static priorities to tasks, i.e the priority of a task does not change during system runtime. The static priority assignment demands very simple support from the system kernel, and hence imposes low overhead. Fixed priority scheduling is at the heart of commercial operating systems such as VxWorks or OSE.

[1] from: http://www.ttagroup.org/technology/tta.htm

- Dynamic priority scheduling [Liu 73] assigns priority to tasks based on the runtime state of the system. Thus priorities do not follow a fixed pattern, changing dynamically. Dynamic priority scheduling provides for improved resource utilization with computationally tractable feasibility analysis and low runtime overhead. However, priority assignment rules to handle arbitrary constraints may impose high runtime overhead, and the system kernel must provide appropriate data structures to support priority handling operations.

Scheduling algorithms can be further classified into *preemptive* and *non-preemptive*. Preemptive scheduling algorithms may interrupt a running task in order to assign the processor to another active task according to the scheduling policy. Preemptions provide for improved feasibility testing, but come at the expense of more complex system kernel support. They also impose computational overhead that may compromise the system performance if not limited. Non-preemptive scheduling algorithms guarantees that all tasks run to completion without interruption. These algorithms demand simple kernel support, and have lower overhead, as the scheduler takes over only upon task completion. However, the system may not react to events at any arbitrary point in time, hence limiting flexibility. Furthermore, non-preemptive scheduling algorithms tend to under-utilize system resources when compared to similar preemptive approaches.

Each of the basic scheduling paradigms is selected for a set of specific advantages. However, concurrent systems running applications with heterogeneous properties often demand advantages of different schemes. Therefore, researchers have proposed combined approaches in order to exploit the benefits of more than one scheme, thus meeting the individual requirements of concurrent applications. For example, the network of avionic systems relays mixed-criticality messages, such as system control data and multimedia streams from entertainment applications. In this context, TTEthernet [Kopetz 05] combines the concepts of time and event triggered architectures to expand classical Ethernet with services to meet time-critical, deterministic or safety-relevant conditions.

In hierarchical scheduling [Regehr 01], a meta algorithm arbitrates between a set of diverse scheduling algorithms. Thus, individual set of applications may arbitrate for a specific scheduling algorithm capable of meeting their requirements. Furthermore, the amount of the CPU portion can be set individually for each scheduler and application. A meta algorithm for the general case leads to resource under-utilization though. Therefore, many work in the literature explore specialized meta algorithms to combine particular sets of scheduling algorithms.

Slot shifting [Fohler 95] combines off-line and on-line scheduling in order to integrate event-driven workload into an existing static table-driven schedule. Off-line, the scheduling table is analyzed for leeway of tasks, and the amount and location of unused resources, which are represented via intervals and associated spare capacities. At runtime, these resources are used to schedule additional tasks according to Earliest Deadline First (EDF), while maintaining the feasibility of the off-line guaranteed tasks. Slot shifting can handle complex task constraints for off-line tasks, and include firm and soft aperiodic tasks at runtime, as well as off-line and on-line handling of sporadic tasks. Slot Shifting is a reactive method in the sense that it does not address how to produce

a static schedule for successful integration with a specific event-driven workload.

I.2 Adaptive real-time systems

Some applications have highly variable and data dependent resource needs over time. In this context, a system that can adapt its execution to changing environment is more important than to apply pessimistic techniques based on worst case scenarios. Moreover, many real-time systems have limited resource availability. Resource provisioning may be limited e.g. due to production cost and physical space limitations, as for Wireless Sensor Networks (WSN), and mobile multimedia devices.

In adaptive RTSs, applications are capable of reacting to variable resource availability in order to deliver functionally correct behavior. This way system design may provision resources for average cases, under which the system utility is maximized — utility is an abstraction which relates to how well the goal of tasks are satisfied. Under situations of scarce resource availability, the system adequately reacts to minimize the utility decrease. The type of adaptation and the impact on the system utility may vary among applications.

Standard scheduling techniques for HRTS behave poorly under overload, potentially not completing any task in time and sharply decreasing the system utility. Adaptive and flexible scheduling techniques must handle applications with partially known properties at design time, relaxed constraints that cannot be expressed solely by execution windows, and co-existence of activities with diverse properties and demands. Scheduling algorithms may overcome overload situations selecting tasks for abortion, hence allowing the remaining tasks to meet their constraints [Baruah 97]. Abortion must account for functionally correct system behavior and minimum utility decrease. In a distributed system, overload may occur only on some nodes, while others have processing reserves. Then, overload handling may include task migration. Migration imposes time overhead, and hence, must account for stability of adaptation in order to effectively result in minimum utility decrease.

Another technique is changing task parameters at runtime based on resource availability. For example, in the *elastic task model* [Buttazzo 02] the system may change the period of tasks in order to adjust the resource demand to the resource availability under overload. This technique follows the reasoning of a spring system, which can be stretched or compressed to a certain extent in presence of an external resistance. Applications that can tolerate some adjustment for a limited time benefit from this technique. The elastic task model also considers the effect of these changes on other tasks. *Tardiness based scheduling algorithms* relax the timing constraints by allowing tasks to miss deadlines at a certain degree and with bounded values [Bernat 99, Stoyenko 91].

Some applications support multiple types of adaptation, with varied impact on the system utility. This imposes extra challenge to the adaptation technique, which must explore a multidimensional space in order to provide for optimum adaptations.

Clearly expressing timing constraints solely with worst case execution times, activation times and execution windows restrict the system capability to adequately provide for adaptivity. Those attributes alone limit the expression of flexibility in temporal

constraints. As a result, scheduling algorithms must find a workaround in order to handle the need for adaptivity. In the next section, we describe a new concept of timing constraint which accounts for both system utility and flexibility for adaptations.

I.3 Informal description of the problem

Some RT applications have tight optimum execution windows for maximum system utility, but accept some flexibility to enlarge those windows for the sake of feasibility. This flexibility comes at the expense of a utility decay, though. For example, tasks of *target sensitive applications* should preferably execute at a specific target point within its execution window, called *target point*, but can execute around this point, albeit at lower utility — the intensity of the utility decay depend on the importance of the application to the system. In this case, the optimum execution window is extremely tight. Ideally, all executions would be scheduled directly at the respective target points, but it might not be feasible due to overlapping executions. Under this condition, the execution of tasks must be scheduled so that no timing constraints are violated and the accrued system utility is maximized. More important applications are less tolerant to deviations from the target point.

The target point may not relate to the whole execution of a task, which cannot run to completion instantaneously in a real computing system. Therefore, the utility of an application to the system relates to the instants the application communicates with the system, i.e. performs input/output operations. The system is aware of applications' output, and not of their internal computational state. Furthermore, the internal state of a program may only change due to some input, in which case the output may also change.

Execution windows alone establish a direct relation between feasibility and utility, hence failing to express that the system utility may vary among feasible scenarios. Classic task models can use tight execution windows to enforce execution at the target point, but this approach compromises feasibility, hence leading to resource underutilization. Large execution windows, on the other hand, do not provide for maximum system utility, even though they are able to guarantee correct system behavior. The ideal approach is to provide some flexibility in the time constraints depending on the system load and applications' importance. However, the task model must provide for expressiveness of such constraints, and scheduling algorithms must take those constraints into account in order to increase utility accrual.

Multimedia applications have target sensitive constraints, for example. In high quality media processing, frame displays are periodic at target points. Time variation in frame display degrades the PQV, and this degradation may vary among frames. Since frame buffering is limited in these applications [Isovic 03], frames have to be displayed right after decoding. Therefore, the utility of the decoding task to the system relates to the decoding completion and subsequent frame display (the output). Periodicity is enforced by tight start times and deadlines of the decoding tasks; if the decoding takes too long and cannot be completed by its deadline, the whole frame is dropped. The PQV may be higher if the frame is displayed with a small delay in these cases, rather than not at

all, but only when this delay is small and its utility clear — frame delay may affect the timeliness of subsequent frames.

Another example is control, where sampling and actuation are ideally executed at their target points for optimum output [Marti 02]. Shifting them a little bit for the sake of feasibility is acceptable provided the system response remains within bounded limits. Thus, the utility of the control task to the system depends on both input and output operations. As can be seen, the sensitivity to deviations from the target points may vary among applications and even among jobs within the same application. Chapter II brings a detailed description of these applications, and a few other examples.

Besides the necessary expressiveness, task models must also provide simple abstractions for easy understanding by applications developers. Moreover, on-line adaptivity requires scheduling algorithms with low overhead, and the compromise among tasks with overlapping executions involves reordering the execution of jobs, shifting them and possibly aborting some executions. One of the difficulties is to express whether two not very important tasks are more important than a very important task. Scheduling must also account for the execution of future jobs before shifting executions, and a complete knowledge of jobs that will execute in the system and the exact temporal constraints may be hard, if not impossible (e.g. aperiodic tasks). Finally, reordering the execution sequence of jobs is a combinatorial problem. Therefore, scheduling is non-trivial, and the need for a simple, yet effective, solution imposes an extra challenge.

I.4 Related work

Time Utility Function (TUF) scheduling as presented in [Chen 96, Wang 04, Li 04] and earliness/tardiness schedulers [Bülbül 07] go beyond the execution window notion to express tasks' temporal constraints. In TUF schedulers, tasks aggregate a given amount of utility to the system as a function of when they execute; the goal of the scheduler is to maximize system utility. A study on several types of time utility functions is presented in [Jensen 92], but no scheduling algorithm. In [Prasad 03], an utility function based approach to express the importance of tasks in order to assign resources to them is proposed. It covers multiple resources a task needs, and investigates how utility functions could be used to solve the resource allocation problem. This work also contains no scheduling algorithm.

A best-effort algorithm for resource allocation in computing systems is proposed in [Locke 86]. However, scheduling decisions are based only on the utility that tasks accrue to the system at the current point in time and not on the shape of their utility functions over time. As a result, potential increased utility accrual due to delayed execution of jobs cannot be exploited. A TUF scheduler assuming any kind of utility function is proposed in [Chen 96]. It uses a heuristic to find an ordering on average close to the optimum in $O(n^3)$, under the constraint that all tasks are within the same busy period. This constraint severely limits the application of this method in a real case. The authors of this work assume the busy periods in a work-conserving schedule in order to circumvent this limitations, but this approach limits the utility accrual potential.

In [Wang 04], the authors propose a TUF scheduling algorithm with $O(n^2)$ assuming

only non-increasing TUFs. The result is used in Ethernet packet scheduling to define the ordering. As the utility of a packet decreases upon delayed arrival, packets are sent as early as possible after being ordered. This work is extended in [Wu 05] and [Li 06] to support variable cost functions and mutual exclusion of resources, respectively. In [Wu 06] an energy-aware TUF scheduler is proposed, but the focus is to satisfy statistical performance requirements. In [Liu 10], the authors also propose a TUF scheduling algorithm assuming non-increasing TUFs as in [Wang 04], but they consider that tasks may execute for a shorter time than the provided worst case execution time. This scheduling algorithm then uses the concept of opportunistic cost to order the execution of jobs based on their utilities and probability density function of the execution times.

Earliness/tardiness schedulers seek the minimization of the overall penalty of the system due to tasks that complete too early or too late with respect to the deadline. However, the penalty function is linear with the distance from the deadline and with no bounds, which hinders the use of this scheduling paradigm by target sensitive RT applications. The computational time and quality of such schedulers depend mainly on the execution sequence of jobs, and the traditional approach is to formulate the scheduling as an optimization problem. A small survey on earliness/tardiness schedulers presented in [Bülbül 07] shows that many are based on branch and bound and most of them can only schedule tasks in a busy period.

In classic task models, target sensitivity can be enforced by tightening the offsets and deadlines, at the expense of decreasing the maximum feasible utilization. For example, a method to decrease the variation in the completion time of tasks under EDF is proposed in [Baruah 99]. This problem is solved by decreasing the deadlines of tasks based on their importance, provided that with the new deadline assignment the schedule is still feasible. Although this work states the possibility of adjusting offsets as well, this approach does not take them into account. Therefore, target points within the execution windows cannot be addressed. The same rationale of tightening the offsets and deadlines has been used for other scheduling algorithms, but the same drawbacks remain.

The elastic scheduling of [Buttazzo 02] presents a method that improves the acceptance of tasks under overload situations in EDF. In order to accept a task, it increases the period of tasks to reduce the utilization of the system based on a spring system analogy. This analogy provides for intuition of the solution, but the algorithm has quadratic complexity with the number of tasks. This work does not consider the target sensitivity when readjusting periods, which also implies in enlarging the execution windows. The same approach is used for adaptive resource management, where the compression algorithm shrinks the resource allocation of applications under overload.

I.5 Contributions of this work

A task model for target sensitive real-time applications

In this thesis, we present a novel task model, called *gravitational task model*, which is based on a physical system: the pendulum. A pendulum is an object (or bob) that is attached to a pivot point and can swing freely. The rest position of a single object in a

pendulum is the projection of the pivot point (central point). An object placed in this position will not swing, and the system is said to be in an *equilibrium state*. If there is more than one object, they will push each other aside and their rest position will depend on their relations between weight and size. The heavier an object is, the stronger the gravity drags it to the central point.

We draw an analogy between real time systems and pendulum systems, with instances of tasks (jobs) as objects in a pendulum and the target points as the pivot point. A point within the execution of a job, called *anchor point*, relates the job to the target point. The anchor point can be an operation of input, output, or both. The equilibrium state of the physical problem is, then, equivalent to the best compromise among the jobs' interests. This model assumes non-preemptive tasks running on single processor systems.

The gravitational model allows jobs to express their target points and to provide some resistance to shift their execution away from this point based on their importance. Therefore, jobs can actually execute at their target points when the system is idle without the need of tight deadlines. Moreover, the intuition from the analogy with pendulums eases the understanding of the abstractions of temporal attributes in the model. Most application developers lack deep knowledge of RT scheduling theory, and hence, this intuition is very important.

Computing the trade-off among competing jobs

We propose a solution for the trade-off among competing jobs based on the equilibrium of pendulums. The equilibrium state in the physical problem depends on the weights of the bobs, and can be seen as the best compromise among the jobs' importances: This equivalence makes it possible to use the equilibrium equation from physics to schedule jobs aiming at maximizing the utility accrual of the system for a given execution sequence of jobs. We present both an approximation and an optimum solution to the best compromise among jobs with conflicting interests. We call these solutions *pendulum equilibrium* and *generic equilibrium*, respectively.

The proposed equilibriums account for the target sensitivity of tasks to take scheduling decision, hence providing for increased utility accrual without restricting the feasibility of the schedule with tight execution windows. During underload the equilibrium schedules all tasks at their target points, and during overload tasks can push each other around to find the best compromise of their interests for maximum utility accrual. Both equilibriums have linear complexity, which provides for on-line deployment. The pendulum equilibrium implicitly assumes that all tasks have elliptical utility functions, yet it is a good approximation even when this assumption does not hold, as evidenced in our experiments. Therefore, there is no need for tasks to express their exact utility functions; only importance as abstraction is enough. Besides, in many cases applications cannot provide the exact utility functions, e.g. multimedia and control (see sections II.1 and II.5).

Scheduling target sensitive real-time tasks

Not all jobs in a schedule compete with one another for their target points. Therefore, in order to apply the equilibrium, the scheduler must know which jobs have conflicting target points. Identifying those jobs is not trivial, and requires equilibrium recomputations that result in increased complexity. As a result, reducing the complexity to recompute the equilibrium is crucial.

Furthermore, the equilibrium may not shift the execution of tasks freely, as it may cause deadline misses and compromise the system utility accrual upon arrival of future jobs. On the other hand, accounting for the arrival of all future jobs is computationally expensive. Therefore, the scheduler must limit the amount of future jobs that the scheduler takes into account at any point in time, and guarantee all timing constraints, yet accounting for increased utility accrual.

We present an algorithm based on bob pendulums to identify which jobs in the schedule compete for their target points — those jobs comprise a job chain. This algorithm inserts jobs one by one in the schedule and, similar to inserting bobs one by one in a pendulum system, recomputes the equilibrium upon collision detection. These equilibrium recomputations lead to a complexity $O(N^2)$. Then, we present a method to avoid recomputing the whole equilibrium upon collision, given the scheduler uses the pendulum equilibrium. This method consists of storing intermediate values for jobs already in equilibrium in order save computational steps in case the scheduler must recompute the equilibrium. This new equilibrium recomputation method has linear complexity, and does not impact on the output of the equilibrium calculation. We also propose an *equilibrium window* for the computation of the equilibrium, and guarantee that the equilibrium within this window never affects the schedulability of jobs outside this window.

The scheduling algorithms that we propose account for increased utility accrual, have low complexity (up to linear complexity), hence providing for on-line deployment, and do not over-compromise the feasibility. Moreover, they exploit the pendulum intuition, hence providing for clarity.

Reordering the execution sequence of jobs for increased utility accrual

The execution sequence of jobs also impacts on the trade-off; the equilibrium itself is limited to finding local optimums. Ordering the execution of non-preemptive tasks is a Non-deterministic Polynomial-time hard (NP-hard) problem [Chen 96] — check all possible permutations. Therefore, an optimum solution has high overhead, and hence, is unfeasible for on-line scheduling algorithms. Heuristic solutions compromise between overhead, acceptance ratio, and utility accrual; affording higher overhead tends to provide for better scheduling decisions. Traditional execution windows based scheduling algorithms aim only at feasibility, e.g. EDF, Rate Monotonic (RM), etc.

In this work, we propose a few heuristics to reorder the execution sequence of jobs for increased utility accrual. Those heuristics are inspired by the physics of liquids: higher density liquids come to rest closer to the bottom and, following the same rationale, tasks with higher utility density should execute closer to their target points. The utility

density of a job is the ratio of its importance to its WCET. Those heuristics differ in how timing constraints are taken into account, which affects the acceptance ratio.

Our ordering heuristics account for increased utility accrual, have low complexity (up to linear complexity), and do not over-compromise the feasibility. Moreover, these heuristics exploit the pendulum intuition, hence providing for clarity.

I.6 Outline of the thesis

The rest of this thesis is organized as follows:

Chapter II — in this chapter, entitled *Examples of target sensitive real-time aplications*, we describe 5 examples of target sensitive real-time applications: multimedia, resource management, body area networks, dataflow systems, and control systems. The descriptions that we provide cover the technological relevance of the respective application, a brief description of its functionality, and the associated temporal constraints. Those applications serve as evidence of the need for task models, as well as scheduling algorithms, which are able to take into account the temporal requirements of target sensitive real-time applications.

Chapter III — in this chapter, entitled *Gravitational Task Model: a bob pendulum based approach to express trade-offs*, we present the gravitational task model, which provides simple abstractions for the expression of timing constraints of target sensitive RT applications based on the analogy between the target sensitive task scheduling and bob pendulum systems.

Chapter IV — in this chapter, entitled *Scheduling target sensitive real-time tasks*, we present a few scheduling algorithms for the gravitational task model. We cover the trade-off among tasks with conflicting targets using the equilibrium state concept for increased system utility, and the execution sequence of jobs in the schedule. Reordering this sequence has a significant impact on the equilibrium of jobs, and dominates the complexity of the problem — the optimum solution is NP-hard [Chen 96].

Chapter V — in this chapter, entitled *Reducing the complexity of periodic tasks' scheduling*, we propose an on-line scheduling algorithm for periodic tasks which is based on EDF. This algorithm uses an equilibrium window to limit the number of future jobs that the scheduler takes into account, while guaranteeing all timing constraints, and accounting for increased utility accrual. Moreover, this algorithm achieves higher acceptance ratio due to the reordering of the execution sequence of jobs based on EDF, yet not compromising the utility accrual.

Chapter VI — in this chapter, entitled *Scheduling tasks for increased system utility under scarce resource availability*, we address the scheduling of target sensitive tasks under scarce resource availability. Traditional approaches to handle overload are shifting and aborting the execution of jobs. We propose a trade-off between

shifting and aborting the execution of jobs based on the gravitational task model for increased system utility accrual. The abortion of a job frees resources which other jobs may use to decrease their deviation from their target point, and yield an extra amount of utility.

Chapter VII — in this chapter, entitled *Applications of the gravitational task model*, we present 3 applications enhanced with the gravitational task model: a video stream adaptation strategy for increased PQV under scarce resource availability, an algorithm for adaptive resource management under scarce resource availability, and an opportunistic packet scheduling strategy for improved reliability and reduced energy consumption in Body Area Networks (BAN). The final results evidence the benefits of the gravitational task model.

Chapter VIII — in this chapter, entitled *Discussion*, we cover preemption, multicore scheduling, and early completion times. These topics were not the focus of the research, but we consider that they should not be ignored. Therefore, we discuss how can the gravitational task model address them, what are the difficulties, and potential research directions.

Chapter IX — this chapter summarizes the contributions of this thesis and brings the concluding remarks.

Chapter II

Examples of target sensitive real-time applications

In this chapter, we describe 5 examples of target sensitive real-time applications: multimedia, resource management, body area networks, dataflow systems, and control systems. The descriptions that we provide cover the technological relevance of the respective application, a brief description of its functionality, and the associated temporal constraints. Those applications serve as evidence of the need for task models, as well as scheduling algorithms, which are able to take into account the temporal requirements of target sensitive real-time applications. Later in this thesis, we will show how the work proposed here improves the functionality of some of these applications.

II.1 Multimedia applications

II.1.1 Overview

The importance of multimedia systems in our daily activities has increased in the past years. Multimedia is present in the ever growing game industry, movie industry, surveillance systems, and publicity. Moreover, thanks to the increasing performance of embedded devices, multimedia takes over a whole new market that spans mobile devices, medical care, and entertainment and control systems of automobile and avionic industry.

Multimedia computing presents challenges from the perspectives of both hardware and software. For example, multimedia standards such as MPEG-2 and MPEG-4 involve the execution of complex media processing tasks in real-time (RT). The need for RT processing of complex algorithms is further accentuated by the increasing interest in 3-D image and stereoscopic video processing. Each media in a multimedia environment requires different processes, techniques, algorithms and hardware. The complexity and variety of techniques and tools, and the high computation, storage, and input/output bandwidths associated with multimedia processing further requires features such as scalability and maximal resource utilization.

MPEG-2 is the video standard in consumer electronics for Digital Versatile Disc (DVD), and still widely used non-High Definition (HD) digital TV satellite streams. In the next sections, we briefly describe the MPEG-2 standard, its temporal requirements, and the decoder design to meet these strict temporal requirements. Finally, we discuss video playout under scarce resource availability.

II.1.2 Layer structure of MPEG-2 streams

MPEG-2 video is broken up into a hierarchy of layers (see figure II.1) to help with error handling, random search and editing, and synchronization. The top layer is the *video sequence layer*, which contains the information about the video resolution (e.g. width, height, display aspect ratio, *framerate*, and the stream Bit Rate (BR). Each sequence consists of one or more Groups Of Pictures (GOP), which consist of a header and a series of frames (or pictures), and allows random access into the sequence. GOPs may comprise any arbitrary number of frames.

A GOP starts with an *intra frames* or I frame, which are simply frames coded as still images, and ends before the next I frame in the stream. I frames contain absolute picture data and are self-contained, meaning that they require no additional information for decoding. Therefore, their data compression provides the least compression among all frame types, and they are not transmitted more frequently than necessary.

The other frames may be of types P or B. P frames or *predicted frames* achieve additional compression by referencing to data present in the most recently reconstructed I or P frame. In order words, they contain a set of instructions to convert the previous picture into the current one. This compression is also called *temporal compression*. Because P frames are not self-contained, decoding is impossible if the previous reference frame is lost. B frames or *bi-directionally predicted frames* use both forward and back-

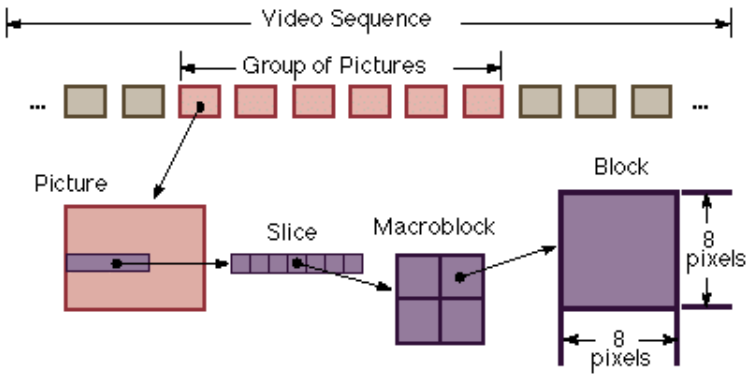


Figure II.1: *MPEG-2 stream structure.*[1]

ward prediction, i.e., decoding a B frame requires the previous I or P frame, and next I or P frame. B frames require resource-intensive compression techniques, but they also exhibit the highest compression ratio. Figure II.2a summarizes the dependency relations among frames.

Each frame consists of *slices*, which are important for error handling. If the bit stream contains an error, the decoder can skip to the start of the next slice. Having more slices in the bit stream allows better error handling, but use space that could otherwise be used to improve picture quality. Slices are further subdivided into *macroblocks*, which are subdivided into *blocks*, and finally, the smallest element of a video stream, a pixel.

II.1.3 Temporal requirements of MPEG-2 playout

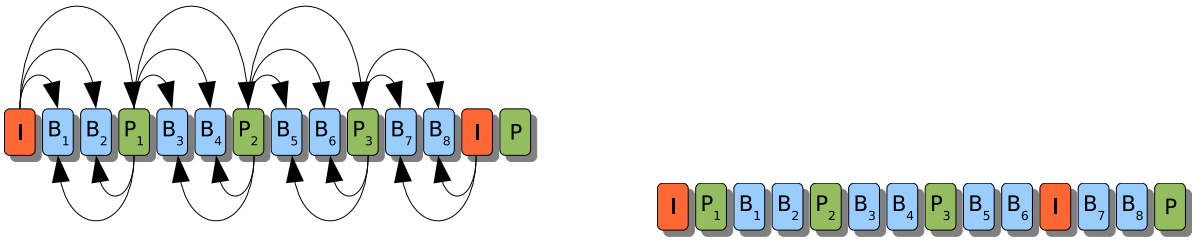
A video consists of the periodic display of a series of frames. The playout of MPEG-2 streams requires that the inter-display time of frames (or frame period) is constant for maximum perceived quality of video (PQV). Variations in the inter-display time, also called *inter-display jitter*, degrade the PQV, but this degradation is hard to quantify analytically. A video which has many differences between frames has a high temporal aspect, and hence, is more sensitive to inter-display jitter. The pattern of movements are also likely to alter the impact of the inter-display jitter on the perceived quality of video. For example, a scene with lots of objects moving in random directions is less sensitive to inter-display jitter than a scene with objects moving in a linear steady movement. On the other hand, a video with low temporal aspect is less sensitive to jitter. See [Claypool 99] for a more detailed discussion on this topic.

The decoding time of frames represent a further challenge in the design of MPEG-2 decoders. Remember that the encoding of frames uses references to other frames in order to achieve higher compression ratios by skipping the encoding of common objects among scenes. This strategy causes the decoding time to be highly dependent on the tempo-

[1] Figure downloaded from `www.fh-friedberg.de`

ral aspect of the scene, as temporal compression/decompression is a computationally expensive task.

Furthermore, as mentioned above, decoding B frames requires input data from a backward and a forward reference frame. Clearly, decoding frames in the same order they are displayed fails for B frames, as the forward reference is not yet available. The decoding sequence requires frames to be sent out of the display sequence and temporarily stored to solve decoding dependencies. Figure II.2 shows that although the display sequence is I B B P, the decoding sequence is I P B B, so that the forward reference is already available in the decoder before bi-directional decoding begins.



(a) *Displaying order and dependencies* **(b)** *Decoding order*

Figure II.2: *Frame decoding and display.*

Picture reordering requires additional buffers at the decoder, and latency in order to put the display order right again. The number of bi-directionally coded frames between I and P frames must be restricted to reduce cost and minimize latency, if latency is an issue.

II.1.4 MPEG-2 decoder model

Basic design

In its simplest form, playing out an MPEG-2 video stream requires three activities: *input*, *decoding*, and *display*. These activities are performed by three tasks, which are separated by an *input buffer*, a *decoding buffer*, and a *display buffer* (see figure II.3). Recent dedicated media processors follow this design, e.g. the DaVinci Digital Media Processors from Texas Instruments [Anderberg 07].

The *input task* directly responds to the incoming stream. It places encoded video stream in the input buffer at a certain rate, expressed in bits per second, the BR. In the simple case, the input activity is very regular, and only determined by the fixed, constant BR. In a more general case, the input may be of a more bursty character due to an irregular source, e.g. the Internet. The input task drops an incoming frame if the input buffer is full (*input buffer overflow*).

The *decoding task* extracts an encoded frame from the input buffer following the arrival order, decodes the frame, and puts the result (decoded frame) in the decoding buffer. The decoding times for frames can vary, depending on the frame bit size and the

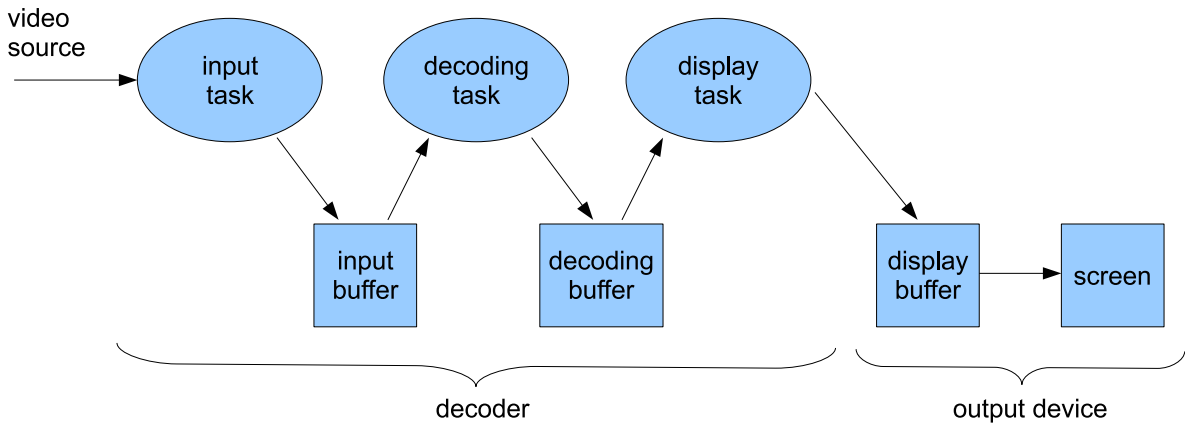


Figure II.3: *Decoding model.*

used compression technique. Usually the decoding task is asynchronous, and decodes a frame upon availability in the input buffer, and given that there is free space in the decoding buffer to store the decoded frame. If the decoding of a frame is not finished by the time this frame has to be displayed, the decoding task drops the frame, hence wasting resources.

The *display task* regularly copies a decoded frame from the decoding buffer into the display buffer of the *output screen device*. This task also resizes the decoded frame in order to match the output screen resolution. The framerate of the video dictates the regularity of the display task, which once started, must always find a frame to be displayed. If there is no frame to fetch in the decoding buffer, the display task leaves the previous frame in the display buffer (frame dropping). The output screen device refreshes the screen at a *display rate* with the content of the display buffer. The display rate may vary among devices; for example, TV sets have less diverse values than computer screens.

Latency and buffer requirements

Latency requirements. The *decoding latency* is the amount of time elapsed between the input task receives the first bit and the decoding task stores the first bit in the decoding buffer. This latency depends on the stream BR and the frame decoding time. Thus, it may vary a lot.

The *display latency* of a forward reference frame is the amount of time elapsed between the decoding task writes the first bit in the decoding buffer and the display task displays the first pixel of the first dependent frame on the screen. In the example depicted in figure II.2, P_1 is a forward reference frame, and B_1 is its first dependent frame. For all other frames, the display latency is the amount of time elapsed between the decoding task writes the first bit in the decoding buffer and the output screen device displays the first pixel on the screen.

Those latencies dictate the buffer requirements of the decoder. Next, we analyze the

buffer requirements as a function of the desired latencies.

Input buffer requirement. The input buffer serves several purposes. First, it has to compensate for the irregular data size of different encoded frames, and possible variation in the incoming BR due to an irregular source, e.g. the Internet. Second, the input buffer has to compensate for varying decoding times, which are not foreseen by the decoder. Therefore, this compensation cannot be bounded a priori.

The input buffer size IBS is essentially a design choice, and is closely related to the maximum tolerable decoding latency from which the decoding task may be able to recover without dropping frames. Once the size of the input buffer is chosen, the maximum decoding latency of a reference frame is $RDL_{max} = IBS/BR$, and the maximum decoding latency of a non-reference frame is $NRDL_{max} = RDL_{max} - FP$.

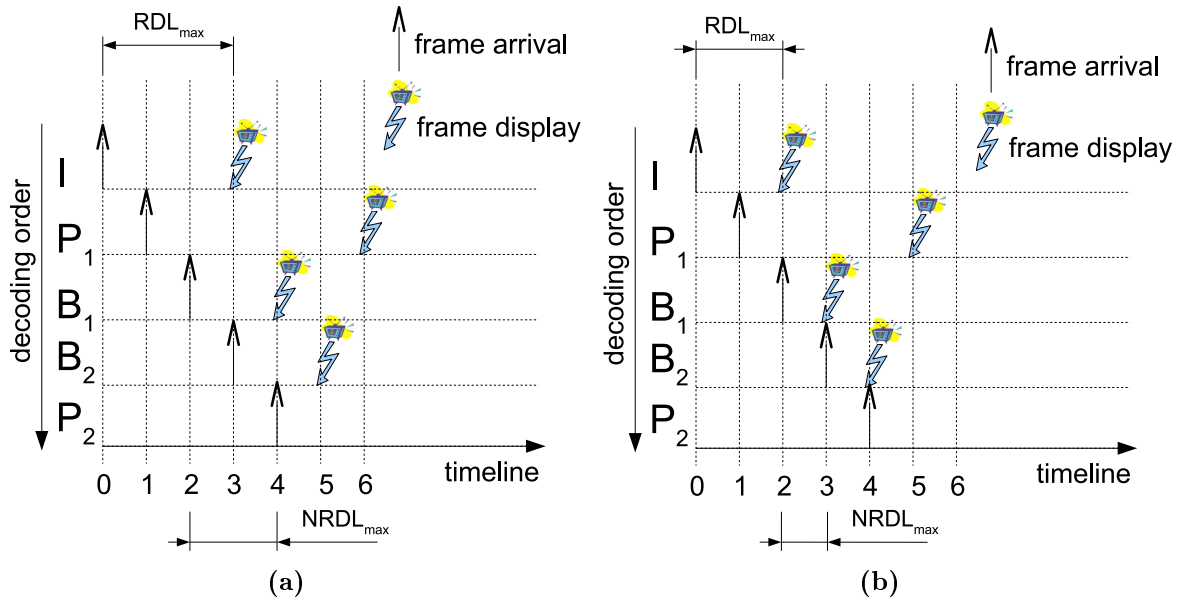


Figure II.4: *Example of decoding latencies.*

Let us analyze the example depicted in figure II.4, where we assume the frame period $FP = 1$. An $\uparrow$ symbol represents the arrival of an encoded frame in the input buffer (frames arrive in decoding order), and a $\lightning$ symbol represents the display of the frame. As can be seen in figure II.4a, for $RDL_{max} = 3 \times FP$ the maximum number of encoded frames in the input buffer is 3 (the frame being decoded plus two subsequent frames). Remember that the maximum decoding latency of P_1 is the length of the interval from its arrival time to the display of B_1, which is the first dependent frame of P_1. The size of the input buffer can be easily derived from the bit rate, framerate, and the maximum number of encoded frames in the input buffer. $NRDL_{max} = 2 \times FP$. Looking at figure II.4b, we can see that for $RDL_{max} = 2 \times FP$, the input buffer has to store at most two frames, and that $NRDL_{max} = FP$.

Decoding buffer requirement. The decoding buffer serves a dual purpose. It serves as reference buffer for the decoding task, and as input buffer for the display task. This makes decoding buffer management somewhat more complicated than input buffer management. The decoded frame may only be removed from the decoding buffer after it has been displayed, and is no longer needed as reference for the decoding of other frames. This buffer also allows the decoding task to recover if the decoding of a frame takes longer than the frame period FP [Anderberg 07]. Recovering from a long decoding latency requires that subsequent frames have short decoding latencies.

The decoding buffer size DBS is also related to the maximum decoding latency. If the stream contains at least one B frame, $DBS_{min} \geq 3 \times DFS$, where DFS is the size of a decoded frame. The decoding buffer must store at the same time the backward reference, the forward reference, and the B frame being decoded. In this case, $RDL_{max} = DBS/DFS$, and $NRDL_{max} = RDL_{max} - 2 \times FP$.

Let us analyze once again the example depicted in figure II.4. As can be seen in figure II.4a, for $RDL_{max} = 3 \times FP$ the maximum number of decoded frames in the decoding buffer is 4 (2 reference frames and 2 B frames, because B_2 may be available in the input buffer before B_1 leaves the decoding buffer). $NRDL_{max} = FP$. Looking at figure II.4b, we can see that for $RDL_{max} = 2 \times FP$, the decoding buffer has to store at most 3 frames (2 reference frames and one B frame), and that $NRDL_{max} = FP$.

Display buffer requirement. This buffer belongs to the output screen device, and hence, does not depend on the decoder design. This buffer stores only one frame with the same resolution as the screen, and the content of the buffer is the output on the screen.

II.1.5 MPEG-2 video playout under scarce resource availability

The combination of highly variable decoding times, demand for high PQV, and dependency constraints during decoding makes the decoder design very challenging. As shown in the previous section, one way to compensate for the highly variable decoding times typical of MPEG-2 streams is to introduce buffers in the video playout. This approach provides for a decoder design with processing power provision for the average case, but imposes larger latency. Moreover, buffering incurs in higher production cost and energy consumption due to the ever increasing energy leakage of new memory architectures. On-chip decoding buffer of dedicated multimedia processors provides for lower latencies, but have size limitation due to space restriction on the die [Sung 08].

One way to reduce the need for frame buffering is to reduce the maximum tolerable decoding time latency, which reduces also the resilience to long decoding times. In case of system overload, reactive quality-aware stream adaptation strategies such as frame skipping [Isovic 04] provide for reduced degradation of the PQV. Another way to reduce the need for frame buffering is to display the frame right after decoding. This approach requires procrastination of the decoding task in order to not display frames too early. Moreover, the decoder must account for the variability of the decoding times with early/late displays, as buffering is not an option.

Therefore, multimedia applications are target sensitive because of this strict no inter-display jitter requirement for maximum PQV. However, such applications tolerate some flexibility in this temporal requirements for the sake of feasibility.

II.2 Adaptive resource management

II.2.1 Basic concepts

The goal of the resource management is to guarantee availability of required resources to applications. This guarantee implies that applications are to some degree isolated from the behaviour of other applications on the same system. Adaptive resource management adjusts the resource allocations on-demand based on the runtime resource requirements of applications. This approach is particularly advantageous in systems with variable resource requirements, where providing for the worst case scenario of each application leads to resource underutilization most of the system lifetime. In such systems, resource provision for the average case can meet the current resource requirements of each application most of the time. Shall system overload occur, not all applications may obtain all required resources simultaneously. In this case, the resource manager must guarantee a minimum availability of required resources to each application, and the resource distribution among applications must provide for minimized degradation of the overall Quality of Service (QoS).

Adaptive algorithms are required so that all applications can deliver functionally correct results under overload. These algorithms trade output quality against resource requirements. If no compromises in output quality are acceptable, then the application is no candidate for adaptivity, and has to run with the full required resources. Therefore, there is no opportunity for adaptive resource management in a system without adaptive applications.

For example, anytime algorithms provide a useful result after any amount of execution time. The longer they can execute the better the result is. Imprecise computation algorithms are a subtype of this type of algorithm. These algorithms require a minimum execution time in which they produce a baseline result. If more time is available optional steps of the algorithm are executed, resulting in more precise results. Examples include iterative algorithms like Newton's root-finding algorithm [Sun 06].

Another way to achieve adaptability in applications are multi-version algorithms. The idea is straightforward: provide multiple implementations of the same functionality, e.g. a fast low-quality version and a slow high-quality version. Then, the application may switch between implementations depending on the currently available resources. An example of such scheme is the quality-aware frame skipping algorithm [Isovic 04].

Therefore, adaptive applications are target sensitive because they have a target resource requirement, but accept some flexibility for the sake of feasibility. The resource manager arbitrates the resource distribution based on the flexibility of each application for minimized system utility degradation, which in this case relates to the overall QoS.

II.2.2 Example of a resource management framework

Figure II.5 depicts the three major components of the ACTORS Framework architecture: the CAL — a dataflow-based programming language — applications, the resource manager [Rizvanovic 07] and the operating system. Applications are adaptive and the operating system is capable of enforcing resource reservations for each application. The resource manager negotiates between the applications and the underlying operating system. The main functionality of the resource manager is to distribute system resources among the applications in order to fulfill the global optimization objective. This objective is user-defined and can be, for instance, maximize the system performance or maximize the system life-time. The **interfaces** among these components use the following abstractions: *service levels*, *happiness*, *reservation setup*, and *resource usage*. Those abstraction serve the purpose of abstracting away specific internal details of the components.

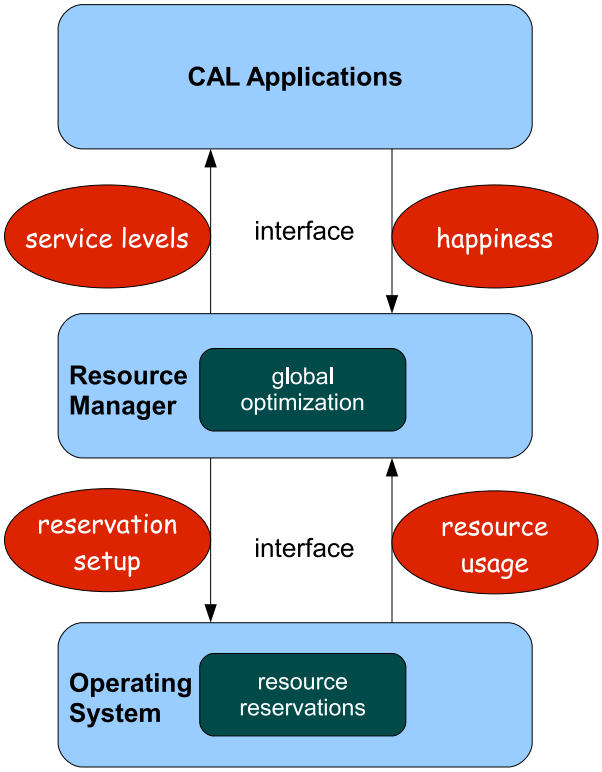


Figure II.5: *The resource manager architecture at run-time.*

CAL applications are capable of adapting their behavior to different resource availability, thus consuming different amount of resources and delivering different QoSs, called *service levels*. Applications consist of independent thread groups called *virtual processors*, each with a resource requirement for a given service level. Applications express the resource requirements using the (α, Δ) model [Mok 01]. The idea of this model is to express the timeliness constraints of the applications using only two minimal key

features. The first key feature is the bandwidth α, which is a measure for the amount of resources that are required by the application. The second key feature is the delay Δ, which specifies the longest amount of time that the application may need to wait for being assigned some resource. Applications also have an importance parameter, which the system uses to determine the distribution of resources. Furthermore, each application features a happiness parameter to express to which extent its current service level is fulfilled based on its runtime state. The service levels and the happiness are abstractions that the application uses to interface with other components.

The **operating system** features real-time schedulers capable of enforcing resource reservations, i.e. it implements the scheduling algorithm that decides which application to execute at which moment in time. For instance, it supports the standard Linux Completely Fair Scheduler, which together with Control Groups allows to assign shares of CPU time for applications. Moreover, it supports a new implementation of a real-time scheduling algorithm named SCHED_EDF, which also provides resource reservations through the hard Constant Bandwidth Server mechanism [Lipari 01]. Since the reservation mechanism is hard an application may not execute more than its assigned budget also if there are free CPU resources. To specify the individual reservations, the operating system internally uses periods and budgets, the so-called reservation setup parameters. At run-time, the operating system monitors the actual resource usage of each reservation and, if necessary, adjusts these parameters. Each reservation is associated to a virtual processor, and may span only one physical core.

The **resource manager** is the core component of the framework and possesses global knowledge about the applications and the underlying system. Each application has a resource demand associated to each service level it can deliver, and it is the goal of each application to provide the best quality of service. However, under scarce resource availability not all applications can achieve this goal simultaneously. For this reason, the resource manager contains a control based resource allocation mechanism which is in charge of distributing resources among competing applications. This mechanism distributes resources so that the global optimization goal of the system is achieved.

The interaction between the resource manager and the applications uses *service levels* and *happiness* as abstractions. The purpose of this interaction is to decide whether resource reallocations are needed based on the monitored behavior of the applications in order to improve the system performance. Applications register with the resource manager and announce their available service levels. After successful completion of the registration, the resource manager constantly monitors the happiness of the applications in order to decide whether redistribution of resources is needed or not. The resource manager tells each application at which service level they must run based on the resource allocation mechanism.

The interaction between the resource manager and the operating system, on the other hand, uses *reservation setup* and *resource usage* as abstractions. The resource manager translates the (α,Δ) values from the applications into reservation setup parameters which are used to specify the reservations in the operating system. The resource manager creates/changes reservations as instructed by the resource allocation mechanism. The operating system periodically reports the usage of the reservation back to the resource

manager, which redistributes unused resources among applications. The resource manager has, for each reservation, a dedicated feedback-loop mechanism that keeps track of long-term changes in the resource usage. This mechanism avoids frequent reallocations of resources which are caused by short-term fluctuations in the resource usage.

II.3 Body area networks

Networks of sensors around as well as inside the human body, referred to as Body Area Networks (BAN) (see figure II.6), promise to revolutionize health-care practices as they facilitate, among other things, better diagnosis, fast emergency response and personalized medication [Prabh 11]. "BANs enable early detection of clinical deterioration through real-time patient monitoring in hospitals, enhance first responders' capability to provide emergency care in large disasters through automatic electronic triage, improve the life quality of the elderly through smart environments, and enable large-scale field studies of human behavior and chronic diseases"[Ko 10]. Examples of currently applications include electrocardiogram sensors for monitoring heart activity, electromyogram sensors for monitoring muscle activity, electroencephalogram sensor for monitoring brain electrical activity, blood pressure sensors, tilt sensors for monitoring trunk position, breathing sensors for monitoring respiration, and motion sensors to discriminate the user's status and estimate the level of activity.

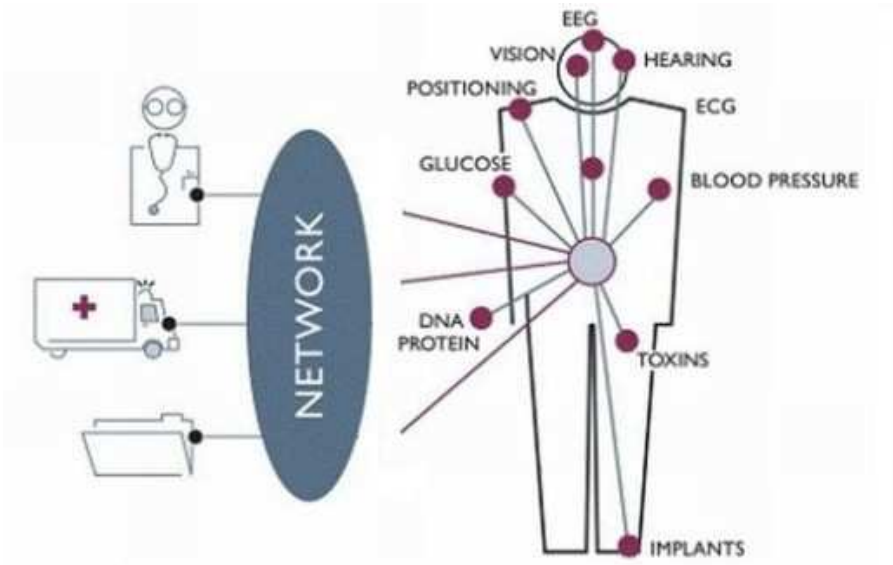


Figure II.6: *A body area network.*[2]

Although BANs have the potential to enable low-cost, personalized health-care systems, it is still unclear whether they can meet the stringent QoS requirements imposed

[2]Figure downloaded from `www.gadgetstech.co.uk`

by some applications. For example, limiting interference to neighboring BANs and keeping the specific absorption rate of Radio Frequency (RF) energy as low as possible — in the interest of protecting the human tissues — demands low transmission power. Therefore, although the distances between devices in a BAN are usually small, the wireless signal may experience severe attenuation. This attenuation causes packet losses, hence negatively impacting on the system reliability.

Furthermore, energy consumption is a serious constraint as many times nodes are deployed within the human body, where access is extremely restricted. Empirical analyses have revealed that the network communication dominates the energy consumption in such systems — the main functionally of nodes is to monitor and relay sensed data, hence demanding very low processing power [Prabh 11]. The packet loss rate has a direct effect on the energy consumption of network communication, as dropped packets must be retransmitted. Therefore, the error-proneness of the low-power wireless communication is a major challenge.

A wide range of metrics can be used to determine the network link quality. Some of the popular metrics include link reliability, which may be measured as the packet success/error rate on the link, or as the expected number of transmissions to successfully send a packet to the other end of the link. Often, the link reliability is measured in terms of the Reveived Signal Strength Indicator (RSSI), which is an indication of the radio energy in the communication channel during the transmission of a packet.

Besides the usual indeterminism of RSSI fluctuations in Wireless Sensor Networks (WSN), BANs are further influenced by factors directly related to human body presence in the environment. Human mobility changes the relative node distance, which influences path-loss, fading, and the relative node orientation. The node orientation in conjunction with the irregular antenna radiation pattern of the nodes can result in different signal strength at the same distance. Additionally, the human body severely attenuates the wireless signal due to the high absorption of RF energy, an effect called *shadowing*. The human body capability to absorb RF energy comes from the composition of body tissues, which are made up primarily of "salt water", an effective absorber of RF energy. High water-content tissues such as muscle and skin can absorb more RF energy than low water-content tissues such as fat and bone or skull [Paul Bousquet 97].

Empirical evaluations of the RSSI effect on the link reliability revealed that the magnitude of the RSSI fluctuation is usually position-dependent and often significant. In addition, the absolute RSSI values are often close to the sensitivity threshold of the radio, where even a small change in RSSI can result in a substantial difference in packet delivery performance. A detailed discussion on the effect of RSSI fluctuations in the link reliability can be found in [Prabh 11].

One way to reduce energy consumption is to send packets during periods of high RSSI, hence reducing the risk of message retransmissions. Therefore, the target sensitivity here relates to the peak RSSI values, and tight*transmission windows* — the interval of time packets can be transmitted — ensure that packet transmission exploits moments of high RSSI. However, tight transmission windows during high workload forces packets compete for the medium, which may cause buffer overflow and packet loss, thus compromising reliability. They also restrict the capability of the packet scheduler to find a feasible

schedule, i.e. meet the deadlines of all packets. Moreover, there is no deterministic relation between RSSI values and successful message transmission, which invalidates the usage of fixed transmission windows. As can be seen, packet scheduling in BANs is target sensitive and requires flexibility in its temporal constraint. The scheduler must also account for the trade-off among packets with conflicting targets.

II.4 Dataflow systems

II.4.1 Basic concepts

In dataflow systems, computations are performed in logical units denoted *actors* (see figure II.7). Actors can be seen as procedures in *procedural programming* (or *imperative programming*). Applications consist of one or more specialized actors, which exchange data over communication channels in order to reach a desired state. The communication between actors uses endpoints called *input ports* and *output ports*, and data is transferred over the channels in quanta called *tokens*. An actor operates in steps called *firings*, during which it consumes a sequence of tokens from its input ports, and produces tokens to its output ports. Firing is generally subject to constraints, such as availability of tokens. When an actor is able to fire it is said to be *enabled*. These systems are modeled as directed graphs in which *nodes* denote actors, and *arcs* denote communication channels.

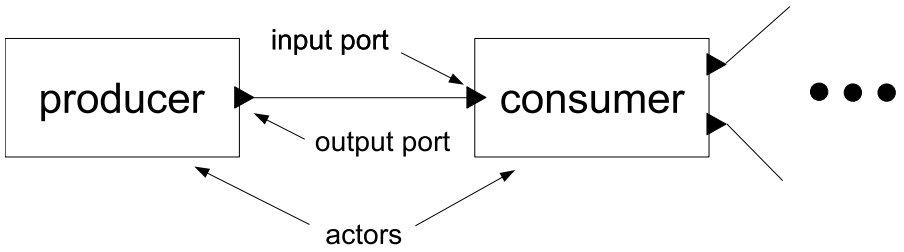


Figure II.7: *Actors of a dataflow system.*

The execution of a dataflow graph is inherently parallel with any dependence between actors explicitly specified as paths in the graph. Therefore, dataflow models offer a representation that efficiently supports parallelization, vectorization and synthesis of both hardware and software (for instance see [Lee 87, Ritz 93]). These features meet the requirements of increasingly complex execution platforms e.g. for embedded multimedia systems: *parallelization* is required to utilize multi-core architectures, *vectorization* is required to utilize the so-called SIMD (Single Instruction Multiple Data) or "multimedia" instructions, and application-specific hardware acceleration emphasizes the need of *hardware/software partitioning*. Low abstraction levels obstruct automated transformations to this end.

II.4.2 Dynamic dataflow

A fundamental property of Dynamic Dataflows (DDF) is the capability to offer a deterministic computation model, which means that the outputs that are computed by a program only depends on the inputs it has consumed; any admissible schedule of the computations produces the same result. Determinism is a result of dataflow process networks being a special case of *Kahn process networks* [Kahn 74].

A DDF actor may have several so called *firing rules*, each of which governs a firing that consumes a distinct set of tokens from the input ports. Such sets may differ in the number of tokens that are consumed and may differ further in the required values of these tokens.

DDF actors use FIFO channels, which means that the actor consumes tokens in the same order as produced by its predecessors in the dataflow graph. Token consumption uses *blocking reads*, whereas token production uses *non-blocking writes*. The use of blocking reads means that the order, in which tokens are consumed for the purpose of checking the firing rules, is crucial. A poorly choosen order could cause an actor to block although there is a satisfied firing rule. The firing rules of a DDF actor must be *sequential*, which means that an appropriate order can be determined beforehand. A further requirement is that the mapping from input tokens to output tokens be functional, that is free from side-effects.

II.4.3 Synchronous dataflow

Synchronous Dataflows (SDF) are a special case of DDFs, in which actors have fixed token rates. A particular actor thus consumes (and produces) the same number of tokens in every firing. This restriction sacrifices expressiveness, but allows several interesting properties to be determined using static (compile-time) analysis.

A model of computation, known as *Boolean dataflow*, results when extending SDFs with two types of conditional actors, *Switch* and *Select*. This is sufficient to render the dataflow language Turing complete [Buck 93]. Conditional actors presents a problem in the SDF model of computation, since the token rates are not known beforehand. We thus attain computational power at the expense of certain properties of SDFs that can no longer be decided in general; the existence of a static schedule is one such property.

However, in several other extensions of the SDF model the task of finding a static schedule remains tractable while expressiveness is improved. One such extension, known as *well-behaved dataflow* [Gao 92], restricts the use of conditional actors in particular patterns: the *conditional schema* (`if-then-else` constructs) and the *loop schema*. Another extension, *cyclo-static dataflow* [Bilsen 96], allows token rates to vary over a fixed period that is associated with each actor. Firing advances an actor's *phase* within its period and thus determines the token rates of the next firing. In yet another extension, *scenario-aware dataflow* [Thelen 06], token rates are parameterized by particular operational modes, called *scenarios*, but remain fixed within a single scenario. Scenario-aware dataflow is primarily intended for static analysis of models with dynamic behavior and includes stochastic modeling of performance metrics, such as throughput. Under certain conditions, it is possible to analyze combinations of scenarios in isolation using

techniques similar to those of SDF graphs.

II.4.4 Scheduling of dataflow graphs

Scheduling concerns the processes of assigning actors to processors and ordering their execution on each processor. Assignment can be either *static* (performed at compile-time) or *dynamic* (performed at run-time) and, given a static assignment, the ordering can be either static or dynamic. In static ordering, we can distinguish the schedule between *fully-static* schedule and *self-timed* [Ha 91]. Actors have exact starting times within the period of a fully-static schedule, whereas self-timed schedules rely on inter-processor synchronization.

The execution of a dataflow graph is usually assumed to be non-terminating, which makes it relevant to find periodic schedules that can execute indefinitely. The schedule must respect precedence constraints, and execution window constraints. In particular, it is desirable to rule out risk of deadlock, which arises when no actor is able to fire.

Executing consumers directly after their respective producers also brings some benefits to the system, as the probability to transport tokens directly using processor registers, i.e. without buffer memory, increases. Reducing amount of tokens buffered on the FIFO channels contribute to reduce buffer size. Therefore, the buffer requirement is a target sensitive property of dataflow graph scheduling. This constraint may be relaxed for the sake of feasibility, and the trade-off among several producers and consumers provides for reduced buffer size.

The SDF model allows the scheduling constraints to be verified beforehand [Lee 87], whereas they are undecidable in general for DDF graphs [Buck 93, Parks 95]. SDF graphs are usually ordered statically, which not only eliminates the run-time overhead of scheduling, but also enables static allocation of buffers and generation of efficient and compact code that can be repeated indefinitely without deadlock. DDF graphs, on the other hand, usually require dynamic reordering and leeway of actors, thus demanding efficient strategies that account for target sensitivity and feasibility.

II.5 Control applications

II.5.1 Basic concepts

The objective of a control system is to make the *system output* of a *dynamic system* behave according to a given *reference* by manipulating some *system input*. In the context of control theory, the controlled system is called *plant*. The difference between the system output and the reference is called *error*. The *control performance* relates, among others, to this error.

The general functionality of control systems can be described as follows (see figure II.8): the *sensory system* samples data from the plant, generating a *feedback* value. Then, the *controller* processes the difference between the reference value and the feedback value, i.e. the error, in order to derive the needed action (*control output*). Finally,

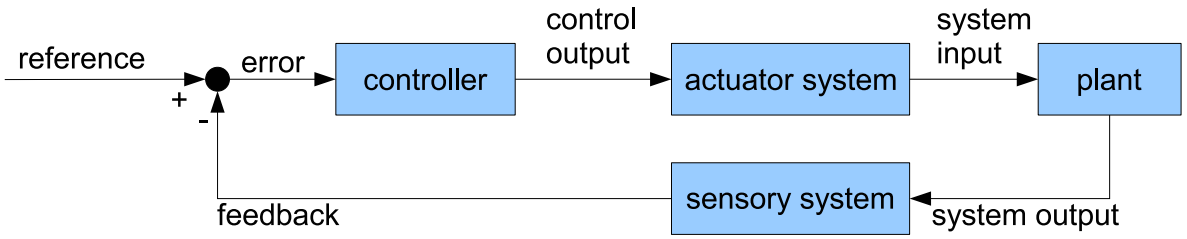


Figure II.8: *Control system.*

the *actuator system* performs the action (*system input*) on the plant. At the design stage, controllers are designed according to dynamics of the plant and control performance specified requirements.

"*Embedded control systems* are embedded systems where control activities are relevant for their correct functionality. Often, they have resource constraints in terms of computing and communication resources. Embedded control systems constitute an important subclass of embedded computing systems. So important that, for example, within automotive systems, computers commonly go under the name Electronic Control Unit (ECU). A top-level modern car contains more than 50 ECUs of varying complexity. A majority of these implement different feedback control tasks, for instance, engine control, traction control, anti-lock braking, active stability control, cruise control, and climate control."[A. Crespo 05]

Embedded control systems are often found in consumer products such as automotive systems, mobile phones, as well as in daily utilities like blenders, mixers, dish washers, wash machines, etc. Therefore, embedded computers by far outnumber desktop computers. The pervasive nature of these systems generate further constraints on physical size and power consumption. These product-level constraints give rise to resource constraints on the computing platform level, for example, limitations on computing speed and memory size. Cost also plays a fundamental role in the design of these systems, thus requiring minimization of system resources, and favoring general-purpose computing components over specially designed hardware and software solutions.

"Traditionally, control algorithms are designed assuming unlimited computing and communication resources. In embedded systems, these resources are often shared between many applications, and the environmental conditions are changing. Thus, the development of the real-time control algorithms must consider these constraints from the very beginning in the control design procedure."[A. Crespo 05]

II.5.2 Control temporal constraints

The basic timing parameters of control tasks are shown in figure II.9. Control tasks are released (e.g., inserted into the ready queue of the real-time operating system) periodically at times r_k, and $r_{k+1} - r_k = p$, where p is the period of the controller. Due to preemption from other tasks in the system, the actual start of the task may

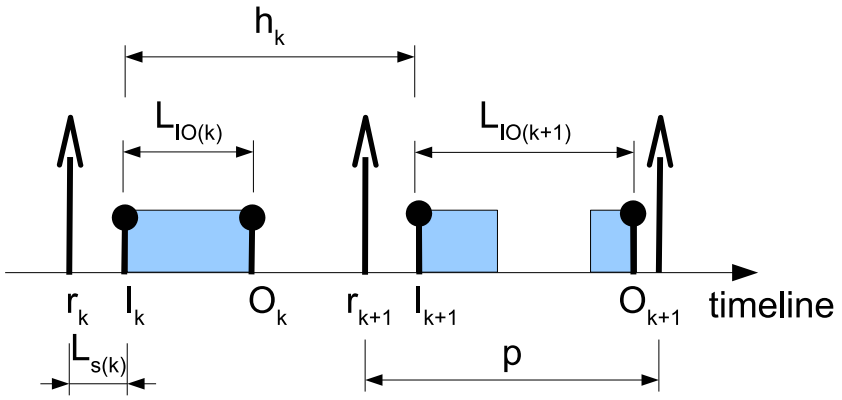


Figure II.9: *Control timing parameters*

be delayed for some time $L_{s(k)}$. This is called the *sampling latency* of the controller. A dynamic scheduling policy will introduce variations in this sampling latency across intervals. These variations are called *sampling jitter*. The maximum sampling jitter is quantified by the difference between the maximum and minimum sampling latencies in all task instances, thus $J_s^{MAX} = L_s^{MAX} - L_s^{MIN}$.

The *sampling interval latency* h_k is the interval of time between two consecutive samplings I_k, thus $h_k = I_{k+1} - I_k$. The *nominal sampling interval* is such that $h_k = p$. Jitter in the sampling latency will of course also introduce jitter in the sampling interval latency, called *sampling interval jitter*. The maximum sampling interval jitter is $J_h^{MAX} = 2 \times \left(L_s^{MAX} - L_s^{MIN}\right)$.

After some computation time and possibly further preemption from other tasks, the controller will actuate the control signal (or control output) at time O_k. The delay from the sampling to the actuation is the input-output latency $L_{IO(k)} = O_k - I_k$. Varying execution times or task scheduling preemptions will introduce variations in this interval. The maximum *input-output jitter* is quantified by the difference between the maximum and minimum input-output sampling latencies in all task instances, thus $J_{IO}^{MAX} = L_{IO}^{MAX} - L_{IO}^{MIN}$.

Basic control theory assumes all latencies to be zero, which also implies that all jitters are zero. These assumptions provide for optimum control performance. However, control tasks must run concurrently in systems with scarce resource availability, and strictly enforcing no latencies leads to resource underutilization. Moreover, computer-based control systems suffer from inherent input-output latencies $L_{IO(k)}$, since computations are not instantaneous.

II.5.3 Effects of temporal properties on control performance

The effects of jitters and latencies on the control performance are not always simple to analyze and quantify. From a control theory perspective, sampling latency and sampling jitter can be interpreted as disturbances acting on the control system. They have in general a negative effect on performance, although there are counterexamples. “As a

rule of thumb, sampling interval jitters that are smaller than ten percent of the period need no compensation. The sensitivity to sampling interval jitter is larger with systems that use slow sampling and for systems with small phase margins; for such systems a small sampling interval jitter can lead to instability."[A. Crespo 05]

The input-output latency and jitter decrease the stability margin and introduce fundamental limitation on the control performance, but can be ignored if small. Otherwise, they must be accounted for in the control design or, if possible, compensated for at runtime. The work in [Marti 01] presents a run-time jitter compensation that guarantees stability and improves the control performance. However, there are always exceptions. For example, latency can have a stabilizing effect in systems that are conditionally stable with respect to the phase shift. However, the latency must be accounted for in the controller and additional latencies caused by the implementation will only have a detrimental effect. It is also in most cases so that a shorter but varying latency is better with respect to control performance than a longer, but constant, latency, also if the latter latency is compensated for. Please refer to [A. Crespo 05] for further details on this topic.

As can be seen, the temporal properties of control system go beyond the expressibility of execution windows. Execution windows alone can limit latency and jitter, but at the expense of compromised feasibility. Moreover, traditionally RT scheduling algorithms assign priority to tasks based on deadlines, rather than on functional properties of the applications. This approach leads to a direct relation between application utility and deadlines, which is not true. For example, some control tasks may tolerate large deadlines, but at the expense of poor control performance. Thus scheduling algorithms should minimize the latency and jitter of those tasks, allowing for large values only for the sake of feasibility and provided that the overall system utility degradation is minimized. Therefore, control applications are target sensitive and accept flexibility for the sake of feasibility. Enforcing the target sensitivity leads to optimum control performance, and the flexibility can be compensated for on-demand at runtime.

II.6 Summary

In this chapter, we described 5 examples of target sensitive real-time applications in order to evidence the need for task models, as well as scheduling algorithms, which are able to take into account the temporal requirements of such applications.

Section II.1 presented a multimedia application. We showed that multimedia systems are embedded in our routine nowadays, with deployment in diverse areas such as health-care systems, games, publicity, automobile industry, etc. We also used the MPEG-2 video standard to illustrate the main activities involved in video playout, and the resource and temporal constraints involved in these activities.

In section II.2, we described the adaptive management of RT application's resources. Adaptive applications running on system which do not provide for the worst case resource requirements demand such management in order to decrease the QoS degradation shall overload occur.

In section II.3, we presented the deployment of WSNs in assisted health-care systems,

the so called BANs. BANs have very restricted resource availability, and high QoS requirements, being the network the most critical resource. This section presented the temporal requirements and resource constraints of packet scheduling in such networks, which has target sensitive RT constraints.

In section II.4, we presented the relevance of dataflow-based programming paradigms to raise the level of abstraction when writing codes tailored to multicore systems. We then described the issues involved the dataflow graph scheduling, which contains target sensitive properties.

Finally, in section II.5, we described the resource and temporal constraints of control systems, which are the original source of the temporal requirements studied in Real-Time Systems (RTS). We also showed that these temporal requirements are not as strict as assumed by many task models and scheduling algorithms, accepting some flexibility for the sake of feasibility and increased resource utilization. Therefore, also control can benefit from task models and scheduling algorithms designed for target sensitive RT applications.

Chapter III

Gravitational Task Model: a bob pendulum based approach to express trade-offs

In this chapter, we introduce a simple gravity pendulum (or bob pendulum) system as a visualization model for trade-offs among target sensitive real-time (RT) applications. Analogies with well-known systems are helpful to fill in the gap between theory and practice. For instance, the so-called nature algorithms use key elements of physical processes to form the basis of an optimization algorithm [Carnahan 01]. Examples include the knapsack problem, traveling salesman problem, ant colony optimization, and simulated annealing.

We introduce, then, the gravitational task model, which provides simple abstractions for the expression of timing constraints of target sensitive RT applications based on the analogy between the target sensitive task scheduling and bob pendulum systems. We consider jobs as objects in a pendulum system, and the target points as the central point. Then, the equilibrium state of the physical problem is equivalent to the best compromise among jobs with conflicting targets. Although the gravitational task model demands a slight change in the description of the pendulum system, the final intuition remains.

We start with a detailed description of the problem, and the basic idea of our solution in section III.1. Section III.2 covers our terminology and assumptions, section III.3 describes some basic properties of target sensitive task scheduling, and section III.4 describes the bob pendulum system. Section III.5 presents the analogy between target sensitive task scheduling and bob pendulums, and section III.6 describes the equilibrium state computation. Finally, section III.7 brings our concluding remarks.

III.1 Introduction

Tasks of target sensitive RT applications have a target point in time for maximum utility, and an *execution window* for correct system behavior. Moreover, these tasks have an importance which accounts for the utility decay as a function of the moment of execution. Classic RT task models, which are based on execution windows alone, implicitly assume that correct system behavior implies in maximized system utility. Hence, they fail to express that the utility of the system may vary among scenarios which yield correct behavior. In these models, tasks of target sensitive RT applications can tighten the execution windows to enforce the target point for maximum utility. However, this approach compromises the task set feasibility, thus leading to resource underutilization. In a concurrent system several tasks may compete for their target points, and the best compromise accounts for the importance of the tasks.

RT task models must allow target sensitive RT applications to express their timing constraints, yet providing simple abstractions for ease of understanding and development. Classic RT task models demand complex scheduling algorithms to address the requirements of target sensitive RT applications. Time Utility Function (TUF) based RT task models lack of low complexity scheduling algorithms which are able to fully exploit the expressiveness power of TUFs. Furthermore, there is no systematic method to accurately define the utility function of applications. For example, we are able to say that deviation from the target point negatively affects the perceived quality of video in multimedia applications, and that this impact is more intense in scenes with a lot of motion. Nonetheless, to the best of our knowledge the exact relation between deviation and perceived quality of video is unknown.

Besides the expressiveness in the task model, scheduling algorithms must be able to use these extra information to take scheduling decisions. The trade-off among several competing tasks with different importances is not trivial. Those trade-offs can be modeled as optimization problems, which consist of a goal function to be maximized/minimized, and a set of constraints on the variables that comprise the goal function. However, optimization theory based models and solutions are difficult for non-specialists, hence creating a gap between problem (practice) and solution (theory).

In this chapter, we propose an analogy between bob pendulum systems and adaptive RT systems. This analogy provides a visualization model for trade-offs, and makes the solution intuitive. The bob pendulum consists of an object that is attached to a pivot point, and can swing freely. The rest position of a single object in a pendulum is the central point. An object placed in this position will not swing, and the system is in an *equilibrium state*. If there is more than one object, they will push each other aside, and their rest position will depend on their relations between weight and size. The heavier an object is, the stronger is the force that drags it to the central point.

We introduce, then, the gravitational task model, which provides simple abstractions for the expression of timing constraints of target sensitive RT applications based on the analogy between target sensitive task scheduling and bob pendulum systems. We consider jobs as objects in a pendulum and the target points as the central point. Then, the equilibrium state of the physical problem is equivalent to the best compromise

among jobs with conflicting targets. Although the gravitational task model demands a slight change in the description of the pendulum system, the final intuition remains. We present the terminology, assumptions and abstractions of this model.

Analogies with well-known systems are helpful to fill in the gap between theory and practice. For instance, the so-called nature algorithms use key elements of physical processes to form the basis of an optimization algorithm [Carnahan 01]. Below are a few examples:

Knapsack problem [Martello 90] — Given a set of items, each with a weight and a value, the problem is to determine the amount of each item to include in a collection so that the total weight is less than a given limit, and the total value is as large as possible. The knapsack problem derives its name from the problem faced by someone who must fill a fixed-size knapsack with the most useful items. Knapsack related problems are encountered in numerous industrial domains such as transportation, logistics, cutting and packing, telecommunication, reliability, advertisement, investment, budget allocation, and production management.

Traveling salesman problem [Applegate 07] — Given a list of cities and their pairwise distances, the problem is to find the shortest possible tour that visits each city exactly once. There are many variations of the problem, e.g. the cities are visited by car, the goal is to save gas, and the cost of a path depends on the direction taken (going uphill consumes more gas than going downhill). The traveling salesman problem has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips.

Simulated annealing [Laarhoven 87] — "Simulated annealing is a single-objective optimization technique inspired by the natural process of annealing solids. The physical process of annealing is the cooling of a metal sufficiently slowly so that it adopts a low-energy, crystalline state. When the temperature of the metal is high, the particles within the metal are able to move around, changing the structure of the metal, freely. As the temperature decreases, the movements of the particles are increasingly limited to only those configurations with lower energy than the previous state. Simulated annealing draws inspiration from the physical process, in a computational model of the physical system"[Smith 06]. Simulated annealing has been used in various combinatorial optimization problems, and has been particularly successful in circuit design problems [Laarhoven 87].

Ant colony optimization [Dorigo 05] — Real ants are capable of finding the shortest path from a food source to the nest, and of adapting to changes in the environment — for example finding a new shortest path once the old one is no longer feasible due to a new obstacle. Ants deposit a certain amount of hormone called pheromone while walking, and those which choose a faster path rapidly reconstitute the interrupted pheromone trail. Because each ant probabilistically prefers to follow a direction rich in pheromone rather than a poorer one, very soon all the ants will choose the shorter path. This behavior serves as inspiration for a whole family of probabilistic techniques for solving computational problems which

can be reduced to finding good paths through graphs, e.g. routing vehicles, protein folding, power electronic circuit design, connection-oriented network routing, image processing, etc.

Like in the examples above, the intuition from the analogy with a pendulum system serves well to represent trade-offs in adaptive systems. This analogy makes the abstractions used in the gravitational task model and the trade-off among tasks with conflicting target points intuitive, and extends the traditional execution window based timeliness criteria.

The gravitational task model allows jobs to express their target points, and to provide some resistance to shift their execution away from this point based on their importance. Therefore, jobs can actually execute at their target points when the system is idle without the need of tight deadlines. Moreover, we present an approximation to the best compromise among jobs with conflicting target points based on the equilibrium of pendulums. This approximation has linear complexity, which represents a significant improvement compared to currently known TUF schedulers. We, then, derive a method to compute the best compromise, yet keeping the pendulum intuition.

III.2 Terminology and assumptions

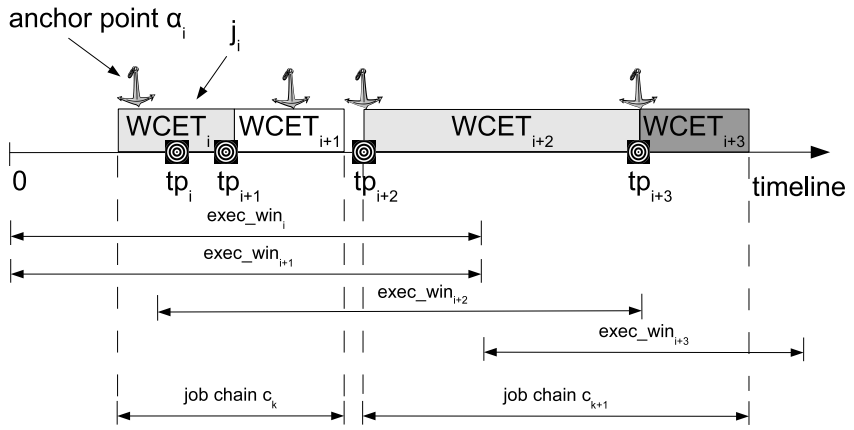


Figure III.1: *Job parameters in the gravitational task model.*

In the gravitational task model, we define the real-time requirements of tasks at job level; those jobs may be instances of recurring tasks or not. Let j_i be the i^{th} job to execute in the schedule (see figure III.1). This job has start time est_i, relative deadline rel_dl_i (hence, absolute deadline $dl_i = est_i + rel\_dl_i$), worst case execution time $WCET_i$, absolute target point tp_i, and importance imp_i. The importance is proportional to the utility decay, i.e. the need to execute as close as possible to the target point. Since the entire execution of a job cannot be performed at a single point in time, each job expresses a point α_i within its execution, called *anchor point*, which relates the job to its target

point. The value of α_i is the fraction of $WCET_i$ executed before the anchor point, and ranges from 0 to 1 — 0 corresponds to the beginning of execution, and 1 corresponds to the end of execution. Finally, jobs have an utility function g_i as a function of the deviation x_i from the target point (more details in section III.6).

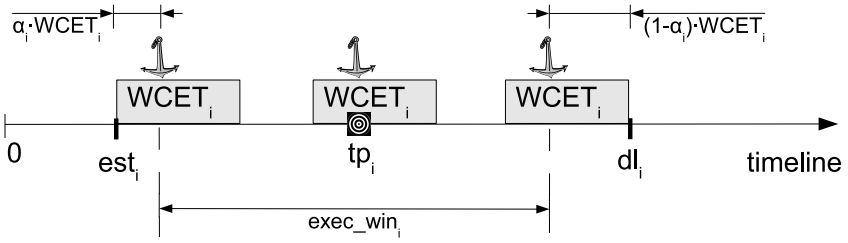


Figure III.2: *The execution window of a job.*

The distance d_i between the anchor points of two consecutive jobs (j_i and j_{i+1}) in a busy period is $(1-\alpha_i) \times WCET_i + \alpha_{i+i} \times WCET_{i+1}$. The execution window $exec\_win_i$ of j_i is the interval where the anchor point can be placed without violating the timing constraints, i.e. $exec\_win_i = [est_i + \alpha_i \times WCET_i, dl_i - (1-\alpha_i) \times WCET_i]$ (see figure III.2, which depicts 3 possible positions for a job j_i within its execution window). The length of this interval is $|exec\_win_i| = rel\_dl_i - WCET_i$. The anchor point cannot lie outside the execution window, at the extremes of the execution window the utility is 0, the flexibility of a job to deviate to the left or to the right side of the target point is the same, and placing the anchor point on the target point results in maximum utility accrual. Finally, in the gravitational task model, the k^{th} busy period is called *job chain* c_k for the sake of intuition resulted from the analogy with pendulum systems.

III.3 Scheduling non-preemptive target sensitive real-time tasks

Non-preemptive target sensitive real-time tasks accrue maximum utility to the system when executing at their target points. If the execution of 2 or more tasks overlap when they execute simultaneously at their target points, tasks must deviate away from their targets; the best compromise maximizes the utility accrual. Scheduling the execution of those tasks involves 2 steps: shifting them, and reordering their execution sequence. Shifting the execution of jobs with conflicting target points limits the schedule to a local optimum — best compromise for this particular execution sequence of jobs. Reordering the execution sequence has impact on the amount of jobs that compete with one another for their target points, and relates to the global optimum. The utility of a schedule depends on the importances, execution times, and target points of the tasks.

III.4 Inspiration from bob pendulum system

A pendulum is an object attached to a pivot point that can swing freely. A basic example is the simple gravity pendulum or bob pendulum. As depicted in figure III.3, it consists of a bob at the end of a massless string, which, when given an initial push, will swing back and forth along the *swinging range* under the influence of gravity over its central (lowest) point in a circular trajectory. Placed at the lowest point, the bob will come to rest there (*rest position*).

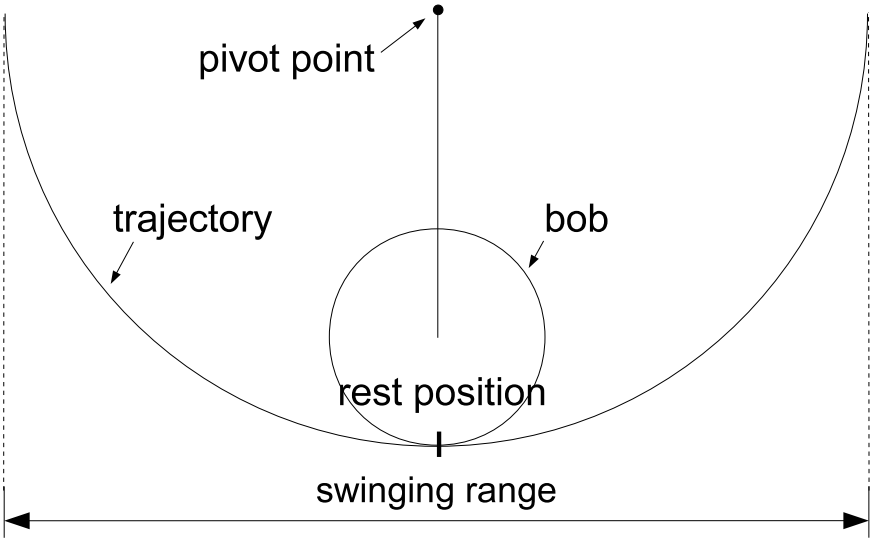


Figure III.3: *Bob pendulum.*

If the bob pendulum contains more than one bob, they cannot be all at the same time in the lowest part, and hence, will push each other aside to find a new rest position — the *equilibrium state*. The pendulum system depicted in figure III.4 consists of N bobs with radius r_i, weight $\vec{W}_i$, hanging by massless strings of length $R - r_i$ with a trajectory of radius R. Two adjacent bobs are an angle θ_i apart from each other. φ_i is the angle between the string attached to bob i and $\vec{W}_i$, and represents the position of the bob. For now, we assume that the trajectory of all bobs in a pendulum are the same.

The *equilibrium condition* is that the sum of all torques in the system is equal to zero and the distance between the centers of two consecutive bobs is the sum of their radii. These conditions assure that the system is neither spinning nor translating. How far from the central point each bob comes to rest in the equilibrium state depends on their weights and sizes. In a pendulum system no translation is possible, which reduces the equilibrium condition to ensure that the sum of all torques is zero. The torque of a bob is the component of its weight perpendicular to the string ($\vec{F}_i$) times the length of the string ($R - r_i$). The angle θ_i between two given consecutive bobs is constant and can be calculated as a function of the radii of the bobs and lengths of the strings using the law of cosines, as shown in equation III.1. The first equation of equation

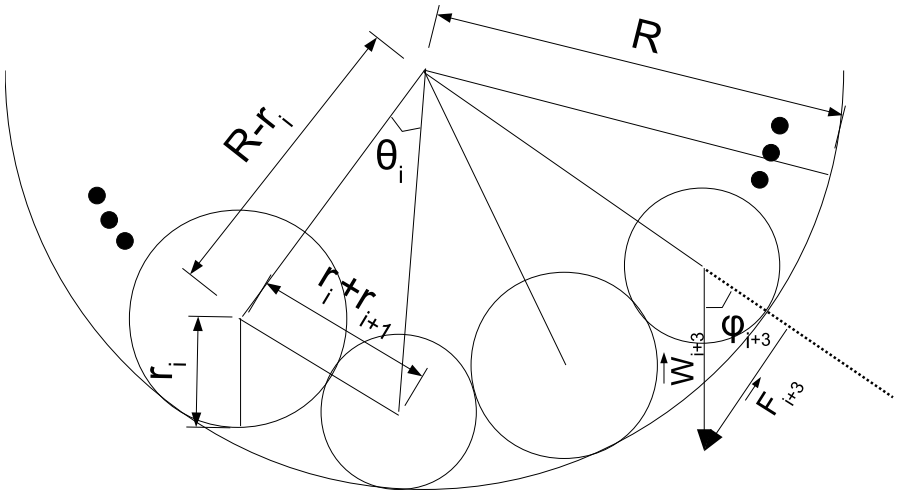


Figure III.4: *Several bob pendulums.*

system III.2 expresses the distance constraint between bobs, the second equation the torque constraint. The solution of this equation system is given by equation III.3, where φ_1 is the position of the first bob in the equilibrium state. The position of all other bobs can then be calculated using the first equation of equation system III.2. The complexity to calculate the equilibrium is linear with the number of bobs.

$$cos(\theta_i) = \frac{(R - r_i)^2 + (R - r_{i+1})^2 - (r_i + r_{i+1})^2}{2\,(R - r_i) \times (R - r_{i+1})} \tag{III.1}$$

$$\begin{cases} \varphi_1 - \varphi_i = \sum_{j=2}^{i} \theta_{j-1}, & i = 2..N \\ \sum_{i=1}^{N} |\vec{W}_i| \times sin\,(\varphi_i) \times (R - r_i) = 0 \end{cases} \tag{III.2}$$

$$tg(\varphi_1) = \frac{\sum_{i=1}^{N} |\vec{P}_i|\,(R - r_i)\,sin\left(\sum_{j=2}^{i} \theta_{j-1}\right)}{\sum_{i=1}^{N} |\vec{P}_i|\,(R - r_i)\,cos\left(\sum_{j=2}^{i} \theta_{j-1}\right)} \tag{III.3}$$

In nature, objects tend to change in such a way that their total energy is minimized [Winterbone 96]. A more general definition of equilibrium that applies to conservative systems is [Goldstein 02]: "A system is in mechanical equilibrium if its position in configuration space is a point at which the gradient with respect to the generalized coordinates of the potential energy is zero". In the pendulum system described above there are only 2 coordinates (horizontal and vertical), and the trajectory of a bob in this space describes a circular function. Zeroing out the gradient of the potential energy in a pendulum system with multiple bobs guarantees only a local optimum, which is valid for a given permutation of the bobs. The absolute minimization of the potential energy of the system involves also reordering the bobs. Liquid systems are good to give an idea of which permutation of bobs leads to a global optimum: liquids with higher density rest closer to the bottom. Likewise, higher density bobs tend to rest closer to the bottom of the trajectory in the global optimum configuration as well. However, unlike in a bob

pendulum system, liquids can go through each other without extra energy cost to reach the rest position for global energy minimization.

III.5 Analogy between pendulum systems and target sensitive real-time applications

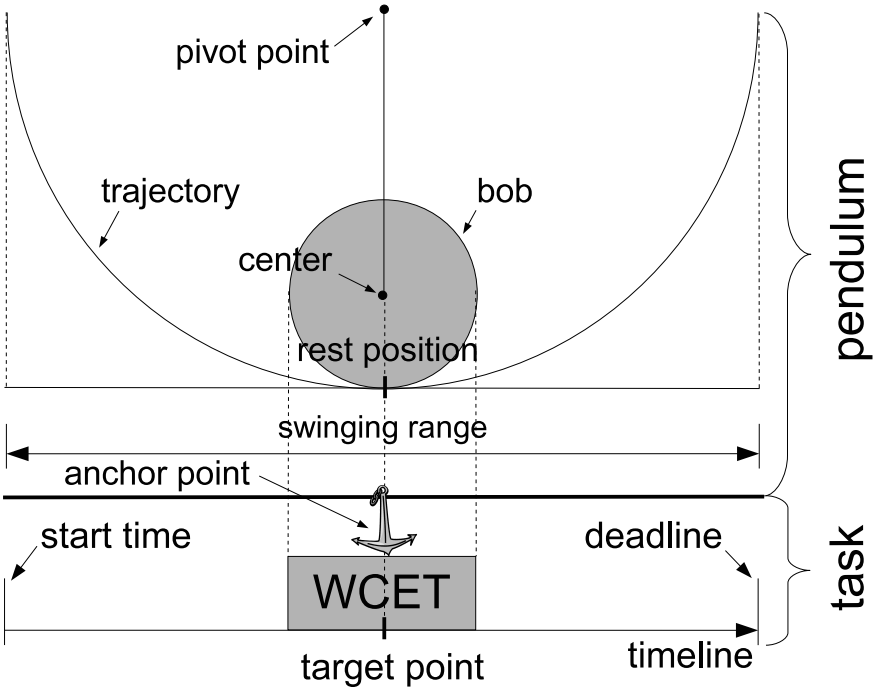


Figure III.5: *Analogy between bob pendulum and RT task set.*

Drawing the analogy, we can think of a bob as a job whose execution time is equivalent to the size of the bob. A job may execute at its target point in the absence of other jobs in the system with the same target point. The target point is equivalent, thus, to the central (lowest) point of a pendulum trajectory, the anchor point to the center of mass of the bob (center of the bob), and the swinging range to the execution window of the job. The importance of a job, which represents its resistance to shift away from its target point when interacting with other jobs, can be seen as the weight of the bob. The heavier a bob is, the closer to the bottom it will come to rest. Finally, the job utility as a function of its deviation from the target point is similar to the potential energy of a bob as a function of its deviation from the central point. As the equilibrium is the state that minimizes the potential energy of the pendulum, the best compromise of the jobs' interests maximizes the accrued utility of the system. This analogy is depicted in figure III.5, and summarized in table III.1.

pendulum	task set
bob	job
weight	importance
swinging range	execution window
central point	target point
bob's center	anchor point
potential energy function	utility function
equilibrium state	best compromise

Table III.1: *Analogy between bob pendulum and task set*

There is still an issue with the analogy that prevents a direct mapping of the pendulum to the task model. As can be seen in figure III.6, representing jobs by the projection of bobs over a straight line tangent to the swinging trajectory (equivalent to the timeline), overlapping executions occur. We overcome this by changing the first constraint of equation system III.2: instead of assuming the angle θ_i between two consecutive bobs as constant, we consider that the distance d_i between the projection of their centers over the tangent line is constant (see figure III.7). Each center is seen as a massive particle that concentrates the whole weight of the bob.

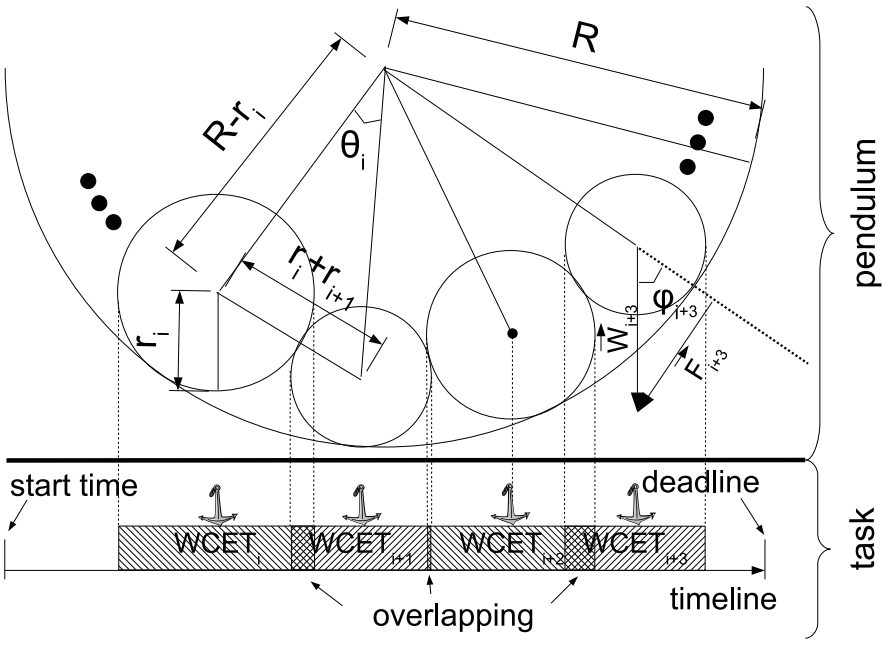


Figure III.6: *Overlap issue.*

In this final analogy summarized in table III.2, we consider the anchor point α_i of a job j_i as a particle in a pendulum. We define the value of α_i as the fraction of $WCET_i$

executed before the anchor point, which ranges between 0 and 1, where 0 corresponds to the beginning of $WCET_i$ and 1 the end of $WCET_i$. Thus, the distance d_i between two consecutive anchor points is $(1 - \alpha_i) \times WCET_i + \alpha_{i+1} \times WCET_{i+1}$.

R_i is the length of the string that hangs job j_i, and is equal to half of the swinging range, thus half of the execution window. The anchor point of j_i can be placed anywhere within this range without violating the earliest start time and the deadline constraints. Therefore $R_i = (|exec\_win_i|)/2$. The target point tp_i is the projection of the pivot point P_i, which is the lowest point of the trajectory of a particle (the central point). Figure III.2 depicts the relation between the execution window, anchor point, and target point.

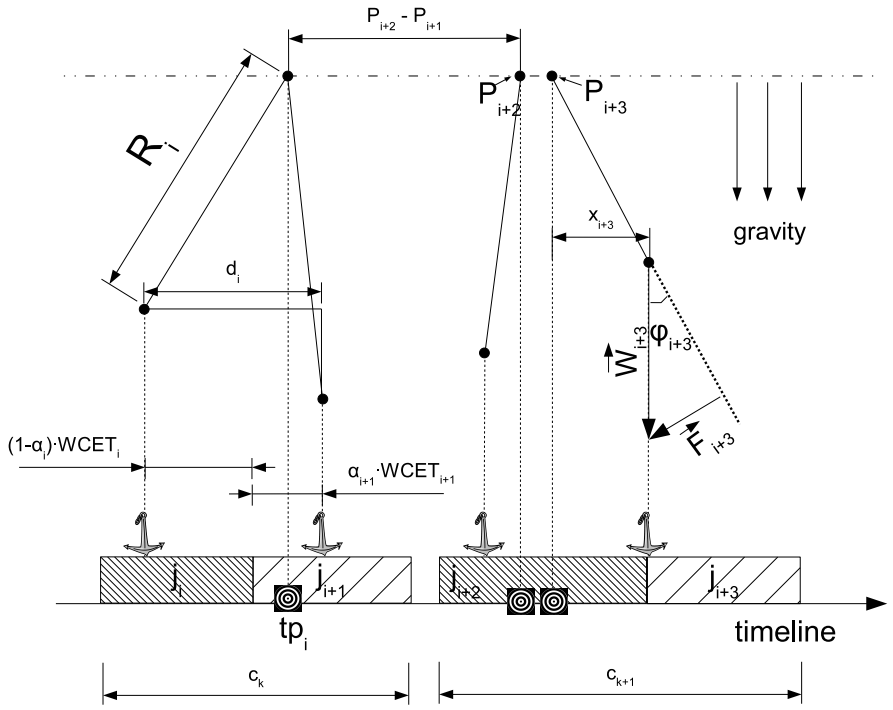


Figure III.7: *Analogy between particle pendulum and jobs with anchor points.*

Let us now define the relationship between weight of a particle and importance of a job. In a particle pendulum, two particles with the same weight hanging by strings of different lengths are not pushed aside in the same proportion with respect to their respective swinging ranges. The particle hanging by the longer string creates a bigger torque, hence moving closer to the target point. However, in a real-time system environment, two jobs with the same importance and different execution windows should be pushed aside in the same proportion with respect to the size of their execution windows. We achieve this proportional deviation by normalizing the importance of a job to the length of its string, hence obtaining $W_i = 2 \times imp_i/(|exec\_win_i|)$.

pendulum	task set
W_i	$2 \times imp_i/(\lvert exec\_win_i \rvert)$
R_i	$(\lvert exec\_win_i \rvert)/2$
P_i	tp_i
d_i	$(1-\alpha_i) \times WCET_i + \alpha_{i+1} \times WCET_{i+1}$

Table III.2: *Mapping jobs into particle pendulum parameters.*

III.6 The equilibrium state

In this section, we describe how to compute the equilibrium of pendulum in order to obtain the best compromise among competing jobs. Section III.6.1 presents a method to approximate the best compromise based on the particle pendulum equilibrium, and section III.6.2 presents a method for the best compromise that we call *generic equilibrium*.

III.6.1 Particle pendulum equilibrium

Equilibrium equation

In this section, we present the mathematical formulation of the equilibrium of the particle pendulum, and propose an approximation with linear complexity. First, let us assume that all particles hang by the same pivot point, and then, extend the solution to cover particles hanging by different pivot points.

The equilibrium is the state that minimizes the potential energy of the system, and is formulated as in the following Non-Linear Optimization Problem (NLOP) in the case of the particle pendulum (recall that x_i represents the deviation of a particle from its target point in the horizontal axis (see figure III.7)):

$$\begin{array}{lll} \min: & \sum_{i=1}^{N} -W_i \times \sqrt{R_i^2 - x_i^2} & \\ \text{s.t}: & P_{i+1} + x_{i+1} - (P_i + x_i) \geq d_i & \forall i = 1..N-1 \\ & |x_i| \leq R_i & \forall i = 1..N \end{array} \tag{III.4}$$

Solving this NLOP gives the deviation of each particle from its central point that minimizes the potential energy of the system, subject to the constraints that each particle i must be at least a distance d_i from the subsequent particle, and that the deviation must not be larger than the length of its string. Solving this problem through mathematical optimization methods for constrained functions is computationally expensive.

We can compute an approximation of the equilibrium by changing the distance constraint of the bob pendulum in equation system III.2 to the distance constraint of the particle pendulum, obtaining equation system III.5.

$$\begin{cases} x_{i+1} - x_i = d_i, & i = 1..N-1 \\ \sum_{i=1}^{N} |\vec{F_i}| \times R_i = 0 \end{cases} \tag{III.5}$$

After a few algebraic steps with the first equation, we achieve the result in III.6.

$$\begin{aligned} x_i &= x_{i+1} - d_i \\ &= x_{i+2} - d_i - d_{i+1} \\ &\dots \\ &= x_N - \sum_{j=i}^{N-1} d_j \end{aligned} \tag{III.6}$$

Finally, replacing equation III.6 in the second equation of the equation system III.5, we achieve the result in equation III.7. As we can see, the complexity to compute this equilibrium state is linear. Replacing the particle pendulum variable from this equation as described in table III.2 we find an approximation to the best compromise among the jobs' interests, which maximizes the accrued utility of the system.

$$\begin{aligned} \sum_{i=1}^{N} \left(|\vec{F_i}| \times R_i \right) &= 0 \\ \sum_{i=1}^{N} \left(|\vec{W_i}| \times sin(\varphi_i) \times R_i \right) &= 0 \\ \sum_{i=1}^{N} \left(|\vec{W_i}| \times x_i \right) &= 0 \\ \sum_{i=1}^{N-1} \left(|\vec{W_i}| \times x_i \right) + |\vec{W_N}| \times x_N &= 0 \\ \sum_{i=1}^{N-1} \left(|\vec{W_i}| \times (x_N - \sum_{j=i}^{N-1} d_j) \right) + |\vec{W_N}| \times x_N &= 0 \\ -\sum_{i=1}^{N-1} \left(|\vec{W_i}| \times (\sum_{j=i}^{N-1} d_j) \right) + \sum_{i=1}^{N} |\vec{W_i}| \times x_N &= 0 \\ x_N = \frac{\sum_{i=1}^{N-1} \left(|\vec{W_i}| \times (\sum_{j=i}^{N-1} d_j) \right)}{\sum_{i=1}^{N} |\vec{W_i}|} & \end{aligned} \tag{III.7}$$

Equation III.7 assumes that all particles hang by the same pivot point though. Since jobs will most likely have different target points, we must extend equation III.7 to be applied in RT scheduling. Therefore, we modify the horizontal distance constraint d_i

in equation system III.5 to add the horizontal distance between the pivot points of subsequent particles ($P_{i+1} - P_i$). The new distance constraint is $d_i = x_{i+1} - x_i + (P_{i+1} - P_i)$. With this new condition, equation III.6 becomes $x_i = x_N - \sum_{j=i}^{N-1}(d_j) - P_i + P_N$. The solution to the equilibrium state using this condition is shown in equation III.8, and holds for a set of N particles that push each other aside when in equilibrium.

$$x_N = \frac{\sum_{i=1}^{N-1}\left(|\vec{W}_i| \times (\sum_{j=i}^{N-1}(d_j) + P_i - P_N)\right)}{\sum_{i=1}^{N}|\vec{W}_i|} \tag{III.8}$$

Although this equation has two nested sums and a multiplication, it can be solved with linear complexity with respect to the number of particles if computing, initially, $\sum_{j=1}^{N-1}(d_j)$ and then, for every iteration of the outer sum, subtracting d_i from the inner sum (see algorithm 1, where function `sum(v[j],...,v[k])` computes $\sum_{i=j}^{k} v_i$).

Algorithm 1 Computing x_N.

```
sum_d = sum(d[1],...,d[N-1])
i = 1
numerator = 0
while i <= N − 1
{
  numerator = numerator + W[i] x (sum_d + P[i] - P[N])
  sum_d = sum_d - d[i]
  i = i + 1
}
return numerator/sum(W[1],...,W[N])
```

Implicit utility function

In the gravitational analogy, the system utility maximization problem is equivalent to the potential energy minimization of a pendulum system. This section describes the implicit utility function which is present in the gravitational task model, and is a consequence of the analogy with the particle pendulum.

A particle swinging in a pendulum describes a circular trajectory centered at the pivot point. Let the horizontal and the vertical position be, respectively, on the x-axis and y-axis of a cartesian plane centered at the pivot point. The set of points belonging to the trajectory is given by the function $y_i = -\sqrt{R_i^2 - x_i^2}$. Observe that the height is negative because we assume the pivot point as a reference for the vertical position, and the particle will always be below this point.

The potential energy (E_i) of a particle is equal to its vertical position times its weight ($E_i = W_i \times y_i$), which is given in equation III.9 as a function of the horizontal position of a particle in its trajectory. This function is an ellipse centered at 0 with horizontal axis of length R and vertical axis of length $W_i \times R_i$.

$$E_i = -W_i \times \sqrt{R_i^2 - x_i^2} \tag{III.9}$$

A maximization problem is a minimization problem where $max\, g = -min\,(-g)$. Based on the analogy between pendulums and task sets in the gravitational task model, we have $min\, E \equiv max\, g$ (where $E = \sum_{i=1}^{N} E_i$). Making $g_i = -E_i$ we have $max\, g = max\,(-E) = -(min\, E)$. Therefore, the system utility maximization problem becomes the potential energy minimization problem. g_i is defined replacing W_i and R_i in equation III.9 for job variables as described in table III.2 (see equation III.10). x_i is equivalent to the deviation of job j_i from its target point tp_i.

$$g_i = \frac{2 \times imp_i}{|exec\_win_i|} \times \sqrt{\left(\frac{|exec\_win_i|}{2}\right)^2 - x_i^2} \tag{III.10}$$

A property of ellipses (independent of their size and shape) is that in the region around their centers a variation Δx corresponds to a variation Δy, but the same variation Δx in a region farther from the center corresponds to a variation $\Delta y' > \Delta y$. For instance, a job deviation of $R/2$ represents a utility decrease of approximately 14%, but a deviation of R drops the utility to 0. This property is convenient to use with applications that are target sensitive, and can tolerate small variances around the target with low impact on its utility, but as the distance increases the utility drops abruptly.

III.6.2 Generic equilibrium

Generic equilibrium equation

The constrained maximization problem III.11 expresses the trade-off in a job chain of N jobs maximizing the accrued utility of the system for a given order of jobs. The goal function is to maximize the sum of the utilities of each job, subject to the constraints that there is no idle time between consecutive jobs ($1st$ constraint in (III.11)), and that the anchor points are within the respective execution windows ($2nd$ and $3rd$ constraint in (III.11)). This mathematical formulation is equivalent to the equilibrium condition of pendulums. In this section, we describe how to convert this constrained multi-dimensional maximization problem into one single unconstrained uni-dimensional maximization problem which has an optimum solution. Unlike the equilibrium equation derived directly from the pendulum analogy, this solution holds assuming jobs with any continuously differentiable concave utility function g_i. The utility function g_i of each job can be of different types, e.g. elliptical, parabolic, hyperbolic, etc.

$$\begin{aligned}
&\max: \quad g(x_1, \ldots, x_n) = \textstyle\sum_{i=1}^{N} g_i(x_i) \\
&\text{s.t}: \\
&\quad tp_{i+1} + x_{i+1} - (tp_i + x_i) = d_i \quad \forall i = 1..N-1 \\
&\quad x_i \leq dl_i - (1-\alpha_i) \times WCET_i - tp_i \;\; \forall i = 1..N \\
&\quad x_i \geq est_i + \alpha_i \times WCET_i - tp_i \;\; \forall i = 1..N
\end{aligned} \tag{III.11}$$

The goal function g is an N dimensional problem as function of $x_1, \ldots, x_N$. However, the $1st$ constraint can be rewritten to express x_i as a function of x_N, hence reducing the problem to 1 dimension. The steps that lead to equation III.12 show this transformation.

$$tp_{i+1} + x_{i+1} - (tp_i + x_i) = d_i$$

$$\sum_{j=i}^{N-1} (tp_{j+1} + x_{j+1} - (tp_j + x_i)) = \sum_{j=i}^{N-1} d_j$$

$$tp_N + x_N - (tp_i + x_i) = \sum_{j=i}^{N-1} d_j$$

finally:

$$x_i(x_N) = tp_N - tp_i - \sum_{j=i}^{N-1} d_j + x_N \tag{III.12}$$

As a result, $g(x_1, \ldots, x_n)$ becomes a sum of composite functions of g_i and x_i as a function of x_N (see equation III.13).

$$g(x_N) = \textstyle\sum_{i=1}^{N}(g_i(x_i(x_N))) = \sum_{i=1}^{N}(g_i \circ x_i)(x_N) \tag{III.13}$$

Replacing equation III.12 in the $2nd$ and $3rd$ constraint, which describe the space where $g(x_1, \ldots, x_n)$ is defined, we obtain inequality system III.14.

$$\begin{cases} x_N \leq dl_i - (1-\alpha_i) \times WCET_i - tp_N + \sum_{j=i}^{N-1} d_j \\ x_N \geq est_i + \alpha_i \times WCET_i - tp_N + \sum_{j=i}^{N-1} d_j \\ \forall i = 1..N \end{cases} \tag{III.14}$$

Solving this inequality system gives the interval I where equation III.13 is defined. If I is empty, there exists no trade-off that meets the timing constraints of all jobs at the same time. Otherwise, the schedule that maximizes the accrued utility in a job chain is given by $x_N \in I$ that maximizes equation III.13. Once x_N is found, the deviations of the other jobs from their target points are calculated using equation III.12.

Next, we show how to find the maximum value of $g(x_N)$ in any interval I provided $g(x_N)$ is concave. The sum of continuously differentiable concave functions is also a continuously differentiable concave function [Sun 06]. Therefore, if each job j_i has a concave and continuously differentiable utility function $g_i(x_i)$, then $g(x_N)$ is concave and continuously differentiable in interval I. We assume applications provide utility functions to the system along with their closed-form derivatives.

A function is called concave in a interval iff its derivative is monotonically decreasing in this interval [Sun 06]. Figure III.8 shows a concave function and its derivative. If $g(x_N)$ is concave and continuously differentiable, $g'(x_N)$ has at most one root in any interval I, where I must fall in one of the 3 categories below (depicted in figure III.8):

1. the interval contains only positive values of $g'(x_N)$, thus $g(x_N)$ is monotonically increasing with maximum at the rightmost point of I.

2. the interval contains only negative values of $g'(x_N)$, thus $g(x_N)$ is monotonically decreasing with maximum at the leftmost point of I.

3. the interval contains both positive and negative values of $g'(x_N)$, thus $g(x_N)$ has maximum at $g'(x_N) = 0$, which is the absolute maximum.

We differentiate $g(x_N)$ applying the chain rule to solve $\sum_{i=1}^{N}(g_i \circ x_i)'(x_N)$, and obtain equation III.15, which we call *generic equilibrium*. Note that $x_i'(x_N) = 1,\ \forall i = 1..N$. If I falls in either case 1) or 2), finding x_N that maximizes the accrued utility is straightforward. Otherwise, we must find the root of the generic equilibrium.

$$g'(x_N) = 0$$

$$\sum_{i=1}^{N}(g_i \circ x_i)'(x_N) = 0$$

$$\sum_{i=1}^{N}(g_i' \circ x_i)(x_N) \times x_i'(x_N) = 0$$

finally:

$$\sum_{i=1}^{N}(g_i' \circ x_i)(x_N) = 0 \tag{III.15}$$

The root of the generic equilibrium can be found either analytically, or using any numerical (iterative) root-finding algorithm [Sun 06], e.g. the bisection method, Newton's method, etc. A discussion on these methods is beyond the scope of this work. An analytical solution is preferable, but it is not always possible.

Solving equation III.15 consists of applying the numerical method rule to the sum of N elements ($\sum_{i=1}^{N}(g_i' \circ x_i)(x_N)$) until the return value of this sum converges to 0. Each iteration of the numerical root-finding algorithms has complexity $O(1)$ and $\sum_{i=1}^{N}(g_i' \circ x_i)(x_N)$ is computed with complexity $O(N)$ as in algorithm 2.

Figure III.9 shows some examples of types of continuously differentiable concave utility functions. Note that since the gravitational task model defines that the utility of jobs is maximized at the target points, $g_i(x_i)$ is maximized when the deviation from the target point is zero, i.e. $x_i = 0$. We assume that a negative deviation happens to the left side of the target point, and a positive to the right.

The complexity to solve the generic equilibrium is $O(K * N)$, where K denotes the number of iterations necessary to converge, and is independent of the number of jobs. Note that the efficiency of the solution also depends on the rate of convergence of the chosen root-finding algorithm. Solving $g'[i](x[i])$ analytically is independent of any other job rather than j_i, hence having complexity $O(1)$. Therefore, an analytical solution of the generic equilibrium has complexity $O(N)$. The next section brings an example of an analytical solution of the generic equilibrium.

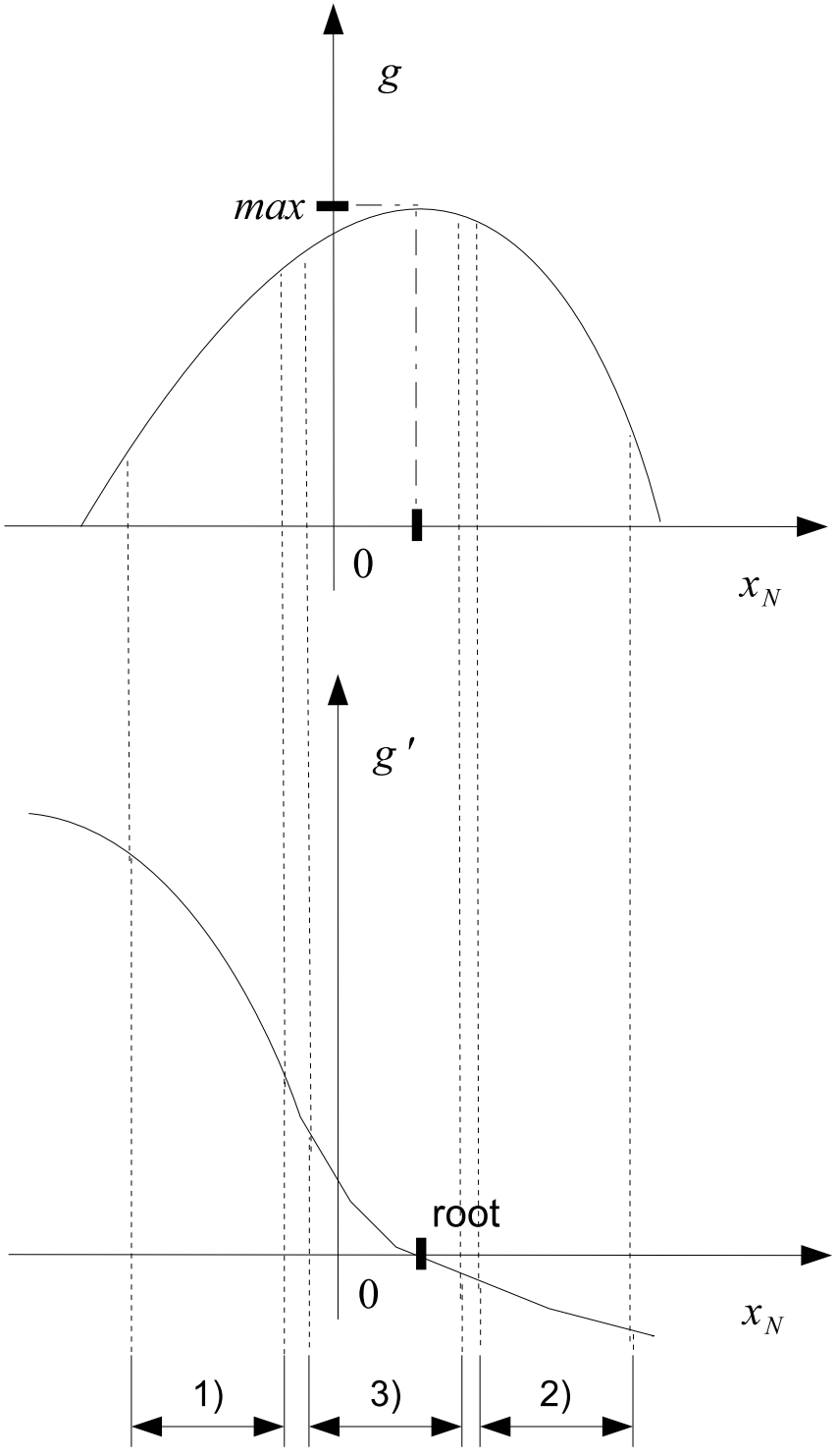


Figure III.8: *Example of a concave function and its derivative.*

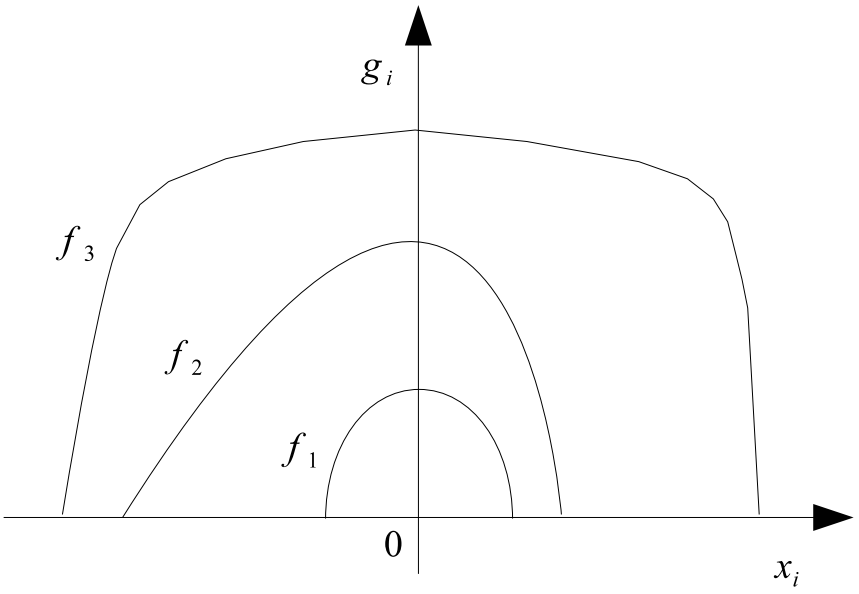


Figure III.9: *Some continuously differentiable concave utility functions.*

Algorithm 2 Computing $\sum_{i=1}^{N}(g_i' \circ x_i)(x_N)$.

```
sum_d = - d[N]
result = 0
i = N
while i > 0
{
  sum_d = sum_d + d[i]
  x[i] = tp[N] - tp[i] - sum_d + x[N]
  result = result + g'[i](x[i])
  i = i - 1
}
return result
```

In the next section, we show an example of an analytic solution assuming that the utility function of jobs are quadratic polynomials.

Example of an analytical solution of the generic equilibrium

In this section, we present an example of an analytical solution of the generic equilibrium assuming quadratic utility functions as in equation III.16.

$$g_i(x_i) = a_i x_i^2 + b_i x_i + c_i \tag{III.16}$$

Figure III.10 depicts a valid quadratic utility function. As $g_i(x_i)$ must be concave, we have $a < 0$. Furthermore, $g_i(x_i)$ must be maximized at $x_i = 0$ to be a valid utility function, and hence, we have $g_i'(0) = 0$. As can be seen in equation III.17, $g_i'(0) = 0$ only if $b_i = 0$. At $x_i = 0$ the return value is c_i.

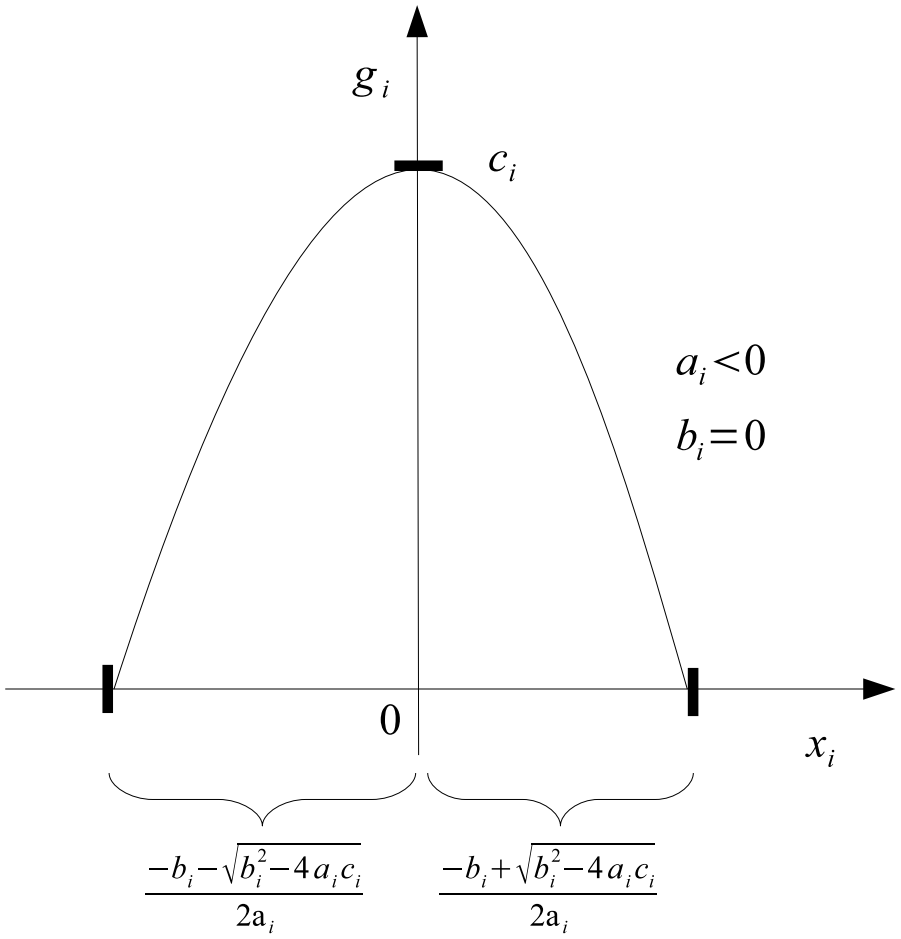


Figure III.10: *A quadratic utility function.*

$$g_i'(x_i) = 2a_i x_i + b_i \tag{III.17}$$

Replacing equation III.17 in the generic equilibrium (equation III.15), we obtain equation III.18 after a few steps for $b_i = 0$. The outer sum in the numerator can be reduced to range from 1 to $N - 1$ because, for $i = N$, $tp_i = tp_N$ and $\sum_{j=i}^{N-1} d_j = 0$, which zeros out the term.

Note that for $a_1 = W_i$ equation III.18 becomes equation III.8, which is the particle pendulum equilibrium. Therefore, besides being an approximation of the trade-off for elliptical utility functions, the particle pendulum equilibrium is optimum for a particular class of quadratic utility functions. This class of quadratic utility functions has $a_i = (2 \times imp_i)/(dl_i - WCET_i)$, $b_i = 0$ and $c_i = imp_i$.

$$\sum_{i=1}^{N}(g_i' \circ x_i)(x_N) = 0$$

$$\sum_{i=1}^{N} 2a_i x_i(x_N) = 0$$

$$\sum_{i=1}^{N}\left(2a_i \times \left(tp_N - tp_i - \sum_{j=i}^{N-1} d_j + x_N\right)\right) = 0$$

$$\sum_{i=1}^{N} 2a_i x_N + \sum_{i=1}^{N} 2a_i \times \left(tp_N - tp_i - \sum_{j=i}^{N-1} d_j\right) = 0$$

finally:

$$x_N = \frac{\sum_{i=1}^{N-1} a_i(\sum_{j=i}^{N-1}(d_j) + tp_i - tp_N)}{\sum_{i=1}^{N} a_i} \tag{III.18}$$

III.7 Summary

In this chapter, we introduced the simple gravity pendulum (or bob pendulum) system as a visualization model for trade-offs among target sensitive RT applications. As depicted in figure III.3, the pendulum system consists of a bob at the end of a massless string, which can swing back and forth along the *swinging range* under the influence of gravity over its central (lowest) point in a circular trajectory. Placed at the lowest point, the bob will come to rest there (*rest position*). If the bob pendulum contains more than one bob, they cannot be all at the same time in the lowest part, and hence, will push each other aside to find a new rest position (*equilibrium state*). The equilibrium state implies in minimized overall potential utility. The pendulum system depicted in figure III.4 consists of N bobs with radius r_i, weight $\vec{W}_i$, hanging by massless strings of length $R - r_i$ with a trajectory of radius R. Two adjacent bobs are an angle θ_i apart from each other. φ_i is the angle between the string attached to bob i and $\vec{W}_i$, and represents the position of the bob.

Trade-offs are present, for example, in the scheduling of target sensitive RT applications. We showed that classic task models, which are based on execution windows alone, either compromise feasibility for the sake of maximized utility accrual or fail to use resources efficiently, i.e. for maximized utility accrual. We introduced, then, the gravitational task model, which provides simple abstractions for expression of timing constraints of such applications based on the analogy between the task model and pendulum systems. This expressiveness provides for adaptivity in order to improve system utility.

In the gravitational task model, each job j_i can express (see figures III.1 and III.2): earliest start time est_i; relative deadline rel_dl_i (hence, absolute deadline $dl_i = est_i +$

rel_dl_i); worst case execution time $WCET_i$, target point tp_i; an anchor point α_i within its execution that must coincide with the position of the target point upon execution for maximum utility accrual; the execution window $exec\_win_i = [est_i + \alpha_i \times WCET_i, dl_i - (1 - \alpha_i) \times WCET_i]$ is the interval within which the anchor point must lie for correct system behavior and deviating the anchor point away from the target point causes the utility to decay; and, finally, importance imp_i which relates to the need of the job to execute as close as possible to its target point. Scheduling algorithms can exploit those parameters for better scheduling decisions. Those algorithms must shift the execution of jobs with conflicting targets to make room for execution such that the system utility is maximized.

Drawing the analogy, the execution time of a job is equivalent to the size of the bob. A job is allowed to execute at its target point in the absence of other jobs in the system with the same target point. The target point is equivalent, thus, to the central (lowest) point of a pendulum trajectory and the swinging range is the execution window of the job. The importance of a job, which represents its resistance to be shifted away from its target point when interacting with other jobs, can be seen as the weight of the bob. The heavier a bob is, the closer to the bottom it will come to rest. Finally, the job utility as a function of its deviation from the target point is similar to the potential energy of a bob as a function of its deviation from the central point. As the equilibrium is the state that minimizes the potential energy of the pendulum, the best compromise of the jobs' interests maximizes the accrued utility of the system. This analogy is depicted in figure III.5 and summarized in table III.1.

However, following this analogy to schedule jobs might result and execution overlap, as can be seen in figure III.6. Then, we introduced the particle pendulum, where the basic difference lies in the fact that adjacent particles have a constant horizontal distance constraint d_i. This analogy is depicted in figure III.7 and table III.2 summarizes the mapping of the parameters from both systems.

We also calculated an approximation for the equilibrium of the particle pendulum inspired by the equilibrium condition for bob pendulums. Combined with the conversion provided in table III.2, the equilibrium of particle pendulums calculates an approximation of the best compromise among jobs with conflicting target points. Then, we showed, based on this new analogy, that the gravitational task model implicitly assumes that jobs have elliptical utility functions as in equation III.10. Finally, we generalized the equilibrium equation in section III.6.2 to allow jobs to have any arbitrary continuously differentiable concave utility function. This generalization keeps the original intuition from the analogy with pendulum systems.

Chapter IV

Scheduling target sensitive real-time tasks

In this chapter, we present a few scheduling algorithms for the gravitational task model. The scheduling of target sensitive real-time (RT) applications must account for timing constraints, and the trade-off among tasks with conflicting targets. Those scheduling algorithms use the equilibrium state concept to compute the deviation of jobs from their target points for increased system utility. Furthermore, the execution sequence of jobs in the schedule has a significant impact on the equilibrium of jobs, and dominates the complexity of the problem — the optimum solution is Non-deterministic Polynomial-time hard (NP-hard) [Chen 96].

We start with a detailed description of the problem, and the basic idea of our scheduling algorithms in section IV.1. Section IV.2 describes how to schedule jobs using the equilibrium to compute trade-offs, and in section IV.3 we present the importance of the execution sequence of jobs in the schedule, and a few ordering heuristics. Section IV.4 brings some experimental results of simulations, where we compare the different ordering heuristics, and the 2 proposed methods for equilibrium calculation: the pendulum equilibrium and the generic equilibrium. Finally, section IV.5 brings our concluding remarks.

IV.1 Introduction

In chapter III, we proposed a method based on the equilibrium of bob pendulums to compute the trade-off among jobs with conflicting target points. However, not all jobs in a schedule compete with one another for their target points. Therefore, in order to apply the equilibrium, the scheduler must know which jobs have conflicting target points. Identifying those jobs is not trivial, and requires equilibrium recomputations that result in increased complexity. As a result, reducing the complexity to recompute the equilibrium is crucial.

The execution sequence of jobs also impacts on the trade-off; the equilibrium itself is limited to finding local optimums. Ordering the execution of non-preemptive tasks is a NP-hard problem [Chen 96] — check all possible permutations. Therefore, an optimum solution has high overhead, and hence, is unfeasible for on-line scheduling algorithms. Heuristic solutions compromise between overhead, acceptance ratio, and utility accrual; affording higher overhead tends to provide for better scheduling decisions. Traditional execution windows based scheduling algorithms aim only at feasibility, e.g. Earliest Deadline First (EDF), Rate Monotonic (RM), etc.

In this chapter, we present an algorithm based on bob pendulums to identify which jobs in the schedule compete for their target points — those jobs comprise a job chain. This algorithm inserts jobs one by one in the schedule and, similar to inserting bobs one by one in a pendulum system, recomputes the equilibrium upon collision detection. These equilibrium recomputations lead to a complexity $O(N^2)$. Then, we present a method to avoid recomputing the whole equilibrium upon collision. This method consists of storing intermediate values for jobs already in equilibrium in order save computational steps in case the scheduler must recompute the equilibrium. This new equilibrium recomputation method has linear complexity, and does not impact on the output of the equilibrium calculation.

We also propose 3 heuristics to reorder the execution sequence of jobs: DST-1, DST-2 and DST-3. Those heuristics are inspired by the physics of liquids: higher density liquids come to rest closer to the bottom and, following the same rationale, tasks with higher utility density should execute closer to their target points. The utility density of a job is the ratio of its importance to its WCET. Heuristic DST-1 uses only the utility density concept to come up with a execution ordering, ignoring the execution windows of jobs. Hence, feasibility may be compromised. Heuristic DST-2 extends DST-1 to account for execution windows on ordering decision. Both DST-1 and DST-2 have complexity $O(N \times log(N))$ because the arrival of a job may cause reordering of previously scheduled jobs. Heuristic DST-3, finally, modifies the previous ones to avoid reordering of the execution sequence upon job arrival, and hence, has linear complexity.

IV.2 Trade-off among competing jobs

In this section, we describe how to schedule jobs using the equilibrium to compute trade-offs. The equilibrium is a method that shifts the execution of jobs with conflicting target points for increased system utility accrual. However, the absolute maximum utility of

a given task set must also consider the execution sequence of jobs — the equilibrium itself is limited to finding a local optimum. The optimum solution is to completely enumerate all possible permutations of the execution sequence of jobs. As we focus on the equilibrium now, we order the jobs statically according to their target points.

Section IV.2.1 describes the algorithm to compute the equilibrium of jobs upon scheduling; section IV.2.2 shows a method to reduce the complexity to calculate the equilibrium of jobs; section IV.2.3 shows the on-line job admission strategy; finally, sections IV.2.4 and IV.2.5 illustrate the scheduling algorithm with examples.

IV.2.1 Computing the equilibrium of jobs

The pendulum equilibrium (see equation III.8) and the generic equilibrium (see equation III.15) compute deviations of jobs with overlapping execution for increased utility accrual. Those equations hold given that there is no idle period between any two consecutive jobs in the equilibrium state — those jobs comprise a job chain. A schedule with 2 job chains is depicted in figure IV.1.

We identify the job chains appending jobs one by one in the schedule, and recomputing the equilibrium state when executions overlap. This is similar to inserting bobs one by one in a pendulum system, and the equilibrium state changes upon collisions. Notice that upon equilibrium recomputation a ripple effect might happen, and adjacent job chains that merge must have their equilibrium recomputed.

Our algorithm to compute the equilibrium of jobs maintains a list of job chains. Whenever we append a job in the schedule, we append a job chain containing the appended job in the job chain list. The algorithm computes, then, the equilibrium for the last job chain, and verifies if the first job of the last job chain overlaps execution with the last job of the previous job chain. If so, the algorithm merges the last and the penultimate chains, and recomputes the equilibrium for this new last chain. This may trigger new merges, which we call *ripple effect*. When the ripple effect stops, the algorithm appends the next job, and repeats this process until all jobs are in the schedule.

Recomputing the equilibrium may shift the execution of some jobs before the current time, which is not feasible. Therefore, when scheduling periodic tasks, we consider all jobs in the hyper-period at the beginning of each hyper-period. Note that the number of jobs that must be considered grows, in the worst case, in factorial scale with respect to the number of periodic tasks. This worst case happens when the periods of all tasks are prime.

Appending N jobs to the schedule, and N job chains to the job chain list, has complexity $O(N)$. The maximum number of merged chains is $N-1$, and for each merge, the algorithm scans all jobs in the new chain, and computes the equilibrium. Updating the job chain list to merge two chains has complexity $O(1)$, and computing the equilibrium has complexity $O(N)$. As merges may happen $N-1$ times, the complexity to identify the job chains is $O(N^2)$ due to the equilibrium recomputations upon merge. Sections IV.2.4 and IV.2.5 illustrate the calculation of the equilibrium of jobs during scheduling with an example. There is an example with the pendulum equilibrium, and

an example with the generic equilibrium.

In the next section, we present a method to recompute the equilibrium of merged chains with complexity $O(1)$. Using this method, the complexity of our algorithm to identify job chains becomes linear.

IV.2.2 Reducing the complexity of equilibrium recomputations

We present in this section a method to dispense revisiting all jobs when computing the equilibrium of merged chains using the particle equilibrium. Instead of visiting all jobs in the new chain and computing the equilibrium as in equation III.8, this method computes the equilibrium of the new chain only using intermediate values stored for each of the merged chains.

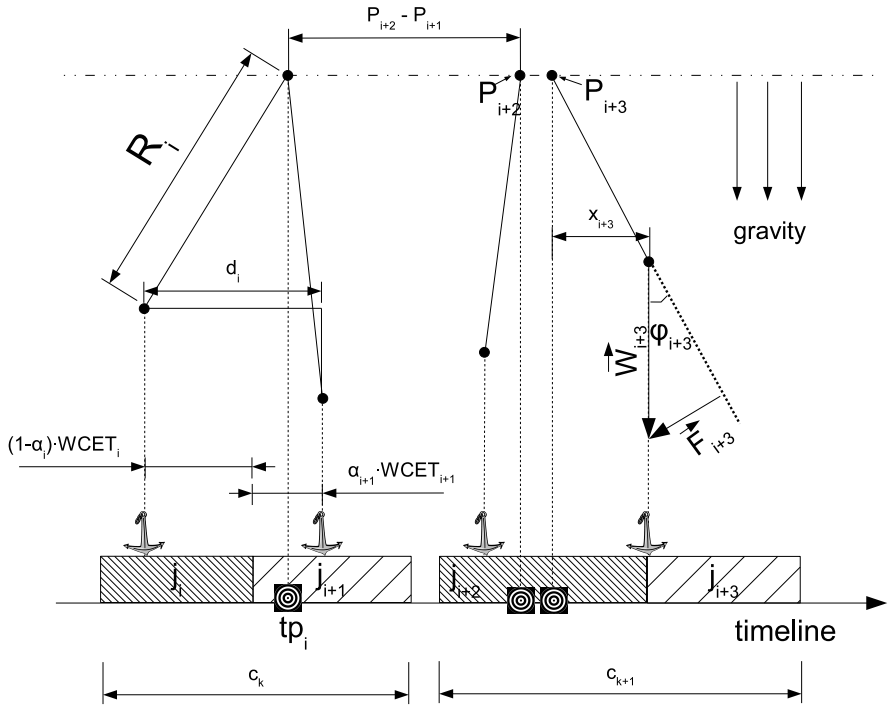


Figure IV.1: *Analogy between particle pendulum and tasks*

In a job chain, we are interested in knowing what is the position of the first and the last job in the equilibrium state. Let j_{fj} be the first job and j_{lj} the last job of a job chain ($fj, lj \in \mathbb{N}^*$). We define the position of job j_i ($pos(j_i)$) as the point in time in the schedule at which its anchor point is placed. We obtain $pos(j_{lj})$ by summing its deviation (computated as in equation III.8) and its target point. Once $pos(j_{lj})$ is known, we can determine $pos(j_{fj})$ by subtracting the distance d_i of the jobs scheduled to execute in between ($fj \leq i < lj$). See equation system IV.1.

$$\begin{cases} pos(j_{lj}) &= \frac{\sum_{i=fj}^{lj-1} W_i \times (\sum_{j=i}^{lj-1}(d_j)+P_i-P_{lj})}{\sum_{i=fj}^{lj} W_i} + P_{lj} \\ pos(j_{fj}) &= pos(j_{lj}) - \sum_{i=fj}^{lj-1} d_i \end{cases} \quad \text{(IV.1)}$$

In the sequence, we will explain how to compute the equilibrium of 2 chains that merge using only intermediate values stored for each chain, and a fixed number of operations that is independent of the number of jobs in each chain. The first step is to decide which intermediate values must be stored for a job chain, and how to compute the equilibrium using these values. Following the rationale of storing the result of sums and multiplications that must be used in future equilibrium recomputations, we choose 3 intermediate values, X, Y and Z. They are computed as shown in equation system IV.2. Using these values, we compute $pos(j_{lj})$ and $pos(j_{fj})$ as in IV.3. Equation systems IV.1 and IV.3 are equivalent.

$$\begin{cases} X &= \sum_{i=fj}^{lj-1} \left(W_i \times \left(\sum_{j=i}^{lj-1}(d_j) + P_i \right) \right) \\ Y &= \sum_{i=fj}^{lj} W_i \\ Z &= \sum_{i=fj}^{lj-1} d_i \end{cases} \quad \text{(IV.2)}$$

$$\begin{cases} pos(j_{lj}) &= \frac{X - P_{lj} \times (Y - W_{lj})}{Y} + P_{lj} \\ pos(j_{fj}) &= pos(j_{lj}) - Z \end{cases} \quad \text{(IV.3)}$$

The second step is to compute the values X, Y and Z of a new chain only using the intermediate values of each merged chain, instead of using equation system IV.2. This way, we avoid going through all the jobs in the new chain, and we can compute X, Y and Z with a fixed number of operations that is independent of the number of jobs in each merged chain (i.e. $O(1)$). Let X', Y' and Z' be the intermediate values of the first chain, and X'', Y'' and Z'' be the intermediate values of the second chain. Then $fj = fj'$, $lj = lj''$, and $lj' + 1 = fj''$ ($j_{lj'}$ and $j_{fj''}$ are adjacent jobs). For instance, in the particular case depicted in figure IV.1, if the chains c_k and c_{k+1} merge then we have that $fj = fj' = j_i$, $lj = lj'' = j_{i+3}$, and that $lj' + 1 = fj'' = j_{i+2}$.

$$Y = \sum_{i=fj}^{lj} W_i \quad \text{(IV.4)}$$

$$Y = \sum_{i=fj'}^{lj''} W_i$$

$$Y = \sum_{i=fj'}^{lj'} W_i + \sum_{i=fj''}^{lj''} W_i$$

$$Y = Y' + Y'' \quad \text{(IV.5)}$$

Expressions IV.4 and IV.5 show how to compute Y as a function of Y' and Y''. Expressions IV.6, IV.7 and IV.8 show how to compute Z as a function of Z', Z'' and $d_{lj'}$.

$$Z = \sum_{i=fj}^{lj-1} d_i \tag{IV.6}$$

$$Z = \sum_{i=fj'}^{lj'-1} d_i + d_{lj'} + \sum_{i=fj''}^{lj''-1} d_i \tag{IV.7}$$

$$Z = Z' + d_{lj'} + Z'' \tag{IV.8}$$

In order to compute X, we break the outer sum in expression IV.9 into 3 parts. These parts correspond to terms IV.11, IV.12 and IV.13. Term IV.13 is equal to X'', as can be seen by the description of X in equation system IV.2.

$$X = \sum_{i=fj}^{lj-1} \left(W_i \times \left(\sum_{j=i}^{lj-1} (d_j) + P_i \right) \right) \tag{IV.9}$$

$$X = \sum_{i=fj'}^{lj''-1} \left(W_i \times \left(\sum_{j=i}^{lj''-1} (d_j) + P_i \right) \right) \tag{IV.10}$$

$$X = \sum_{i=fj'}^{lj'-1} \left(W_i \times \left(\sum_{j=i}^{lj''-1} (d_j) + P_i \right) \right) \tag{IV.11}$$

$$+ W_{lj'} \times \left(\sum_{j=lj'}^{lj''-1} (d_j) + P_{lj'} \right) \tag{IV.12}$$

$$+ \sum_{i=fj''}^{lj''-1} \left(W_i \times \left(\sum_{j=i}^{lj''-1} (d_j) + P_i \right) \right) \tag{IV.13}$$

Term IV.12 becomes term IV.14 after a few mathematical steps. The sum $\sum_{j=lj'}^{lj''-1}(d_j)$ in term IV.12 becomes the expression $\sum_{j=fj''}^{lj''-1}(d_j) + d_{lj'}$ since $d_{lj'}$ and $d_{fj''}$ are adjacent. Notice that $\sum_{j=fj''}^{lj''-1}(d_j) = Z''$.

$$W_{lj'} \times \left(\sum_{j=lj'}^{lj''-1} (d_j) + P_{lj'} \right) =$$

$$W_{lj'} \times \left(\sum_{j=fj''}^{lj''-1} (d_j) + d_{lj'} + P_{lj'} \right) =$$

$$W_{lj'} \times (Z'' + d_{lj'} + P_{lj'}) \tag{IV.14}$$

Finally, let us analyze term IV.11. We break the inner sum $\sum_{j=i}^{lj''-1}(d_j)$ into the term $\sum_{j=i}^{lj'-1}(d_j) + d_{lj'} + \sum_{j=fj''}^{lj''-1}(d_j))$ (see (IV.15)). Then, we use the distribution property to

break term IV.15 into terms IV.16 and IV.17. The former term is equal to X', and the latter one is the sum of a variable multiplied by a constant. We remove the constant part from the sum, and after some basic mathematical steps — which include replacing IV.2 in IV.18 — we obtain IV.19. The sum of expressions IV.11, IV.12 and IV.13 is the value of X without revisiting jobs.

$$\sum_{i=fj'}^{lj'-1}\left(W_i \times \left(\sum_{j=i}^{lj''-1}(d_j)+P_i\right)\right) =$$

$$\sum_{i=fj'}^{lj'-1}\left(W_i \times \left(\sum_{j=i}^{lj'-1}(d_j)+d_{lj'}+\sum_{j=fj''}^{lj''-1}(d_j)+P_i\right)\right) = \quad \text{(IV.15)}$$

$$\sum_{i=fj'}^{lj'-1}\left(W_i \times \left(\sum_{j=i}^{lj'-1}(d_j)+P_i\right)\right)+ \quad \text{(IV.16)}$$

$$\sum_{i=fj'}^{lj'-1}\left(W_i \times \left(\sum_{j=fj''}^{lj''-1}(d_j)+d_{lj'}\right)\right) = \quad \text{(IV.17)}$$

$$X'+\left(\sum_{j=fj''}^{lj''-1}(d_j)+d_{lj'}\right)\times\sum_{i=fj'}^{lj'-1}(W_i) = \quad \text{(IV.18)}$$

$$X'+(Z''+d_{lj'})\times(Y'-W_{lj'}) \quad \text{(IV.19)}$$

The calculations of X, Y and Z without revisiting jobs are summarized in equation system IV.20. This alternative solution to equation system IV.2 uses only the intermediate values of the adjacent chains, and a fixed number of operations. These calculations have, hence, complexity $O(1)$, and they must be computed whenever there is a merge of adjacent chains. For an incoming job j_i, we compute its intermediate values as in equation system IV.2 with $lj = fj = i$, thus $X = 0$, $Y = W_i$ and $Z = 0$. We apply this method recursively upon ripple effects.

$$\begin{cases} X &= X'+(Z''+d_{lj'})\times(Y'-W_{lj'}) \\ & \quad +W_{lj'}\times(Z''+d_{lj'}+P_{lj'})+X'' \\ Y &= Y'+Y'' \\ Z &= Z'+d_{lj'}+Z'' \end{cases} \quad \text{(IV.20)}$$

Notice that using equations IV.20 and IV.3 we obtain the positions of the first and last jobs without considering the earliest start times and deadlines constraints of jobs within the chain. Since these constraints must not be violated in order to yield a feasible schedule, the boundary within which a chain can swing is limited. As a result, we introduce the parameters S_{left} and S_{right}, which represent, respectively, the amount of slack that a job chain can shift to the left and to the right. These slacks must always be larger than or equal to zero to ensure that no job within the chain violates its timing constraints (see equation system IV.21).

$$\begin{cases} S_{left} &= \min\{pos(j_i) - est_i - \alpha_i \times WCET_i\} & \forall i = fj..lj \\ S_{right} &= \min\{dl_i - pos(j_i) - (1-\alpha_i) \times WCET_i\} & \forall i = fj..lj \end{cases} \quad \text{(IV.21)}$$

A chain with one job has, hence, $S_{left} = tp_i - \alpha_i \times WCET_i - est_i$ and $S_{right} = dl_i - tp_i - (1-\alpha_i) \times WCET_i$, since this job executes at its target point. Then upon chain collision, the equilibrium of chains that merge is computed as described in this section and, at each merge, the slacks of the new chain are computed. Assume 2 consecutive chains c_k and c_{k+1} that merge into a new chain c_k (the new chain is the k^{th} chain in the schedule). Their overlap is given by $overlap = pos'(j_{lj'}) + (1 - \alpha_{lj'}) \times WCET_{lj'} - (pos''(j_{fj''}) - \alpha_{lj''} \times WCET_{lj''})$ and the slacks of the new chain are computed as in equation IV.22. As $j_{lj} = j_{lj''}$, let us use the term $pos''(j_{lj''})$ to represent the position of j_{lj} before merging, and $pos(j_{lj})$ to represent the position of j_{lj} after merging. Figure IV.2 depicts this situation. The target points and the pendulum analogy were omitted to avoid overcrowding the figure.

$$\begin{cases} S_{left} &= \min\{S^*_{left}, S^{**}_{left}\} \\ S_{right} &= \min\{S^*_{right}, S^{**}_{right}\} \\ S^*_{left} &= S'_{left} + pos(j_{lj}) - pos''(j_{lj''}) - overlap \\ S^*_{right} &= S'_{right} - (pos(j_{lj}) - pos''(j_{lj''})) + overlap \\ S^{**}_{left} &= S''_{left} + pos(j_{lj}) - pos''(j_{lj''}) \\ S^{**}_{right} &= S''_{right} - (pos(j_{lj}) - pos''(j_{lj''})) \end{cases} \quad \text{(IV.22)}$$

If after merging $S_{left} + S_{right} < 0$, then the schedule is not feasible. Otherwise, if either S_{left} or S_{right} is negative, the chain must be appropriately shifted. If $S_{left} < 0$, then at least one job in the chain is scheduled at most $|S_{left}|$ before its start time. In this case, we increase the position of the first and last job in the chain by $|S_{left}|$, S_{right} decreases by $|S_{left}|$, and $S_{left} = 0$. In case $S_{right} < 0$, then at least one job in the chain is scheduled at most $|S_{right}|$ after its deadline. In this case, we decrease the position of the first and last job in the chain by $|S_{right}|$, S_{left} decreases by $|S_{right}|$, and $S_{right} = 0$. As we can see, the complexity of merging 2 chains remains $O(1)$.

Notice that this method does not update the position of jobs within the chain, as their positions are not needed during scheduling, and doing so implies in visiting all jobs. We update the positions of all jobs in the end of the scheduling process (see section IV.2.1). Section IV.2.4 illustrates a scheduling algorithm which uses this method with an example.

IV.2.3 On-line admission

A job j_a arriving on-line gets its own job chain c_a. The scheduler inserts c_a in the job chain list so that j_a is ordered with the other jobs by target point, and computes the equilibrium for c_a. Inserting c_a in the list has complexity $O(N)$, and calculating the equilibrium of c_a has complexity $O(1)$. This insertion may cause an existing chain c to split into c' and c'': c' contains all jobs which execute before j_a and c'' contains

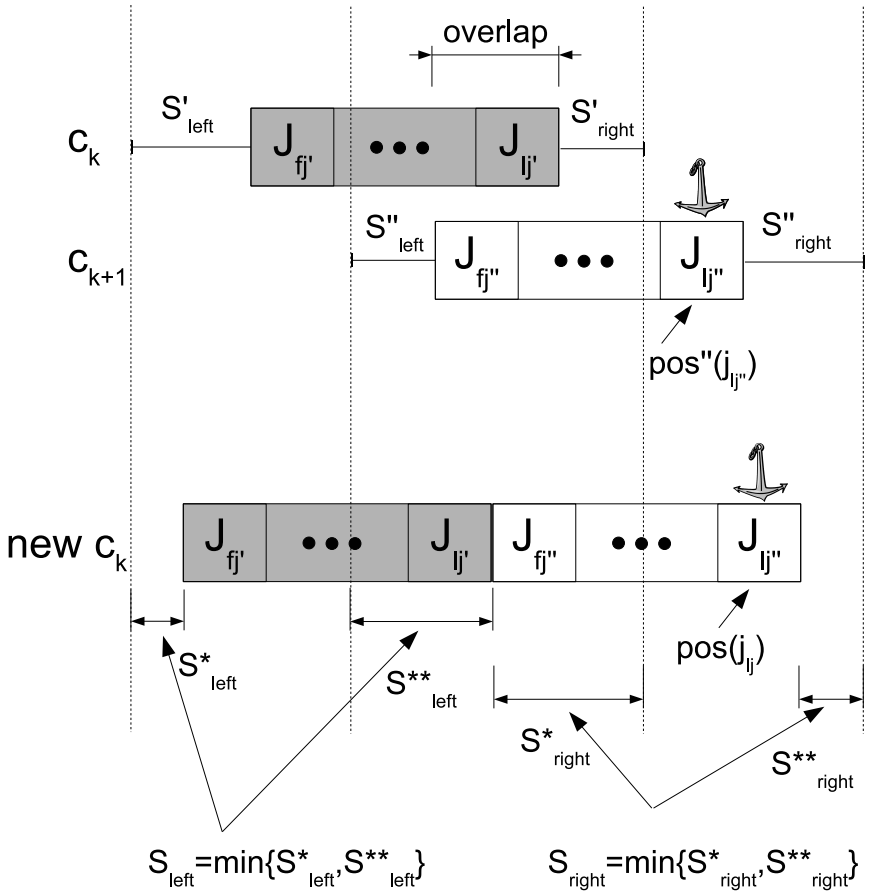


Figure IV.2: *Slack of merged chains*

all jobs which execute after j_a. The algorithm computes the equilibrium of c' and c'' using equation IV.3, their respective intermediate values using equation IV.2, and their respective slacks using equation system IV.21. Computing the equilibrium for chains c' and c'' has linear complexity.

Next, the algorithm checks if c_a merges with any of the adjacent chains as described in sections IV.2.1 and IV.2.2. The necessary updates have worst case complexity $O(N)$. If after the necessary equilibrium recomputations — which happen due to execution overlapping and ripple effect — jobs are scheduled for execution before the current time, the scheduler postpones the execution of the next ready job; during this process chains may merge again. Scanning the job chain to look for merges has complexity $O(N)$. The scheduler admits the incoming job only if the schedule is feasible, and the resulting utility accrual is higher than before admission.

The worst case complexity of the on-line admission is, hence, linear with the number of jobs. The next section brings an example of this method in a scheduling scenario.

IV.2.4 Scheduling example using pendulum equilibrium

Without intermediate values

	τ_1	τ_2	τ_3
start time	0	0	0
period	6	12	12
relative deadline	6	6	12
WCET	2	1	4
importance	1	6.25	2
anchor point	0	0	0

Table IV.1: *Task set.*

Consider the task set shown in Table IV.1. Within the first hyper-period ($[0, 12]$), we have the following jobs ordered by target point: $\tau_{1,1}$, $\tau_{2,1}$, $\tau_{3,1}$ and $\tau_{1,2}$, where $\tau_{i,j}$ represents the j^{th} instance of task τ_i (see table IV.2). Given this order, we now refer to them as jobs j_1, j_2, j_3 and j_4, respectively. Their target points are 2, 2.5, 4 and 8, respectively, as we assume that target points lie in the middle of the execution window ($tp_i = est_i + \alpha_i \times WCET_i + |exec\_win_i|/2$); for the sake of simplicity, we assume their anchor points are all 0 (all anchor points are mapped to the beginning of the execution of a job).

Following the algorithm described in section IV.2.1, we first place j_1 at time 2. The system utility at this point is 1. Next, we try to put j_2 at time 2.5, but j_1 finishes its execution at time 4, so we have to find the equilibrium state for them. We compute x_2 with Equation III.8 and obtain the result 0.25. Hence, $x_1 = x_2 - d_1 - P_1 + P_2 = -1.25$, and the system utility is $0.5 \times \sqrt{4 - (-1.25)^2} + 2.5 \times \sqrt{6.25 - 0.25^2} = 6.99929475$. So, j_1 shifts 1.25 units of time to the left, and j_2 shifts 0.25 units of time to the right of its target point, hence scheduled to execute at times 0.75 and 2.75, respectively. Then, we schedule j_3 at time 4 and no job is pushed, since j_2 finishes at time 3.75. Finally, we schedule j_4 at time 8 and no job is pushed, since j_3 finishes exactly at time 8. The final system utility is $6.99929475 + 2 + 1 = 9.99929475$, and the final schedule has 2 job chains (c_1 and c_2). This schedule is depicted in figure IV.3.

Now suppose that an on-line job j_a arrives at time 3 with deadline 13, anchor point 0, WCET 2 and importance 2; hence weight 0.5, execution window $[3, 11]$, utility function $0.5 \times \sqrt{16 - x_a^2}$, and target point 7. In this case, we have to apply the on-line phase described in section IV.2.3. First, we recompute the equilibrium state for jobs j_3, j_a and j_4, as j_2 executes currently and cannot shift; j_3 and j_4 together accrue utility 3 to the system before the arrival of j_a. The result of the equilibrium is $x_3 = -1$, $x_a = 0$ and $x_4 = 1$. However, x_3 cannot by smaller than -0.25 in order to not affect the schedule of j_2. Therefore, we have $x_3 = -0.25$, $x_a = 0.75$ and $x_4 = 1.75$, and the final system utility is $1.996 + 1.9645 + 0.484 = 4.4445$, which is higher than the previous system utility, i.e.

	j_1	j_2	j_3	j_4
instance	$\tau_{1,1}$	$\tau_{2,1}$	$\tau_{3,1}$	$\tau_{1,2}$
tp_i	2	2.5	4	8
W_i	0.5	2.5	0.5	0.5
d_i	2	1	4	2
$exec\_win_i$	$[0, 4]$	$[0, 5]$	$[0, 8]$	$[6, 10]$
$g_i(x_i)$	$0.5 \times \sqrt{4 - x_1^2}$	$2.5 \times \sqrt{6.25 - x_2^2}$	$0.5 \times \sqrt{16 - x_3^2}$	$0.5 \times \sqrt{4 - x_4^2}$

Table IV.2: *Jobs.*

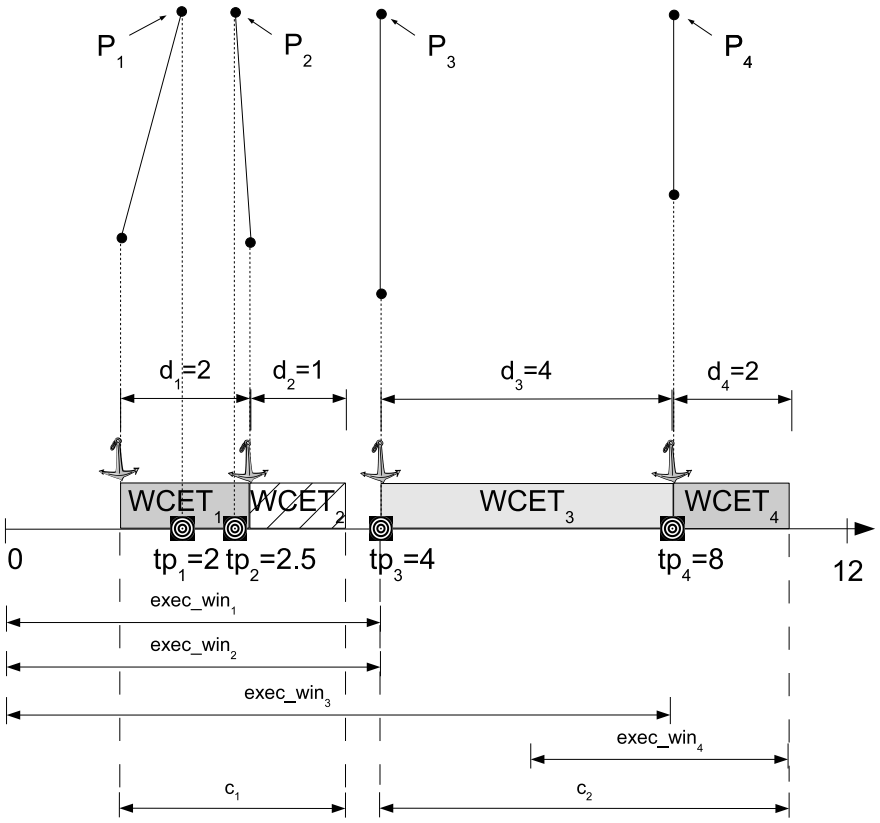


Figure IV.3: *Schedule before on-line job arrival.*

3. Hence, j_a is accepted, and the final schedule with the pendulum analogy is depicted in figure IV.4.

With intermediate values

Let us schedule the same task set used in the previous example. We first place j_1 at time 2, and create a job chain c_1 with $X_1 = 0$, $Y_1 = 0.5$, $Z_1 = 0$, $S_{left_1} = 2$ and $S_{right_1} = 2$;

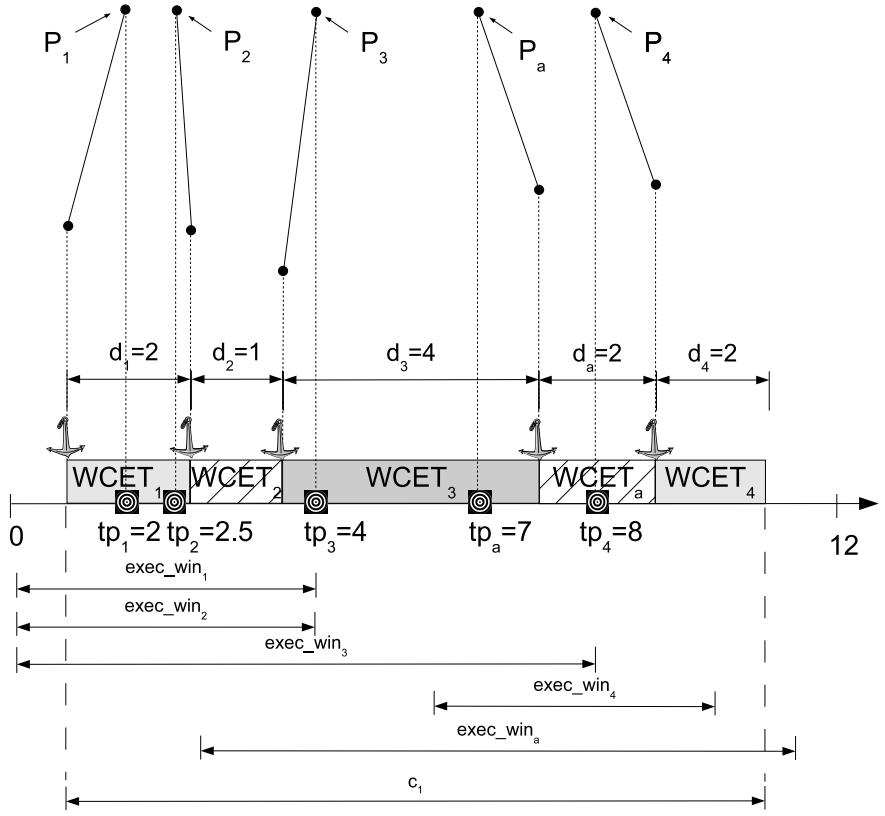


Figure IV.4: *Schedule after on-line job arrival.*

these intermediate values are computed using equations IV.2 and IV.21. Next, we put j_2 at time 2.5 in a job chain c_2 with $X_2 = 0$, $Y_2 = 2.5$, $Z_2 = 0$, $S_{left_2} = 2.5$ and $S_{right_2} = 2.5$. However, j_1 finishes its execution at time 4, and chains c_1 and c_2 overlap. Therefore, we merge them into a new chain c_1 with $X_1 = 2$, $Y_1 = 3$, $Z_1 = 2$; these values are computed using equation IV.20. Then, using equation system IV.3, we obtain $pos(j_2) = 2.75$ and $pos(j_1) = 0.75$. Finally, using equation system IV.21, we obtain $S_{left_1} = 0.75$ and $S_{right_1} = 2.25$. Then, we schedule j_3 at time 4 in a job chain c_2 with $X_2 = 0$, $Y_2 = 0.5$, $Z_2 = 0$, $S_{left_2} = 4$ and $S_{right_2} = 4$. There is no merge, since j_2 finishes at time 3.75. Finally, we schedule j_4 at time 8 in a job chain c_3 with $X_3 = 0$, $Y_3 = 0.5$, $Z_3 = 0$, $S_{left_3} = 4$ and $S_{right_3} = 4$. Since j_3 finishes exactly at time 8, we merge chains c_2 and c_3, obtaining a new chain c_2 with $X_2 = 4$, $Y_2 = 1$ and $Z_2 = 4$. Using equation system IV.3 we obtain $pos(j_4) = 8$ and $pos(j_3) = 4$. Finally, using equation system IV.21, we obtain $S_{left_2} = 2$ and $S_{right_2} = 2$. The final system utility is $6.99929475 + 2 + 1 = 9.99929475$, and the final schedule has 2 job chains (c_1 and c_2). There are no jobs whose positions need to be updated. This schedule is depicted in figure IV.3.

To finalize the example, let us consider the arrival of the on-line job j_a at time 3 with deadline 13, anchor point 0, WCET 2 and importance 2; hence weight 0.5, execution window $[3, 11]$, utility function $0.5 \times \sqrt{16 - x_a^2}$ and target point 7. In this case, we have to apply the on-line admission algorithm described in section IV.2.2. Ordering

by target point, j_a executes in between j_3 and j_4, hence splitting job chain c_2 into 2 chains; j_3 and j_4 accrue utility 3 before the arrival of j_a. Now jobs j_3, j_a and j_4 have their own chains, respectively c_2, c_3 and c_4. Computing the equilibrium for each of those chains independently using equations IV.3, and their intermediate values with equation systems IV.2 and IV.21 results in: $X_2 = 0$, $Y_2 = 0.5$, $Z_2 = 0$, $S_{left_2} = 4$ and $S_{right_2} = 4$; $X_3 = 0$, $Y_3 = 0.5$, $Z_3 = 0$, $S_{left_3} = 4$ and $S_{right_3} = 4$; and, $X_4 = 0$, $Y_4 = 0.5$, $Z_4 = 0$, $S_{left_4} = 4$ and $S_{right_4} = 4$. Obviously, scheduling the execution of j_3, j_a and j_4 at their target points causes these chains to overlap, and hence, we have to merge c_3 and c_4, obtaining a new chain c_3 with $X_3 = 4.5$, $Y_3 = 1$ and $Z_3 = 2$. Then, $pos(j_4) = 8.5$, $pos(j_a) = 6.5$, $S_{left_3} = 2.5$ and $S_{right_3} = 1.5$. As this new c_3 overlaps with c_2, we merge them into a new chain c_2 with $X_2 = 9.5$, $Y_2 = 1.5$ and $Z_2 = 6$. Then, $pos(j_4) = 9$, $pos(j_3) = 3$, $S_{left_2} = 3$ and $S_{right_2} = 1$. However, $pos(j_3)$ cannot be smaller than 3.75, which is the finishing time of j_2; j_2 cannot shift because it already executes when j_a arrives. Therefore, we have $pos(j_3) = 3.75$, $pos(j_4) = 9.75$, $S_{left_2} = 3.75$ and $S_{right_2} = 0.25$. Then, we update the position of jobs in the middle of chains, in this case only j_a. So, $pos(j_a) = 7.75$, and the final system utility is 1.996+1.9645+0.484 = 4.4445, which is higher than the previous system utility, i.e 3. Hence, j_a is accepted, and the final schedule with the pendulum analogy is depicted in figure IV.4. As expected, the final schedule is exactly the same as in the previous example, which does not use intermediate values.

IV.2.5 Scheduling example using generic equilibrium

In this section, we show an example of how to schedule jobs using the generic equilibrium. The scheduling algorithm is the same as presented in section IV.2.1, but now we use the generic equilibrium to compute the compromise of jobs with conflicting target points. The task set is also the same for comparison of utility accrual, but we skip the on-line admission step; scheduling only the periodic tasks is enough to exemplify the use of the generic equilibrium.

The algorithm first places j_1 at time 2, and the system utility at this point is 1. Next, the algorithm schedules j_2 at time 2.5. However, j_1 finishes its execution at time 4, and hence, the equilibrium state between j_1 and j_2 is computed with the generic equilibrium (equation III.15). This equation is solved as a function of x_2 (deviation of the last job, i.e. j_2), and hence, the algorithm uses equation III.12 to define x_1 as a function of x_2, obtaining $x_1 = 2.5 - 2 - 2 + x_2 = x_2 - 1.5$. Then, using equation III.15:

$$\begin{aligned} g_2'(x_2) + g_1'(x_2 - 1.5) &= 0 \\ -\frac{2.5 \times x_2}{\sqrt{6.25 - x_2^2}} - \frac{0.5 \times (x_2 - 1.5)}{\sqrt{4 - (x_2 - 1.5)^2}} &= 0 \end{aligned} \qquad \text{(IV.23)}$$

Next, the algorithm uses equation system III.14 to find the interval where equation IV.23 is valid, obtaining $x_2 \in [-0.5, 2.5]$. The algorithm uses, then, a root finding algorithm (e.g. the bisection method) to find the value of $x_2 \in [-0.5, 2.5]$ that zeros out equation IV.23, obtaining $x_2 = 0.3486$. Hence, $x_1 = x_2 - 1.5 = -1.1514$, and the system

utility is $0.5 \times \sqrt{4-(-1.1514)^2} + 2.5 \times \sqrt{6.25-0.3486^2} = 7.0066$. So, j_1 shifts 1.1514 units of time to the left, and j_2 shifts 0.3486 units of time to the right of its target point, hence scheduled to execute at times 0.8486 and 2.8486, respectively. Then, we schedule j_3 at time 4 and no job is pushed, since j_2 finishes at time 3.8486. Finally, we schedule j_4 at time 8 and no job is pushed, since j_3 finishes exactly at time 8. The final system utility is $7.0066 + 2 + 1 = 10.0066$. This utility accrual is higher than in the schedule of section IV.2.4 because the generic equilibrium is optimum, and the particle equilibrium is not.

IV.3 Reordering the execution sequence of jobs

Although nature seeks the minimization of the potential energy in such a system, nature also tends to go to a local optimum. The basic premise is that if a lot of energy is needed to change the actual configuration of the system, then the configuration will not change. In fact, to the best of our knowledge, there is no closed formula to solve the bobs ordering in a pendulum, but rather statistical methods based on stochastic processes to come up with a possible configuration. Therefore, the pendulum analogy cannot be further exploited to solve the ordering problem.

In section IV.2, we used a intuitive simple static ordering by target points. The simple example depicted in figure IV.5 demonstrates the shortcoming of this simple static ordering for maximizing the accrued utility of the system. Consider 3 jobs j_1, j_2, and j_3 with the same WCET, anchor points equal to zero, $tp_2 = tp_3$ and $tp_1 = tp_2 - \epsilon$ ($\epsilon \mapsto 0$). Clearly, if $imp_2 = imp_3$ and $imp_2/imp_1 \mapsto 0$ (imp_1 is a significantly larger importance than those of the other tasks), then j_1's deviation will map to zero and j_3 will be approximately at a distance $2 \times WCET$ from its target point in the equilibrium state. A reasonable way to avoid this situation is to place j_3 on the left side of j_1, so that its deviation can be minimized (this time the deviation is equal to WCET) and, hence, accrue more utility to the system.

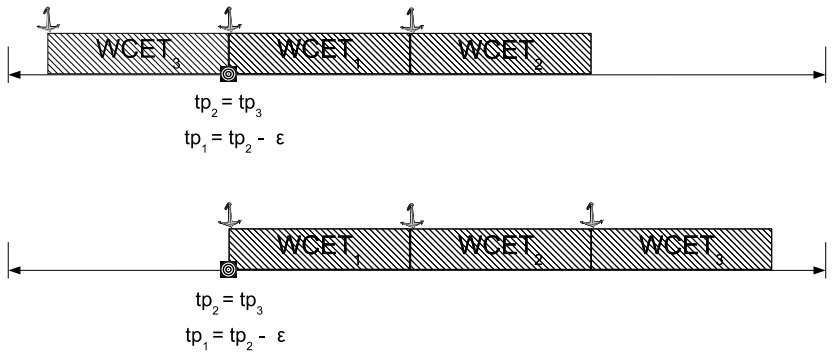


Figure IV.5: *Job ordering example.*

The heuristic we propose here to solve the ordering problem is based on a liquid system analogy. In a container with liquids of different densities, the higher the density

the closer to the bottom this liquid will come to rest. Mapping jobs to liquids and target points to the bottom of a container, jobs with higher density should execute closer to their target points. The same density principle has been used to develop heuristics to solve other optimization problems, like the knapsack problem [Martello 90], and the traveling salesman problem [Applegate 07].

The physical definition of density is the ratio of the mass to the volume of a body: density measures how tightly the matter is packed together. While in nature the mass of a body and gravity define the force that drags the body down, in a task set the importance of a job drags it to the target point. Similarly, a body occupies space (volume) and a job occupies time (WCET). Thus, we define the density of a job j_i as $dens_i = imp_i/WCET_i$. This density is called *utility density*.

The utility density represents the information of how much a job contributes to the accrued utility of the system per unit of execution. The current definition of utility depends only on the importance of the job, being independent of its WCET. Consider a job with a large and a job with a short WCET, but with the same importance and utility function. Although they provide the same utility to the system, the short job gives other jobs the chance to execute and contribute to maximize the accrued utility of the system. For instance, consider 3 jobs j_1, j_2 and j_3 with importances $imp_1 = imp_2 = imp_3 = imp$ and WCETs $WCET_1 = WCET_2 = WCET_3 = WCET$ and a job j' with importance $imp' = 2 \times imp$ and WCET $WCET' = 3 \times WCET$; all jobs have the same deviation from their target points. j_1, j_2 and j_3 execute for the same amount of time as j' and accrue more utility to the system, though j' has the highest importance.

We use the utility density to derive a job ordering similar to the ordering of liquids. A straightforward application of the analogy, however, is not practical: in a liquid system liquids have the same target, which is not the case in the task set environment. Besides, there is no best compromise concept; the liquid with higher density goes straight to the bottom and pushes the others up.

The next sections bring 3 different utility density based ordering heuristics: DST-1, DST-2 and DST-3. Those heuristics assume that job parameters are available before the job arrival, e.g. jobs are instances of strictly periodic tasks.

IV.3.1 Heuristic DST-1

This heuristic positions the anchor points of jobs with higher density directly at their target points. However, if positioning the anchor point of a job j_k directly at its target point results in an execution overlap with a previously scheduled job chain, the job is placed either on the left or on the right side of this job chain (wherever the job is closer to its target point in order to minimize the disturbance in the actual equilibrium state). Expression IV.25 returns on which side of the job chain to place j_k, where dev_{left} and $\text{dev}_{\text{right}}$ represent how far from the target point the anchor point lies if j_k is placed on the left or on the right side, respectively (see equation system IV.24 and figure IV.6). Then, the incoming job gets its own job chain, both chains are merged, and necessary

equilibrium recomputations are performed [1] as described in section IV.2.2.

$$\begin{aligned} \text{dev}_{\text{left}} &= (tp_k - pos(j_{fj})) + \alpha_{fj} \times WCET_{fj} + (1 - \alpha_k) \times WCET_k \\ \text{dev}_{\text{right}} &= (1 - \alpha_{lj}) \times WCET_{lj} + \alpha_k \times WCET_k + (pos(j_{lj}) - tp_k) \end{aligned} \tag{IV.24}$$

$$(\text{dev}_{\text{left}} < \text{dev}_{\text{right}}) \ ? \ \text{left} \ : \ \text{right} \tag{IV.25}$$

This heuristic tries to favor jobs that pack more utility per time of execution (density) by scheduling their execution first and as close as possible to their target points. Therefore, more jobs with high utility accrual contribution execute closer to their target points, and hence, less jobs have to execute far away from their target points. As a result, the negative impact of jobs competing for their target points decreases.

This heuristic has complexity $O(N \times log(N))$ to order the jobs by utility density, e.g. using an algorithm like *merge sort*, and complexity $O(1)$ to recompute the equilibrium every time chain merge happens. Finding the correct position and inserting a chain in the chain list has complexity $O(log(N))$. As jobs are never inserted in the middle of a job chain, chains can only merge. Therefore, the heuristic recomputes the equilibrium at most $N - 1$ times, and the final complexity of the heuristic is $O(N \times log(N))$.

Figure IV.6 depicts the heuristic with an example. In this scenario j_m is inserted in the schedule. Jobs j_k and j_l have higher densities and were already scheduled. α_m can be placed directly at tp_m because no execution overlap happens. That is not the case for job j_n, though. As a result, the heuristic determines the free spot where α_n is closer to tp_n, in this case the right side. Then, it computes the equilibrium for j_k, j_l, and j_n, checks for ripple effect, picks the next non-scheduled job with highest density, and repeats the heuristic steps until all jobs are scheduled.

A scheduling algorithm using heuristic DST-1 schedules a group of N jobs at once. For periodic tasks with deadline smaller than or equal to the period, we schedule all jobs in the hyper-period at the beginning of a hyper-period.

Example

Let us consider the task set used in section IV.2.4. The heuristic initially orders the jobs in decreasing order of utility density, obtaining $\tau_{2,1}$, $\tau_{3,1}$, $\tau_{1,1}$, and $\tau_{1,2}$ (see table IV.3). The heuristic places, then, $\tau_{2,1}$ at its target point. No execution overlap occurs and $\tau_{2,1}$ gets a job chain $c_{\tau_{2,1}}$ with $X_{\tau_{2,1}} = 0$, $Y_{\tau_{2,1}} = 2.5$, $Z_{\tau_{2,1}} = 0$, $S_{left_{\tau_{2,1}}} = 2.5$ and $S_{right_{\tau_{2,1}}} = 2.5$; these intermediate values are computed using equations IV.2 and IV.21. Next comes $\tau_{3,1}$, which can also execute at its target point without causing any execution overlap. $\tau_{3,1}$ gets a job chain $c_{\tau_{3,1}}$ with $X_{\tau_{3,1}} = 0$, $Y_{\tau_{3,1}} = 0.5$, $Z_{\tau_{3,1}} = 0$, $S_{left_{\tau_{3,1}}} = 4$ and $S_{right_{\tau_{3,1}}} = 4$.

[1] If we calculate dev_{left} and $\text{dev}_{\text{right}}$ after equilibrium recalculations, then in the worst case, after each job insertion, one choice will result in $N - 1$ merges, the other one no merge, and the latter will be the final choice. Hence, $N - 1$ unnecessary merges are computed N times, and the final complexity of the heuristic is $O(N^2)$.

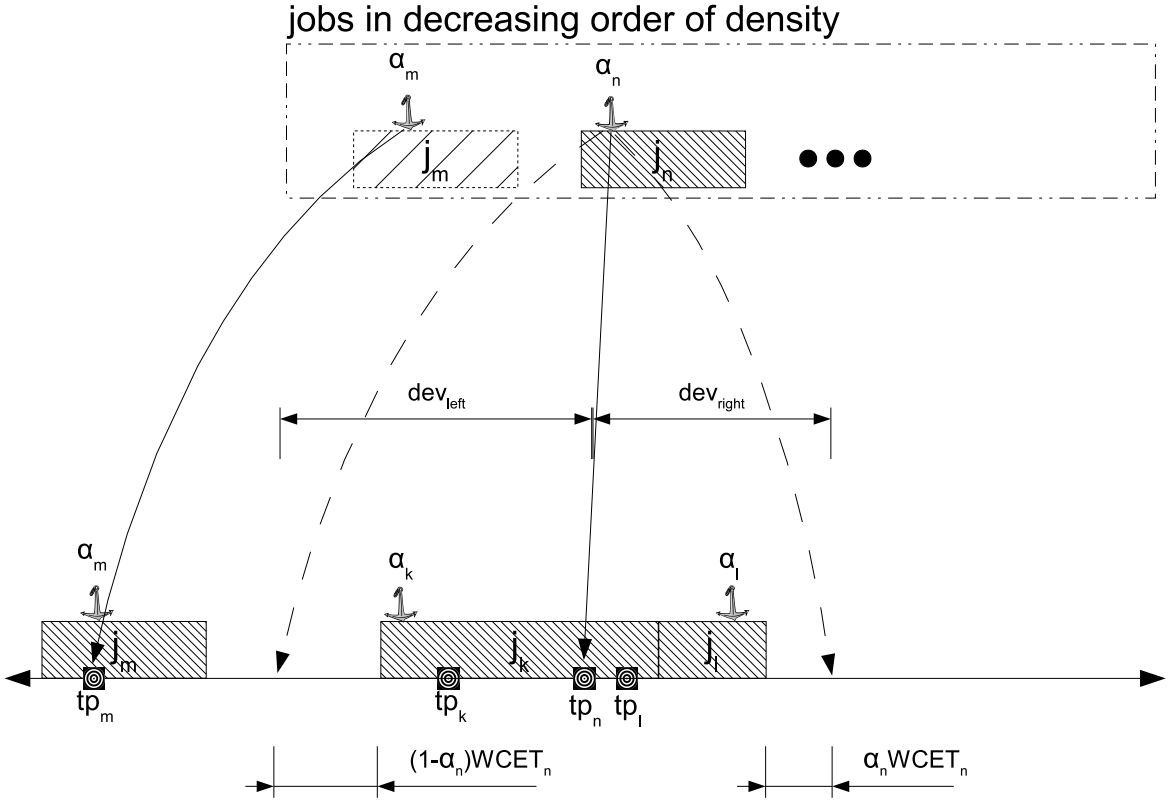


Figure IV.6: *Liquid based job ordering heuristic.*

job	$\tau_{1,1}$	$\tau_{2,1}$	$\tau_{3,1}$	$\tau_{1,2}$
density	0.5	2.5	0.5	0.5

Table IV.3: *Density of each job.*

The heuristic puts, then, $\tau_{1,1}$ at its target point, causing an execution overlap with $\tau_{2,1}$. The heuristic assigns to $\tau_{1,1}$ a job chain $c_{\tau_{1,1}}$ with $X_{\tau_{1,1}} = 0$, $Y_{\tau_{1,1}} = 0.5$, $Z_{\tau_{1,1}} = 0$, $S_{left_{\tau_{1,1}}} = 2$ and $S_{right_{\tau_{1,1}}} = 2$, and then, merges $c_{\tau_{1,1}}$ and $c_{\tau_{2,1}}$. The new chain $c_{\tau_{1,1},\tau_{2,1}}$ has intermediate values $X_{\tau_{1,1},\tau_{2,1}} = 2$, $Y_{\tau_{1,1},\tau_{2,1}} = 3$, $Z_{\tau_{1,1},\tau_{2,1}} = 2$. Then, using equation system IV.3, we obtain $pos(\tau_{2,1}) = 2.75$ and $pos(\tau_{1,1}) = 0.75$. Finally, using equation system IV.21, we obtain $S_{left_{\tau_{1,1},\tau_{2,1}}} = 0.75$ and $S_{right_{\tau_{1,1},\tau_{2,1}}} = 2.25$.

Then, the heuristic inserts $\tau_{1,2}$ in the schedule at its target point within a new chain $c_{\tau_{1,2}}$ with $X_{\tau_{1,2}} = 0$, $Y_{\tau_{1,2}} = 0.5$, $Z_{\tau_{1,2}} = 0$. No overlap happens, but chains $c_{\tau_{3,1}}$ and $c_{\tau_{1,2}}$ touch each other, and hence, are merged into a new chain $c_{\tau_{3,1},\tau_{1,2}}$ with $X_{\tau_{3,1},\tau_{1,2}} = 4$, $Y_{\tau_{3,1},\tau_{1,2}} = 1$ and $Z_{\tau_{3,1},\tau_{1,2}} = 4$. Using equation system IV.3 we obtain $pos(\tau_{1,2}) = 8$ and $pos(\tau_{3,1}) = 4$. Finally, using equation system IV.21, we obtain $S_{left_{\tau_{3,1},\tau_{1,2}}} = 2$ and $S_{right_{\tau_{3,1},\tau_{1,2}}} = 2$. The final system utility is $6.99929475 + 2 + 1 = 9.99929475$ and the final schedule has 2 job chains ($c_{\tau_{1,1},\tau_{2,1}}$ and $c_{\tau_{3,1},\tau_{1,2}}$).

IV.3.2 Heuristic DST-2

Heuristic DST-1 does not take the execution window into account to decide where to place the jobs, hence over compromising feasibility. Heuristic DST-2 slightly modifies DST-1 in order to account for feasibility when the insertion of a job in the schedule causes execution overlap. In DST-2, if positioning the anchor point of a job j_k directly at its target point results in an execution overlap with a previously scheduled job chain, the job is placed either on the left or on the right side of this job chain using the rule in expression IV.27.

$$\begin{aligned} \text{flex}_{\text{left}} &= \text{dev}_{\text{left}} - est_k - \alpha_k \times WCET_k \\ \text{flex}_{\text{right}} &= dl_k - (1 - \alpha_k) \times WCET_k - \text{dev}_{\text{right}} \end{aligned} \tag{IV.26}$$

$$\begin{aligned} &(\\ &\quad \Big((\text{dev}_{\text{left}} < \text{dev}_{\text{right}}) \text{ and } (\text{flex}_{\text{left}} > 0) \Big) \\ &\quad \text{or} \\ &\quad (\\ &\qquad \Big(\text{not } \Big((\text{dev}_{\text{right}} < \text{dev}_{\text{left}}) \text{ and } (\text{flex}_{\text{right}} > 0) \Big) \Big) \\ &\qquad \text{and} \\ &\qquad \Big(\text{flex}_{\text{left}} - \text{dev}_{\text{left}} > \text{flex}_{\text{right}} - \text{dev}_{\text{right}} \Big) \\ &\quad) \\ &) \ ? \ \text{left} \ : \ \text{right} \end{aligned} \tag{IV.27}$$

Additionally to the terms dev_{left} and $\text{dev}_{\text{right}}$, this expression introduces the flexibility terms $\text{flex}_{\text{left}}$ and $\text{flex}_{\text{right}}$ (see equation system IV.26 and figure IV.7). These flexibility terms represent, respectively, the distance from est_k, if j_k goes to the left side, and dl_k, if j_k goes to the right side of the job chain. This expression places the job on the side that minimizes the distance from the target point, given that on this side the respective flexibility term remains positive. Else, a compromise between the deviation terms and the flexibility terms is used for decision. A job tends to go to the left side for lower dev_{left} and higher $\text{flex}_{\text{left}}$; analogously, a job tends to go to the right side for lower $\text{dev}_{\text{right}}$ and higher $\text{flex}_{\text{right}}$. This compromise is given by the expression $(\text{flex}_{\text{left}} - \text{dev}_{\text{left}} > \text{flex}_{\text{right}} - \text{dev}_{\text{right}})$? left : right.

The rest of the heuristic remains exactly as DST-1, and hence, DST-2 has complexity $O(N \times log(N))$ as well.

IV.3.3 Heuristic DST-3

This heuristic ordering sorts an incoming job in the ready queue so that this job executes at its target point. In case its execution overlaps with other jobs, the heuristic searches for the closest place to the target point with either an idle period or a job with lower density. The job is inserted, then, in this position. Both searching for the correct

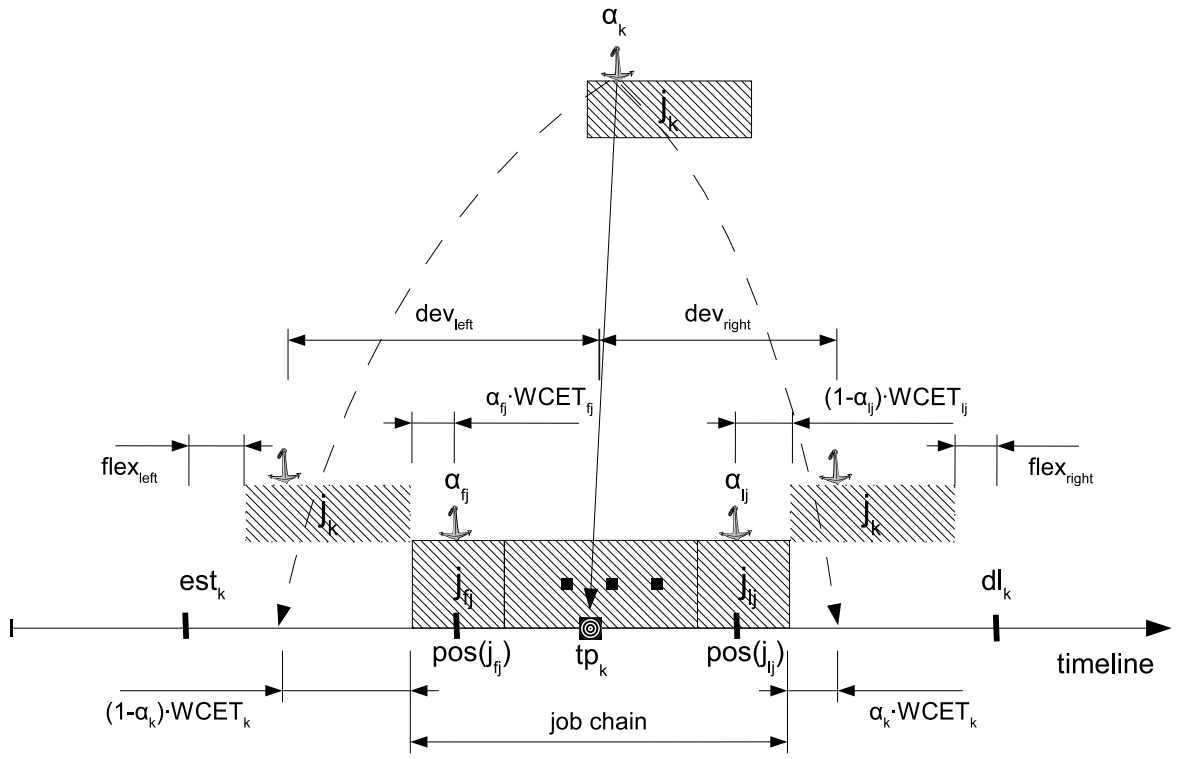


Figure IV.7: *Parameters of equation system IV.26.*

position and the insertion operation have linear complexity. Therefore, this heuristic ordering has linear complexity.

Consider the situation depicted in figure IV.8, where j_k, j_l, j_m and j_n are jobs already scheduled to execute and $density(j_k) > density(j_l) > density(j_m) > density(j_n)$. tc represents the current time at which an aperiodic job j' arrives in the system with anchor point α', target point tp' and $density(j_l) > density(j') > density(j_m)$. α' cannot be positioned at tp' because the execution of j' would overlap with the execution of other jobs. So, we scan the left and the right side of tp' looking for either a free spot or a job with less density. To the left side we find first a free spot before j_k; to the right side we find j_n, whose density is smaller than $density(j')$. At this point we have 2 options: either position j' before j_k or after j_l. We choose, then, the side where α' will be closer to tp', i.e., the right side.

Upon job arrival, the equilibrium might schedule the execution of some jobs before the current time tc. In order to avoid this phenomenon, at time tc the schedule must also consider jobs that will arrive within an intervals of time in the future of length ew, which we call *equilibrium window*. Therefore, we anticipate the job arrivals by ew units of time, i.e. a job arriving at time $tc + ew$ is taken into account in the schedule at time tc. ew must be large enough to make sure that jobs arriving after $tc + ew$ cannot change the schedule at time tc during equilibrium recomputations. Theorem IV.1 proves that, for ew equal to the length of the largest job chain, the new equilibrium state never

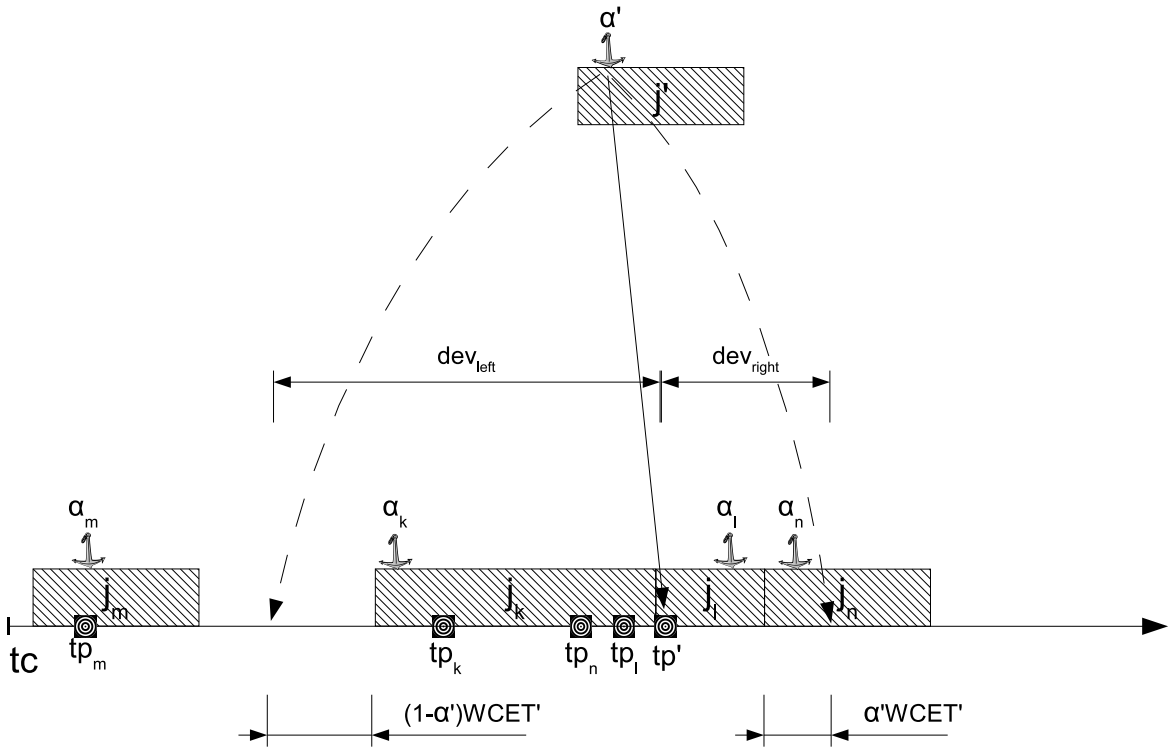


Figure IV.8: *The heuristic ordering DST-3.*

causes a change of the schedule at time tc.

Theorem IV.1. *If ew is larger than or equal to the length of the longest job chain, no job j arriving at $tc + ew$ or later can change the schedule at time tc or before.*

Proof. Assume that all jobs available in the system in the interval $[tc, tc+ew[$ are already scheduled (i.e. in equilibrium). Lets assume that, after j is inserted in the schedule and the equilibrium is recomputed, j belongs to the job chain c_k. Jobs whose execution are scheduled before the execution of j will have their schedule altered if, and only if, they belong to c_k (the pendulum analogy helps to understand this phenomenon). Therefore, the schedule in tc will only be changed if c_k reaches time t. However, c_k can reach time tc and contain job j if, and only if, the length of c_k is larger than ew, which contradicts the initial assumption that ew is larger than or equal to the longest job chain. □

The scheduling algorithm with heuristic DST-3 consists of an *off-line* and an *on-line part*.

Off-line. First, the scheduler sorts all jobs by arrival time. If the jobs are instances of periodic tasks, the scheduler considers all jobs until the next point in time that the schedule repeats itself. For instance, the schedule of synchronized periodic tasks with deadlines smaller than or equal to the period repeats itself at the end of the first hyper-period. Then, for each job in increasing order of arrival time, the scheduler inserts the

job in the schedule using heuristic DST-3, and then applies the equilibrium as described in sections IV.2.1 and IV.2.2, which also checks for feasibility. Once all jobs are in the schedule and meet their deadlines, the scheduler sets the equilibrium window ew to the length of the longest job chain.

Online. At system startup, all jobs that arrive in the interval $[0, ew[$ are in the scheduling table. The scheduler, then, includes new jobs in the scheduling table at runtime when their arrival time is $tc + ew$, and updates the scheduling table appropriately (apply heuristic DST-3 and compute necessary equilibriums). Upon completion of a job, its entry is removed from the scheduling table. Notice that, despite of having a scheduling table the method is on-line, and the scheduling table does not contain entries for all jobs that will execute in the life time of the system. Table entries are only for jobs that are ready for execution, and jobs that will arrive within an interval of time ew. If the system is feasible in the off-line part, it is also feasible in the on-line part, as each job has exactly the same schedule in both parts.

The complexity to schedule the execution of a job upon its anticipated arrival involves the complexity of choosing the order to insert it in the schedule, and potential recomputations of equilibrium in the case of ripple effects. The ordering has complexity $O(N)$, and the number of equilibrium recomputations due to merges cannot be larger than $N - 1$, where N is the number of jobs in the scheduling table at tc. Performing all necessary updates in the scheduling table has linear complexity as well. Therefore, scheduling the execution of a job has linear complexity.

IV.4 Evaluation

In this section, we initially describe the simulation setup. Then, we describe the 4 sets of experiments performed, and their respective results. The first set of experiments compares the pendulum equilibrium with the generic equilibrium; the second set of experiments compares scheduling algorithms using the different heuristics to order the execution sequence of jobs; in the third set of experiments, we compare a scheduling algorithm using the pendulum equilibrium and heuristic DST-3 with Generic Utility Scheduler (GUS) [Li 06], and EDF with different execution windows; finally, in the fourth set of experiments, we compare heuristic DST-3 with the optimum solution.

IV.4.1 Simulation setup

In our simulations, task sets comprise periodic tasks and the system utilization varies in the range $[0.1, 0.9]$ with granularity 0.1. Each utilization category comprises 1000 randomly generated task sets, including infeasible ones. The number of periodic tasks in each task set is a random integer uniformly distributed in the interval $[2, 10]$. The period and importance of each task are integer numbers uniformly distributed in the interval $[1, 10]$. Deadlines are equal to the period, earliest start times are equal to 0 and target points lie in the middle of the execution window of the jobs, thus allowing

the same amount of deviation to both sides of the target point. The computation times were uniformly distributed such that the generated task set has the desired utilization. In each schedule, we considered all jobs within the hyper-period and ordered them by their target points. Our results are within a confidence level of 95% with significance level of 0.05.

We consider 5 different types of continuously differentiable utility functions (where $R_i = \frac{dl_i - WCET_i}{2}$):

- $g1_i(x_i) = imp_i \times \sqrt{1 - \left(\frac{x_i}{R_i}\right)^2}$
- $g2_i(x_i) = imp_i \times \sqrt{1 - \left(\frac{x_i}{R_i}\right)^4}$
- $g3_i(x_i) = imp_i \times \left(1 - \left(\frac{x_i}{R_i}\right)^4\right)$
- $g4_i(x_i) = imp_i \times \left(2 - \frac{e^{\frac{1.31695 \times x_i}{R_i}} + e^{-\frac{1.31695 \times x_i}{R_i}}}{2}\right)$
- $g5_i(x_i) = imp_i \times \left(1 - \left(\frac{x_i}{R_i}\right)^2\right)$

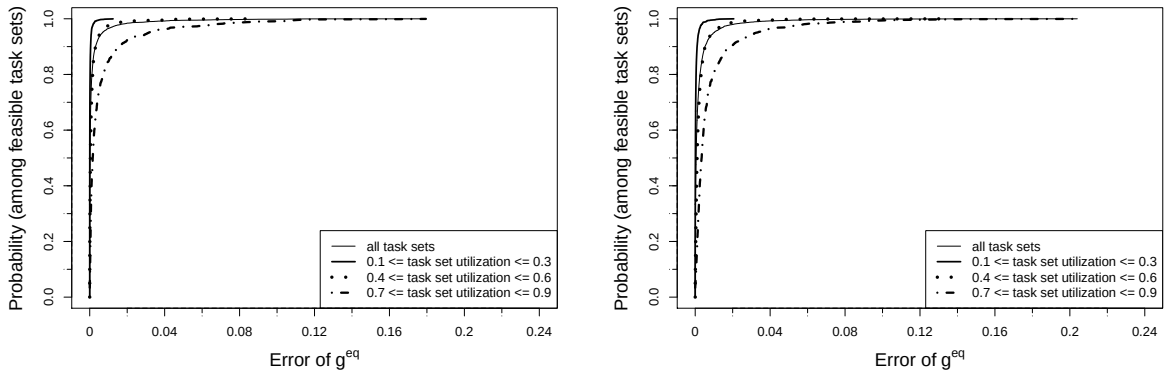
The respective derivatives are:

- $g1_i'(x_i) = -\frac{imp_i \times \left(\frac{x_i}{R_i}\right)}{R_i \times \sqrt{1 - \left(\frac{x_i}{R_i}\right)^2}}$
- $g2_i'(x_i) = -2 \times \frac{imp_i \times \left(\frac{x_i}{R_i}\right)^3}{R_i \times \sqrt{1 - \left(\frac{x_i}{R_i}\right)^4}}$
- $g3_i'(x_i) = -4 \times imp_i \times \left(\frac{x_i}{R_i}\right)^3 \times \frac{1}{R_i}$
- $g4_i'(x_i) = -imp_i \times \frac{1.31695}{R_i} \times \left(\frac{e^{\frac{1.31695 \times x_i}{R_i}} - e^{-\frac{1.31695 \times x_i}{R_i}}}{2}\right)$
- $g5_i'(x_i) = -2 \times imp_i \times \left(\frac{x_i}{R_i^2}\right)$

The utility function $g1_i(x_i)$ is the same as in equation III.10 after a few algebraic steps to convert from the pendulum domain to the task set domain, and $g5_i(x_i)$ is a quadratic polynomial function as in equation III.16 (see section III.6.1). The other utility functions are arbitrary.

IV.4.2 Experiment set 1

In the first set of experiments, we compare the pendulum equilibrium with the generic equilibrium. The metric of comparison is the accrued utility of each task set; acceptance ratio is left out of this comparison because we are interested in evaluating the equilibriums, rather than the scheduling algorithm example. Therefore, we calculate for each feasible task set the error of the accrued utility obtained by the pendulum equilibrium, which is $1 - \frac{g^{eq}}{g^{gen}}$ (g^{eq} stands for the utility accrual of the pendulum equilibrium and g^{gen} stands for the utility accrual of the generic equilibrium, which is optimum).



(a) *Jobs with utility function* $g1_i(x_i)$

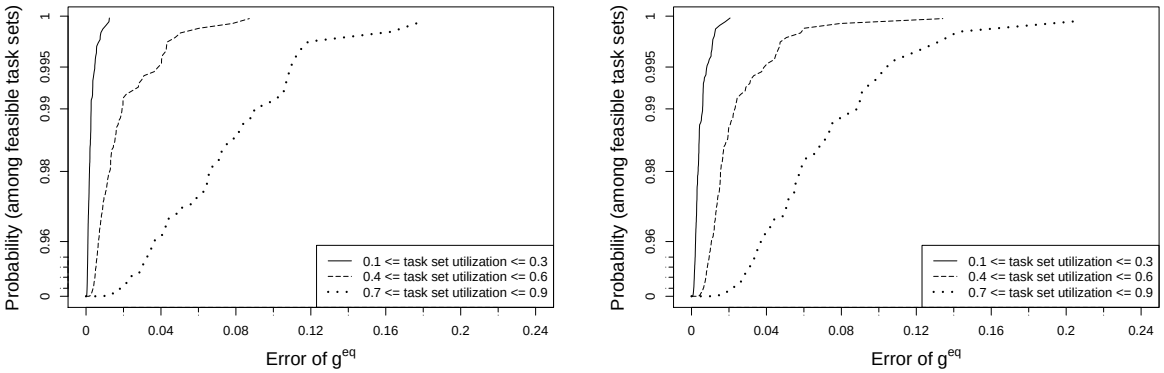
(b) *Jobs with either utility function* $g1_i(x_i)$, $g2_i(x_i)$, $g3_i(x_i)$, $g4_i(x_i)$ *or* $g5_i(x_i)$

Figure IV.9: *C.D.F. of error of* g^{eq} *(job ordered by target point)*

In the experiment depicted in figure IV.9a, all jobs have utility function $g1_i(x_i)$, and are ordered by target point. To the best of our knowledge, a closed-form of the generic equilibrium for elliptical functions is not possible; we solve the generic equilibrium using the bisection method [Sun 06]. This figure contains the cumulative distribution functions (C.D.F.) of the error of g^{eq}. Task sets are categorized by their utilization in the ranges $[0.1, 0.3]$, $[0.4, 0.6]$ and $[0.7, 0.9]$, and we also plot the C.D.F. of the error considering all task sets. Looking at the line that represents all task sets we observe that the error of g^{eq} is smaller than 2% in almost 100% of the cases. Under low system utilization jobs rarely overlap execution when scheduled at their target points. Hence, we can see that for task set utilization within $[0.1, 0.3]$ the equilibrium and the generic equilibrium have very close results. As the utilization increases, the approximation of the pendulum equilibrium loses accuracy. However, the line for task set utilization within $[0.7, 0.9]$ shows that even for task sets with high utilization the error of g^{eq} is smaller than 2% in approximately 90% of the cases. Therefore, the approximation obtained by g^{eq} is very good.

In figure IV.10a, we plot the same results with polynomial scale to the power 40 in the y-axis. The line for task set utilization within $[0.7, 0.9]$ shows that the error in the pendulum equilibrium is smaller than 4% in approximately 96% of the cases, but in 4%

of the cases this error ranges from 4% to approximately 18%. This superiority of the generic equilibrium does not come at the expense of a higher complexity.



(a) *Jobs with utility function* $g1_i(x_i)$

(b) *Jobs with either utility function* $g1_i(x_i)$, $g2_i(x_i)$, $g3_i(x_i)$, $g4_i(x_i)$ *or* $g5_i(x_i)$

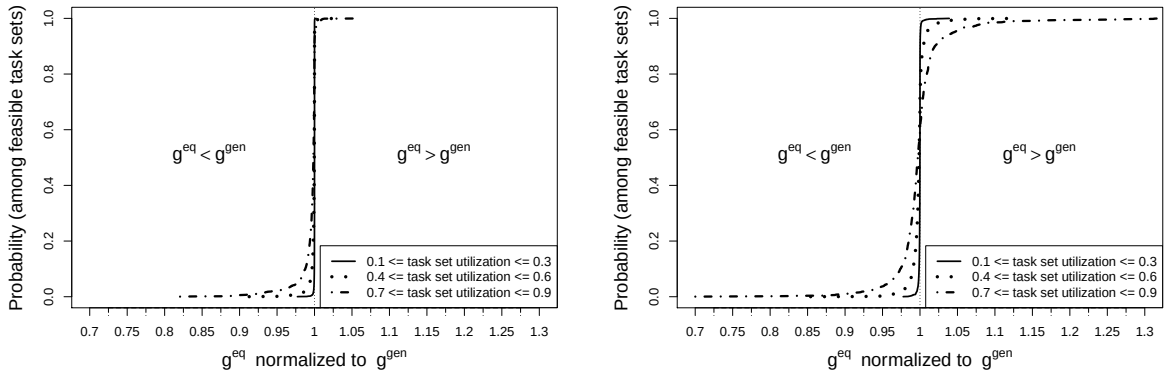
Figure IV.10: *C.D.F. of error of* g^{eq} *(job ordered by target point)*

In the experiment depicted in figure IV.9b, each job has one of the 5 utility functions listed earlier. All other parameters remain as in the previous experiment. Once again we observe that for task set utilization within $[0.1, 0.3]$ the original equilibrium and the generic equilibrium have very close results, and that for high system utilization the error of g^{eq} is smaller than 2% in approximately 90% of the cases. In fact, the experiments depicted in figures IV.9a and IV.9b have very similar results, revealing that the approximation obtained by the pendulum analogy holds even for variable shapes of utility functions.

In figure IV.10b, we plot the same results of the previous experiment with polynomial scale to the power 40 in the y-axis. Once again, the line for task set utilization within $[0.7, 0.9]$ shows that the error in the pendulum equilibrium is smaller than 4% in approximately 96% of the cases, but in 4% of the cases this error ranges from 4% to approximately 20%. Therefore, the shape of the functions do not have much impact on the approximation of the pendulum equilibrium.

In the experiments depicted in figures IV.11a and IV.11b, we investigate how the choice of equilibrium calculation impacts on the utility accrual when the ordering heuristic depends on the equilibrium calculation. We use heuristic DST-3 in these experiments. This figure contains the plot of g^{eq} normalized to g^{gen} for each feasible task set, and categorized by system utilization. Values smaller than 1 indicate that $g^{eq} < g^{gen}$, equal to 1 indicate that $g^{eq} = g^{gen}$ and higher than 1 indicate that $g^{eq} > g^{gen}$.

In the experiment depicted in figures IV.11a, jobs have elliptical utility functions. We can observe that scheduling a task set using g^{eq} might accrue more utility than using g^{gen}. This is a consequence of the different order of jobs' execution, which impacts on the utility accrual. However, in most of the cases the generic equilibrium provides for



(a) *Jobs with utility function* $g1_i(x_i)$

(b) *Jobs with either utility function* $g1_i(x_i)$, $g2_i(x_i)$, $g3_i(x_i)$, $g4_i(x_i)$ *or* $g5_i(x_i)$

Figure IV.11: *C.D.F. of* g^{eq} *normalized to* g^{gen} *(jobs ordered using heuristic DST-3)*

higher utility accrual.

In the experiment depicted in figure IV.11b, jobs may have any of the 5 utility functions listed in section IV.4.1. Here, independently of the utilization category, all curves are symmetric around point 1. This result reveals that the ordering heuristic is also sensitive to the type of utility functions, and in this case, the method for equilibrium calculation does not make much difference in the utility accrual.

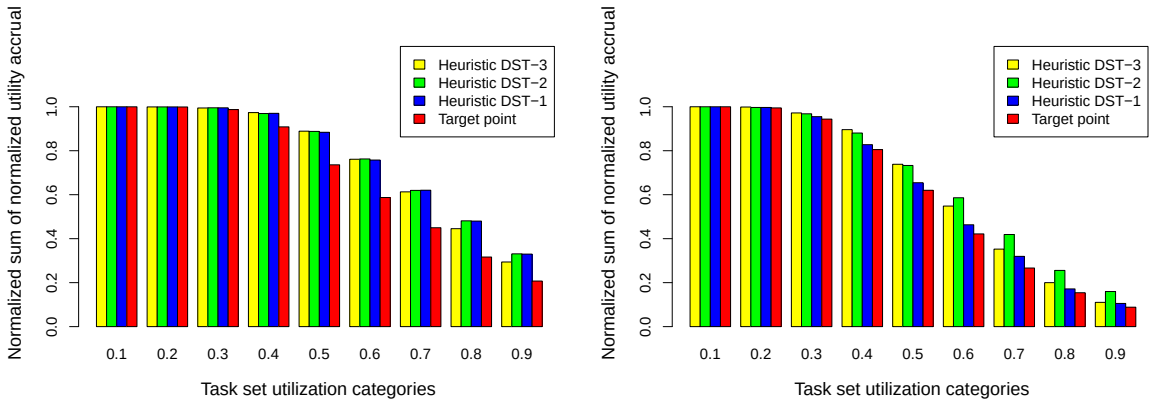
We conclude from this first set of experiments that the approximation of the equilibrium obtained from the pendulum analogy is very close to the optimum solution, independent of the shape of the utility functions. However, in a very few cases the generic equilibrium may increase the utility accrual up to 20%. Moreover, the generic equilibrium may not improve the decisions of the ordering heuristic if jobs have arbitrary utility functions. The particle equilibrium has the additional advantage of constant complexity for equilibrium recomputations, which allows for the design of a scheduling algorithm with linear complexity (as described in sections IV.2.1 and IV.2.2).

IV.4.3 Experiment set 2

In the second set of experiments, we compare all ordering heuristics. The scheduling algorithms use the pendulum equilibrium, and jobs have utility function $g1_i(x_i)$, which is the implicit utility function of the gravitational task model. This way we focus only on the impact of ordering on the system utility accrual. As metrics of comparison we consider the utility accrual and the acceptance ratio of task sets.

We compute the performance of a given schedule in accruing utility as the accrued utility normalized to the utility that would be accrued if all jobs executed directly at their target points. Although executing all jobs directly at their target points might be infeasible, it gives an upper bound to define how good a schedule is from the utility accrual point of view. Moreover, this quotient is also comparable among different utilization categories, hence giving an insight on the impact of the system utilization

on the utility accrual. For each utilization category, we plot the sum of these quotients divided by the number of task sets in the category (including unfeasible task sets, which we consider to accrue no utility to the system). In our experiments, the number of task sets per utilization category is 1000.



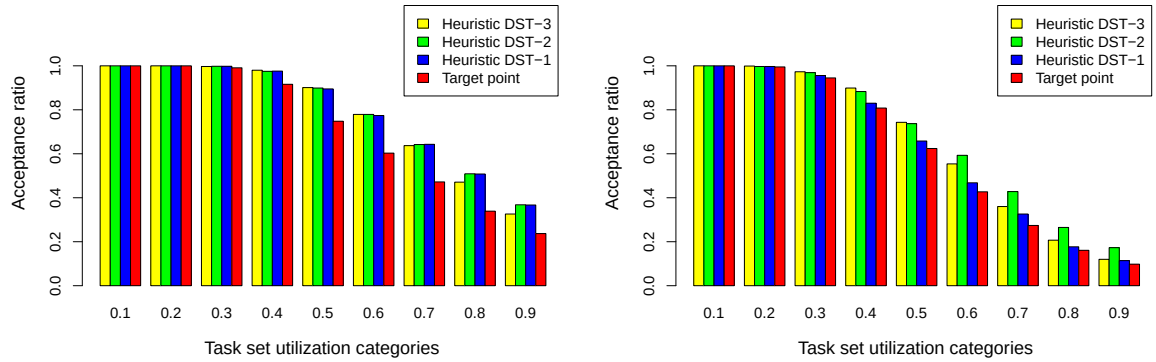
(a) *Target points equal to* 0.5 **(b)** *Target points variable*

Figure IV.12: *Utility accrual for all heuristics*

The experiment depicted in figure IV.12 contains the utility accrual for each schedule under 2 simulation scenarios: target points in the middle of the execution window, and target points randomly distributed within the execution window. We can observe that heuristics based on the utility density of jobs (DST-1, DST-2, and DST-3) outperform the static ordering by target points in both scenarios. Furthermore, DST-3 has statistically identical results to DST-1 and DST-2. This result shows that the lower complexity of DST-3 does not come at the expense of decreased utility accrual. We can also see that the trade-off among jobs with conflicting target points has more impact on the utility accrual for system utilization larger than or equal 40%, which is the point where the utility accrual starts to decrease. Comparing between both scenarios, we can observe that variable target points have a negative impact on the utility accrual for high system utilization. Heuristic DST-1 suffers more utility degradation due to its inability to account for the execution window of jobs.

The experiment depicted in figure IV.13 contains the utility accrual for each schedule under 2 simulation scenarios: target points in the middle of the execution window, and target points randomly distributed within the execution window. Interestingly, both plots in this figure are identical to the respective results for the utility accrual in figure IV.12. Obviously, the system utilization has a negative impact on both acceptance ratio and utility accrual. This result reveals that the system utilization has actually the same impact on both acceptance ratio and utility accrual.

These results reveal that DST-2 obtains the best results for utility accrual, but at the expense of a high complexity. On the other hand, DST-3 obtains results close to



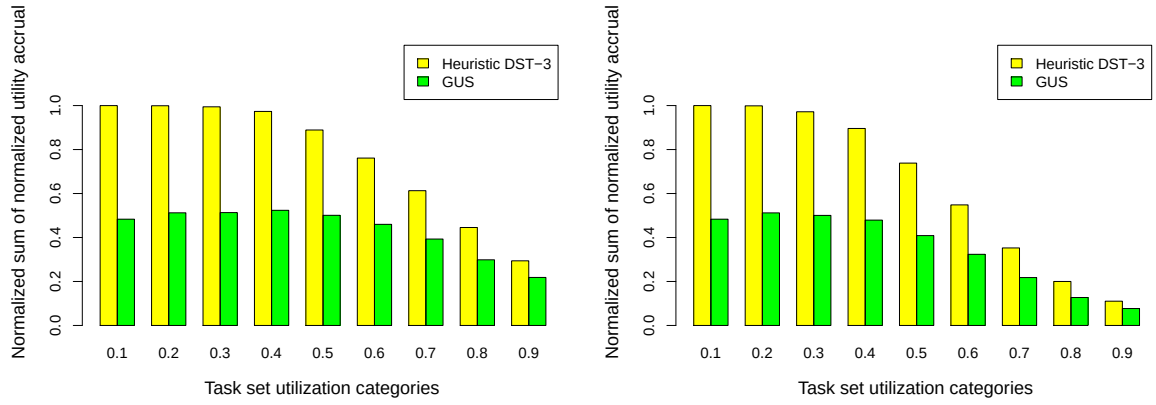
(a) *Target points equal to* 0.5 **(b)** *Target points variable*

Figure IV.13: *Acceptance ratio for all heuristics*

DST-2 at the expense of a linear complexity. This complexity significantly reduces the scheduling overhead in comparison to the other heuristics.

IV.4.4 Experiment set 3

In the third set of experiments, we compare the gravitational task model with the Time Utility Function (TUF) task model, and with the deadline based task model. The scheduling algorithms are DST-3 with pendulum equilibrium, GUS [Li 06], and non-preemptive EDF with constrained execution window. As metrics of comparison we consider the utility accrual and the acceptance ratio of task sets. We use the same simulation scenarios as in experiment set 2 (see section IV.4.3).

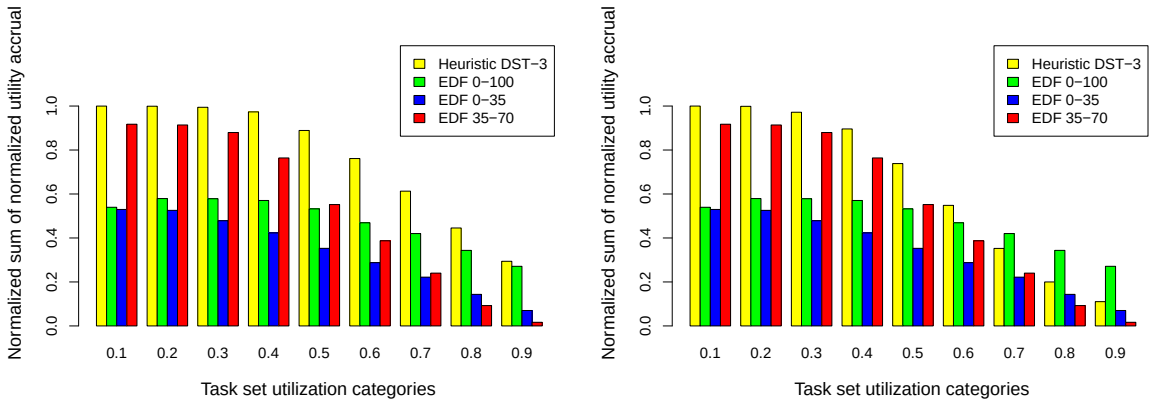


(a) *Target points equal to* 0.5 **(b)** *Target points variable*

Figure IV.14: *Utility accrual for DST-3 and GUS [Li 06]*

In figure IV.14 we compare DST-3 with GUS, which is a non-preemptive TUF sched-

uler that considers the utility of a job only at the current point in time to take scheduling decisions. Therefore, GUS does not account for the target sensitivity of tasks. GUS also aborts the execution of jobs that will miss their deadlines because they accrue no utility to the system. Therefore, this figure contains plots only for the utility accrual, as an comparison of acceptance ratio is meaningless. In order to fairly compare both algorithms, we consider only schedules where all jobs execute and meet their deadlines. As can be seen, DST-3 accrues more utility due to its capability to account for the target points, and the task set utilization has a negative impact on the utility accrual of both algorithms. Furthermore, we observe that variable target points also have a negative impact on GUS as the system utilization increases.



(a) *Target points equal to* 0.5

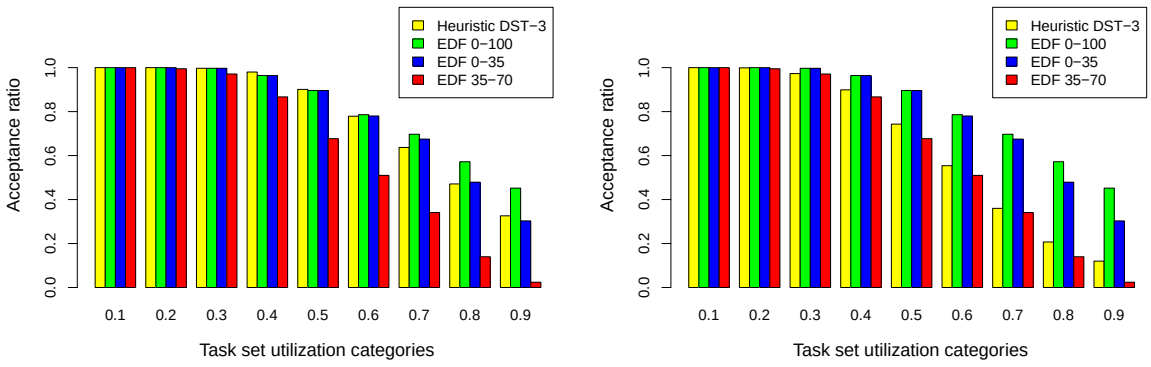
(b) *Target points variable*

Figure IV.15: *Utility accrual for DST-3 and EDF*

In figures IV.15 and IV.16, we compare DST-3 with non-preemptive EDF where tardiness is set to the period (original deadline), and we vary the values of offsets and deadlines — EDF expresses target sensitivity by tightening the execution window of tasks. In other words, we schedule the tasks under EDF using the constrained deadline, but a task set is infeasible if at least one task misses the original deadline. We have 3 configurations for EDF: no offset and deadline equal to the period (EDF 0-100), no offset and deadline 35% of the execution window (EDF 0-35), and offset 35% of the execution window with deadline 70% of the execution window (EDF 35-70).

In figure IV.15, we plot the utility accrual of the scheduling algorithms. As can be seen in this figure, scheduling the task sets under EDF only constraining the deadline to 35% of the original deadline (EDF 0-35) is not enough to improve the utility accrual, since the target point is in the middle of the execution window. Adjusting also the offset (EDF 35-70) improves the utility accrual of task sets with low utilization significantly, but large offsets degrade the utility accrual of task sets with high utilization. Of course, EDF without any adjustment does not achieve high utility accrual under low system utilization, but this result changes for high system utilization. We can observe in figure IV.15a that for system utilization above 80% EDF 0-100 and DST-3 have statistically identical results. In the experiment of figure IV.15b, where we vary the location

of the target points, EDF 0-100 may achieve values higher than DST-3, but only at very high system utilization.; DST-3 is still superior in most cases. The values for EDF in both figures IV.15a and IV.15b do not change because only the target points vary among scenarios, and EDF is unaware of target points.



(a) *Target points equal to* 0.5 **(b)** *Target points variable*

Figure IV.16: *Acceptance ratio for DST-3 and EDF*

In figure IV.16, we plot the acceptance ratio of the scheduling algorithms. As can be seen in this figure, EDF 0-100 and EDF 0-35 have very high acceptance ratio compared to the other approaches, and that large offset decrease the acceptance ratio of EDF. However, the offset is a necessary artifact to improve the utility accrual under low system utilization. On the other hand, using the gravitational task model the negative impact on the feasibility of the task sets is small (see figure IV.16a), and the utility accrual significantly increases. However, variable target points negatively impact on the acceptance ratio of DST-3, which also explains the low utility accrual in this scenario (see figure IV.15a). We conclude, then, that scheduling algorithms for the gravitational task model may further increase in the utility accrual by increasing the acceptance ratio.

In the next chapter, we propose a scheduling algorithm for the gravitational task model based on both EDF — in order to account for feasibility — and utility density — in order to account for utility accrual. We will observe significant improvement in the utility accrual over the heuristics presented in this chapter.

IV.4.5 Experiment set 4

In this last experiment, we compare the heuristic DST-3 with the optimum execution sequence of jobs, which we obtain with an exhaustive search — we use the generic equilibrium for scheduling. Due to the complexity of the optimum solution, we have to severely restrict the number of jobs in the task set. Therefore, we generate 9000 task sets with 10 jobs which have the same execution windows and target points, and all task sets are feasible. This setup allows us to evaluate only the impact of the execution

sequence of jobs on the utility accrual. The generation of other random parameters remain as described in section IV.4.1. As can be seen in figure IV.17, the utility accrual of heuristic DST-3 is very close to the optimum among feasible scenarios. As expected, the error increases as the utilization of the task set increases.

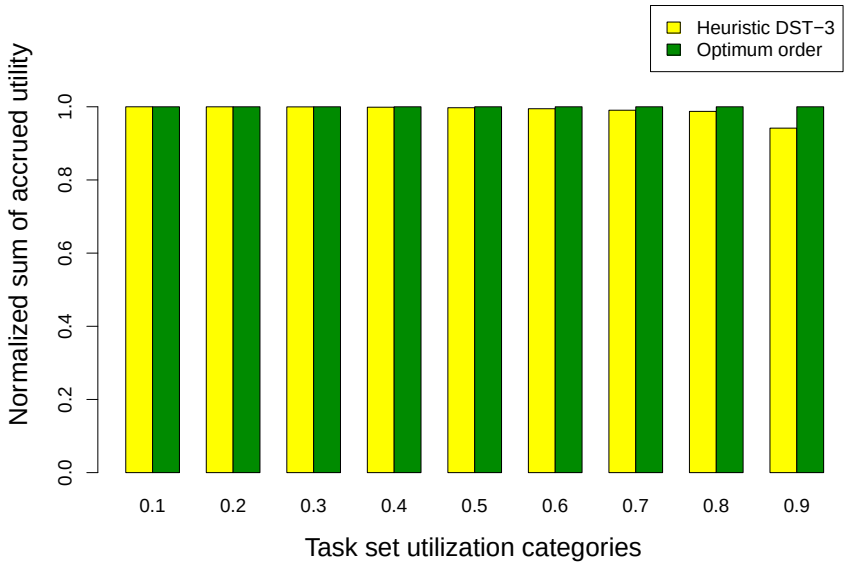


Figure IV.17: *Utility accrual for heuristic DST-3 and optimum order.*

IV.5 Summary

In this chapter, we presented a few scheduling algorithms for the gravitational task model. Initially, we presented in section IV.2 how to schedule jobs using the equilibrium to compute trade-offs. Section IV.2.1 describes a method inspired by the bob pendulum to find job chains in a schedule. Appending bobs one by one in a pendulum reveals which groups of jobs push one another, and the equilibrium state changes upon collisions — appending one bob may cause more than one collision. Similarly, the algorithm to find job chains appends jobs one by one to the schedule, and recomputes the equilibrium for job chains that merge; execution sequence of jobs must be known. Appending one job might cause more than one merge (*ripple effect*), but there can be no more than N merges; chains do not split upon job append, only merge. Therefore, the final complexity is $O(N^2)$. This algorithm allows the equilibrium to schedule jobs that do not compete altogether with one another for their target points.

In section IV.2.2, we presented a method with constant complexity to recompute the equilibrium of 2 adjacent chains that merge. As those equilibrium recomputations have

constant complexity, and there can be no more than $N-1$ merges, the complexity to schedule jobs is linear. This method is valid for trade-off calculations using the particle pendulum equilibrium, which we described in chapter III. The basic idea is to store intermediate values for each job chain in order to save computational steps when computing the equilibrium of the new job chain which appears upon merge. We called those intermediate values X, Y, Z, S_{left}, and S_{right}. The 3 former values are used for equilibrium recomputations (see equation IV.3), and the 2 latter ones are used to identify timing constraint violations. S_{left} and S_{right} express how much the chain can shift to the left or right, respectively, without violating any constraint. Equations IV.2 and IV.21 compute those values for a job chain visiting all jobs, and equations IV.20 and IV.22 compute those values using the intermediate values of adjacent merged chains.

In section IV.2.3, we described the on-line job admission procedure. Upon on-line job admission, the incoming job may be inserted anywhere in the schedule. In this case, there can be at most one chain split and N chain merges. Upon chain split, the intermediate values of the separated chains are computed using equations IV.2 and IV.21; merges are handled as before. Therefore, on-line admission also has linear complexity. Sections IV.2.4 and IV.2.5 illustrated the scheduling algorithm with examples.

In section IV.3, we presented the importance of the execution sequence of jobs in a scheduling algorithm, and a few ordering heuristics. The execution sequence of jobs dominates schedulability and utility maximization problems. Moreover, it is an NP-hard problem — check all possible permutations for the execution of jobs. Therefore, an optimum solution has high overhead, and is unfeasible for on-line scheduling algorithms. Heuristic solutions compromise between overhead, acceptance ratio, and utility accrual; affording higher overhead tends to provide for better scheduling decisions. We also presented a few ordering heuristics which are based on the rationale of physics for fluids. The basic idea is that in a container with liquid of different densities, the higher the density the closer to the bottom it will come to rest. Mapping jobs to liquids and target points to the bottom of a container, jobs with higher density should execute closer to their target points.

The physical definition of density is the ratio of the mass to the volume of a body: density measures how tightly the matter is packed together. While in nature the mass of a body and gravity define the force that drags the body down, in a task set the importance of a job drags it to the target point. Similarly, a body occupies space (volume) and a job occupies time (WCET). Thus, we defined the density of a job j_i as $dens_i = imp_i/WCET_i$. This density is called *utility density*.

In section IV.3.1, we presented heuristic DST-1, which has complexity $O(N \times log(N))$. This heuristic tries to favor jobs that pack more utility per time of execution (density) by scheduling their execution first, and as close as possible to their target points. If positioning the anchor point of a job directly at its target point results in an execution overlap with a previously scheduled job chain, the heuristic places the job either on the left or on the right side of this job chain (wherever the job is closer to its target point in order to minimize the disturbance in the actual equilibrium state). Therefore, more jobs with high utility accrual contribution execute closer to their target points, and hence, less jobs have to execute far away from their target points. As a result,

the negative impact of jobs competing for their target points decreases. However, this heuristic does not consider the execution window of jobs for taking ordering decision, hence compromising feasibility. This section also contains a scheduling example.

In section IV.3.2, we presented heuristic DST-2, which slightly modifies DST-1 in order to account for feasibility when the insertion of a job in the schedule causes execution overlap. In DST-2, if positioning the anchor point of a job directly at its target point results in an execution overlap with a previously scheduled job chain, the heuristic places the job either on the left or on the right side of this job chain using the rule in expression IV.27. This expression finds a compromise between deviations from the target points and how close the execution of jobs lie to the edge of their execution windows. This heuristic also has complexity $O(N \times log(N))$.

In section IV.3.3, we presented heuristic DST-3, which modifies heuristic DST-1 and DST-2 to avoid reordering of the execution sequence upon job arrival, and hence, has linear complexity. In this heuristic, the jobs considered in the schedule at run-time are all jobs ready for execution plus all jobs that will arrive within an interval of time called *equilibrium window*. This equilibrium window is necessary to account for the disturbance of jobs that have not yet arrived in the system on the schedule at the current time. We proved that it suffices that this equilibrium window is as large as or greater than the length of the longest job chain to account for all possible disturbances.

Simulation results in section IV.4 showed that heuristics based on the utility density of jobs yield good results. Moreover, DST-3 has results almost as good as DST-2, and lower computational complexity. Results also showed the benefits of the gravitational task model over scheduling algorithms for other task models — we considered the TUF scheduler GUS [Li 06], and EDF as a deadline based task model. The gravitational task model is able to accrue more utility without over-compromising the feasibility of the task sets.

Chapter V

Reducing the complexity of periodic tasks' scheduling

In this chapter, we propose a gravitational task model based on-line scheduling algorithm for periodic tasks which is inspired on a mix of Earliest Deadline First (EDF) and heuristic DST-3: *EDF-swap*. This algorithm limits the number of future jobs that the scheduler takes into account at any point in time to n^2 (n is the number of periodic tasks), guarantees all timing constraints, and accounts for increased utility accrual. Other scheduling algorithms for the gravitational task model must account for the arrival of all jobs in the hyper-period — $n!$ in the worst case — when computing the equilibrium of jobs in order to guarantee timing constraints, and increase utility accrual. Therefore, we achieve significant reduction on the computational complexity to schedule periodic tasks.

We start with a detailed description of the problem, and the basic idea of our solution in section V.1. Section V.2 describes how to compute the equilibrium of jobs, and in section V.3 we present the algorithm to reorder the execution sequence of jobs for increased utility accrual. Section V.4 brings some experimental results from simulations, and finally, section V.5 brings our concluding remarks.

V.1 Introduction

In chapter IV, we presented a few scheduling algorithms for the gravitational task model. These algorithms consider a finite number of jobs, and assume that jobs' parameters are available prior to their arrival in the system. These algorithms differ in the execution sequence of jobs, and the computation of the equilibrium of jobs. Scheduling tasks using the pendulum equilibrium and heuristic DST-3 has linear complexity with the number of jobs, and simulation results showed the good performance of this approach. In order to schedule periodic tasks, we limited the deadline to be smaller than or equal to the period, and then, scheduled all jobs within a hyper-period at the beginning of the hyper-period.

Let N be the number of jobs for which the scheduler computes the equilibrium, and n be the number of periodic tasks. In a task set, the number of jobs within the hyper-period can grow in factorial scale with the number of periodic tasks, thus $N = n!$. Therefore, the previously introduced scheduling algorithms have a factorial worst case complexity with the number of periodic tasks ($O(N = n!)$). Those algorithms, then, have a very high overhead when scheduling periodic tasks, and hence, are impractical as an on-line approach.

In scheduling algorithms for the gravitational task model, inserting jobs in the schedule only upon their arrival restricts the freedom of the equilibrium in shifting the execution of tasks. This approach may cause deadline misses, and compromise the system utility accrual — an incoming job might trigger an equilibrium recomputation that results in jobs being scheduled before the current point in time, which is not possible. On the other hand, accounting for the arrival of all future jobs is computationally expensive. Therefore, the scheduler must limit the amount of future jobs that the scheduler takes into account at any point in time, and guarantee all timing constraints, yet accounting for increased utility accrual.

In this chapter, we propose a gravitational task model based scheduling algorithm for periodic tasks which is inspired on a mix of EDF and heuristic DST-3. This algorithm uses an interval of time called *equilibrium window* (ew) to limit the amount of jobs considered in the schedule. Only jobs which have arrival time within this window are in the schedule, and we limit the number of jobs to $N = n^2$. We, then, propose a method to compute the equilibrium of jobs which guarantees the timing constraints of jobs outside the equilibrium window.

Both ordering the execution sequence of jobs and computing the equilibrium of jobs have complexity $O(N)$. The equilibrium window limits the number of jobs that the scheduler must consider at runtime, and hence, significantly reduces the overhead to schedule periodic tasks ($O(N = n^2)$). This lower overhead does not come at the expense of restricted feasibility, and simulation results show that there is a negligible impact on the utility accrual compared to methods that consider full knowledge of the arrival of future jobs.

V.2 Trade-off among competing jobs

In section IV.2, we described a method based on the pendulum equilibrium to compute the equilibrium of jobs for a given execution sequence. In this section, we describe how to apply the equilibrium of jobs in non-preemptive EDF, which is an optimum work-conserving deadline-based non-preemptive scheduling policy.

An optimum scheduling policy in the context of deadline-based scheduling means that if there is a feasible schedule, then this policy also generates a feasible schedule. The advantages of ordering the execution sequence of jobs as in non-preemptive EDF include the optimality of the schedule, and reuse of its feasibility analysis, as well as other work on its timeliness properties (e.g. [Kim 06, Baruah 06, Jeffay 91]). Moreover, EDF orders the execution sequence of jobs based only on the deadlines of jobs ready for execution — no need to consider the arrival of future jobs to take scheduling decisions.

There is an issue that prohibits the direct application of the equilibrium of jobs in EDF: the equilibrium generates a non-work-conserving schedule among the ready jobs — i.e. there are moments of idle processing even though there are jobs ready for execution —, whereas EDF is work-conserving. Therefore, applying the equilibrium without further consideration may cause EDF to produce a different order of the execution sequence of jobs.

	τ_1	τ_2
start time	0	0
period	2	5
WCET	1	1
relative tp	0.5	0.875
anchor point	0.5	0.5
importance	1	1

Table V.1: *Task set.*

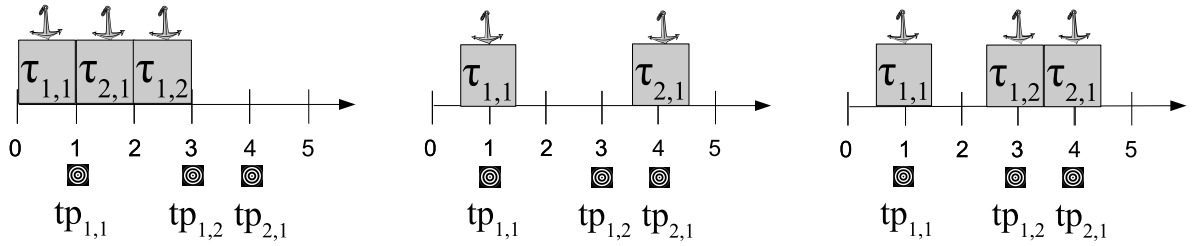
	$\tau_{1,1}$	$\tau_{1,2}$	$\tau_{2,1}$
start time	0	2	0
deadline	2	4	5
WCET	1	1	1
target point	1	3	4
anchor point	0.5	0.5	0.5
importance	1	1	1

Table V.2: *Job set.*

Consider the following example. Table V.1 contains the parameters of 2 periodic tasks, and table V.2 the first 3 jobs that arrive in the system ($\tau_{i,j}$ is the j^{th} instance of task τ_i). Scheduling these jobs with EDF results in the schedule depicted in figure V.1a. Let tc represent the current time. At $tc = 0$ jobs $\tau_{1,1}$ and $\tau_{2,1}$ are ready for execution, and $\tau_{1,1}$ executed first because its deadline is earlier than the deadline of $\tau_{1,2}$. $\tau_{2,1}$ executes at $tc = 1$, and job $\tau_{1,2}$ at $tc = 2$.

If we compute the equilibrium of jobs at $tc = 0$ with the execution sequence that EDF generates, we obtain the schedule in figure V.1b. In this schedule, job $\tau_{1,1}$ starts to execute at $tc = 0.5$. Job $\tau_{1,2}$ arrives at time $tc = 2$, when job $\tau_{2,1}$ is still ready for execution. In this case, EDF schedules $\tau_{1,2}$ to execute next, as this is the ready job with the earliest deadline. Figure V.1c depicts the resulting schedule after computing the equilibrium of jobs. As can be seen in this example, combining the equilibrium of jobs with EDF without further consideration may alter the execution sequence of jobs. Also

notice that all jobs have deviation 0 from their target points, hence accruing maximum utility to the system.



(a) *Schedule under EDF*

(b) *Schedule at $tc = 0$ with EDF and equilibrium*

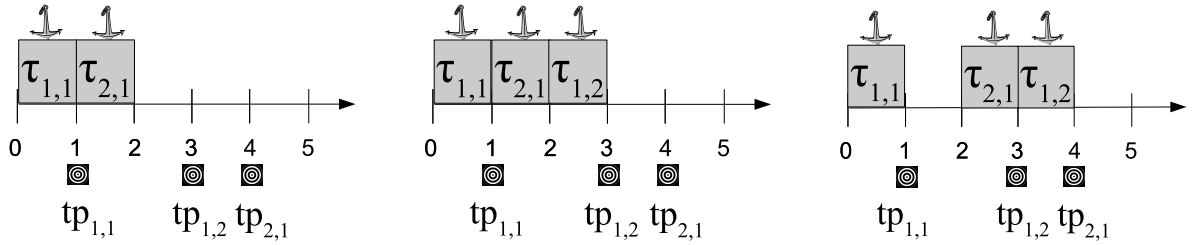
(c) *Schedule at $tc = 2$ with EDF and equilibrium*

Figure V.1: *Intermediate schedules for EDF with equilibrium.*

Our scheduling algorithm combining equilibrium of jobs and EDF works as follows. We define an interval of time ew called *equilibrium window* — similar to the equilibrium of heuristic DST-3 (see section IV.3.3). This interval starts at tc, and ends at ew_end (hence, $ew = [tc, ew\_end[$).

At system start-up, we calculate ew_end so that n^2 jobs arrive within the interval $[0, ew\_end[$. Then, the scheduler schedules the execution of all jobs that arrive within the equilibrium window as in work-conserving non-preemptive EDF. Then, the scheduler computes the equilibrium of all jobs under the constraint that the schedule in the interval $[ew\_end, \infty[$ must be exactly as in work-conserving non-preemptive EDF.

Upon completion of a job, the scheduler sets ew_end to the arrival time of the next job outside the current equilibrium window, orders the execution of incoming jobs with the other jobs using EDF, computes the equilibrium of all jobs as described above, and the process repeats itself. The new constraint on the equilibrium guarantees that the execution sequence that EDF generates prevails.



(a) *Schedule at $tc = 0$*

(b) *Schedule at $tc = 1$ before equilibrium*

(c) *Schedule at $tc = 1$ after equilibrium*

Figure V.2: *Intermediate schedules for EDF with adapted equilibrium.*

Let us repeat the example in table V.1 using this new equilibrium algorithm. For the sake of simplicity, we assume $ew\_end = 2$. The equilibrium window is $ew = [0, 2[$ at $tc = 0$, and the jobs arriving in this interval are $\tau_{1,1}$ and $\tau_{2,1}$. Scheduling these jobs

with EDF results in the schedule depicted in figure V.2a. At this point, the equilibrium cannot do anything, because the scheduler may not alter the schedule in the interval $[2, \infty[$. Upon completion of job $\tau_{1,1}$ at $tc = 1$, the scheduler sets $ew\_end = 4$ — which is the arrival time of the next job, i.e. $\tau_{1,3}$. Now the equilibrium window is $ew = [1, 4[$, and the scheduler inserts $\tau_{1,2}$ in the schedule using EDF, resulting in the schedule of figure V.2b. Finally, the scheduler computes the equilibrium, resulting in the schedule depicted in figure V.2c. Observe that now all jobs deviate from their target points, hence accruing less utility than the scheduling example in figure V.1. This result is a consequence of both the execution sequence of jobs and the constrained equilibrium window.

The complexity of this scheduling algorithm is linear with the number of jobs in the equilibrium window, which is the complexity to compute the equilibrium. As we limit the number of jobs within the equilibrium window to n^2, the complexity is $O(n^2)$.

V.3 Reordering the execution sequence of jobs

In section IV.3, we proposed a few ordering heuristics based on the utility density of jobs, and simulation results showed their efficacy in accruing utility. Those heuristics had lower acceptance ratio than EDF in some cases, though. In this section, we describe a heuristic to reorder the execution sequence that EDF generates for increased utility accrual — EDF does not account for the utility of jobs. We call this heuristic *EDF-swap*. Our reordering heuristic swaps the execution sequence of adjacent jobs based on their utility density and target points, and does not compromise the feasibility of the original schedule. Therefore, the final schedule accounts for both high acceptance ratio and increased utility accrual.

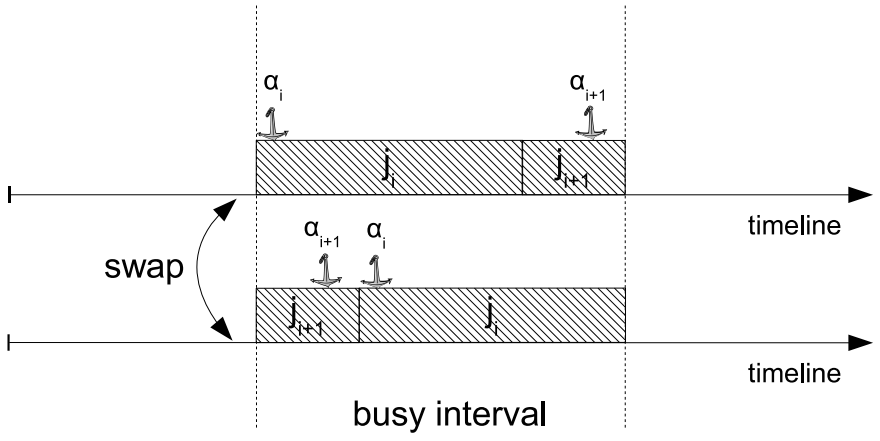


Figure V.3: *Busy interval on swap.*

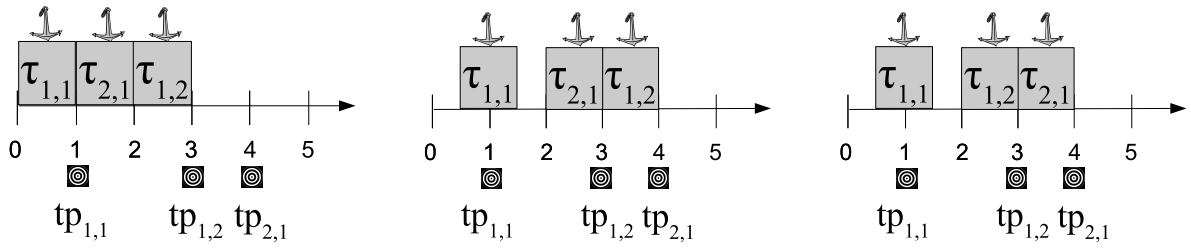
The swap heuristic resembles the bubble sort algorithm [Cormen 01]. The bubble sort algorithm goes N times through an unsorted list with N items, compares adjacent

items, and swaps them if they are in the wrong order. The name comes from the notion that items are raised or "bubbled up" to the top.

EDF-swap scans the jobs within the equilibrium window, and applies the swap algorithm 3; the apostrophe on a job parameter indicates its value upon swap without any equilibrium recomputation. The `if` condition returns true whenever the adjacent jobs belong to the same job chain ($pos(j_{i+1}) - pos(j_i) = d_i$) and, upon swap, no timing constraint is violated ($pos'(j_i) \in exec\_win_i$ and $pos'(j_{i+1}) \in exec\_win_{i+1}$), and the job with higher utility density lies closer to its target point. In case both jobs have the same utility density, the sum of their absolute deviations must decrease. Only jobs in the same job chain compete for their target points, and hence, are considered for swap. The heuristic swaps the execution of jobs so that the busy interval to which they belong does not change (see figure V.3). After scanning all jobs, EDF-swap computes the equilibrium for the jobs within the equilibrium window as described in section V.2.

In the scheduling algorithm of section V.2, we apply the reordering heuristic every time the scheduler admits a new job in the schedule. Upon each admission, the scheduler may apply the reordering heuristic more than once, but simulations results show that running the reordering heuristic more than once brings no extra increase in the utility accrual.

Let us repeat the example in table V.1, this time using EDF-swap. For the sake of simplicity, we assume $ew\_end = 4$. The equilibrium window is $ew = [0, 4[$ at $tc = 0$, and the jobs arriving in this interval are $\tau_{1,1}$, $\tau_{2,1}$, and $\tau_{1,2}$. Scheduling these jobs with EDF results in the schedule depicted in figure V.1a. Then, the scheduler computes the equilibrium, resulting in the schedule depicted in figure V.4b. Jobs $\tau_{2,1}$ and $\tau_{1,2}$ compete for their target points in this scenario. We apply, then, the reordering heuristic. Jobs $\tau_{2,1}$ and $\tau_{1,2}$ have adjacent executions, and the same utility density — same importance and same execution time. Their deviations before swap are $x(\tau_{2,1}) = -1.5$ and $x(\tau_{1,2}) = 0.5$, and upon swap $x'(\tau_{2,1}) = -0.5$ and $x(\tau_{1,2}) = -0.5$. As upon swap the sum of their absolute deviations from their target points gets smaller, the heuristic swaps them, resulting in the schedule of figure V.4c. The scheduler computes, finally, the equilibrium, which does not change the schedule at this point because the completion of $\tau_{2,1}$ may not be deferred beyond the equilibrium window.



(a) *Schedule at $tc = 0$ before equilibrium*

(b) *Schedule at $tc = 0$ after equilibrium*

(c) *Schedule at $tc = 0$ after equilibrium and swap heuristic*

Figure V.4: *Intermediate schedules for EDF-swap.*

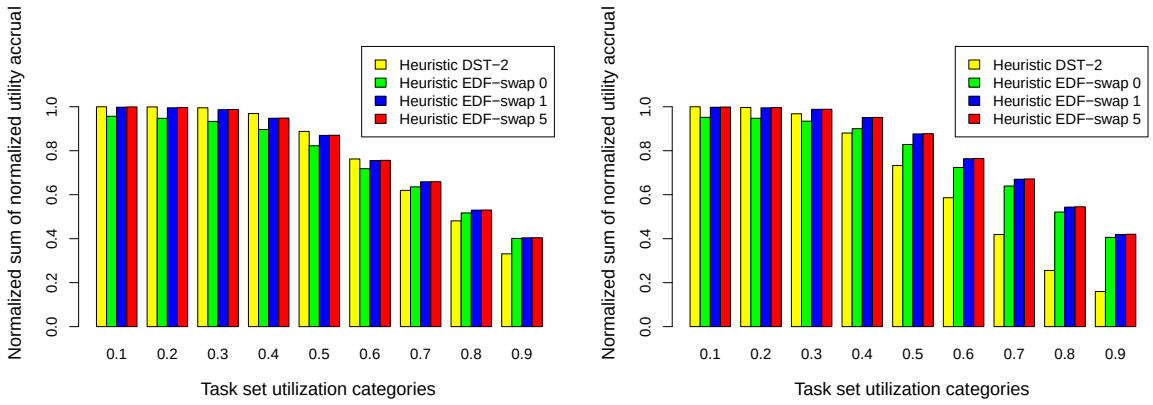
Algorithm 3 The swap heuristic.

```
if(
    pos(j[i+1])-pos(j[i]) == d[i]
    and
    pos'(j[i]) ∈ exec_win[i]
    and
    pos'(j[i+1]) ∈ exec_win[i+1]
    and
    (
      (
        dens[i] > dens[i+1]
        and
        |x[i]| > |x'[i]|
      )
      or
      (
        dens[i+1] > dens[i]
        and
        |x[i+1]| > |x'[i+1]|
      )
      or
      (
       dens[i] == dens[i+1]
        and
        |x[i]| + |x[i+1]| > |x'[i]| + |x'[i+1]|
      )
    )
  )
{
  swap(j[i],j[i+1])
  pos(j[i+1]) = pos(j[i+1])-WCET[i]
  pos(j[i]) = pos(j[i+1]) + d[i+1]
}
```

Scanning the job list has complexity $O(N = n^2)$, swapping adjacent jobs has constant complexity, and may happen at most $N - 1$ times (complexity $O(N = n^2)$). Finally, the equilibrium calculation performed at the end has linear complexity. Therefore, the complexity of EDF-swap is $O(N = n^2)$.

V.4 Evaluation

For the sake of comparability, we use the same simulation setup of the experiments in section IV.4. We assume all jobs have the same utility function $g1_i(x_i)$. The goal of our experiments is to investigate the impact of the swap heuristic on the system utility accrual, and compare the results with heuristic DST-2, which is the heuristic for the gravitational task model that obtained the best results in the experiments of section IV.4. We also measure the utility accrual in the same way as described in section IV.4. We skip a comparison with Generic Utility Scheduler (GUS) because of its bad performance in scheduling target sensitive real-time (RT) tasks (see the experiment depicted in figure IV.14 in section IV.4.4). We also skip the acceptance ratio as metric of measurement, as the acceptance ratio for EDF-swap is the same as for EDF, and section IV.4.4 already contains those results (see the experiments depicted in figure IV.16).



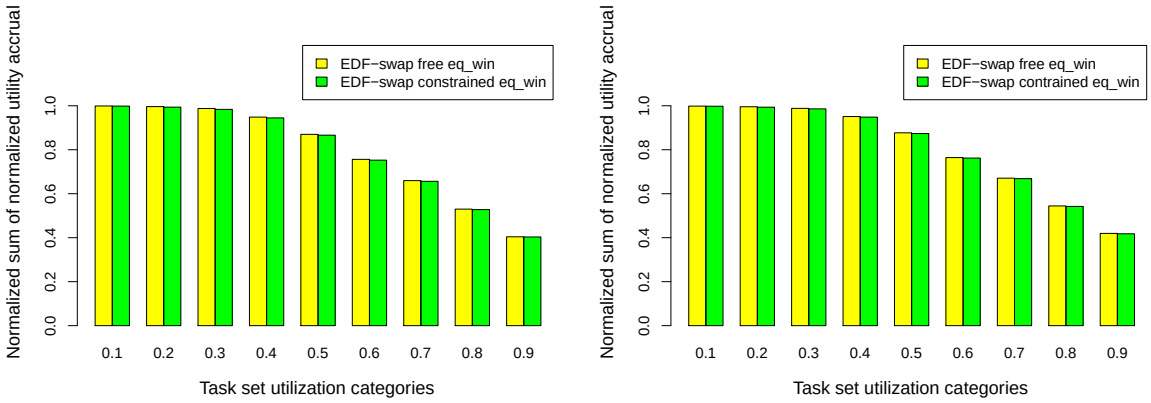
(a) *Target points equal to* 0.5 **(b)** *Target points variable*

Figure V.5: *Comparison of utility accrual among scheduling algorithms.*

In the experiments depicted in figure V.5, we compare heuristic DST-2, EDF-swap without any round of the swap heuristic (EDF-swap 0), EDF-swap with 1 round of the swap heuristic (EDF-swap 1), and EDF-swap with 5 rounds of the swap heuristic (EDF-swap 5). We limit the number of jobs within the execution window to n^2 in those experiments. As can be seen, applying one round of the swap heuristic brings a slight utility increase ($\sim 5\%$), while more rounds of the swap heuristic brings no extra increase. Therefore, we conclude that the acceptance ratio of the schedule has more impact on the utility accrual than the actual execution sequence of jobs. Moreover, comparing the result for EDF-swap and pure EDF without equilibrium (in figure IV.15), we conclude that most of the increase in the utility accrual comes from the equilibrium of jobs, and not from the execution sequence of jobs.

We can also observe that 'EDF-swap 1' is always superior or equivalent to DST-2, and that varying the target points does not affect the performance of EDF-swap, which is a very good result. The same does not apply for DST-2, as can be seen comparing figures V.5a and V.5b. Therefore, EDF-swap has significantly lower complexity due to

the limitation on the number jobs within the equilibrium window, and is superior to DST-2.



(a) *Target points equal to* 0.5 **(b)** *Target points variable*

Figure V.6: *Impact of equilibrium window length on the utility accrual.*

In the experiments depicted in figure V.6, we investigate the impact of the length of the equilibrium window on the utility accrual of the schedule. In these experiments, we compare EDF-swap with n^2 jobs in the equilibrium window, and with equilibrium window as large as the hyper-period. As can be seen, limiting the length of the execution window to contain n^2 jobs has negligible impact on the utility accrual, and is independent of the location of the target points. Therefore, we confirm that the significant reduction of the computational complexity does not come at the expense of a lower utility accrual.

V.5 Summary

In this chapter, we proposed a gravitational task model based on-line scheduling algorithm for periodic tasks which is inspired on a mix of EDF and heuristic DST-3. We called this algorithm EDF-swap. EDF-swap accounts for both high acceptance ratio and increased utility accrual. Moreover, it also significantly reduces the worst case complexity to schedule n periodic tasks from $n!$ to n^2. We achieved high acceptance ratio by ordering the execution sequence of jobs as in non-preemptive work-conserving EDF, and increased utility accrual by computing the equilibrium of jobs and swapping the execution of jobs based on their utility density. Finally, we achieved the complexity reduction by limiting the number of jobs in equilibrium — other gravitational task model based on-line scheduling algorithms must consider all jobs within the hyper-period.

In section V.2, we showed that the equilibrium of jobs may not be applied directly on EDF without further consideration. Doing so may lead to a execution sequence of jobs that differs from the order that non-preemptive work-conserving EDF generates, hence invalidating the timeliness properties of EDF. We proposed, then, a method to combine the equilibrium of jobs with EDF, and illustrated this method with an example. This

method also limits the number of jobs in equilibrium, which is the key contribution for the reduced computational complexity.

In section V.3, we proposed a heuristic to reorder the execution sequence of jobs within the equilibrium window for increased utility accrual. This heuristic is based on the utility density of jobs, and does not compromise the feasibility of the original schedule. We illustrated the heuristic with a scheduling example.

Simulation results in section V.4 showed that EDF-swap accrues more utility and has higher acceptance ratio than any other scheduling algorithm for the gravitational task model. Moreover, varying the target points within the execution window of jobs does not have a negative impact on the performance of EDF-swap. This superiority is also followed by a significant reduction of the computational complexity.

Chapter VI

Scheduling tasks for increased system utility under scarce resource availability

In this chapter, we address the scheduling of target sensitive tasks under scarce resource availability. Traditional approaches to handle overload are shifting and aborting the execution of jobs. We propose a trade-off between shifting and aborting the execution of jobs based on the gravitational task model for increased system utility accrual. The abortion of a job frees resources which other jobs may use to decrease their deviation from their target point, and yield an extra amount of utility.

We start with a brief description of the problem and basic idea of our solution in section VI.1. Then, we describe our overload handling mechanism in section VI.2. We show the benefits of our mechanism with a target sensitive application example: a multimedia application. Section VI.3 contains the description of the application and experimental results. Finally, we summarize our results and draw our conclusions in section VI.4.

VI.1 Introduction

Target sensitive real-time (RT) applications must execute within a time interval bounded by earliest start time and deadline constraints to yield some utility to the system, and have this utility maximized when executing at a target point. We say that the system is overloaded if at least one job cannot execute at its target point. The traditional approaches to handle overload are shifting and aborting the execution of jobs.

There is no direct relation between the amount of jobs that execute and the final accrued utility. For example, jobs that do not meet the start time-deadline constraints do not accrue any utility to the system. The abortion of a job frees resources which other jobs may use to decrease their deviation from their target point, hence yielding an extra amount of utility. However, abortion/deadline-miss may contribute negatively to the system utility. The trade-off between abortion and deviation from the target point must account for a resulting increased utility accrual, and is a Non-deterministic Polynomial-time hard (NP-hard) problem [Chen 96].

In this chapter, we propose an overload handling mechanism for target sensitive real time applications. This mechanism differs from previous work because we consider the trade-off between aborting and shifting the execution of jobs in order to account for increased system utility. The abortion heuristic of our mechanism is based on the utility density of jobs — which is the ratio between utility accrual and execution time —, and on the pendulum analogy of the gravitational task model.

Our mechanism has quadratic complexity, and is application independent; any application can benefit from it by modeling its requirements into the parameters of the gravitational task model. The evaluation section brings a multimedia case study where we use our mechanism to reduce frame display jitter and improve resource usage.

VI.2 Handling overload

The system utility is not only related to the amount of work done. If system resources are not enough to allow all jobs to execute at their target points (overload condition), a compromise for maximized utility accrual might imply the abortion of some jobs — resources freed by an aborted job may allow other jobs in the system to accrue an extra amount of utility which results in an increase in the resulting system utility. Notice that an aborted job can either accrue no utility, or accrue a negative utility to the system. In this work we will only consider the former case.

Our overload handling mechanism uses a heuristic which discards jobs based on their utility density, and accounts for target sensitivity based on the equilibrium of the gravitational task model. As in the gravitational task model all jobs have elliptical utility functions, the importance alone implicitly accounts for the utility variation as a function of the deviation from the target point. The smaller this ratio is, the smaller is the utility accrual of the job per unit of execution. If these units of execution are used by other jobs with higher utility density that compete for the same units of execution the resulting utility accrual might be higher (example in the end of this section).

The overload handling mechanism works as follows. Assume any existing scheduling

algorithm based on the gravitational task model. Our overloading handling mechanism maintains a job list with the order jobs execute (*exec_list*), a job list ordered by decreasing utility density (*density_list*), and a job list with the final schedule (*sched_list*). The sorting criteria for *exec_list* depends on the scheduling algorithm. An incoming job is inserted in *exec_list* and *density_list* accordingly. Then, our mechanism inserts jobs in *sched_list* in decreasing order of utility density at the position defined by *exec_list*, and applies the equilibrium. Whenever inserting a job results in a smaller system utility, the scheduler aborts the job execution, and reapplies the equilibrium. This process is repeated until there is no job left to be inserted in the schedule. See algorithm 4. Upon completion of a job, the scheduler updates all lists accordingly.

Algorithm 4 Overload handling mechanism upon job arrival.

```
input: exec_list, density_list, incoming_job

sched_list = new_empty_list()
insert_sorted(incoming_job,exec_list)
insert_sorted(incoming_job,density_list)
for (i=0; i<length(density_list); i++)
{
  job = density_list[i]
  utility_before = utility(sched_list)
  insert_sorted(job,sched_list,exec_list)
  equilibrium(sched_list)
  if (utility(sched_list)<utility_before)
  {
    abort(job,sched_list)
    equilibrium(sched_list)
  }
}

return sched_list
```

The complexity to insert and remove a job from all lists is linear, and reinserting all jobs in the scheduler has quadratic complexity. Chapter IV.2.2 describes a method to compute the equilibrium of N jobs with linear complexity. Our method may compute the equilibrium N times in the worst case, hence leading to a quadratic complexity. Therefore, the final complexity of this overload handling mechanism is quadratic.

Consider the following example, where we schedule the jobs described in table VI.1; in this example, for the sake of simplicity, we order *exec_list* by target point. The execution order in the schedule is, then, j_1, j_2 and j_3. Our overload handling mechanism inserts those jobs in the schedule in decreasing order of utility density. Therefore, we initially insert job j_1 in the schedule directly at its target point. The accrued utility of the system is 10 at this point and this state is depicted in figure VI.1a. Next, we insert

j_3 in the schedule, which can also be scheduled directly at its target point. The accrued utility of the system is 18 at this point and this state is depicted in figure VI.1b. At last, we insert job j_2 in the schedule, which demands a recomputation of the equilibrium of the system. With these job parameters, the equilibrium schedules job j_1 at time 2.57, job j_2 at time 3.57 and job j_3 at time 5.57. This state is depicted in figure VI.1c. In this state the accrued utility of the system is 17.5, which is smaller than the system utility without job j_2. Therefore, the overload handling mechanism aborts the execution of j_2, and the final schedule is as depicted in figure VI.1b.

	j_1	j_2	j_3
start time	0	0	2
deadline	4	6	6
WCET	2	1	2
target point	3	4	5
anchor point	1	1	1
importance	10	1	8
util. density	5	1	4

Table VI.1: *Job set.*

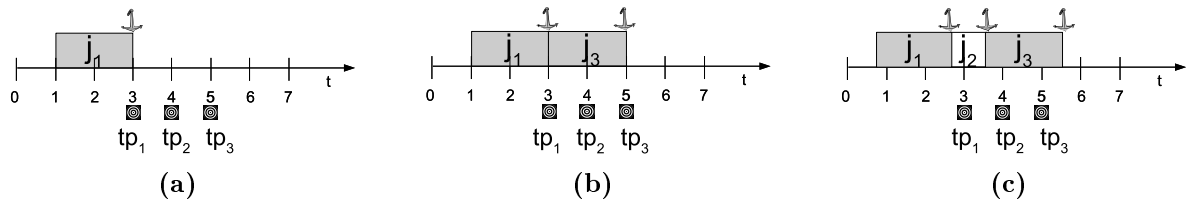


Figure VI.1: *The overload handling heuristic*

Assume now that the importance of j_2 is 6, hence having a utility density of 6. Applying the overload handling mechanism results in j_1 executing at time 2.6, job j_2 at time 3.6 and job j_3 at time 5.6. In this schedule the accrued utility of the system is 21.5, which is more than the utility accrued when only j_1 and j_2 are scheduled to execute. Therefore, j_3 (the job with lowest utility density) is not aborted.

VI.3 Evaluation

We evaluate our overload handling mechanism with a multimedia case study, which is an application example that can benefit from the offered trade-off. To the best of our knowledge, there is no previous work in the literature which accounts for the trade-off between aborting and shifting the execution of jobs for increased utility accrual. Therefore, we compare our results to Generic Utility Scheduler (GUS) [Li 06], which is

a Time Utility Function (TUF) scheduler with an overload handling mechanism based on the utility of jobs. We first describe the case study, and then, show the simulation results.

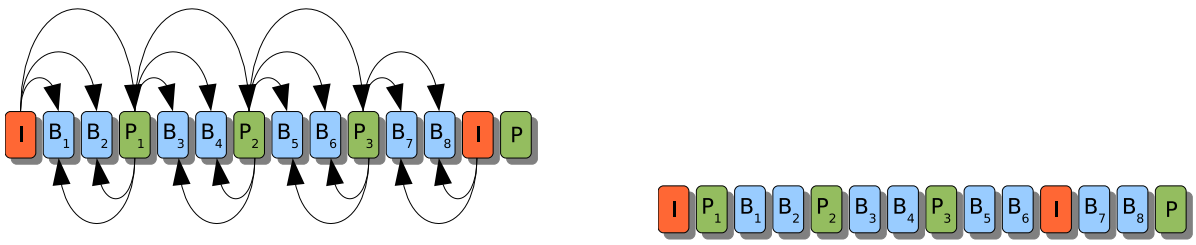
VI.3.1 Multimedia case study

High quality media processing, such as consumer electronics, is target sensitive because each frame must be displayed at a specific point in time for maximized perceived quality of video (PQV). Buffering frames in advance is not an option in these applications [Isovic 03], hence demanding frames to be displayed before the next frame starts being decoded. Imposing tight timing constraints reduces the PQV degradation under low utilization, but the degradation is sharp as the system load increases.

Since system cost drives the system development into optimized resource utilization, such systems must be designed in order to exploit low utilization periods for higher PQV, yet providing the necessary flexibility to adapt the application for overload conditions. Common strategies to circumvent lack of resources are delayed frame decoding (and hence display) and frame skipping [Isovic 03].

Here, we show how our overload handling mechanism can express the trade-off between delaying and skipping the display of a frame. For this case study, we consider the MPEG-2 and the MPEG-4 Simple Profile (SP) as the video compression standards. Notice that the capabilities of our overload handling mechanism are not limited neither to any particular standard, nor to multimedia applications.

MPEG-2 is the standard used in Digital Versatile Disc (DVD), and MPEG-4 Simple Profile (MPEG-4 SP) is the most common High Definition (HD) compression standard for Internet streaming and mobile devices nowadays. Among several other contributions, the MPEG-2 standard defines 3 different types of frames: I, P and B; MPEG-4 SP contains only I and P frames. While I frames are self-contained, P and B frames depend on other frames to be decoded. P frames depend on the previous P or I frame (which ever comes first), and B frames depend on the previous and the next P or I frame. This dependency graph is shown in figure VI.2a. This strategy allows better compression by avoiding the replication of data that remains unchanged in-between scenes, but imposes more challenges to the real-time aspect of the application. All frames in-between two I frames (including the first I frame) comprise a Group Of Pictures (GOP). A GOP finishing with a B frame is called dependent, else it is called independent.



(a) *Displaying order and dependencies* **(b)** *Decoding order*

Figure VI.2: *Frame decoding and display*

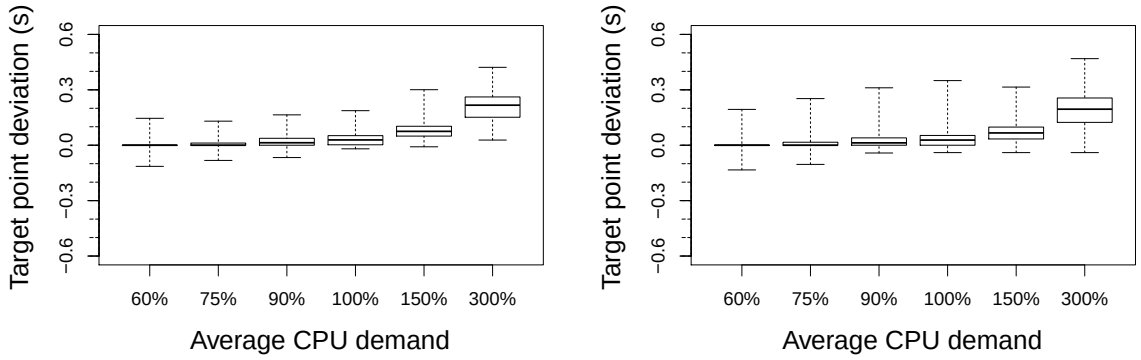
The compression standard affects both frame skipping and delayed frame decoding. Dependencies in between frames defined by the compression standard dictate how the delay or abortion of one frame decoding affects other frames of the stream. By selecting the appropriate set of parameters in the gravitational task model, the overload handling mechanism can address the trade-off between skipping and delaying frame decoding for reduced PQV degradation.

In the gravitational task model, future jobs that affect the schedule of jobs currently in the system must be considered in the equilibrium computation. For this case study we consider all frames belonging to the current GOP; we assume that all GOPs are independent. The start time of frames is the start of its GOP and the relative deadline is the length of the GOP. Having a large execution window gives the multimedia application more flexibility for adaptations under overload, yet allowing for maximum PQV under no overload. The anchor point of each frame decoding is 1, since we are interested in the decoding completion and frame display. We assume the frame decoding time is known, e.g. given by a decoding time estimation tool.

Assuming a constant framerate, the target point of I and B frames is the start of the GOP plus the position of the frame in display order (see figure VI.2a) multiplied by the frame inter-display time. For example, suppose a frame rate of $25fps$, which has an inter-display time of $1/25s = 40ms$. If the current GOP starts at time $1200ms$, its $3rd$ frame must be displayed at $1200 + 40 * 3 = 1320ms$. For P frames, its position in the equation is subtracted by the number of B frames until the previous P or I frame in the display order. For example, the $4th$ frame of the GOP in figure VI.2a, which is a P frame, has target point $1200 + 40 * (3 - 2) = 1240ms$. This way, ordering the execution of the decoding jobs by target point resolves all the dependencies. See figure VI.2b.

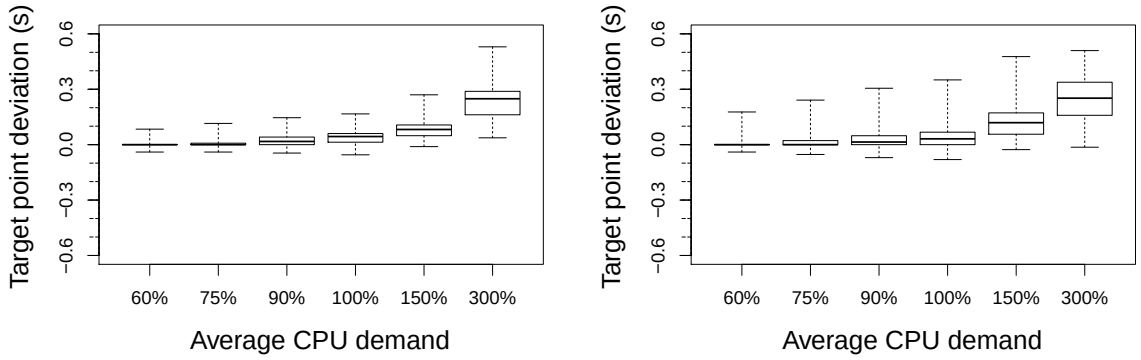
Lastly, we assign importances to the decoding jobs based on the frame skipping algorithm presented in [Isovic 04]. As criteria we consider the frame type and the frame position in the GOP. The I frame is the most important in a GOP because all other frames are dependent on it. Next come the P frames, whose importances are dependent on their position in the GOP. The closer to the I frame in the same GOP, the higher is the importance. Last comes the B frames, whose importance are also based on their position on the GOP. However, since no frame depends on a B frame to be decoded, the importance here depends only on the frame distribution on the GOP. Skipping too many frames in a row affects the smoothness of the video, hence being preferable to give different importances to odd and even B frames; we assign higher priority to even frames.

In our importance assignment policy, even B frames have double the importance of odd B frames. Each P frame has double the importance of the next frame, and the last P frame in the GOP has importance equal to the double of the importance of even B frames. Finally, the importance of the I frame is the double of the importance of the next P frame.



(a) *BBC News*

(b) *EuroSport Tennis*



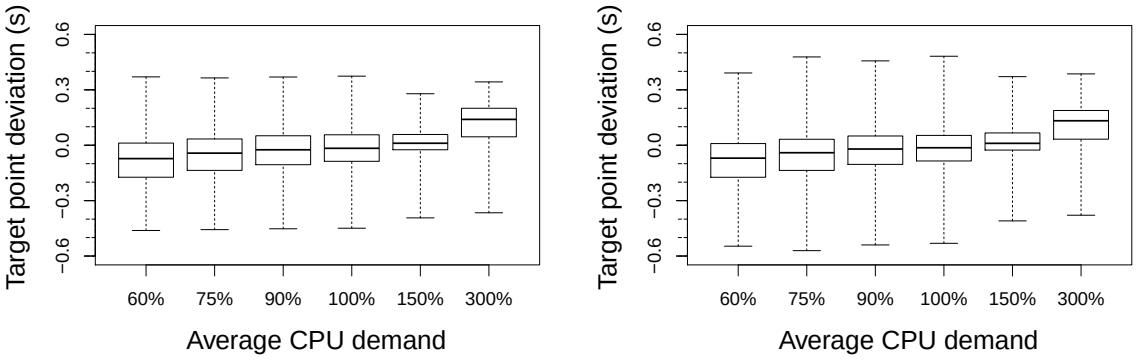
(c) *HD documentary*

(d) *Matrix movie (DVD)*

Figure VI.3: *Jitter dispersion for proposed overload handling mechanism*

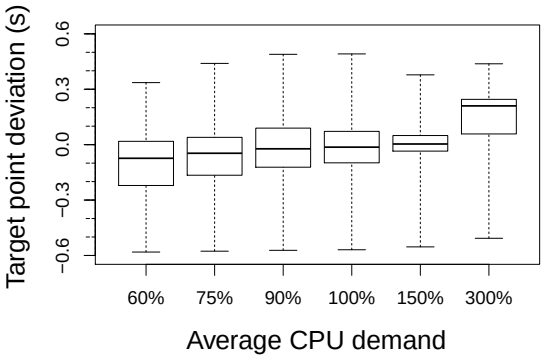
VI.3.2 Experiments

In our experiments, we measure the percentage of skipped frames and the dispersion of the deviation from the target point. We chose these parameters over a perceived quality of video analysis because, to the best of our knowledge, there is no existing method that can properly quantify it as a function of jitter and frame skipping. We use 4 streams of different types, each one having between 100 and 200 GOPs and different properties. The 1*st* stream is a BBC news satellite stream recorded using the Dreambox [dre] set-top box, and has all GOPs with 12 frames. The 2*nd* stream is a tennis match recorded from EuroSport also using the Dreambox, and with 12 frames in GOPs. The 3*rd* stream is a high definition documentary downloaded from the Internet, and has variable GOP sizes. Finally, the 4*th* stream is a scene of the movie Matrix recorded from a DVD, has no B frames, and has varying GOP sizes. For each stream, we assume 6 scenarios for the average CPU demand needed to decode all frames: 60%,75%,90%,100%,150%,300%. The different scenarios are created by linearly scaling the original decoding times obtained from the trace files.

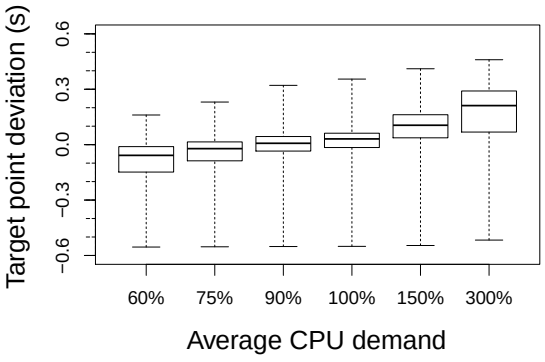


(a) *BBC News*

(b) *EuroSport Tennis*



(c) *HD documentary*

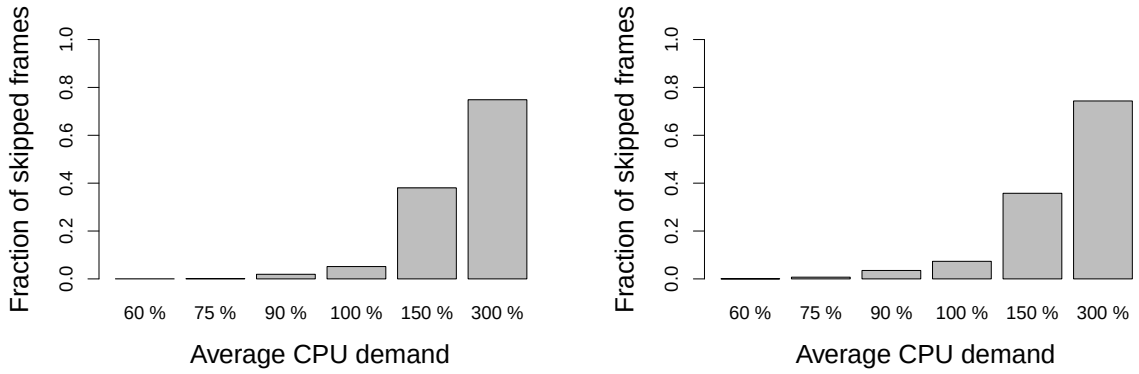


(d) *Matrix movie (DVD)*

Figure VI.4: *Jitter dispersion for GUS [Li 06]*

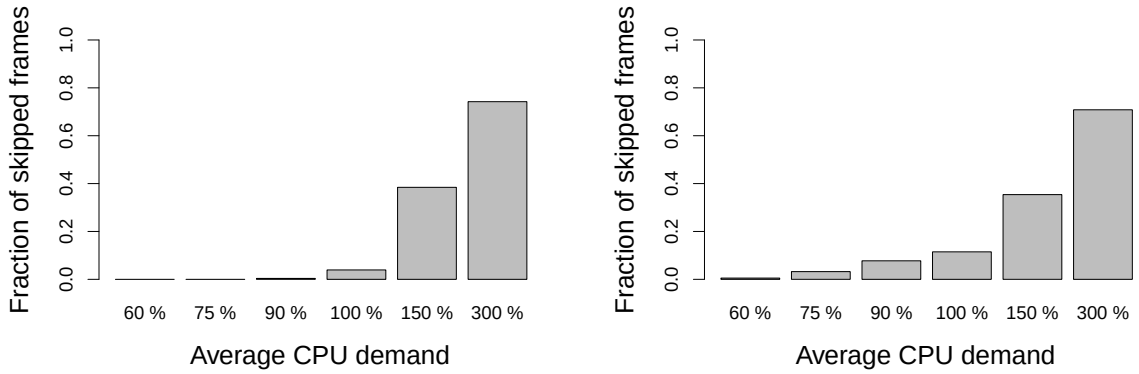
In figure VI.3 we show, for our overload handling mechanism, the boxplot with the deviation of each frame in a stream from the target point. In a boxplot, 50% of all points are located in the range of the box, being 25% above the thick line that crosses the box and 25% below. The whiskers above and below the box comprise, each one, 25% of all the points. As we can see in this figure, for all streams considered, if the average utilization is 60% or 75%, many of the frames finish their decoding exactly by their target point. This is concluded from the fact that the box comprising 50% of all jitter is very short and indicates a jitter of zero. Nonetheless, conflicting target points happen anyway and our method schedules the frames to keep the deviation under limited boundaries. As the CPU demand increases, the deviation increases in order to avoid skipping the whole movie. Even with the streams being so different, our method shows to be very stable based on the similarities of the 4 graphics. We run the same experiment for GUS (see figure VI.4). In this case, we observe a significantly larger jitter dispersion, independently of the system workload.

In figure VI.5 we show, for our overload handling mechanism, the fraction of skipped



(a) *BBC News*

(b) *EuroSport Tennis*

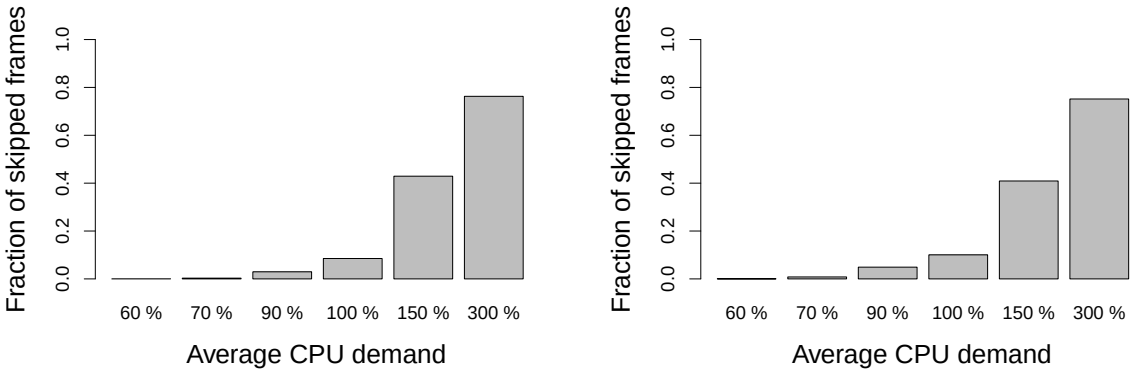


(c) *HD documentary*

(d) *Matrix movie (DVD)*

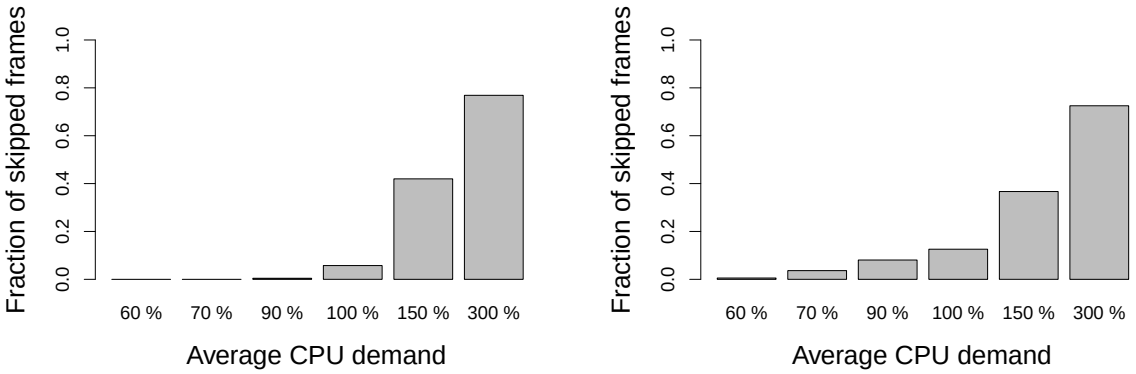
Figure VI.5: *Fraction of skipped frames for proposed overload handling mechanism*

frames in each of the streams for the CPU demand scenarios proposed. Since we consider average CPU demand, it is possible that some GOPs demand more CPU than others and for these GOPs, frames must be skipped even if resources are enough to decoded all frames by the end of the stream. Therefore, even in the case of CPU demand less than 100% we can observe some skipped frames. Once again, observing all streams analyzed, we see a stable behavior of our method. All decoded frames were displayed in time and accounting for reduced jitter. Even for an average CPU demand of 300%, we observed that on average two frames are displayed per GOP, hence still allowing the user to appreciate a moving picture with some smoothness. In figure VI.6 we plot the fraction of skipped frames for GUS. In this experiment GUS has results statistically identical to ours, an expected result, as the skipping heuristic is very similar in both approaches. Therefore, our method is able to reduce the frame display jitter without compromising the number of displayed frames, hence providing for a smoother video.



(a) *BBC News*

(b) *EuroSport Tennis*



(c) *HD documentary*

(d) *Matrix movie (DVD)*

Figure VI.6: *Fraction of skipped frames for GUS [Li 06]*

VI.4 Summary

In this chapter, we showed that executing as many tasks as possible does not imply in maximized system utility. In fact, rejecting some tasks frees resources that other tasks can use to accrue more utility. For instance, tasks that do not meet their starttime-deadline requirements do not accrue utility to the system, yet consume resources that other tasks can use to increase the utility accrual. The overload situation can be overcome by either delaying or dropping jobs, and the choice must account for final increased utility accrual. Dropping jobs is a combinatorial problem.

In section VI.2, we proposed a heuristic overload handling mechanism for target sensitive RT applications. This mechanism increases the system utility by both aborting jobs and computing the compromise among the deviation of the ones scheduled for execution. The abortion heuristic is based on the utility density of jobs, which is the ratio between utility accrual and execution time. This mechanism has quadratic complexity, and is application independent. Hence, any application can benefit from it by modeling its requirements into the parameters of the gravitational task model.

Section VI.3 described a multimedia case study where we use this mechanism for reduced frame display jitter and improved resource usage under scarce resource availability. The frame decoding timing constraint assignment is based on the frame skipping algorithm [Isovic 04] for quality aware video adaptation. Simulation results showed the efficacy of the overload handling mechanism.

Chapter VII

Applications of the gravitational task model

In this chapter, we present 3 applications enhanced with the gravitational task model in order to demonstrate the validity of our work. Section VII.1 brings a multimedia application where we use the gravitational task model to reduce the degradation in the perceived quality of video (PQV) during overload. The basic idea is to relax the temporal constraints of the frame display. This way, the decoder may prepone/postpone the frame display in order to cope with the longer decoding times without the need to drop frames. Of course, without further consideration, this approach leads to large variations of inter-display time, which degrades the PQV. We use the gravitational task model to schedule the completion of the decoding task as close as possible to the optimum display instant, hence accounting for both optimum PQV under low system workload, and adaptivity under overload with decreased degradation of the PQV. This section presents both quantitative and qualitative evaluations of the resulting video output that show the benefits of our approach.

Section VII.2 brings an algorithm for adaptive resource management. This algorithm uses a gravitational task model based compression method which has linear complexity for reallocation of resources among applications under scarce resource availability. The pendulum analogy provides for the intuition of the solution.

Finally, section VII.3 describes an opportunistic packet scheduling strategy which aims at reduced retransmission ratio of packets in Body Area Networks (BAN). Reducing the retransmission ratio increases communication reliability and reduces energy consumption. This application has been entirely developed by researchers not involved with studies on the gravitational task model, and evidences the contribution of the gravitational task model's intuition to ease the cooperation among different research fields.

VII.1 Video stream adaptation

VII.1.1 Introduction

In section II.1, we exemplified the tasks involved in video playout using the MPEG-2 video encoding/decoding standard. We showed that digital video playout imposes tight execution window constraints on frame display for maximum PQV; the inter-display time must be strictly periodic and constant. Provisioning computational performance for the worst case scenario leads to resource underutilization, as the decoding time of frames may vary a lot due to advanced image compression techniques, e.g. temporal compression. Temporal compression/decompression is computationally expensive and its effort depends on the temporal aspects of the video.

We also showed that the standard MPEG-2 decoder design assumes computational performance for the average case, and add frame buffers to cope with variable decoding times, and hence, guarantee the strict inter-display time requirement for maximum PQV. Larger buffers allow the decoder to recover from the long decoding time of a frame, provided that subsequent frames have shorter decoding times. However, larger buffers imposes longer latencies, require changes in the decoder design, and increase energy consumption and production cost. Usually, decoders have decoding buffer capacity for 3 frames, the minimum necessary to cope with precedence constraints.

We propose a relaxation of the temporal constraints of the frame display. This way, the decoder may prepone/postpone the frame display in order to cope with the longer decoding times without the need to drop frames, and without buffering extra frames. Of course, without further consideration, this approach leads to large variations of inter-display time, which degrades the PQV. We use the gravitational task model to schedule the completion of the decoding task as close as possible to the optimum display instant, hence accounting for both optimum PQV under low system workload, and adaptivity under overload with decreased degradation of the PQV.

Our solution does not require any change in the standard architectural design of video decoders, as we only alter the timing requirements of tasks and the scheduling algorithm. Moreover, we reduce the degradation of the PQV without requiring additional buffering, which comes at the expense of longer display latencies. The evaluation section brings both a quantitative and a qualitative experimental analysis of the resulting video that shows the benefits of our solution.

VII.1.2 Temporal constraints of tasks

In this section, we describe the temporal constraint assignment for the decoding and display tasks. The input task works asynchronously, responding directly to the video stream source and storing frames in the input buffer, and has negligible execution time, as a Direct Memory Access (DMA) module handles the data transfer between buffers. Our goal is to provide some flexibility to the display task for early/late display in order to avoid frame drop during overload. Early display allows the earlier decoding of subsequent frames which would miss their deadlines otherwise, and late display avoids frame drops.

The earliest start time of the decoding task is the moment of frame arrival, and the deadline is customizable — larger deadlines provide for late display, and hence, flexibility under overload. Without loss of generality, we assume that the frame arrival has the same periodicity of the frame display. The decoding task does not have a target point, since we are interested in the moment of display. Therefore, we assign importance 0 to the decoding tasks, which implies in total flexibility to shift the decoding jobs within their execution windows.

The execution window of the display task varies depending on the frame type. B frames are not used as reference, and hence, can be displayed right after decoding in order to free the buffer and allow the decoding of the next frame. Thus, the display task of B frames have the same execution window as the decoding task of the respective frame; I and P frames have earliest start time equal to the desired display instant. The relative deadline of the display task is the same as for the decoding task.

The execution time of the display task is 0, as a DMA module handles the data transfer between decoding buffer and display buffer. The desirable frame display instant is the target point of the display task, which has importance equal to the importance of the video stream. Ordering the execution of all jobs using Earliest Deadline First (EDF) ensures the correct decoding and display sequence [Isovic 03].

VII.1.3 Evaluation

Experiment setup

In our experiments, we measured the decoding times of MPEG-2 frames recorded from satellite streams using a DreamBox [dre] satellite receiver. The measurement tool uses the `Libmpeg2` library [lib], which is a free library for decoding MPEG-2 video streams. This library is highly optimized for high performance, has been ported to x86, PowerPC, SPARC, ARM and SH4, and can be easily ported to other architectures. Due to its high performance, `Libmpeg2` is a very popular codec used in many free multimedia players, such as MPlayer, Xine and VLC.

We use the measured decoding times to schedule the frame decoding and display using a scheduling simulator. Finally, we use the output of the simulator to generate a video output that displays frames according to the schedule of the display tasks. This video output is a *transport stream* file, and we alter the display time by editing the *presentation time stamp* field of the transport stream packet header. Please refer to [Tra 00] for more information about the transport stream syntax.

In order to investigate the impact of the workload on the PQV, we generate task sets with variable number of video streams. However, our video adaptation strategy is also applicable for systems with one single stream and insufficient computing performance for the worst case.

Scenarios

We generate task sets from the decoding times of 5 MPEG-2 satellite streams. Each stream has frame rate of $25fps$ (hence, frame display periodicity is $40ms$), and com-

prises 7500 interpolated frames with different properties. The $1st$ stream is a sport car documentary from BBC news, and we assign importance 10; the $2nd$ stream is the Bloomberg business news, and we assign importance 2; the $3rd$ stream is a volleyball match recorded from Live Sport, and we assign importance 2; the $4th$ stream is a soccer match recorded from DSF, and we assign importance 5; and finally, the $5th$ stream is the cartoon Lucky Luke from RTL, and we assign importance 2.

In our experiments, we use 7 task sets with different workloads. Task set A contains the $1st$ stream; task set B contains the $1st$ and $2nd$ streams; task set C contains the $1st$, $2nd$ and $3rd$ streams; task set D contains the $1st$, $2nd$, $3rd$ and $4th$ streams; task set E contains all streams; task set F contains all streams twice; and finally, task set G contains all streams 3 times. The average utilization of those task sets vary in between 5% and 70%.

We schedule the task sets using pure EDF and gravitational task model with EDF job ordering. Under EDF, we consider 3 different deadline assignments: deadline equal to period, deadline equal to 4 times the period, and deadline equal to 16 times the period. Under the gravitational task model, we consider only deadlines equal to 16 times the period.

Results

We perform both a quantitative and a qualitative analysis of the resulting video quality. For the quantitative analysis, we use the deviation from the target point (which relates to the inter-display jitter), and the frame drop as metrics of comparison. Here, we show only the graphs for the BBC News documentary, as the results for other streams are very similar.

Figure VII.1 depicts the percentage of dropped frames (due to deadline misses) for each task set and scheduling algorithm. As expected, we can observe that shorter deadlines imply more dropped frame as the workload increases. Although the maximum frame drop rate is below 1%, the resulting video shows that the impact on the PQV may be significant, as dropping random frames may also affect the decoding of other frames due to precedence constraints. An interesting result is that 'EDF d=4p' drops a few frames for a workload with 10 stream, but drops no frames for a workload of 15 stream. We attribute this behavior to the EDF arbitration in scheduling jobs with the same deadline.

In figures VII.2, VII.3, VII.4, and VII.5, we plot the histogram of the deviation from the target point for each frame display. The x-axis is the deviation in milliseconds — each bar of the histogram has a width of 5 milliseconds —, and the y-axis is the number of frames. Remember that the total number of frames in each stream is 7500. Each graph plots the histogram for 6 task sets: A, B, C, D, F, and G. These task sets have number of streams 1, 2, 3, 4, 10, and 15, respectively. Each figure contains two sets of histogram plots: one with the y-axis ranging from 0 to 7500, and one with the y-axis ranging from 0 to 5. The latter plots show that there are a very few frames with very large deviation from the target point, and these bars are not visible in a plot with a

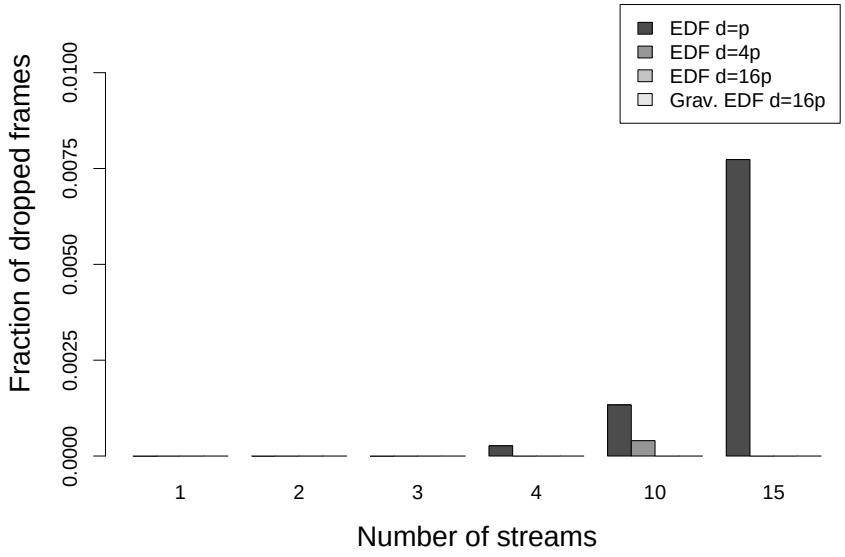
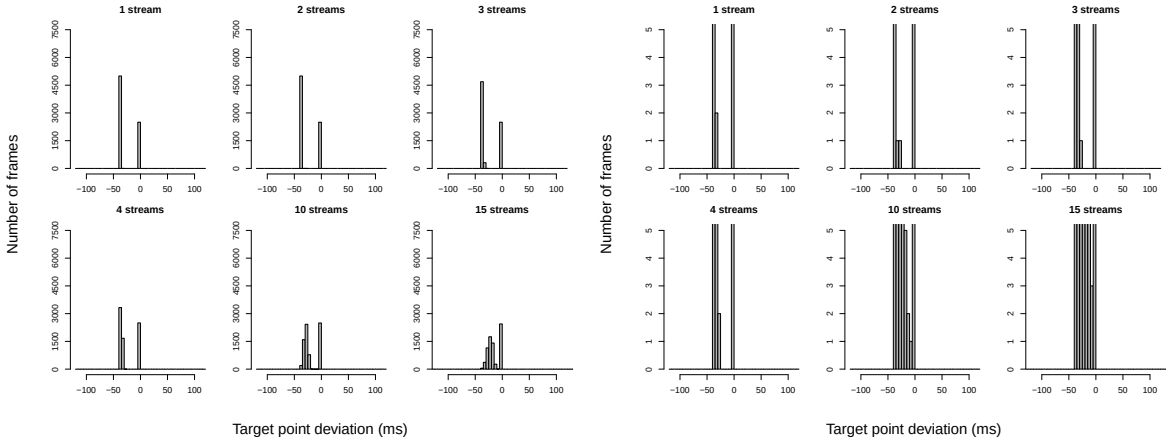


Figure VII.1: *Dropped frames.*

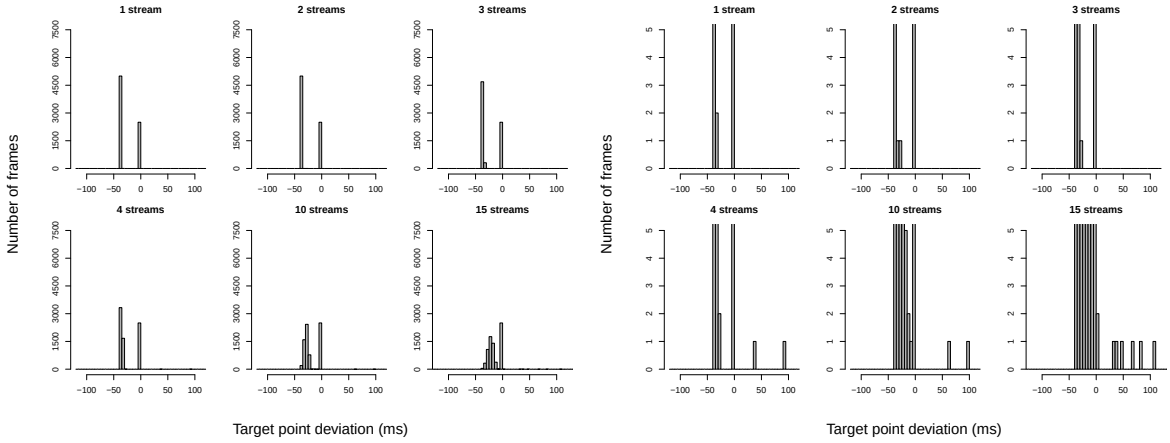


(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure VII.2: *EDF d=p.*

wide y-axis range.

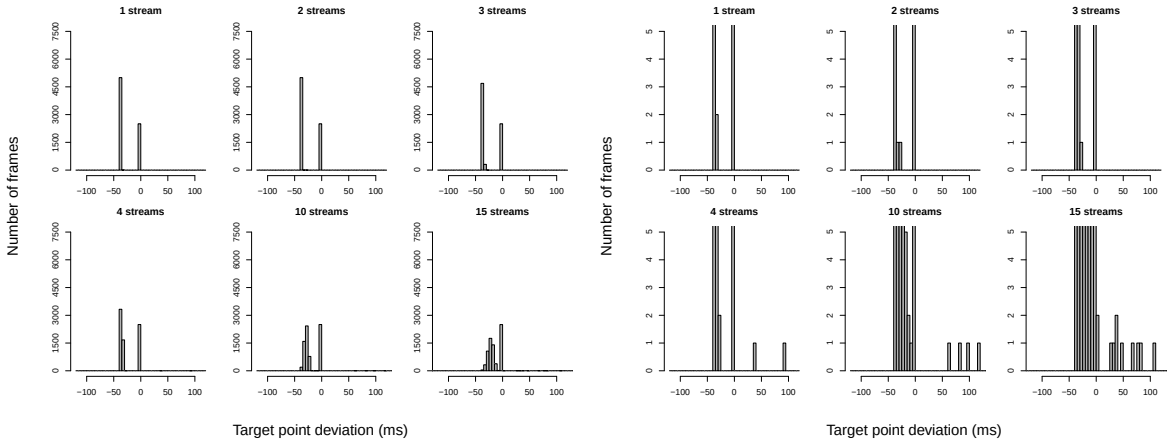
In figure VII.2 we plot the histograms for EDF with deadline equal to the period. We can observe that for low workloads, i.e. less streams, most of the frames have either deviations 0 or −40 milliseconds. This behavior reflects the incapability of EDF to enforce display at the target points — under low workload B frames are displayed too soon, which alters the periodicity of the frame display. These frames are of type B, which are not needed for reference, and hence, displayed right after decoding. We observe that,



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure VII.3: *EDF d=4p.*

as we increase the workload, the deviations become more sparse. These variations of the display time come from the interference of the decoding of other streams.

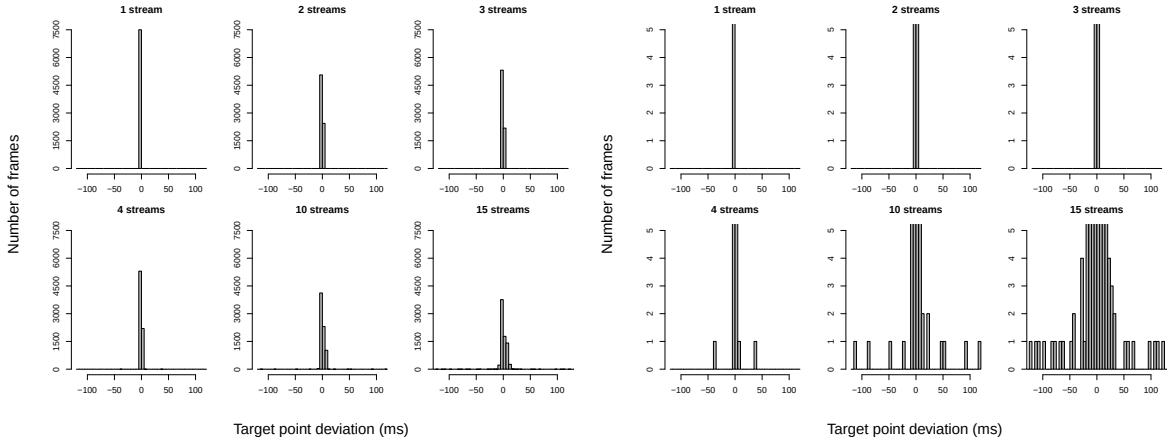


(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure VII.4: *EDF d=16p.*

In figure VII.3 we plot the histograms for EDF with deadline equal to the 4 times period. The pattern of the deviations from the target point remain similar to the previous experiment, but in this experiment we observe a few deviations larger than 0. This is a consequence of the extended deadline, which allows for delayed display, hence avoiding frame dropping. The same pattern can be observed in the experiment depicted in figure VII.4, where we set the deadline equal to 16 times the period. In this case, we observed no frame dropping, but the incapability of EDF to enforce display at the target points still allows for large deviations from the target point.

In figure VII.5 we plot the histograms for a combination of the gravitational task



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure VII.5: *Grav. EDF d=16p.*

model and EDF with deadline equal to the 16 times period. In this case, the scheduler is able to account for the target point of frames under low workload, yet providing the necessary flexibility under high workload in order to avoid frame drops. We can see that for as task set with only one stream, all frames are displayed at their target points, and as the workload increases, the deviations gradually increase. Very few frames have deviations larger than 0, although the maximum deviation may be very high.

Observing the resulting video output, we concluded that the dropped frames may cause significant disturbance, even though the dropping rate is low. Moreover, the larger display jitters in EDF produced jerked scenes, while using the gravitational task model resulted in a smooth video playout.

VII.2 Adaptive resource management

In this section, we describe the adaptive resource management algorithm that we implemented in the ACTORS Framework described in section II.2. This algorithm uses a gravitational task model based compression method which has linear complexity for reallocation of resources among applications. The pendulum analogy provides for the intuition of the solution. An alternative solution is the elastic task model, which has a compression method based on a spring system, and quadratic complexity. Our solution keeps the intuition from an analogy with physical systems, and has reduced complexity. We skip a comparison between the resulting compression of both methods because they are not comparable — they consider different proportional compression rates.

VII.2.1 Terminology and assumptions

Assume a system with capacity C, and M cores with capacity c_k ($\sum_{k=1}^{M} c_k = C$). An application A_i has SL_i service levels, and operates at a service level $sl_i | 0 \leq sl_i \leq SL_i - 1$. A service level sl_i has a quality of service $qos_i(sl_i)$, a maximum band-

width requirement $B_i^{MAX}(sl_i)$ and a minimum bandwidth $B_i^{MIN}(sl_i)$. Here we assume $B_i^{MIN}(sl_i) = B_i^{MAX}(sl_i + 1)$, $B_i^{MIN}(SL_i - 1) = 0.8 \times B_i^{MAX}(SL_i - 1)$, and $qos_i(sl_i) > qos_i(sl_i + 1)$. In order words, service level sl_i requires more bandwidth and provides better quality of service than service level $sl_i + 1$. Each A_i has an assigned bandwidth $B_i | B_i^{MIN}(sl_i) \leq B_i \leq B_i^{MAX}(sl_i)$, and an importance imp_i.

An application A_i may contain 1 to M virtual processors $VP_{i,j}$, each with a maximum bandwidth requirement $BWD_{i,j}^{MAX}(sl_i)$ and a minimum bandwidth $BWD_{i,j}^{MIN}(sl_i)$ — we assume that $\sum_{j=1}^{M} BWD_{i,j}^{MAX}(sl_i) = B_i^{MAX}(sl_i)$ and $\sum_{j=1}^{M} BWD_{i,j}^{MIN}(sl_i) = B_i^{MIN}(sl_i)$. $CA_{i,j}$ represents the core assignment of virtual processor $VP_{i,j}$, and assumes values from 1 to M. The assignment of virtual processors to core k is valid iff $\sum_{CA_{i,j}=k} BWD_{i,j} \leq c_k$, where $BWD_{i,j}$ is the bandwidth assignment to $VP_{i,j}$. We assume that an application may not have more than one virtual processor on the same core in order to exploit potential parallelism.

Upon application registration/unregistration, the resource manager must assign to each application A_i a service level sl_i, a bandwidth B_i, and assign virtual processors to cores. The target of each application is to operate at service level $sl_i = 0$, with bandwidth assignment $B_i = B_i^{MAX}(0)$. However, a bandwidth allocation is feasible if, and only if, $\sum_{i=1}^{N} B_i \leq C$. If $\sum_{i=1}^{N} B_i^{MAX}(0) > C$, the resource manager compresses bandwidth assignments, and changes service levels accordingly in order to fulfill the feasibility condition. Applications with higher importances and providing higher quality of service undergo a smaller compression ratio. Notice that if $\sum_{i=1}^{N} B_i^{MIN}(SL_i - 1) > C$, then there exists no feasible bandwidth assignment. In this case, the resource manager can either terminate some applications, or reject the registering application.

VII.2.2 Service level assignment

Let us assume N applications. In the case of registration, N includes the registering application. In the case of unregistration, N does not include the unregistering application. There exists a feasible bandwidth assignment if, and only if, $\sum_{i=1}^{N} B_i^{MIN}(SL_i - 1) \leq C$. Else, either the resource manager rejects the registering application, or terminates already registered applications.

The service level assignment works as follows. First, the resource manager assigns the minimum service level to each registered application, and service level SL_i to the registering application. If terminations are allowed, the minimum service level is SL_i, which indicates that the application terminates. Else, the minimum service level is $SL_i - 1$. Then, the resource manager initializes an array containing the bandwidth density of each applications for each service level. The bandwidth density, for a given service level sl_i, is $imp_i \times qos_i(sl_i)/B_i$. Each element of this array contains the index of the respective application and service level. The resource manager scans this array in decreasing order of bandwidth density, and changes the service level of the corresponding application if (i) the service level of the array's element upgrades the current service level assignment of the application, and (ii) the new service level assignment does not violate the condition $\sum_{i=1}^{N} B_i^{MAX}(sl_i) \leq C$. At the end of the scanning processes, all applications have their respective service level assignment. Applications with service

level SL_i terminate, hence freeing resources.

VII.2.3 Core assignment

The core assignment for virtual processors of the registering application uses a simple greedy algorithm. The resource manager assigns the virtual processor with more bandwidth demand to the core with more free capacity that does not hold another virtual processor of the same application. This process repeats until all virtual processors have a core assignment.

This algorithm may return an assignment of virtual processors to cores that violates the capacity constraint of some cores, i.e. $\sum_{CA_{i,j}=k} BWD_{i,j} > c_k$ for some k. Therefore, we also propose a migration algorithm in order to distribute the load among cores. The migration algorithm runs also upon unregistration, and consists of the 2 steps that we describe below.

In the first step, for each core $k|\sum_{CA_{i,j}=k} BWD_{i,j} > c_k$, the migration algorithm searches for the virtual processor $VP_{x,y}$ with higher bandwidth, and moves $VP_{x,y}$ to the core $k'|\sum_{CA_{i,j}=k'} BWD_{i,j} + BWD_{x,y} \leq c_{k'}$. For each core k, the migration algorithm repeats this step until either $\sum_{CA_{i,j}=k} BWD_{i,j} \leq c_k$ or there is no $k'|\sum_{CA_{i,j}=k'} BWD_{i,j} + BWD_{x,y} \leq c_{k'}$.

In the second step, for each core k, the migration algorithm searches for the virtual processor $VP_{x,y}$ with higher bandwidth, and moves $VP_{x,y}$ to the core $k'|\sum_{CA_{i,j}=k} BWD_{i,j} > \sum_{CA_{i,j}=k'} BWD_{i,j} + BWD_{x,y}$. For each core k, the migration algorithm repeats this step until there is no core $k'|\sum_{CA_{i,j}=k} BWD_{i,j} > \sum_{CA_{i,j}=k'} BWD_{i,j} + BWD_{x,y}$.

VII.2.4 Bandwidth compression

The assignment of virtual processors to cores may violate the capacity constraint of cores. In this case, for each core, the resource manager compresses the bandwidth of the virtual processors until $\sum_{CA_{i,j}=k} BWD_{i,j} = c_k$. In this section, we present a bandwidth compression algorithm based on an analogy with pendulum systems.

Pendulum system

A pendulum is an object attached to a pivot point that can swing freely. A basic example is the simple gravity pendulum or bob pendulum. As depicted in figure VII.6, it consists of a massive bob (bob_i) hanging by a massless string of length L_i. When given an initial push, the bob will swing back and forth under the influence of gravity over its central (lowest) point in a circular trajectory. Placed at the lowest point, the bob will come to rest there (*rest position*). As depicted in figure VII.7, applying a force F perpendicular to the string will cause the bob to rest at a position x_i, hence, altering the *equilibrium state* of the system. We call the angular displacement $\varphi_i|0 \leq \varphi_i \leq \pi/2$. The value of x_i depends on the intensity of the force F and the weight W_i of the bob. This analogy differs from the one presented in chapter III in the interaction among the bobs. Here each bob is in an isolated pendulum, and all bobs are pushed by a common force F.

$$F = W_i \times sin(\varphi_i) \tag{VII.1}$$

$$F = W_i \times \frac{(x_i - rp_i)}{L_i} \tag{VII.2}$$

$$x_i = \frac{W_i \times rp_i + F \times L_i}{W_i} \tag{VII.3}$$

$$x_i = rp_i + \frac{F \times L_i}{W_i} \tag{VII.4}$$

Let us define the lowest point of a pendulum as rp_i. At the maximum angular displacement (i.e. $\varphi_i = \pi/2$), $x_i = rp_i + L_i$. Given a force F, the pendulum is in equilibrium when $F = W_i \times sin(\varphi_i)$. After a few algebraic steps, we obtain the value of x_i (see equation VII.4). As can be seen is this equation, x_i is inversely proportional to W_i.

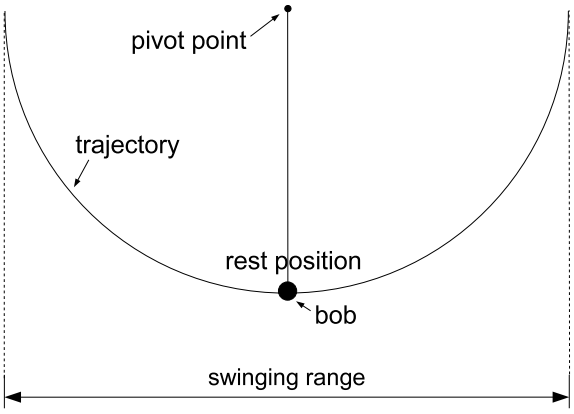


Figure VII.6: *Bob pendulum*

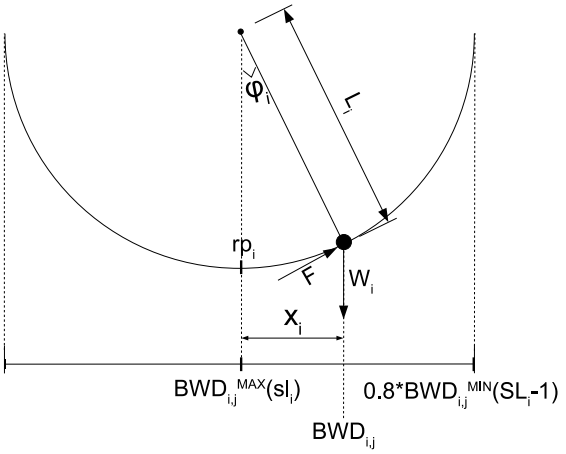


Figure VII.7: *Analogy between pendulum system and bandwidth compression*

Analogy between pendulum system and bandwidth compression

Figure VII.7 depicts the analogy between the pendulum system and the bandwidth allocation for virtual processors. Let us assume each application A_i operates at a service level sl_i. The target of a virtual processor belonging to A_i is to obtain a bandwidth $BWD_{i,j} = BWD_{i,j}^{MAX}(sl_i)$, and the target of a bob is to rest at the central (lowest) point $x_i = rp_i$. The product of the importance and the quality of service of an application is inversely proportional to the bandwidth compression ratio, and similarly, the weight of a bob is inversely proportional to x_i. Therefore, $W_i \equiv imp_i \times qos_i(sl_i)$. Bobs cannot be pushed beyond the swinging range, and a virtual processor $VP_{i,j}$ cannot obtain a bandwidth smaller than $0.8 \times BWD_{i,j}^{MIN}(SL_i - 1)$. Finally, the force F is equivalent to the resistance that the system imposes on the applications to compress the bandwidth assignment. Following this analogy, it is easy to see that the position x_i of each bob in

the equilibrium state is equivalent to a fair compression of the bandwidth assignment for the virtual processors on a core.

$$F = imp_i \times qos_i(sl_i) \times sin(\varphi_i) \quad \text{(VII.5)}$$

$$F = imp_i \times qos_i(sl_i) \times \frac{(BWD_{i,j}^{MAX}(sl_i) - BWD_{i,j})}{BWD_{i,j}^{MAX}(sl_i) - 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1)} \quad \text{(VII.6)}$$

$$BWD_{i,j} = BWD_{i,j}^{MAX}(sl_i) - \frac{F \times (BWD_{i,j}^{MAX}(sl_i) - 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1))}{imp_i \times qos_i(sl_i)} \quad \text{(VII.7)}$$

Expanding the equilibrium condition, i.e. $F = W_i \times sin(\varphi_i)$, using the applications' parameters leads to equation VII.7. This equation gives the bandwidth assignment for each $VP_{i,j}$ upon a system resistance F.

Exploiting the intuition

In this section, we compare the bandwidth compression for resource allocation with the equivalent bob pendulum system. The goal is to observe the dynamics of both systems, and exploit the intuition from the analogy to better understand the meaning of some applications' parameters. Let us consider 2 applications A_1 and A_2 with one virtual processor each, and importances imp_1 and imp_2 respectively. For simplicity, let us assume only one service level ($qos = 1$) with maximum bandwidth requirements $B_1^{MAX} = 10$ and $B_2^{MAX} = 5$, and minimum bandwidth requirements $B_1^{MIN} = 2$ and $B_2^{MIN} = 1$.

A force F compresses these bandwidths as follows:

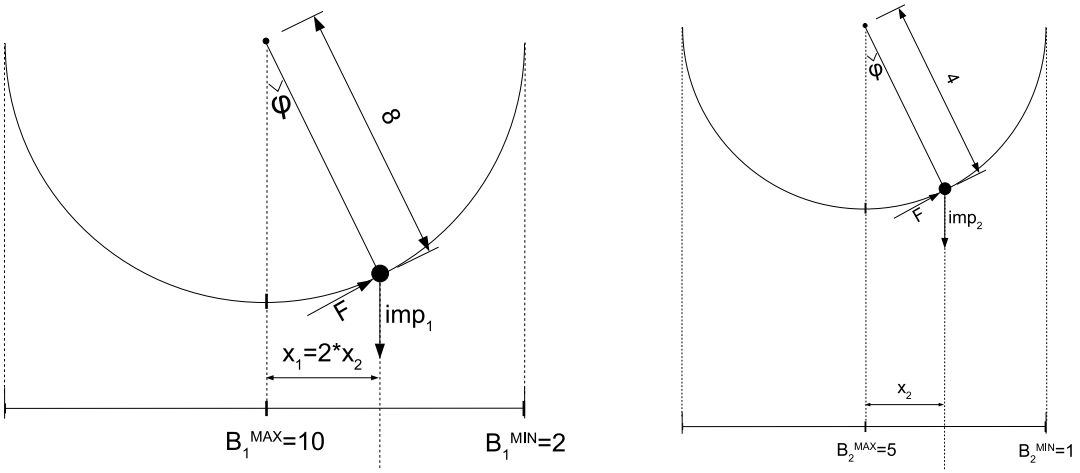
$$B_1 = B_1^{MAX} - \frac{F \times (B_1^{MAX} - B_1^{MIN})}{imp_1} = B_1^{MAX} - \frac{F \times 8}{imp_1} \quad \text{(VII.8)}$$

$$B_2 = B_2^{MAX} - \frac{F \times (B_2^{MAX} - B_2^{MIN})}{imp_2} = B_2^{MAX} - \frac{F \times 4}{imp_2} \quad \text{(VII.9)}$$

As can be seen, $\frac{F \times 8}{imp_1}$ and $\frac{F \times 4}{imp_2}$ stand for the amount of bandwidth compression, being the maximum allowed compression the difference between the maximum and the minimum bandwidth requirements. The actual compression is a fraction of the maximum compression, and proportional to F divided by the importance of the respective application (e.g. F/imp_1 for A_1).

Let us first assume $imp_1 = imp_2 = imp$. In this case, the bandwidth of both applications are compressed by a factor F/imp. However, a compression by the same factor results in different absolute bandwidth compression. As can be see in figure VII.8, the compression factor is equivalent to the relative displacement of the bob (both bobs have the same angular displacement), and the actual bandwidth compression is equivalent to the actual displacement of the bob.

If $imp_1 = k \times imp_2$ for any k, the compression factor scales linearly with k. The compression of A_1 is $\frac{F \times 8}{k \times imp_2}$, and the compression of A_2 is $\frac{F \times 8}{imp_2}$. As can be seen, higher



(a) *Application* A_1 **(b)** *Application* A_2

Figure VII.8: *Analogy between pendulums and applications (same importance).*

importance leads to smaller compression. Similarly, a heavy bob experiences a smaller angular displacement than a light bob under the influence of the same force F.

This model also allows applications to express compression inflexibility. This is the extreme case when $k \to \infty$, which is equivalent to a bob with weight infinity. Applications with importance infinity can still have their resource assignment compressed, but those are the last one's to experience any compression, in which case $F \to \infty$.

Compression algorithm

In the next section, we describe how the bandwidth allocation algorithm chooses F such that $\sum_{i=1}^{N} BWD_{i,j} \leq c_k$, for each core k. For a given a core k, let us define Γ_C as the set of virtual processors whose bandwidth can be compressed, and Γ_L as the set of virtual processors whose bandwidth are compressed to the maximum. The bandwidth of each $VP_{i,j} \in \Gamma_C$ must fulfill equation VII.11. Expanding this equation, we obtain the value of F (see equation VII.12).

The compression algorithm works as follows for each core (see algorithm 5). Initially, Γ_C contains all virtual processors on core k and Γ_L is empty (lines 1 and 2). Then, the algorithm calculates 3 intermediate values `sum_1, sum_2, sum_3` (lines 3 to 5) for the calculation of F (line 8). Notice that if $F > imp_i \times qos_i(sl_i)$, then $BWD_{i,j} < BWD_{i,j}^{MIN}(sl_i)$, and such a bandwidth assignment is not valid because $BWD_{i,j}^{MIN}(sl_i) \leq BWD_{i,j} \leq BWD_{i,j}^{MAX}(sl_i)$. Therefore, for each $VP_{i,j} | F > imp_i \times qos_i(sl_i)$, the algorithm removes $VP_{i,j}$ from Γ_C (line 14), appends it to Γ_L (line 15), and appropriately updates the intermediate values `sum_1, sum_2, sum_3` to reflect the changes in the sets Γ_C and Γ_L (lines 16 to 18). The algorithm, then, recalculates the force F (line 8), and repeats the succeeding lines until $F > imp_i \times qos_i(sl_i)\ \forall VP_{i,j} \in \Gamma_C$ (lines 7 and 12 adjust the repeat loop condition). Finally, the algorithm calculates $BWD_{i,j}$ for all $VP_{i,j} \in \Gamma_C$ (lines 22 to 25). In line 26, the algorithm updates the service level assignment of each application to be consistent with the assigned bandwidth. The algorithm, then, returns

a concatenation of sets Γ_L and Γ_C (line 27).

$$\sum_{VP_{i,j} \in \Gamma_C} BWD_{i,j} = C - \sum_{VP_{i,j} \in \Gamma_L} 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1) \quad \text{(VII.10)}$$

$$\sum_{VP_{i,j} \in \Gamma_C} BWD_{i,j}^{MAX}(sl_i) -$$

$$F \times \sum_{VP_{i,j} \in \Gamma_C} \left(\frac{BWD_{i,j}^{MAX}(sl_i) - 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1)}{imp_i \times qos_i(sl_i)} \right) =$$

$$C - \sum_{VP_{i,j} \in \Gamma_L} 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1) \quad \text{(VII.11)}$$

$$F = \frac{\sum_{VP_{i,j} \in \Gamma_C}(BWD_{i,j}^{MAX}(sl_i)) - C + \sum_{VP_{i,j} \in \Gamma_L} 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1)}{\sum_{VP_{i,j} \in \Gamma_C} \left(\frac{BWD_{i,j}^{MAX}(sl_i) - 0.8 \times BWD_{i,j}^{MIN}(SL_i - 1)}{imp_i \times qos_i(sl_i)} \right)} \quad \text{(VII.12)}$$

The code from line 1 to line 5 has complexity $O(N)$. The `while` loop does not execute more than N times in total (not only for a iteration of the `repeat` loop), since each element of Γ_C is visited only once. Each line of code within this loop has complexity $O(1)$, and hence, the `while` loop has complexity $O(N)$. The `repeat` loop does not execute more than N times in total neither, because the variable `repeat` assumes the value `false` at most N times (within the `while` loop). Finally, the `forall` loop has complexity $O(N)$. Therefore, the complexity of the compression algorithm is $O(N)$.

VII.2.5 Example

This section illustrates the bandwidth compression algorithm with a simple example, where 3 applications register with the resource manager. Table VII.1 shows the service levels information of the 3 applications. For this example, we assume a system with 4 cores, and each core has capacity of 90 to serve actors-aware applications. Therefore, the total system capacity is 360. Let us consider as initial state that applications A_1 and A_2 are already registered with the resource manager, and running at their maximum service level. Table VII.2 shows the resource distribution for both applications.

When A_3 registers with the resource manager, the sum of bandwidth requirement for all applications at their maximum service level is 440. Therefore, system resources are insufficient to allow for all applications to run at their maximum service level. In this case, the resource manager executes the service level assignment algorithm (as described in section VII.2.2), which assigns service level 0 to A_1, service level 2 to A_2, and service level 0 to A_3. The sum of bandwidth requirements for all applications under this service level assignment is 160+120+80 = 360. However, the assignment of virtual processors to

Algorithm 5 Bandwidth compression algorithm.

```
1: Γ_C = VPs;
2: Γ_L = ∅;
3: sum_1 = ∑_{VP_{i,j} ∈ Γ_C} BWD^{MAX}_{i,j}(sl_i);
4: sum_2 = ∑_{VP_{i,j} ∈ Γ_L} BWD^{MIN}_{i,j}(sl_i);
5: sum_3 = ∑_{VP_{i,j} ∈ Γ_C} ( (BWD^{MAX}_{i,j}(sl_i) − BWD^{MIN}_{i,j}(sl_i)) / W_i );
6: repeat:
7:    stop = true;
8:    F = (sum_1 - C + sum_2)/sum_3;
9:    VP = get_first(Γ_C);
10:   while (
                (VP not ∅)
                &&
                (VP.app.imp*VP.app.qos(VP.app.SL-1) < F)
             )
11:   {
12:      stop = false;
13:      VP.BWD = 0.8 * VP.BWD_MIN[VP.app.SL-1];
14:      remove_first(Γ_C);
15:      append(Γ_L);
16:      sum_1 -= VP.BWD_MAX[VP.app.sl];
17:      sum_2 += VP.BWD_MIN[VP.app.sl];
18:      sum_3 -=
                (VP.BWD_MAX[VP.app.sl]-VP.BWD_MIN[VP.app.sl])
                /(VP.app.imp*VP.app.qos(VP.app.sl));
19:      VP = get_first(Γ_C);
20:   }
21:until (stop);
22:forall (VP in Γ_C)
23:{
24:    VP.BWD = VP.BWD_MAX[VP.app.sl]
         - F*(VP.BWD_MAX[VP.app.sl] - VP.BWD_MIN[VP.app.sl])
         /(VP.app.imp*VP.app.qos(VP.app.sl));
25:}
26:update_sl_assignment();
27:return Γ_L ∪ Γ_C
```

cores does not distribute the load equally among the cores. As can be seen in table VII.3, which shows the assignment of virtual processors to cores, core 0 is overloaded. The resource manager, then, applies the compression algorithm 5 to shrink the bandwidth of the virtual processors on core 0, resulting in the bandwidth assignment of table VII.4. This bandwidth assignment results in a service level reassignment for applications A_1

App	Imp	SL	QoS [%]	BW [%]	Δ [ms]	BWD [%]
A1	10	0	100	160	40	[VP_0 =40, VP_1 =40, VP_2 =40, VP_3 =40]
		1	80	120	50	[VP_0 =30, VP_1 =30, VP_2 =30, VP_3 =30]
		2	50	80	100	[VP_0 =20, VP_1 =20, VP_2 =20, VP_3 =20]
A2	1	0	100	200	20	[VP_0 =50, VP_1 =50, VP_2 =50, VP_3 =50]
		1	90	160	40	[VP_0 =40, VP_1 =40, VP_2 =40, VP_3 =40]
		2	70	120	70	[VP_0 =30, VP_1 =30, VP_2 =30, VP_3 =30]
		3	40	80	150	[VP_0 =20, VP_1 =20, VP_2 =20, VP_3 =20]
A3	100	0	100	80	20	[VP_0 =20, VP_1 =15, VP_2 =45]
		1	70	60	100	[VP_0 =20, VP_1 =10, VP_2 =30]

Table VII.1: *Service level table of applications A1, A2, and A3*

Core	VPs
0	[A1 $VP_0 = 40$; A2 $VP_0 = 50$]
1	[A1 $VP_1 = 40$; A2 $VP_1 = 50$]
2	[A1 $VP_2 = 40$; A2 $VP_2 = 50$]
3	[A1 $VP_3 = 40$; A2 $VP_3 = 50$]

Table VII.2: *Assignment of VPs to cores (Apps A1, and A2)*

Core	VPs
0	[A1 $VP_0 = 40$; A2 $VP_0 = 30$; A3 $VP_2 = 45$]
1	[A1 $VP_1 = 40$; A2 $VP_1 = 30$; A3 $VP_0 = 15$]
2	[A1 $VP_2 = 40$; A2 $VP_2 = 30$; A3 $VP_1 = 20$]
3	[A1 $VP_3 = 40$; A2 $VP_3 = 30$]

Table VII.3: *Assignment of VPs to cores before compression (A1, A2, and A3)*

(service level 1) and A_2 (service level 3), which may change their bandwidth assignments on other cores. The final bandwidth distribution is in table VII.4.

Core	VPs
0	[A1 $VP_0 = 29$; A2 $VP_0 = 16$; A3 $VP_2 = 44$]
1	[A1 $VP_1 = 30$; A2 $VP_1 = 20$; A3 $VP_0 = 15$]
2	[A1 $VP_2 = 30$; A2 $VP_2 = 20$; A3 $VP_1 = 20$]
3	[A1 $VP_3 = 30$; A2 $VP_3 = 20$]

Table VII.4: *Assignment of VPs to cores after compression (A1, A2, and A3)*

VII.3 Opportunistic packet scheduling in body area networks

Besides the efficacy and efficiency in computing trade-offs among target sensitive applications, the gravitational task model also offers an easy intuition for the sake of understanding. This intuition is also a very important contribution, given that real-time (RT) scheduling theory has applicability in diverse fields where deep knowledge of scheduling theory is not imperative. For example, cyber-physical systems emerges as a new research field which combines control systems, embedded systems, Wireless Sensor Networks (WSN), and Real-Time Systems (RTS). Although each of these areas are focus of deep research, recent contributions to the state-of-the-art in each domain tend to remain isolated until the knowledge matures and spreads out.

In this section, we describe an application of the gravitational task model in the field of WSNs. This application has been entirely developed by researchers not involved with studies on the gravitational task model, namely Shashi Prabh from the Center of Real-Time Systems Research at Polytechnic Institute of Porto, and Jan-Hinrich Hauer from the Telecommunication Networks Group at Technische Universität Berlin. This application is part of the work presented in [Prabh 11], which has been published in EWSN'11, a top European conference in the field of wireless sensor network. Therefore, this application evidences the contribution of the gravitational task model's intuition to ease the cooperation among those areas of research, and more, the synergy between recent significant contributions to the state-of-the-art in two different fields.

The opportunistic packet scheduling strategy aims at reduced retransmission ratio of packets in BANs (refer to section II.3 for a detailed description of packet scheduling issues in BANs). Therefore, this strategy achieves increased communication reliability and reduced energy consumption. It uses an algorithm that schedules packets based on the Reveived Signal Strength Indicator (RSSI) fluctuations of the wireless communication channel. We describe in the next section the empirical characterization of RSSI fluctuations in a BAN, and then, the packet scheduling algorithm based on the gravitational task model.

VII.3.1 Characterization of RSSI fluctuations

The characterization of RSSI fluctuations varies among scenarios, e.g. indoors, outdoors, activity of the subject, etc. In this section, we present results of an empirical characterization of RSSI fluctuations while the subject is moving outdoors. This investigation is complementary to studies in indoors settings, which receive more research focus. Please refer to [Prabh 11] for a complete description of the experiments and results. Here we focus on the relevant information to later describe the algorithm for opportunistic packet scheduling.

In the experiments, nodes equipped with IEEE 802.15.4-compliant CC2420 radio transceivers were positioned on the subjects as shown in figure VII.9. Experiments contained TelosB [Polastre 05], Shimmer2 [Burns 10], and MicaZ [Hill 02] platforms in isolated scenarios — i.e. no scenario mixed nodes of different platforms — in order to investigate the influence of the type of platform on RSSI fluctuations. An experiment

consisted of one node (sender) continuously broadcasting IEEE 802.15.4 packets with a constant transmission frequency which was experiment dependent (red node in the figure). The other seven nodes (receivers) passively listened for packets (they did not send acknowledgments). The measurement software accesses the CC2420 radio directly, i.e., there is no Medium Access Control (MAC) layer involved and the senders send packets immediately without clear channel assessment. Each of the seven receiver nodes keeps statistics of the number of correctly received packets and the associated RSSI. The receiver nodes are placed on the left and right ankle, left trouser pocket, left and right hand, in the center of the chest, and in the center of the back (blue nodes in the figure).

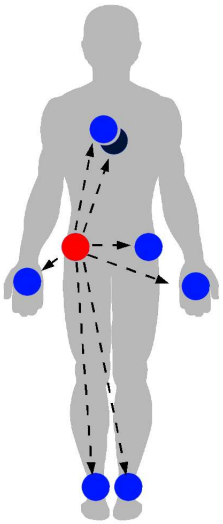


Figure VII.9: *Node positions [Prabh 11]*

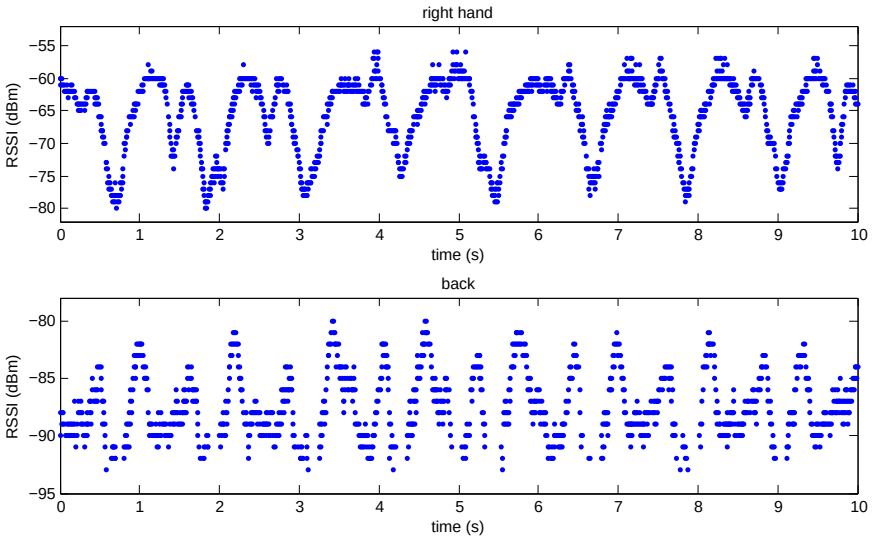


Figure VII.10: *RSSI measurements (right hand and back) [Prabh 11]*

In each experiment the subject that carries the BAN was continuously walking outdoors in a large field with negligible external Radio Frequency (RF) interference, as verified with the help of periodic noise-floor measurements on the nodes. The subjects were walking at even speed of approximately 1.2 steps/s (common walking speed). A single experiment lasted for 5 minutes, and 10 experiments were performed for each scenario.

While a subject was walking, the changes in the relative positions of the limbs manifested as periodic fluctuations in the RSSI. For example, the top graph in figure VII.10 shows a 10-second snapshot of the RSSI obtained in one experiment scenario on a Shimmer2 node that was positioned on the right hand of the subject (recall that the sender is always located in the right trouser pocket). The sender used a transmission power of -10dBm. The graph shows a period of about 1.2s, which matches the step frequency of the subject: often a plateau of about 1s with values of -60dBm is followed by short trough with values as low as -80dBm, resulting in a significant RSSI range of about 20dBm. The node position, however, has an impact on the RSSI pattern: for example, the RSSI time series obtained in the same experiment on the back of the subject is

much more noisy (figure VII.10 bottom). Results for other node platforms have similar RSSI fluctuation patterns, which confirms that the effect is not platform-specific. However, the absolute values of radio signal may vary across platforms, which reflects the difference in transmission power levels of their antennas.

VII.3.2 The packet scheduling algorithm

The goal of this scheduling strategy is to account for the RSSI fluctuations, and schedule packet transmissions for periods of high RSSI values. This way chances are better that the packets are received correctly. The term *opportune transmission window* (OTW) describes a time interval that yields high RSSI values relative to the average RSSI of the link. Assuming that the subject performs regular movements, intuitively, we can predict an OTW by adding the current step period to the time of the previous OTW center.

The main difficulty in using RSSI measurements to predict OTWs arises due to significant noise content in the RSSI measurements. The second challenge arises due to the irregularities of human movements, which are rarely exactly periodic. Consequently, the simplistic approach of locating the peaks and extrapolating the inter-peak separation to predict OTWs fails.

Therefore, the authors proposed a method to predict the OTWs which derives the inter-peak separation of an RSSI time series using Fourier Transformations. The RSSI time series are obtained from a set of initial probe (control) packets which are sent in a very low frequency from the receivers to the sender (coordinator), and assumed to be interspersed between data packets. The coordinator applies Fast Fourier Transformation to convert the RSSI time series to the Fourier domain, where the dominant peak corresponds to the speed of the subject. A tight bandpass filter identifies the phase of the frequency. As human movement is irregular, the coordinator must re-run this method either periodically or on-demand, where the probe packet transmissions is optimized for energy vs. accuracy trade-off. Refer to the original paper [Prabh 11] for a detailed description of the method, as well as evaluations results.

The OTWs are described in terms of the period of RSSI fluctuations (T) and the width of the OTW (Δ) as $OTW = [kT + T/4 - \Delta/2, kT + T/4 + \Delta/2]$, where k is an integer number (see figure VII.11). There is a cost function associated with the time of transmissions. This cost function is directly related to the shape of the RSSI function, and the higher the RSSI, the lower the cost. Thus the cost function has minima at the center of a OTW.

Let us consider a set of n transmissions $\Gamma = \{X_1, \ldots, X_n\}$ to be scheduled in this order during a given OTW, where X_i denotes the events of polling by the coordinator followed by the transmission of data packet and leading probe packets to the coordinator. Each event has an importance ω_i, and the data transmission lasts e_i units of time. If a message comprises m packets, this message is associated with m different events. The goal is to transmit all messages within a OTW minimizing the total cost.

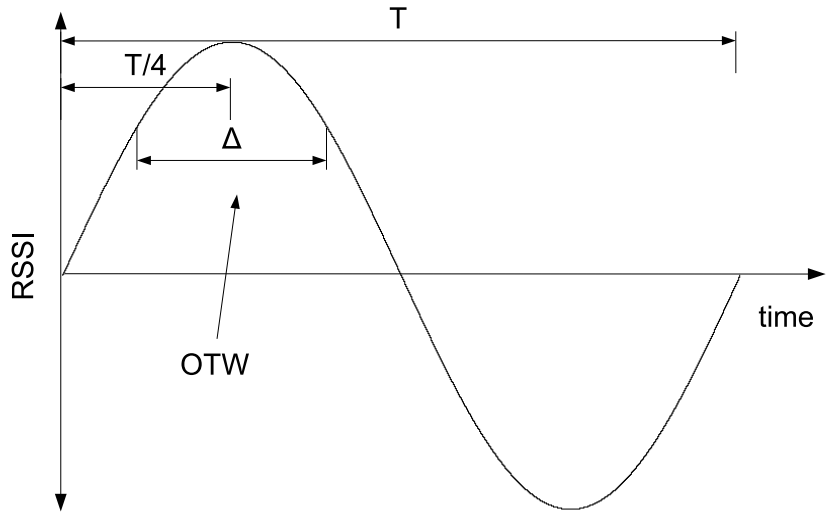


Figure VII.11: *The opportunistic transmission window.*

Pendulum	Transmission events
R_i	$(\Delta - e_i)/2$
W_i	ω_i
P_i	$kT + kT/4$
d_i	e_i

Table VII.5: *Mapping transmission events into particle pendulum parameters.*

Mapping the packet scheduling to the gravitational task model is fairly straight forward, and the implicit utility function of the gravitational task model (see section III.6.1) approximates the cost function. Table VII.5 summarizes the analogy. An transmission event X_i is equivalent to a particle in the particle pendulum systems, with the center of the OTW $(kT + kT/4)$ being the pivot point. The anchor point is the start of the transmission, and thus the distance d_i between adjacent particles is equivalent to the transmission time e_i. The length R_i of the string is equivalent to $(\Delta - e_i)/2$. Finally, the weight W_i of a particle is equivalent to the importance ω_i of a message; there is no need for normalization of the importance to R_i in this case because all transmissions share the same OTW. The equilibrium, then, returns the schedule for the transmission of packets that approximates the minimum total cost.

VII.4 Summary

In this chapter, we presented 3 applications enhanced with the gravitational task model in order to demonstrate the validity of our work. Section VII.1 presented a multimedia application where we use the gravitational task model to reduce the degradation in the PQV during overload. The basic idea is to relax the temporal constraints of the frame display. This way, the decoder may prepone/postpone the frame display in order to cope with the longer decoding times without the need to drop frames. Of course, without further consideration, this approach leads to large variations of inter-display time, which degrades the PQV. We use the gravitational task model to schedule the completion of the decoding task as close as possible to the optimum display instant, hence accounting for both optimum PQV under low system workload, and adaptivity under overload with decreased degradation of the PQV.

We measured the frame display jitter and the frame drop rate, as there is no precise evaluation metric for the PQV. Results showed that with the gravitational task model the jitter distribution is closer to 0, and the frame drop rate is 0. We also produced the resulting video output to visualize the effect of jitter and frame drop, concluding that the gravitational task model provides for better PQV than other approaches.

Section VII.2 presented an algorithm for adaptive resource management under scarce resource availability. We considered a multicore system where applications may contain parallelizable execution flows, and each flow has a resource requirement. Applications may also offer different service levels, depending on the resource availability. Our algorithm uses a gravitational task model based compression method which has linear complexity for reallocation of resources among applications, and service level assignment. The pendulum analogy provides for the intuition of the solution. This algorithm has been implemented in the ACTORS Framework, which is a project of the 7^{th} Framework Programme for Research and Technological Development.

Finally, section VII.3 described an opportunistic packet scheduling strategy which aims at reduced retransmission ratio of packets in BANs (refer to section II.3 for a detailed description of packet scheduling issues in BANs). The basic idea is that sending packets when the RSSI value is high reduces the retransmission ratio, which increases communication reliability and reduces energy consumption. This application has been entirely developed by researchers not involved with studies on the gravitational task model, and evidences the contribution of the gravitational task model's intuition to ease the cooperation among different research fields.

This section described a set of experiments for the characterization of RSSI fluctuations in BANs. The results of this characterization were used to define a transmission window, within which packets are more likely to be successfully transmitted. As all packets share the same target point for transmission, they are scheduled using the equilibrium of the gravitational task model.

Chapter VIII

Discussion

In this chapter, we cover a few scheduling related topics that may impact on the utility accrual of a schedule, and thus, we consider that they should not be ignored. However, due to the complexity to address these topics, we just present a discussion of their impact on scheduling, and potential research directions. These topics are early completion times, preemptive scheduling, and multicore scheduling.

Early completion times, which we discuss in section VIII.1, may deviate the anchor points from their target points, thus altering the utility accrual of the original schedule — the new utility accrual may be higher or lower than the original one. Furthermore, on-line adaptivity upon detection of early completion may provide for higher utility accrual.

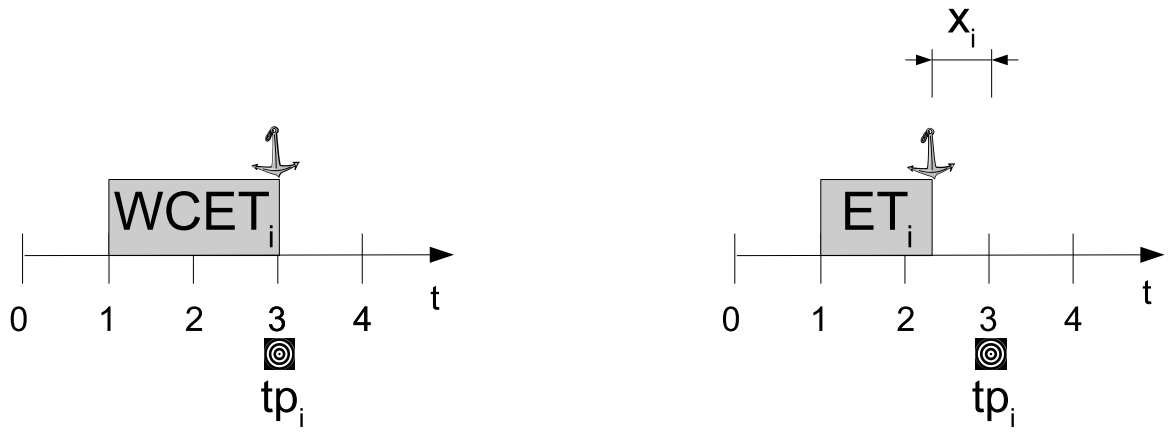
In section VIII.2, we discuss the expressiveness that preemptive scheduling requires from the task model. For that, we use a few examples to understand how preemption relates to the utility accrual in the gravitational task model.

Finally, in section VIII.3, we discuss new issues that arise in multicore scheduling. For example, jobs running in different cores may have precedence constraints, and hence, the equilibrium must consider job chains containing jobs that run in parallel.

VIII.1 Handling early completion times

Throughout this work, we have considered that the actual execution times of jobs are equal to the worst case execution times (WCET)). This assumption is not unrealistic, e.g. the transmission time of a message on a communication bus in packet scheduling is entirely predictable from the message size and the bus frequency. However, most likely execution times are variable. For example, multimedia applications have highly variable execution times. Therefore, in this section we ponder the impact of variable execution times on the schedule of target sensitive applications.

Consider the schedule of a set of jobs in equilibrium assuming the WCETs. There are 2 ways to handle the earlier completion of a job: (i) keep the schedule of subsequent tasks unchanged, and (ii) reschedule the subsequent jobs. The former approach is possible because the gravitational task model is non-work-conserving, i.e. tasks may procrastinate, and clearly does not increase the finishing time of any job. Regarding the utility accrual, early completion of a job may alter the deviation from its anchor point to the target point. In figure VIII.1, we plot the schedule of a job j_i assuming the worst case execution time (see figure VIII.1a), and the actual position of the anchor point upon early completion of j_i (see figure VIII.1b, where ET_i stands for the actual execution time of j_i). The earlier completion of this job deviates its anchor point from the target point by the same amount that the execution is reduced.



(a) *Schedule assuming WCET* **(b)** *Schedule upon early completion*

Figure VIII.1: *Effect of early job completion.*

The latter approach may consider only reapplying the equilibrium, or also reordering the execution of jobs. The equilibrium upon early completion of a job will never shift the execution of any job forward in time (see theorem VIII.1). The computation of the equilibrium of jobs does not require any special consideration, and thus, is trivial. It has linear complexity, and accounts for increased utility accrual. Reordering the execution of jobs also provides for higher utility accrual, but may also increase the finishing time of some jobs, and potentially cause deadline misses. Therefore, reordering the execution

of jobs requires further special consideration upon early job completion (not covered in this work).

Theorem VIII.1. *The early completion of a job will never shift the execution of any job forward in time.*

Proof. Consider a job chain jc with C jobs, and that job j_i completes earlier. We can split this chain into 2 parts: jobs j_1 to j_i in job chain jc', and jobs j_{i+1} to j_C in job chain jc'' (see figure VIII.2). Of course, chains jc' and jc'' have to overlap when in equilibrium in order to merging into jc. This overlap implies that jc'' starts execution earlier in the absence of jc'. Therefore, if the former chain becomes shorter due to the early completion of j_i, the latter chain can only shift backwards upon equilibrium computation. □

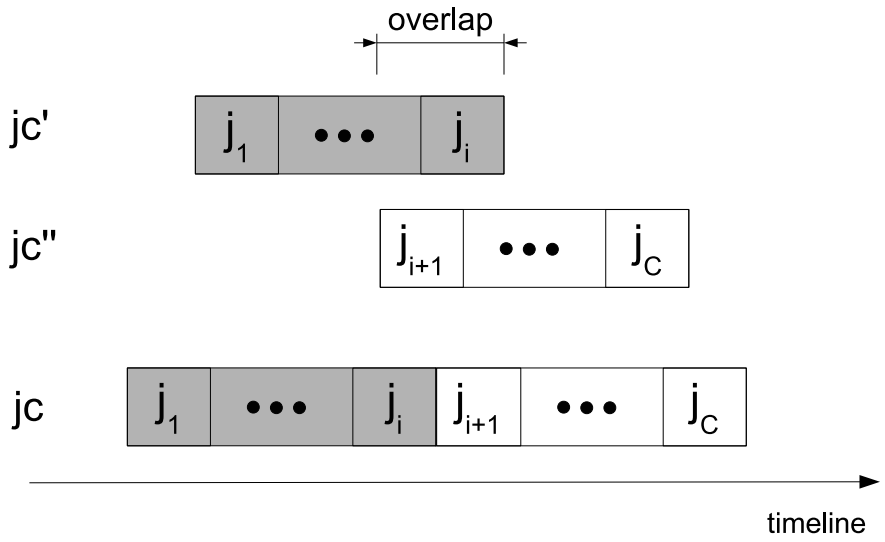


Figure VIII.2: *Splitting a job chain into 2.*

Obviously, highly variable execution times have a bigger impact on the utility accrual. In this case, one approach to mitigate the negative effect of early completion on the utility accrual is to assume average execution times upon the initial equilibrium calculation, and react to the actual execution times at runtime. Upon shorter execution times recomputing the equilibrium allows for increased utility accrual; upon longer execution times, the scheduler can only delay the execution of subsequent jobs, which incurs in less utility accrual. Furthermore, without any further consideration in the original schedule, this delay may cause deadline misses.

The work in [Liu 10], handles variable execution times in Time Utility Function (TUF) schedulers using the concept of opportunistic cost to order the execution of jobs based on their utilities and probability density function of the execution times. The goal is to increase the probability of accruing more utility based on stochastic observations. This

work considers only job ordering, and may cause jobs to miss their deadlines, since their execute time may be longer than assumed initially. We believe on the possibility to develop a scheduling algorithm for the gravitational task model which is based on this concept of opportunistic cost and guarantees timing constraints.

VIII.2 Preemptive scheduling

So far, we have only considered non-preemptive scheduling algorithms. The advantages of non-preemptive scheduling include no job context saving and switching overhead, which may significantly increase the WCET. These overheads involve, for example, loss of cached data which have to be retrieved from main memory again. Moreover, some simple embedded system do not support preemption, and some applications, like network packet scheduling, are inherently non-preemptive. Nonetheless, preemptive scheduling opens up possibility for more efficient resource utilization, and hence, cannot be ignored.

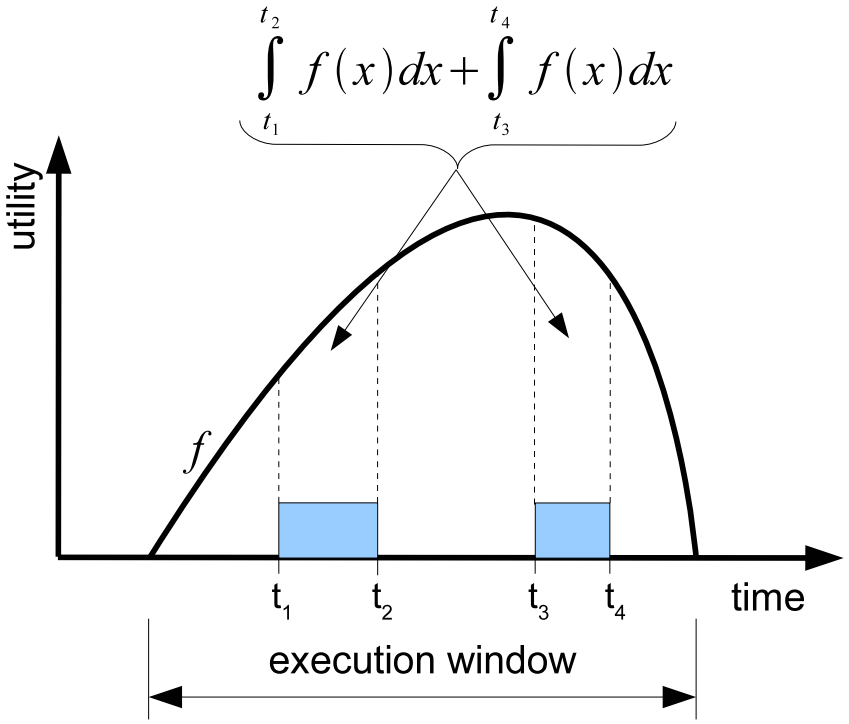


Figure VIII.3: *TUF model in [Farzinvash 09].*

Most work on TUF schedulers are non-preemptive (see the related work in section I.4), and we believe that this tendency is due to the lack of understanding of the impact of preemption on the utility accrual. The work in [Farzinvash 09] proposes a preemptive TUF based scheduler which assumes that each execution quantum of a job accrues utility to the system as a function of the moment of execution, and to the best of our

knowledge, this is the only work on TUF that considers preemption. The utility of a job is, then, the integral of the utility function within the time intervals this job executes (see figure VIII.3). This figure depicts the schedule of a job which executes in the time intervals $[t_1, t_2]$ and $[t_3, t_4]$, and the corresponding utility accrual.

We believe that this task model does not correctly express the utility accrual of applications as a function of time. Our studies led us to conclude that the utility that an application accrues to the system varies as a function of the moment of I/O operations. The internal computational state of an application is not visible to the system, and hence, cannot alter the utility accrual. For example, the authors of [Farzinvash 09] use as motivating example a tracking system which verifies whether objects cross a certain boundary and intercepts them. This system reads the coordinate of the objects from registers that are periodically updated by a sensory system. The utility of this application depends on the age of the sensed position when applying the interception (the older the data, the less accurate the position of the object).

Let us consider now 2 target sensitive applications that we presented in this thesis to analyze the impact of preemption on utility accrual: multimedia applications, and control systems. As discussed in section II.1, video decoding and playout have strict timing constraints for maximum perceived quality of video (PQV). This section also presented the basic data structure of MPEG-2 video files, and the resource requirements for proper decoding and display. The PQV varies as a function of the display time of each frame, and frames can be displayed only after decoded. Frames are decoded on a slice basis, and the exact instant that each slice is decoded has no impact on the PQV. For this reason, the video adaptation strategy that we presented in section VII.1 computes the utility of a frame as a function of an anchor point which represents the display instant.

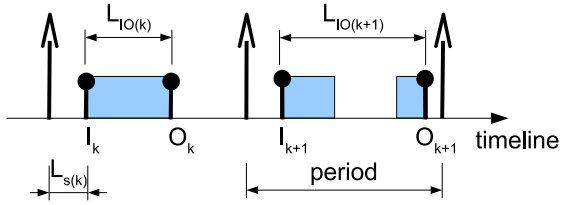


Figure VIII.4: *Control timing constraints.*

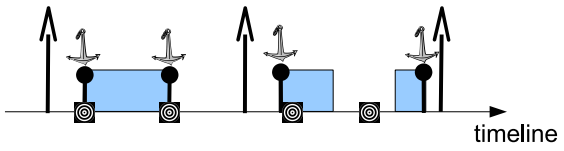


Figure VIII.5: *Gravitational task model with multiple anchor points and target points.*

The utility accrual of control applications, on the other hand, may vary upon preemption. In II.5, we described the basic timing constraints of control tasks. Besides the traditional constraints like periods and deadlines, these tasks have 2 types of latency constraints: I/O latency L_{IO}, and sampling interval latency L_s (see figure VIII.4). The lower the latencies, the higher the utility accrual. Variations of these latencies among task instances also decrease the utility accrual. Please, refer to section II.5 for further details on these timing constraints.

The I/O latency and jitter are not an issue in non-preemptive scheduling, as they depend only on the actual execution times, which do not depend on the scheduler. For

preemptive scheduling, on the other hand, the task model must explicitly express these timing constraints in order to account for increased utility accrual. One possibility is to extend the gravitational task model to allow jobs to express multiple anchor points and target points (see figure VIII.5). Other extensions, such as mobile target points in order to relate anchor points to events, are part of future work on the gravitational task model and respective scheduling algorithms.

VIII.3 Multicore scheduling

Until the 2000s, the processing performance of computing systems had increased exponentially over time. This performance increase was able to attend the demand for IT products and services coming from virtually every sector of society like manufacturing, financial services, education, science, government, military, and entertainment. However, processor performance growth faced an abrupt slowdown in the last decade due to power, architectural, and physical limitations. Multicore system arouse then as a promising alternative to bypass these limitations, yet brought a lot of other issues along. For example, traditional sequential programming models do not provide for parallel processing, and hence, are unable to exploit the performance of multicore architectures. Furthermore, the communication among cores require complex memory hierarchies and data busses which, if not properly managed, can negatively affect the system performance.

Real-time (RT) scheduling theory also had to be revisited in order to account for improved system utilization in multicore architectures. The research in this field includes investigations on timeliness properties [Leontyev 10], proposal of new task models and scheduling algorithms to express and exploit parallelism [Eker 03], reduce scheduling overhead [Bastoni 10], etc.

As mentioned in section II.4, dataflow graphs offer a representation that efficiently supports parallelization, vectorization and synthesis of both hardware and software (for instance see [Lee 87, Ritz 93]). These features meet the requirements of increasingly complex execution platforms e.g. for embedded multimedia systems: *parallelization* is required to utilize multi-core architectures, *vectorization* is required to utilize the so-called SIMD (Single Instruction Multiple Data) or "multimedia" instructions, and application-specific hardware acceleration emphasizes the need of *hardware/software partitioning*. This observation has motivated the research on dataflow-based programming paradigm [Eker 03].

Executing consumers directly after their respective producers in a dataflow graph increases the probability to transport tokens directly using processor registers, i.e. without buffer memory. Therefore data intensive applications, e.g. multimedia applications, can achieve significant performance enhancement. However, this constraint may be relaxed for the sake of feasibility, and the trade-off among several producers and consumers must account for increased system utility.

Multicore architectures also create new issues in the context of target sensitive applications. For example, tasks running on different cores may have precedence constraints. On the one hand, solving these constraints with reduced execution windows compro-

mises feasibility. On the other hand, enlarging the execution windows for the sake of feasibility requires the scheduler to take care of those precedence constraints. Therefore, the equilibrium calculation must be extended to consider the interactions among jobs running on different cores.

The target sensitive constraints must also be considered in combination with cache and other architectural constraints when assigning jobs to cores in order to not compromise the system performance. For example, jobs running on different cores and exchanging large amounts of data impose a large time overhead. In this context, running jobs with conflicting target points on different cores does not necessarily lead to higher utility accrual, as the communication overhead may drastically decrease the system performance. Therefore, accounting for the target sensitivity of tasks must not neglect other issues inherent from multicore architectures, and may require revisiting previous solutions which are unaware of target sensitive constraints.

VIII.4 Summary

In this chapter, we discussed 3 scheduling related topics that may impact on the utility accrual of a schedule: early completion times, preemptions, multicore architectures. Although we have considered the actual execution times of jobs are equal to the WCETs throughout this work, most likely execution times are variable. For example, multimedia applications have target sensitive constraints, and highly variable execution times. We showed in section VIII.1 how can this alter the utility accrual of a schedule, and discussed pros and cons of possible solutions.

In section VIII.2, we discussed the expressiveness that preemptive scheduling requires from the task model. We used the control application described in section II.5 as example to illustrate the impact of preemption on utility accrual, and to identify which the support that task models have to supply.

Finally, in section VIII.3, we briefly discussed the issues involved in scheduling of target sensitive applications on multicore systems.

Chapter IX

Conclusions

Real-time (RT) applications have timing constraints additionally to their logical output for functionally correct system behavior. Real-Time Systems (RTS) commonly consist of multiple tasks that concurrently compete for system resources, and hence, RT applications demand special scheduling algorithms which account for timing constraints. These algorithms, called *RT scheduling algorithms*, define a set of rules to schedule the execution of tasks at system run-time in order to preserve timing constraint. For example, some tasks have earliest start time and deadline constraints, which define the *execution window* of a task. This window is the interval of time where inputs become available and the logical output results in correct system behavior. The scheduler must guarantee that all tasks entirely execute within their respective execution windows.

Some RT applications have tight optimum execution windows for maximum system utility, but accept some flexibility to enlarge those windows for the sake of feasibility. This flexibility comes at the expense of a utility decay, though. For example, tasks of *target sensitive applications* should preferably execute at a specific target point within its execution window, called *target point*, but can execute around this point, albeit at lower utility — the intensity of the utility decay depend on the importance of the application to the system. In this case, the optimum execution window is extremely tight. Ideally, all executions would be scheduled directly at the respective target points, but it might not be feasible due to overlapping executions. Under this condition, the execution of tasks must be scheduled so that no timing constraints are violated and the accrued system utility is maximized. More important applications are less tolerant to deviations from the target point.

On-line adaptivity requires scheduling algorithms with low overhead, and the compromise among tasks with overlapping executions involves reordering the execution of jobs, shifting them and possibly aborting some executions. One of the difficulties is to express whether two not very important tasks are more important than a very important task. Scheduling must also account for the execution of future jobs before shifting executions, and a complete knowledge of jobs that will execute in the system and the exact temporal constraints may be hard, if not impossible (e.g. aperiodic tasks). Finally, reordering the execution sequence of jobs is a combinatorial problem [Chen 96]. Therefore, scheduling is non-trivial, and the need for a simple, yet effective, solution imposes an extra challenge.

In this thesis, we presented a novel task model, called *gravitational task model*, which is based on a physical system: the pendulum. A pendulum is an object (or bob) that is attached to a pivot point and can swing freely. The rest position of a single object in a pendulum is the projection of the pivot point (central point). An object placed in this position will not swing, and the system is said to be in an *equilibrium state*. If there is more than one object, they will push each other aside and their rest position will depend on their relations between weight and size. The heavier an object is, the stronger the gravity drags it to the central point.

We drew an analogy between real time systems and pendulum systems, with instances of tasks (jobs) as objects in a pendulum and the target points as the pivot point. A point within the execution of a job, called *anchor point*, relates the job to the target point. The anchor point can be an operation of input, output, or both. The equilibrium state of the physical problem is, then, equivalent to the best compromise among the jobs' interests. This model assumes non-preemptive tasks running on single processor systems. This analogy provides for simple abstraction of applications' requirements and intuition, which eases the understanding of the task model by applications developers.

We proposed a solution for the trade-off among competing jobs based on the equilibrium of pendulums. The equilibrium state in the physical problem depends on the weights of the bobs, and can be seen as the best compromise among the jobs' importances: This equivalence makes it possible to use the equilibrium equation from physics to schedule jobs aiming at maximizing the utility accrual of the system for a given execution sequence of jobs. We presented both an approximation and an optimum solution to the best compromise among jobs with conflicting interests. We called these solutions *pendulum equilibrium* and *generic equilibrium*, respectively. We also proposed a few scheduling algorithms for the gravitational task model which fulfill the requirements for on-line adaptivity. These algorithms use the equilibrium state concept to reorder the execution sequence of jobs, and compute the deviation of jobs from their target points for increased system utility.

We also presented 3 applications enhanced with the gravitational task model in order to demonstrate the validity of our work:

- A video stream adaptation strategy where we use relax the temporal constraints of video playout so that the decoder may prepone/postpone the frame display in order to cope with the longer decoding times without the need to drop frames. Of course, without further consideration, this approach leads to large variations of inter-display time, which degrades the perceived quality of video (PQV). We use the gravitational task model to schedule the completion of the decoding task as close as possible to the optimum display instant, hence accounting for both optimum PQV under low system workload, and adaptivity under overload with decreased degradation of the PQV.

- Algorithm for adaptive resource management which uses a gravitational task model based compression method for reallocation of resources among applications under scarce resource availability. The pendulum analogy provides for the intuition and low complexity of the solution.

- An opportunistic packet scheduling strategy which aims at reduced retransmission ratio of packets in Body Area Networks (BAN). Reducing the retransmission ratio increases communication reliability and reduces energy consumption. This application has been entirely developed by researchers not involved with studies on the gravitational task model, and evidences the contribution of the gravitational task model's intuition to ease the cooperation among different research fields.

This work covered the task model and scheduling algorithms that account for the specific timing constraints of target sensitive applications. However, there are several scheduling related topics that require further investigation in this new context. Some of them we discussed previous chapter (early completion times, preemption, and multicore scheduling), but possibilities are numerous. For example, scheduling of sporadic tasks, mutual exclusion of shared resources, etc.

Appendix A

Extended results

In this appendix, we plot the histograms of the target point deviation, and the barplots of the fraction of skipped frames for all videos in the experiments described in section VII.1. We omitted these results in their respective section due to the similarities in the results for all videos.

In the histogram plots, the x-axis is the deviation in milliseconds — each bar of the histogram has a width of 5 milliseconds —, and the y-axis is the number of frames. Remember that the total number of frames in each stream is 7500. Each graph plots the histogram for 3 task sets: E, F, and G (description in section VII.1). These task sets have number of streams 5, 10, and 15, respectively. Each figure contains two sets of histogram plots: one with the y-axis ranging from 0 to 7500, and one with the y-axis ranging from 0 to 5.

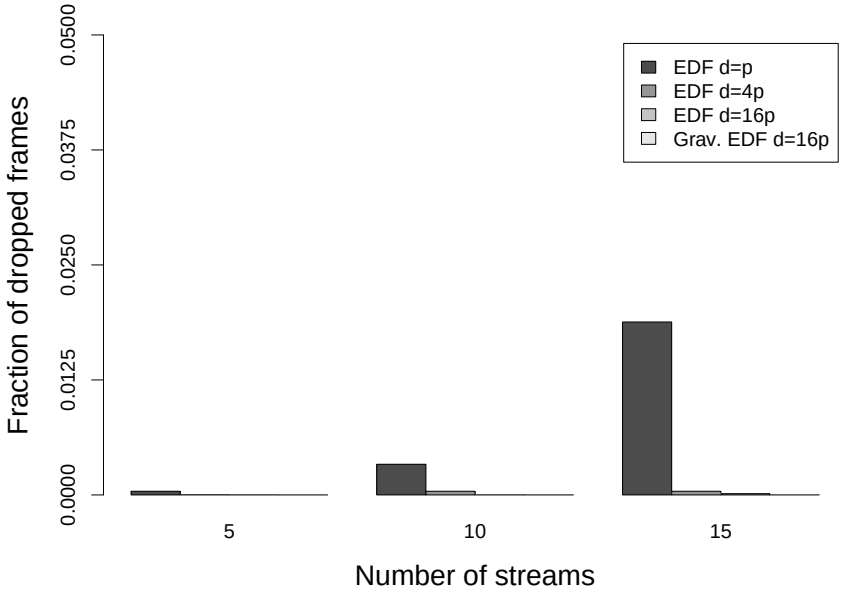
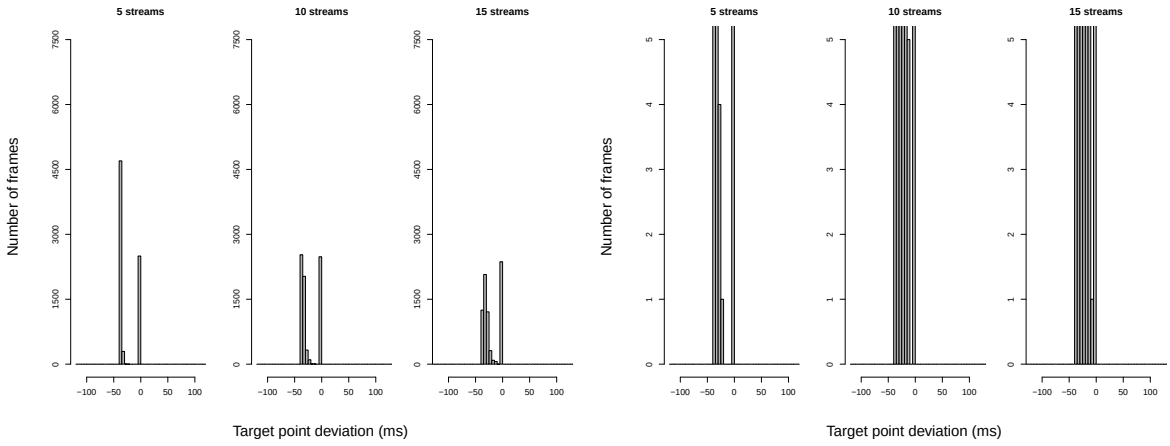
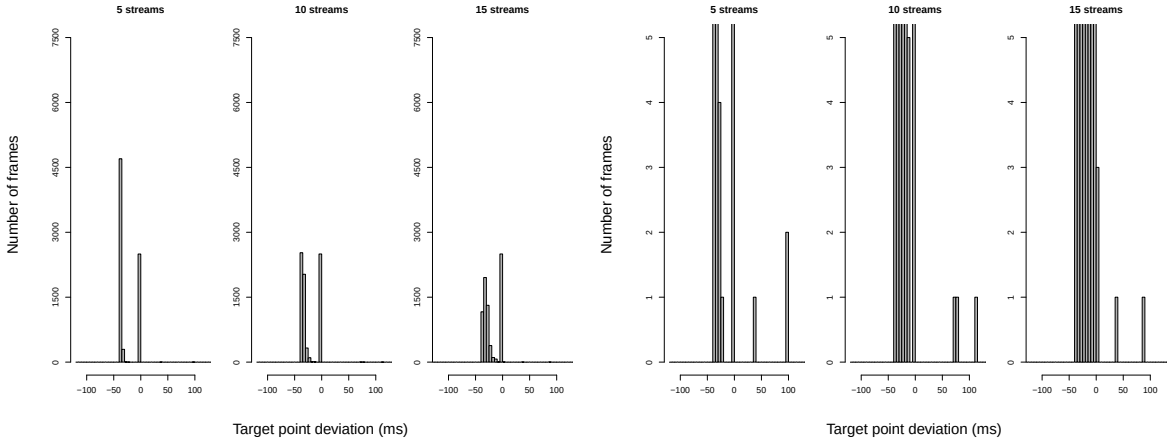


Figure A.1: *Dropped frames for soccer stream.*



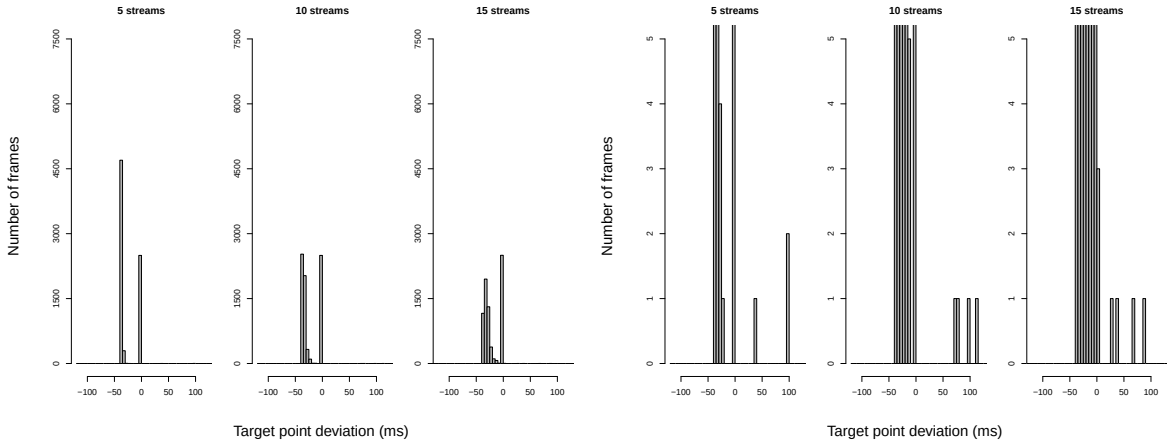
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.2: *Histogram of target point deviation for soccer stream under EDF d=p.*



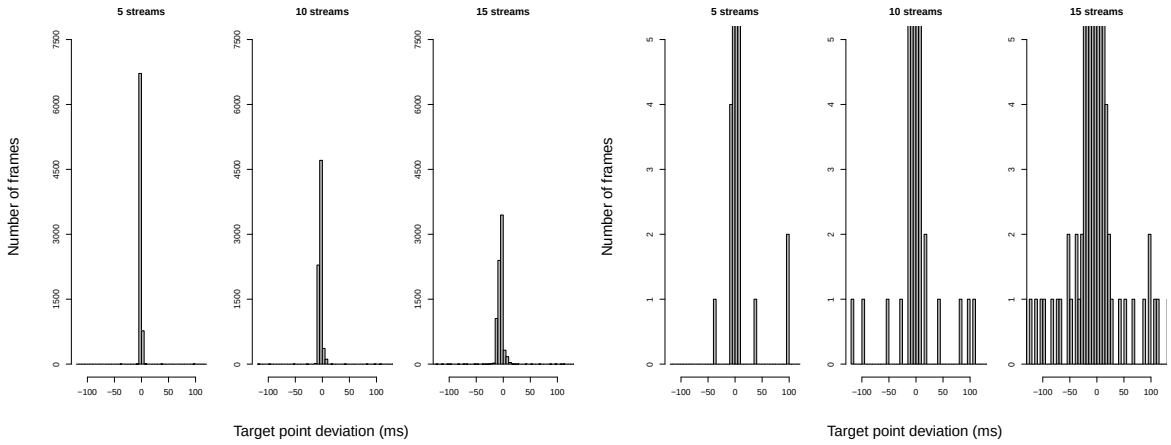
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.3: *Histogram of target point deviation for soccer stream under EDF d=4p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.4: *Histogram of target point deviation for soccer stream under EDF d=16p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.5: *Histogram of target point deviation for soccer stream under Grav. EDF d=16p.*

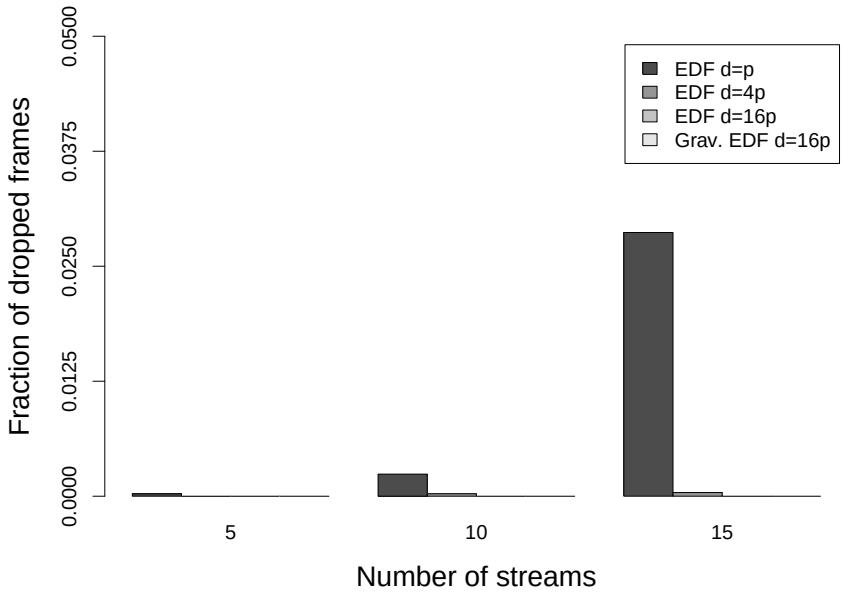
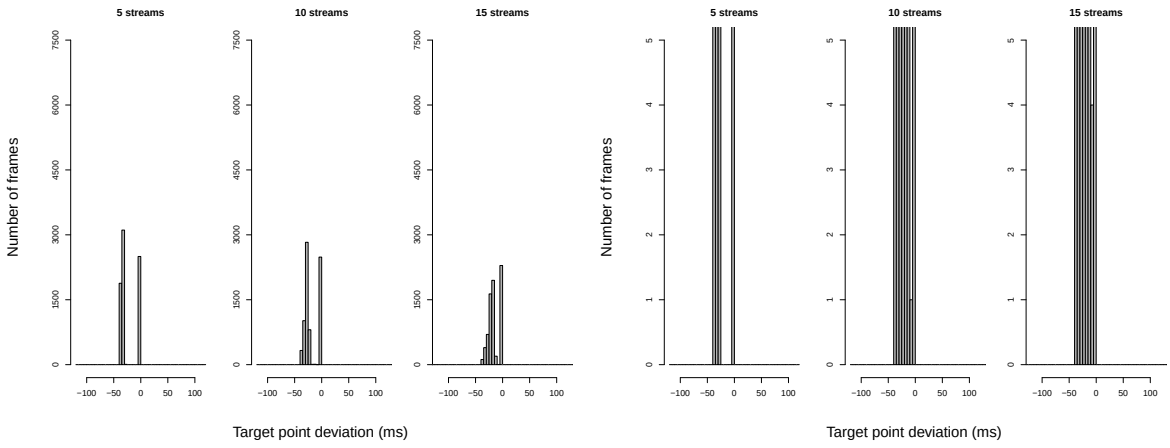
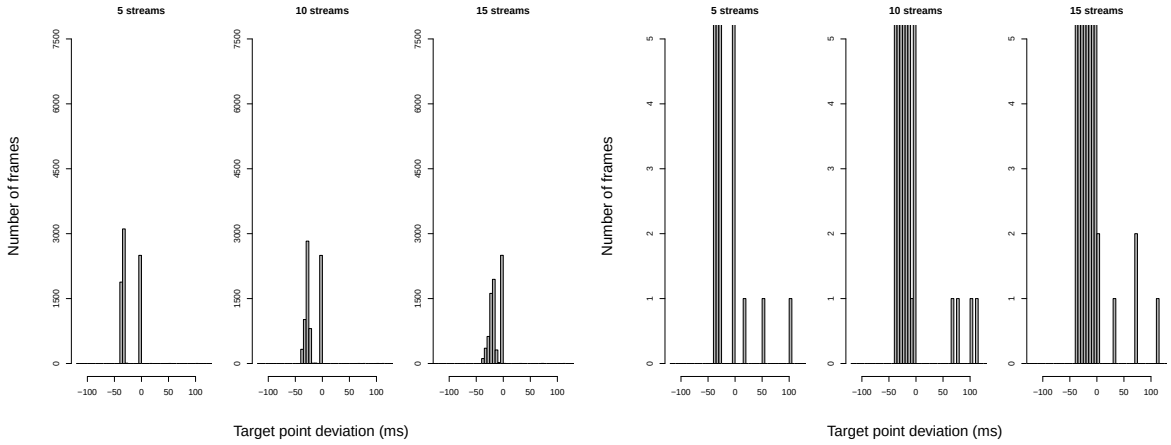


Figure A.6: *Dropped frames for volleyball stream.*



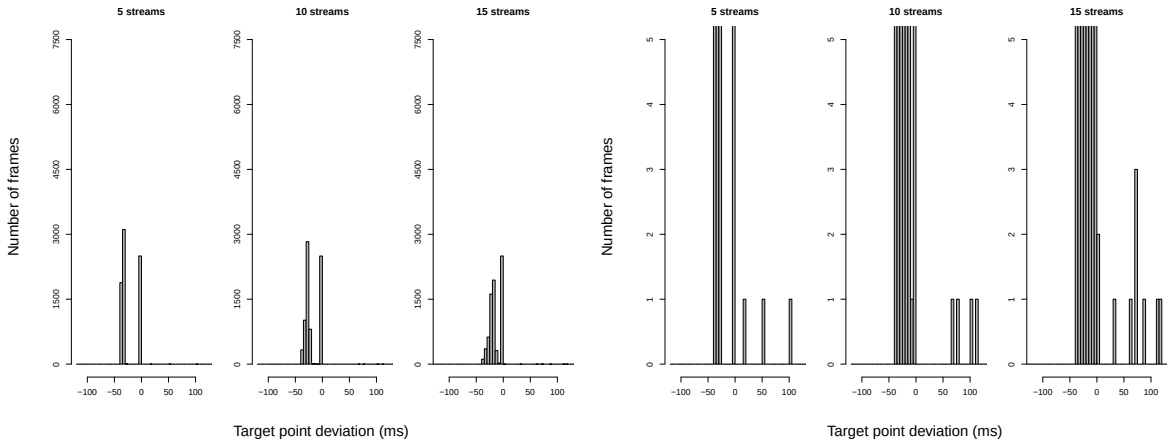
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.7: *Histogram of target point deviation for volleyball stream under EDF d=p.*



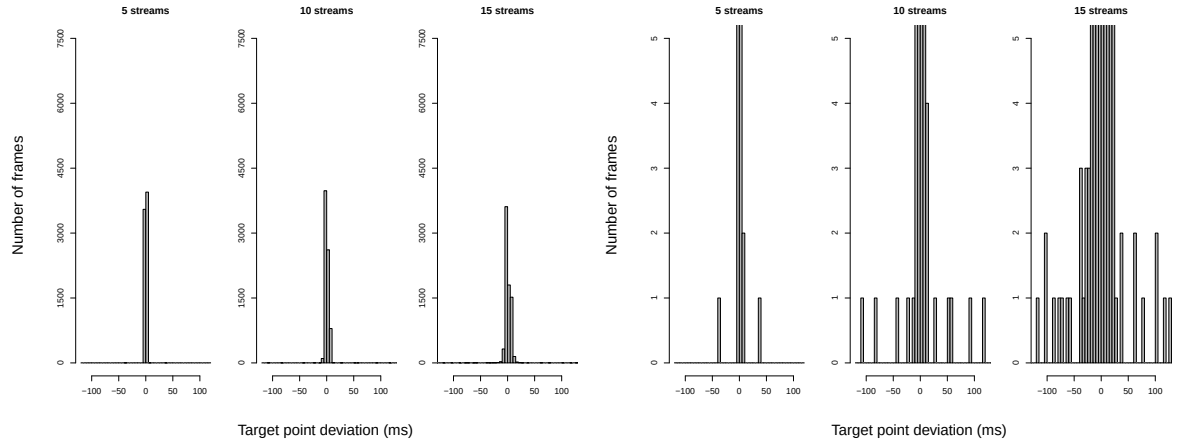
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.8: *Histogram of target point deviation for volleyball stream under EDF d=4p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.9: *Histogram of target point deviation for volleyball stream under EDF d=16p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.10: *Histogram of target point deviation for volleyball stream under Grav. EDF d=16p.*

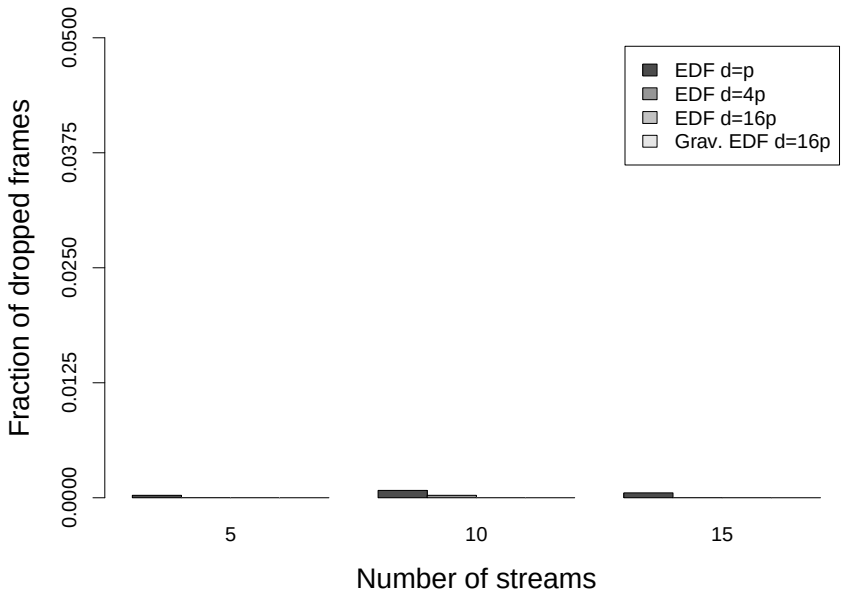
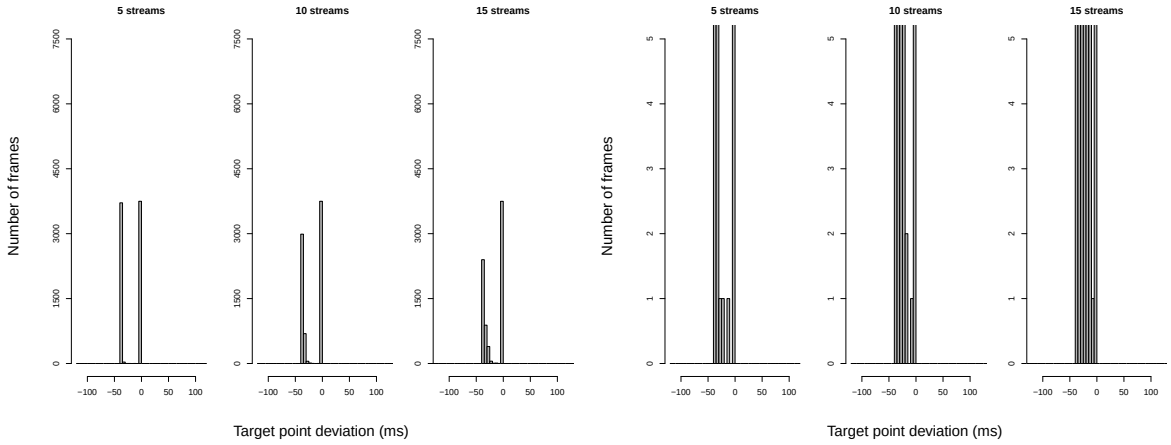
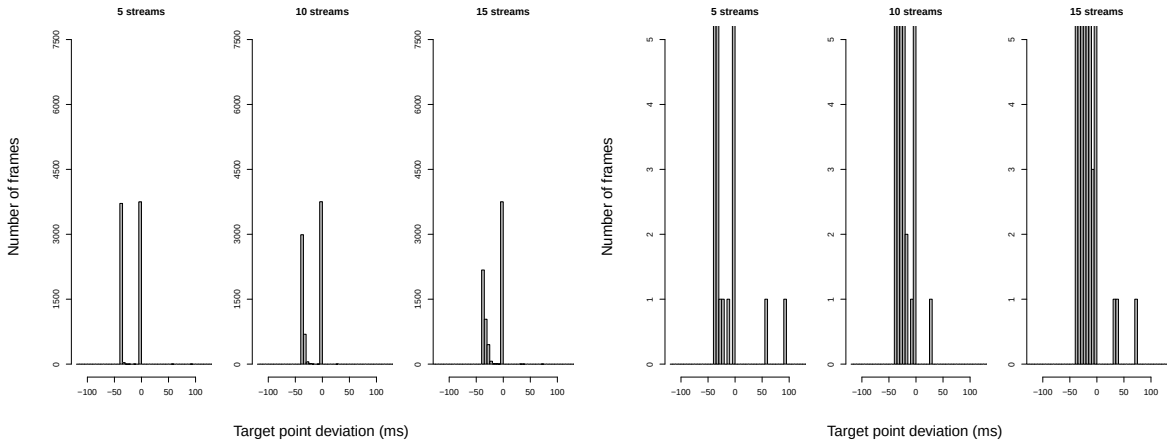


Figure A.11: *Dropped frames for bloomberg stream.*



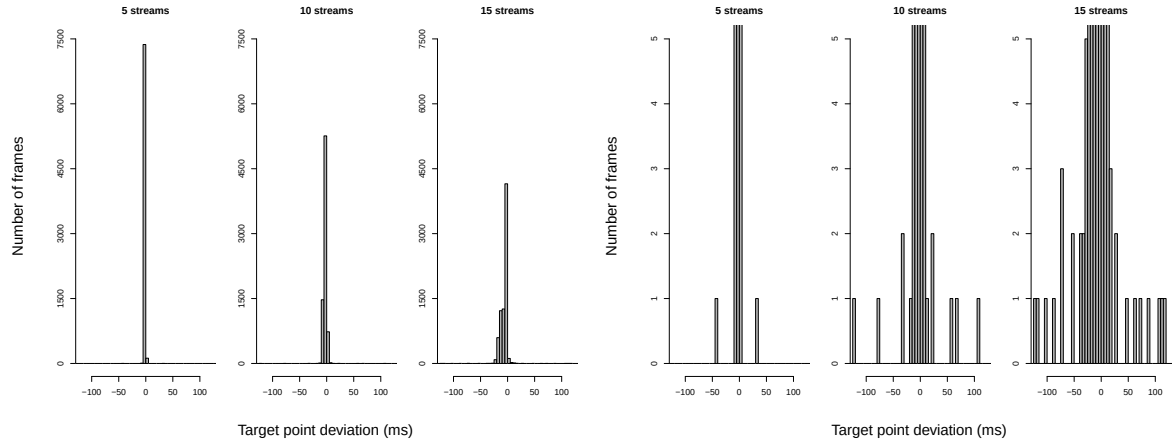
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.12: *Histogram of target point deviation for bloomberg stream under EDF d=p.*



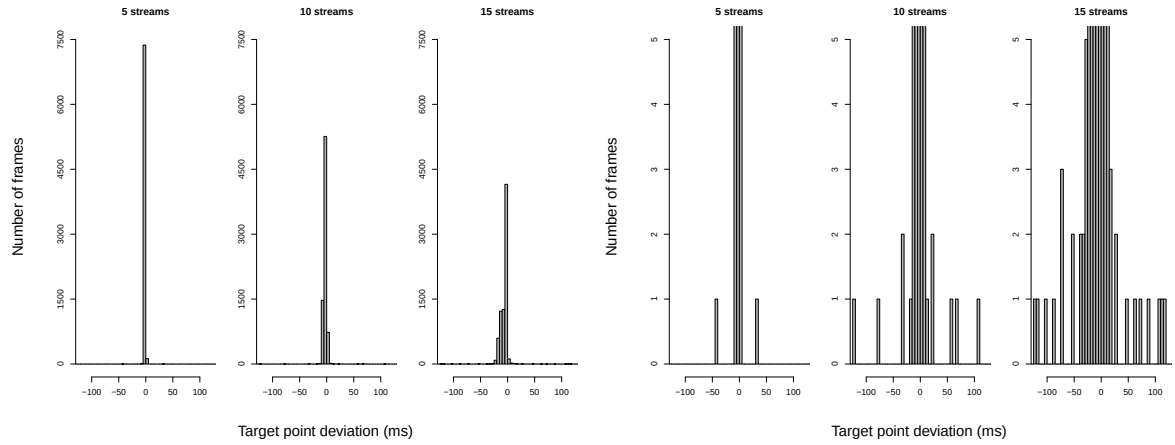
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.13: *Histogram of target point deviation for bloomberg stream under EDF d=4p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.14: *Histogram of target point deviation for bloomberg stream under EDF d=16p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.15: *Histogram of target point deviation for bloomberg stream under Grav. EDF d=16p.*

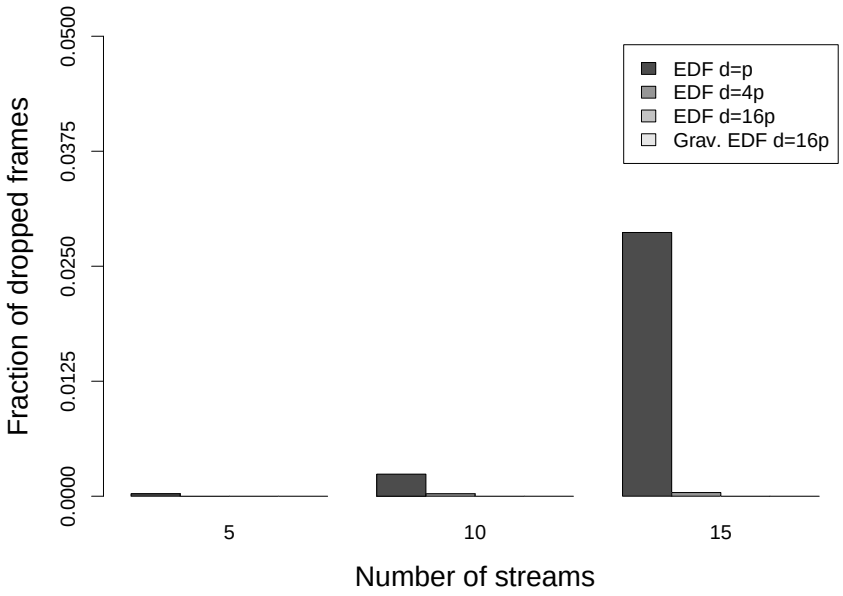
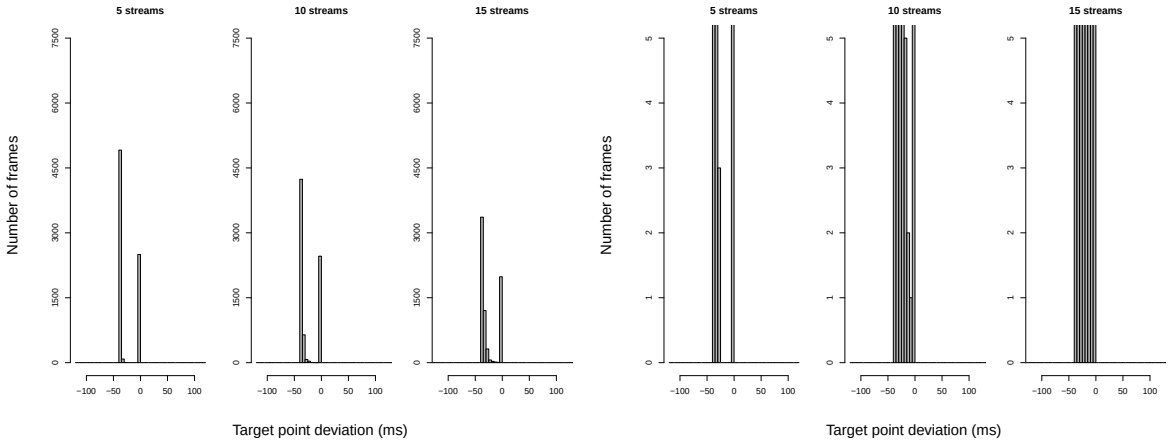
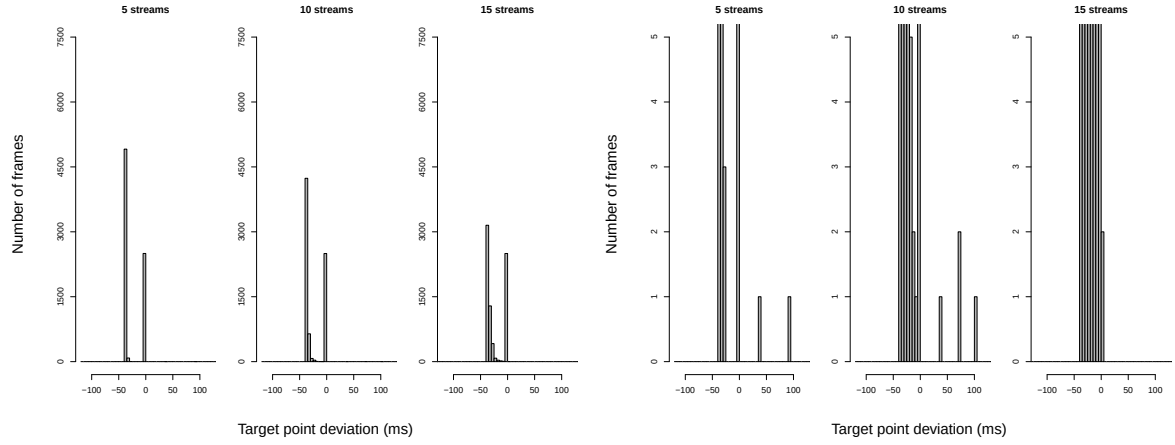


Figure A.16: *Dropped frames for Lucky Luke stream.*



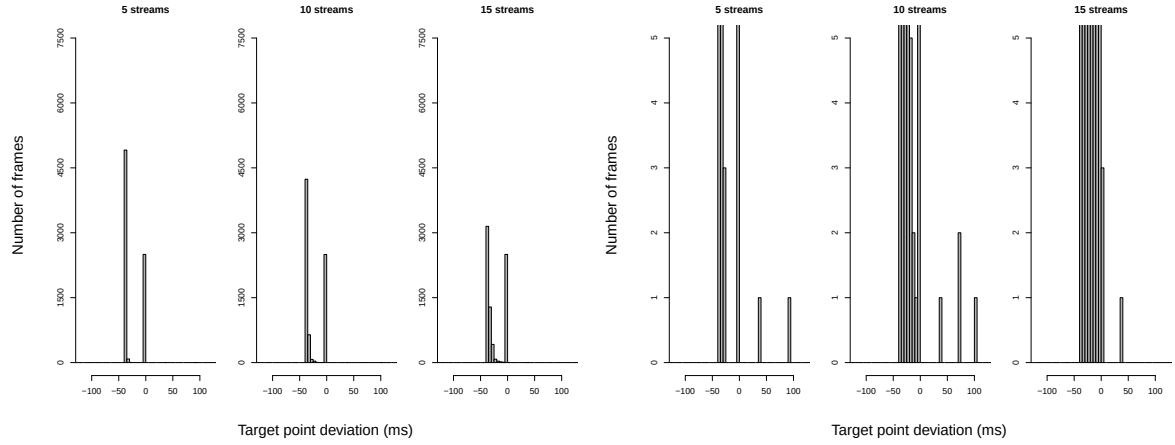
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.17: *Histogram of target point deviation for Lucky Luke stream under EDF d=p.*



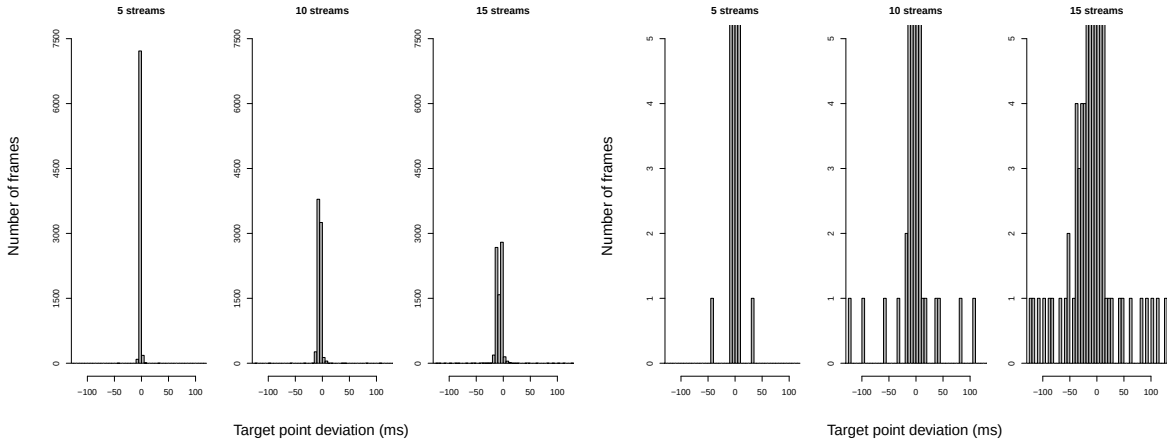
(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.18: *Histogram of target point deviation for Lucky Luke stream under EDF d=4p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.19: *Histogram of target point deviation for Lucky Luke stream under EDF d=16p.*



(a) *Complete y-axis* **(b)** *Reduced y-axis*

Figure A.20: *Histogram of target point deviation for Lucky Luke stream under Grav. EDF d=16p.*

Bibliography

[A. Crespo 05] K. Arzen A. Cervin M. Torgren Z. Hanzalek A. Crespo P. Albertos. *Artist2 Roadmap on Real-Time Techniques in Control System Implementation.* Technical Report EU/IST IST-004527, Cluster: Control for Embedded Systems, 2005.

[Anderberg 07] Niclas Anderberg. *Decode Demo for the DVEVM/DVSDK 1.2.* Rapport technique, Texas Instruments, 04 2007.

[Applegate 07] David L. Applegate, Robert E. Bixby, Vasek Chvatal & William J. Cook. The traveling salesman problem: A computational study (princeton series in applied mathematics). Princeton University Press, Princeton, NJ, USA, 2007.

[Baruah 97] Sanjoy K. Baruah & Jayant R. Haritsa. *Scheduling for Overload in Real-Time Systems.* IEEE Transactions on Computers, vol. 46, pages 1034–1039, 1997.

[Baruah 99] Sanjoy Baruah, Giorgio Buttazzo, Sergey Gorinsky & Giuseppe Lipari. *Scheduling Periodic Task Systems to Minimize Output Jitter.* In RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications, page 62, Washington, DC, USA, 1999. IEEE Computer Society.

[Baruah 06] Sanjoy K. Baruah & Samarjit Chakraborty. *Schedulability analysis of non-preemptive recurring real-time tasks.* In Proceedings of the 20th international conference on Parallel and distributed processing, IPDPS'06, pages 171–171, Washington, DC, USA, 2006. IEEE Computer Society.

[Bastoni 10] A. Bastoni, B.B. Brandenburg & J.H. Anderson. *An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers.* In Real-Time Systems Symposium (RTSS), 2010 IEEE 31st, pages 14 –24, 30 2010-dec. 3 2010.

[Bernat 99] G. Bernat, A. Burns & Albert Llamosí. *Weakly Hard Real-Time Systems*. IEEE Transactions on Computers, vol. 50, pages 308–321, 1999.

[Bilsen 96] Greet Bilsen, Marc Engels, Rudy Lauwereins & Jean Peperstraete. *Cyclo-Static Dataflow*. IEEE Trans. Signal Processing, vol. 44, no. 2, pages 397–408, 1996.

[Bouyssounouse 05] Bruno Bouyssounouse & Joseph Sifakis. Embedded systems design: The artist roadmap for research and development (lecture notes in computer science). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Buck 93] Joseph T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model*. PhD thesis, EECS Department, University of California, Berkeley, 1993.

[Bülbül 07] Kerem Bülbül, Philip Kaminsky & Candace Yano. *Preemption in single machine earliness/tardiness scheduling*. J. of Scheduling, vol. 10, no. 4-5, pages 271–292, 2007.

[Burns 10] A. Burns, B.R. Greene, M.J. McGrath, T.J. O'Shea, B. Kuris, S.M. Ayer, F. Stroiescu & V. Cionca. *SHIMMER x2122 x2013: A Wireless Sensor Platform for Noninvasive Biomedical Research*. Sensors Journal, IEEE, vol. 10, no. 9, pages 1527 –1534, September 2010.

[Buttazzo 02] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo & Luca Abeni. *Elastic Scheduling for Flexible Workload Management*. IEEE Transactions on Compututer, vol. 51, no. 3, pages 289–302, 2002.

[Buttazzo 04] Giorgio C. Buttazzo. Hard real-time computing systems: Predictable scheduling algorithms and applications (real-time systems series). Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[Carnahan 01] J. Carnahan & R. Sinha. *Nature's algorithms [genetic algorithms]*. Potentials, IEEE, vol. 20, no. 2, pages 21–24, Apr/May 2001.

[Chen 96] Ken Chen & Paul Muhlethaler. *A scheduling algorithm for tasks described by time value function*. Real-Time Syst., vol. 10, no. 3, pages 293–312, 1996.

[Claypool 99] Mark Claypool & Jonathan Tanner. *The Effects of Jitter on the Perceptual Quality of Video*. Proc. ACM Multimedia '99(Part 2), pages 115–118, 1999.

[Cormen 01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Introduction to algorithms, second edition. The MIT Press, September 2001.

[Dorigo 05] Marco Dorigo & Christian Blum. *Ant colony optimization theory: a survey.* Theor. Comput. Sci., vol. 344, no. 2-3, pages 243–278, 2005.

[dre] http://www.dream-multimedia-tv.de/.

[Eker 03] Johan Eker & Jorn W. Janneck. *CAL Language Report.* Rapport technique UCB/ERL M03/48, University of California at Berkeley, December 2003.

[Farzinvash 09] L. Farzinvash & M. Kargahi. *A Scheduling Algorithm for Execution-Instant Sensitive Real-Time Systems.* In Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA '09. 15th IEEE International Conference on, pages 511 –518, 2009.

[Fohler 95] G. Fohler. *Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems.* In Proceedings of the 16th Real-Time Systems Symposium, Pisa, Italy, Dec. 1995.

[Fuller 11] Samuel H. Fuller & Lynette I. Millett. *Computing Performance: Game Over or Next Level?* Computer, vol. 44, pages 31–38, 2011.

[Gao 92] G. R. Gao, R. Govindarajan & P. Panangaden. *Well-behaved Dataflow Programs for DSP Computation.* In IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-92, volume 5, pages 561–564. IEEE, March 1992.

[Goldstein 02] Herbert Goldstein, Charles P. Poole & John Safko. Classical mechanics, volume 2010. Addison-Wesley, 2002.

[Ha 91] Soonhoi Ha & Edward A. Lee. *Compile-Time Scheduling and Assignment of Data-Flow Program Graphs with Data-Dependent Iteration.* IEEE Trans. Comput., vol. 40, no. 11, pages 1225–1238, 1991.

[Hill 02] Jason L. Hill & David E. Culler. *Mica: A Wireless Platform for Deeply Embedded Networks.* IEEE Micro, vol. 22, pages 12–24, 2002.

[Isovic 03] Damir Isovic, Gerhard Fohler & Liesbeth F.M. Steffens. *Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions.* In 15th Euromicro Conference on Real-time Systems (ECRTS 03), Porto, Portugal, July 2003.

[Isovic 04] Damir Isovic & Gerhard Fohler. *Quality aware MPEG-2 stream adaptation in resource constrained systems.* In 16th Euromicro Conference on Real-time Systems (ECRTS 04), Catania, Sicily, Italy, July 2004.

[Jeffay 91] Kevin Jeffay & Donald F. Stanat. *On non-preemptive scheduling of periodic and sporadic tasks.* In Proceedings of the Real-Time Systems Symposium, pages 129–139, 1991.

[Jensen 92] E. D. Jensen. *Asynchronous Decentralized Real-Time Computer Systems.* In W. A. Halang & A. D. Stoyenko, editeurs, Real-Time Computing, the NATO Advanced Study Institute. Springer Verlag, October 1992.

[Kahn 74] Gilles Kahn. *The Semantics of Simple Language for Parallel Programming.* In IFIP Congress '74, pages 471–475, 1974.

[Kim 06] Sangwon Kim, Joonwon Lee & Jinsoo Kim. *Runtime feasibility check for non-preemptive real-time periodic tasks.* Inf. Process. Lett., vol. 97, pages 83–87, February 2006.

[Ko 10] JeongGil Ko, Chenyang Lu, M.B. Srivastava, J.A. Stankovic, A. Terzis & M. Welsh. *Wireless Sensor Networks for Healthcare.* Proceedings of the IEEE, vol. 98, no. 11, pages 1947 –1960, 2010.

[Kopetz 05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger & Klaus Steinhammer. *The Time-Triggered Ethernet (TTE) Design.* Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on, vol. 0, pages 22–33, 2005.

[Laarhoven 87] P. J. M. Laarhoven & E. H. L. Aarts, editeurs. Simulated annealing: theory and applications. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[Lee 87] Edward A. Lee & David G. Messerschmitt. *Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing.* IEEE Trans. Comput., vol. 36, no. 1, pages 24–35, 1987.

[Leontyev 10] Hennadiy Leontyev & James H. Anderson. *Generalized tardiness bounds for global multiprocessor scheduling.* Real-Time Syst., vol. 44, pages 26–71, March 2010.

[Li 95] Yau-Tsun Steven Li & Sharad Malik. *Performance analysis of embedded software using implicit path enumeration.* In Proceedings of the 32nd annual ACM/IEEE Design Automation Conference, DAC '95, pages 456–461, New York, NY, USA, 1995. ACM.

[Li 04] Peng Li, Binoy Ravindran & E. Douglas Jensen. *Adaptive Time-Critical Resource Management Using Time/Utility Functions: Past, Present, and Future.* In COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC'04), pages 12–13, Washington, DC, USA, 2004. IEEE Computer Society.

[Li 06] Peng Li. *A Utility Accrual Scheduling Algorithm for Real-Time Activities with Mutual Exclusion Resource Constraints.* IEEE Trans. Comput., vol. 55, no. 4, pages 454–469, 2006. Member-Haisang Wu and Senior Member-Binoy Ravindran and Member-E. Douglas Jensen.

[lib] http://libmpeg2.sourceforge.net/.

[Lipari 01] Guiseppe Lipari & Sanjoy Baruah. *A hierarchical extension to the constant bandwidth server framework.* In 7th IEEE Real-Time Technology and Applications Symposium, 2001.

[Liu 73] C. L. Liu & James W. Layland. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* Journal of the ACM, vol. 20, no. 1, 1973.

[Liu 10] Shuo Liu, Gang Quan & Shangping Ren. *On-Line Scheduling of Real-Time Services for Cloud Computing.* Services, IEEE Congress on, vol. 0, pages 459–464, 2010.

[Locke 86] Carey Douglass Locke. *Best-effort decision-making for real-time scheduling.* PhD thesis, Pittsburgh, PA, USA, 1986.

[Martello 90] Silvano Martello & Paolo Toth. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[Marti 01] Pau Marti, Gerhard Fohler, Krithi Ramamritham & Josep M. Fuertes. *Jitter Compensation in Real-Time Control systems.* In Proceedings of the 22nd IEEE Real-Time Systems Symposium, London, UK, Dec. 2001.

[Marti 02] Pau Marti, Gerhard Fohler, Krithi Ramamritham & Josep M. Fuertes. *Control performance of flexible timing constraints for Quality-of-Control Scheduling.* In Proceedings of the 23rd IEEE Real-Time Systems Symposium, Texas, TX, USA, Dec. 2002.

[Mok 01] Aloysius K. Mok, Xiang Feng & Deji Chen. *Resource Partition for Real-Time Systems.* In 7th IEEE Real-Time Technology and Applications Symposium, 2001.

[Orozco 97] J. Orozco, R. Cayssials, J. Santos & E. Ferro. *Precedence constraints in hard real-time distributed systems.* Engineering of Complex Computer Systems, IEEE International Conference on, vol. 0, page 33, 1997.

[Parks 95] Thomas Martyn Parks. *Bounded Scheduling of Process Networks.* PhD thesis, EECS Department, University of California, Berkeley, 1995.

[Paul Bousquet 97] Kim Brian Michael Coltman Robert Dorer Julie Ebbighausen William Fashouer Gregg Fleming Jeffrey Gordon Glenn Goulet Arnold Kupferman Adrian Hellman Jose Mantilla Stephanie Markos Ronald Mauri Roger Wayson David Valenstein Paul Valihura Paul Bousquet Aviva Brecher. *Evaluating Compliance with FCC Guidelines for Human Exposure to Radiofrequency Electromagnetic Fields*. FCC Office Of Engineering And Technology (OET). Edition 97.01. FCC Bulletin 65 Edition 97.01, 1997.

[Polastre 05] J. Polastre, R. Szewczyk & D. Culler. *Telos: enabling ultra-low power wireless research*. In Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on, pages 364 – 369, 2005.

[Prabh 11] K. Shashi Prabh & Jan-Hinrich Hauer. *Opportunistic Packet Scheduling in Body Area Networks*. In EWSN, pages 114–129, 2011.

[Prasad 03] D. Prasad, A. Burns & M. Atkins. *The Valid Use of Utility in Adaptive Real-Time Systems*. Real-Time Syst., vol. 25, no. 2-3, pages 277–296, 2003.

[Puschner 00] Peter Puschner & Alan Burns. *A Review of Worst-Case Execution-Time Analysis*. Journal of Real-Time Systems, vol. 18, no. 2/3, pages 115–128, May 2000.

[Ramamritham 95] Krithi Ramamritham. *Allocation and Scheduling of Precedence-Related Periodic Tasks*. IEEE Transactions on Parallel and Distributed Systems, vol. 6, pages 412–420, 1995.

[Regehr 01] John Regehr. *HLS: A framework for composing soft real-time schedulers*. In In Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001, pages 3–14. IEEE, 2001.

[Ritz 93] Sebastian Ritz, Matthias Pankert, Vojin Živojnović & Heinrich Meyr. *Optimum Vectorization of Scalable Synchronous Dataflow Graphs*. In Intl. Conf. on Application-Specific Array Processors, pages 285–296. Prentice Hall, IEEE Computer Society, 1993.

[Rizvanovic 07] Larisa Rizvanovic & Gerhard Fohler. *The MATRIX - A Framework for Real-time Resource Management for Video Streaming in Networks of Heterogenous Devices*. In The International Conference on Consumer Electronics 2007, Las Vegas, USA, January 2007.

[Smith 06] K.I. Smith. *A study of simulated annealing techniques for multi-objective optimisation*. PhD thesis, University of Exeter, Exeter, UK, 2006.

[Stoyenko 91] A. D. Stoyenko & L. Georgiadis. *An optimal lateness and tardiness scheduling in real-time systems.* Computing, vol. 47, pages 215–234, August 1991.

[Sun 06] Wenyu Sun & Ya-xiang Yuan. Optimization theory and methods, volume 1. springer, 2006.

[Sung 08] Star Sung & Jacques Baudia. *An LCD driver with on-chip frame buffer and 3 times image compression.* In Proc. SPIE 6807, 2008.

[Thelen 06] B.D. Thelen, M.C.W Geilen, T. Basten, J.P.M Voeten, S.V. Gheorghita & S. Stuijk. *A Scenario-Aware Data Flow Model for Combined Long-Run Average and Worst-Case Performance Analysis.* In 4th ACM-IEEE Intl. Conf. on Formal Methods and Models for Codesign, MEMOCODE 2006, pages 185–194, July 2006.

[Tra 00] *Information technology - Generic coding of moving pictures and associated audio information systems.* Rapport technique, ISO/IEC, 01 2000.

[Wang 04] Jinggang Wang & Binoy Ravindran. *Time-Utility Function-Driven Switched Ethernet: Packet Scheduling Algorithm, Implementation, and Feasibility Analysis.* IEEE Trans. Parallel Distrib. Syst., vol. 15, no. 2, pages 119–133, 2004.

[Winterbone 96] Desmond Winterbone. Advanced thermodynamics for engineers. Elsevier, Orlando, 1996.

[Wu 05] Haisang Wu, Umut Balli, Binoy Ravindran & E. Douglas Jensen. *Utility Accrual Real-Time Scheduling under Variable Cost Functions.* In RTCSA '05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), pages 213–219, Washington, DC, USA, 2005. IEEE Computer Society.

[Wu 06] Haisang Wu, Binoy Ravindran, E. Douglas Jensen & Peng Li. *Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems.* Trans. on Embedded Computing Sys., vol. 5, no. 3, pages 513–542, 2006.

Glossary

Bit Rate
Bit rate is the rate at which bits of a video stream arrive in the decoder.. 16, 18–20

Body Area Network
Body Area Network (BAN) are networks of sensors around as well as inside the human body, with application e.g. in health-care and sports.. 14, 25–27, 33, 111, 126, 127, 130, 141, 169, 174

Central Processing Unit
Central Processing Unit (CPU) is the computing system component responsible for the logical operations and data manipulation.. 4, 6

Digital Versatile Disc
Digital Versatile Disc (DVD) is an optical disc storage media format, and was invented and developed by Philips, Sony, Toshiba, and Time Warner in 1995. Its main uses are video and data storage.. 16, 103, 105

Direct Memory Access
Direct Memory Access (DMA) is a feature of modern computers and microprocessors that allows certain hardware subsystems within the computer to access system memory for reading and/or writing independently of the central processing unit.. 2, 112, 113

Dynamic Dataflow
Dynamic dataflow offers a determinate computation model to dataflows systems, which means that the outputs that are computed by a program only depends on the inputs it has consumed.. 28, 29

Earliest Deadline First
Earliest Deadline First is a dynamic scheduling rule that assigns priorities to jobs based on their deadlines.. 6, 10, 12, 13, 58, 77, 83–85, 88–94, 96, 97, 113–117, 172

Electronic Control Unit
Electronic control units (ECU) is a specialized nomenclature for embedded control systems in automotive systems.. 30

Generic Utility Scheduler
A TUF based scheduling algorithm.. 77, 83, 84, 88, 96, 102, 106, 107, 172

Group Of Pictures
A Group Of Pictures consists of all frames in between two I frames (including the first I frame).. 16, 103–105, 107

High Definition
High Definition (HD) video refers to any video system of higher resolution than standard-definition (SD) video, and most commonly involves display resolutions of $1,280 \times 720$ pixels (720p) or $1,920 \times 1,080$ pixels (1080i/1080p).. 16, 103

Medium Access Control
Medium Access Control refers to the network layer responsible of establishing the communication between two neighbor nodes. Typical duties of this layer include carrier sensing and collision detection/avoidance.. 127

MPEG-2
MPEG-2 is a video standard defined by the Moving Picture Experts Group.. 16–18, 21, 32, 103, 112, 113, 135, 168

MPEG-4
MPEG-4 is a video standard defined by the Moving Picture Experts Group.. 16

MPEG-4 Simple Profile
MPEG-4 Simple Profile is a video standard defined by the Moving Picture Experts Group.. 103

Non-deterministic Polynomial-time hard
Non-deterministic Polynomial-time hard in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP".. i, 12, 13, 57, 58, 87, 100, 171

Non-Linear Optimization Problem
Non-Linear Optimization Problem is an optimization problem with a non-linear goal function.. 45

opportune transmission window
Opportune transmission window is the interval of time in which packets may be transmitted in the opportunistic packet scheduling algorithm proposed in [Prabh 11], and briefly described in section VII.3.. 128, 129

perceived quality of video
Perceived quality of video stands for a formal or informal measure of the video quality from the user's perception.. 3, 8, 14, 17, 21, 22, 103, 104, 111–114, 130, 135, 140, 173

Quality of Service
Quality of Service refers to the quantitative evaluation of one or several qualitative metrics of a system.. 4, 22, 23, 25, 32, 33, 168, 169

Radio Frequency
Radio Frequency refers to the transmission of radio signals through the air.. 26, 127

Rate Monotonic
Rate Monotonic is a static scheduling rule that assigns priorities to jobs based on their deadlines.. 12, 58

real-time
Real-time describes a system or application whose correct behavior depends on timing constraints.. i, xiii, 2, 8, 10, 11, 13, 16, 32, 33, 35, 36, 42, 46, 54, 57, 96, 100, 108, 126, 136, 139, 167–169, 173

Real-Time System
Real-Time Systems are hardware or software systems in which the correctness of their results is subject to predefined timeliness constraints.. 2–5, 7, 33, 126, 139, 167–169

Hard Real-Time Systems are RTS in which the violation of their timeliness constraints could result on fatal consequences. 4, 5, 7

Soft Real-Time System are RTS in which the violation of their timeliness constraints produces the degradation of the system performance, although they do not necessarily result in fatal consequences. 4

Reveived Signal Strength Indicator
Reveived Signal Strength Indicator (RSSI) is a metric of link quality in wireless network.. 26, 27, 126–128, 130, 174

Synchronous Dataflow
Synchronous dataflow is the special case of dynamic dataflow, in which actors have fixed token rates.. 28, 29

Time Utility Function
Time Utility Function describe the utility accrued by a task as a function of the moment of execution.. 9, 10, 36, 38, 83, 88, 103, 133–135, 172

Wireless Sensor Network

A Wireless Sensor Network is constituted by a number of sensor nodes communicating with an ad-hoc infrastructure running at least one common application in a collaborative manner.. 7, 26, 32, 126, 168

worst case execution time

The worst case execution time (WCET) of a computational task is the maximum length of time the task could take to execute on a specific hardware platform.. 3, 13, 132, 134, 137, 174

Summary

Real-time (RT) applications have timing constraints additionally to their logical output for functionally correct system behavior. Control applications, which act on a physical plant to make it behave according to a prescribed reference, are the source of the RT constraints. These systems include safety critical, mission critical, and business critical control applications, which have stringent timing constraints — the violation of a single deadline can jeopardize the entire system behavior, and even cause catastrophic consequences [Bouyssounouse 05].

Real-Time Systems (RTS) commonly consist of multiple tasks that concurrently compete for system resources. Therefore, RT applications demand special scheduling algorithms which account for timing constraints, the so-called *RT scheduling algorithms*. These algorithms define as a set of rules to schedule the execution of tasks at system run-time in order to preserve timing constraint. For example, some tasks have earliest start time and deadline constraints, which define the *execution window* of a task. This window is the interval of time where the logical output results in correct system behavior, and the scheduler must guarantee that all tasks entirely execute within their respective execution windows.

Some RT applications have tight optimum execution windows for maximum system utility, but accept some flexibility to enlarge those windows for the sake of feasibility. This flexibility comes at the expense of a utility decay, though. For example, tasks of *target sensitive applications* should preferably execute at a specific target point within its execution window, called *target point*, but can execute around this point, albeit at lower utility — the intensity of the utility decay accounts for the importance of the application to the system. In this case, the optimum execution window is extremely tight. Ideally, all executions would be scheduled directly at the respective target points, but it might not be feasible due to overlapping executions. Under this condition, the execution of tasks must be scheduled so that no timing constraints are violated and the accrued system utility is maximized. More important applications are less tolerant to deviations from the target point.

Besides the necessary expressiveness, task models must also provide simple abstractions for easy understanding by applications developers. Moreover, on-line adaptivity requires scheduling algorithms with low overhead, and the compromise among tasks with overlapping executions involves reordering the execution of jobs, shifting them and

possibly aborting some executions. The difficulties include expressing whether two not very important tasks are more important than a very important task, and reordering the execution sequence of jobs is a combinatorial problem. Therefore, scheduling is non-trivial, and the need for a simple, yet effective, solution imposes an extra challenge.

In this thesis, we presented a novel real-time task model which provides for easy abstractions to express the timing constraints of target sensitive real-time applications: the gravitational task model. This model uses an analogy with pendulum systems to ease the understanding of its temporal abstractions by application developers, which not always have deep knowledge of real-time scheduling theory. In other words, this model fills in the gap between application requirements and theoretical abstractions used in task models. We also presented a few scheduling algorithms designed for the gravitational task model which fulfill the requirements for on-line adaptivity. These algorithms also exploit the analogy with pendulum system, which provides for clarity. Finally, we presented 3 applications enhanced with the gravitational task model, and highlighted the improvement in the results.

The following sections summarize the content of each chapter of this thesis.

Chapter I

This chapter contains the introduction of this thesis. In section I.1, we briefly described the basic concepts of RT scheduling, like timing constraints, task models, and scheduling algorithms and paradigms. In section I.2, we focused on properties and requirements of adaptive RTS. Then, in section I.3, we informally described target sensitive RT applications and related scheduling issues, followed by the related work in this area in section I.4. In section I.5, we summarized our main contributions to the state-of-the-art of real-time scheduling theory. Finally, in section I.6, we presented the outline of the rest of the thesis.

Chapter II

In this chapter, entitled *Examples of target sensitive real-time aplications*, we described 5 examples of target sensitive real-time applications in order to evidence the need for task models, as well as scheduling algorithms, which are able to take into account the temporal requirements of such applications.

Section II.1 presented a multimedia application. We showed that multimedia systems are embedded in our routine nowadays, with deployment in diverse areas such as healthcare systems, games, publicity, automobile industry, etc. We also used the MPEG-2 video standard to illustrate the main activities involved in video playout, and the resource and temporal constraints involved in these activities.

In section II.2, we described the adaptive management of RT application's resources. Adaptive applications running on system which do not provide for the worst case resource requirements demand such management in order to decrease the Quality of Service (QoS) degradation shall overload occur.

In section II.3, we presented the deployment of Wireless Sensor Networks (WSN) in

assisted health-care systems, the so called Body Area Networks (BAN). BANs have very restricted resource availability, and high QoS requirements, being the network the most critical resource. This section presented the temporal requirements and resource constraints of packet scheduling in such networks, which has target sensitive RT constraints.

In section II.4, we presented the relevance of dataflow-based programming paradigms to raise the level of abstraction when writing codes tailored to multicore systems. We then described the issues involved the dataflow graph scheduling, which contains target sensitive properties.

Finally, in section II.5, we described the resource and temporal constraints of control systems, which are the original source of the temporal requirements studied in RTSs. We also showed that these temporal requirements are not as strict as assumed by many task models and scheduling algorithms, accepting some flexibility for the sake of feasibility and increased resource utilization. Therefore, also control can benefit from task models and scheduling algorithms designed for target sensitive RT applications.

Chapter III

In this chapter, entitled *Gravitational Task Model: a bob pendulum based approach to express trade-offs*, we proposed a gravitational task model for target sensitive RT applications. In section III.2, we introduced the terminology and assumptions of this model, and in section III.3, we showed that the scheduling of target sensitive RT applications involves the computation of trade-offs. We also showed that classic task models, which are based on execution windows alone, either compromise feasibility for the sake of maximized utility accrual or fail to use resources efficiently, i.e. for maximized utility accrual.

In section III.4, we introduced the simple gravity pendulum (or bob pendulum) system as a visualization model for trade-offs among target sensitive RT applications. As depicted in figure III.3, the pendulum system consists of a bob at the end of a massless string, which can swing back and forth along the *swinging range* under the influence of gravity over its central (lowest) point in a circular trajectory. Placed at the lowest point, the bob will come to rest there (*rest position*). If the bob pendulum contains more than one bob, they cannot be all at the same time in the lowest part, and hence, will push each other aside to find a new rest position (*equilibrium state*). The equilibrium state implies in minimized overall potential utility.

In section III.5, we described the analogy between the schedule of target sensitive jobs and the pendulum system. Drawing the analogy, the execution time of a job is equivalent to the size of the bob. A job is allowed to execute at its target point in the absence of other jobs in the system with the same target point. The target point is equivalent, thus, to the central (lowest) point of a pendulum trajectory and the swinging range is the execution window of the job. The importance of a job, which represents its resistance to be shifted away from its target point when interacting with other jobs, can be seen as the weight of the bob. The heavier a bob is, the closer to the bottom it will come to rest. Finally, the job utility as a function of its deviation from the target

point is similar to the potential energy of a bob as a function of its deviation from the central point. As the equilibrium is the state that minimizes the potential energy of the pendulum, the best compromise of the jobs' interests maximizes the accrued utility of the system.

Finally, in section III.6, we calculated an approximation for the equilibrium of the particle pendulum inspired by the equilibrium condition for bob pendulums. Combined with the conversion provided in table III.2, the equilibrium of particle pendulums calculates an approximation of the best compromise among jobs with conflicting target points. Then, we showed, based on this new analogy, that the gravitational task model implicitly assumes that jobs have elliptical utility functions as in equation III.10. Finally, we generalized the equilibrium equation in section III.6.2 to allow jobs to have any arbitrary continuously differentiable concave utility function. This generalization keeps the original intuition from the analogy with pendulum systems.

Chapter IV

In this chapter, entitled *Scheduling target sensitive real-time tasks*, we presented a few scheduling algorithms for the gravitational task model. Initially, we presented in section IV.2 how to schedule jobs using the equilibrium to compute trade-offs. Section IV.2.1 describes a method inspired by the bob pendulum to find job chains in a schedule. Appending bobs one by one in a pendulum reveals which groups of jobs push one another, and the equilibrium state changes upon collisions — appending one bob may cause more than one collision. Similarly, the algorithm to find job chains appends jobs one by one to the schedule, and recomputes the equilibrium for job chains that merge; execution sequence of jobs must be known. Appending one job might cause more than one merge (*ripple effect*), but there can be no more than N merges; chains do not split upon job append, only merge. Therefore, the final complexity is $O(N^2)$. This algorithm allows the equilibrium to schedule jobs that do not compete altogether with one another for their target points.

In section IV.2.2, we presented a method with constant complexity to recompute the equilibrium of 2 adjacent chains that merge. As those equilibrium recomputations have constant complexity, and there can be no more than $N - 1$ merges, the complexity to schedule jobs is linear. This method is valid for trade-off calculations using the particle pendulum equilibrium, which we described in chapter III. The basic idea is to store intermediate values for each job chain in order to save computational steps when computing the equilibrium of the new job chain which appears upon merge.

In section IV.2.3, we described the on-line job admission procedure. Upon on-line job admission, the incoming job may be inserted anywhere in the schedule. In this case, there can be at most one chain split and N chain merges. We compute the intermediate values of the separated chains in linear complexity, as handle possible new merges as before. Therefore, on-line admission also has linear complexity. Sections IV.2.4 and IV.2.5 illustrated the scheduling algorithm with examples.

In section IV.3, we presented the importance of the execution sequence of jobs in a scheduling algorithm, and a few ordering heuristics. The execution sequence of jobs

dominates schedulability and utility maximization problems. Moreover, it is an Non-deterministic Polynomial-time hard (NP-hard) problem — check all possible permutations for the execution of jobs. Therefore, an optimum solution has high overhead, and is unfeasible for on-line scheduling algorithms. Heuristic solutions compromise between overhead, acceptance ratio, and utility accrual; affording higher overhead tends to provide for better scheduling decisions. We also presented a few ordering heuristics which are based on the rationale of physics for fluids. The basic idea is that in a container with liquid of different densities, the higher the density the closer to the bottom it will come to rest. Mapping jobs to liquids and target points to the bottom of a container, jobs with higher density should execute closer to their target points.

The physical definition of density is the ratio of the mass to the volume of a body: density measures how tightly the matter is packed together. While in nature the mass of a body and gravity define the force that drags the body down, in a task set the importance of a job drags it to the target point. Similarly, a body occupies space (volume) and a job occupies time (WCET). Thus, we defined the density of a job its importance normalized to its execution time. This density is called *utility density*.

In section IV.3.1, we presented heuristic DST-1, which has complexity $O(N \times log(N))$. This heuristic tries to favor jobs that pack more utility per time of execution (density) by scheduling their execution first, and as close as possible to their target points. If positioning the anchor point of a job directly at its target point results in an execution overlap with a previously scheduled job chain, the heuristic places the job either on the left or on the right side of this job chain (wherever the job is closer to its target point in order to minimize the disturbance in the actual equilibrium state). Therefore, more jobs with high utility accrual contribution execute closer to their target points, and hence, less jobs have to execute far away from their target points. As a result, the negative impact of jobs competing for their target points decreases. However, this heuristic does not consider the execution window of jobs for taking ordering decision, hence compromising feasibility. This section also contains a scheduling example.

In section IV.3.2, we presented heuristic DST-2, which slightly modifies DST-1 in order to account for feasibility when the insertion of a job in the schedule causes execution overlap. In DST-2, if positioning the anchor point of a job directly at its target point results in an execution overlap with a previously scheduled job chain, the heuristic places the job either on the left or on the right side of this job chain using the rule in expression IV.27. This expression finds a compromise between deviations from the target points and how close the execution of jobs lie to the edge of their execution windows. This heuristic also has complexity $O(N \times log(N))$.

In section IV.3.3, we presented heuristic DST-3, which modifies heuristic DST-1 and DST-2 to avoid reordering of the execution sequence upon job arrival, and hence, has linear complexity. In this heuristic, the jobs considered in the schedule at run-time are all jobs ready for execution plus all jobs that will arrive within an interval of time called *equilibrium window*. This equilibrium window is necessary to account for the disturbance of jobs that have not yet arrived in the system on the schedule at the current time. We proved that it suffices that this equilibrium window is as large as or greater than the length of the longest job chain to account for all possible disturbances.

Simulation results in section IV.4 showed that heuristics based on the utility density of jobs yield good results. Moreover, DST-3 has results almost as good as DST-2, and lower computational complexity. Results also showed the benefits of the gravitational task model over scheduling algorithms for other task models — we considered the Time Utility Function (TUF) scheduler Generic Utility Scheduler (GUS) [Li 06], and Earliest Deadline First (EDF) as a deadline based task model. The gravitational task model is able to accrue more utility without over-compromising the feasibility of the task sets.

Chapter V

In this chapter, entitled *Reducing the complexity of periodic tasks' scheduling*, we proposed a gravitational task model based on-line scheduling algorithm for periodic tasks which is inspired on a mix of EDF and heuristic DST-3. We called this algorithm EDF-swap. EDF-swap accounts for both high acceptance ratio and increased utility accrual. Moreover, it also significantly reduces the worst case complexity to schedule n periodic tasks from $n!$ to n^2. We achieved high acceptance ratio by ordering the execution sequence of jobs as in non-preemptive work-conserving EDF, and increased utility accrual by computing the equilibrium of jobs and swapping the execution of jobs based on their utility density. Finally, we achieved the complexity reduction by limiting the number of jobs in equilibrium — other gravitational task model based on-line scheduling algorithms must consider all jobs within the hyper-period.

In section V.2, we showed that the equilibrium of jobs may not be applied directly on EDF without further consideration. Doing so may lead to a execution sequence of jobs that defers from the order that non-preemptive work-conserving EDF generates, hence invalidating the timeliness properties of EDF. We proposed, then, a method to combine the equilibrium of jobs with EDF, and illustrated this method with an example. This method also limits the number of jobs in equilibrium, which is the key contribution for the reduced computational complexity.

In section V.3, we proposed a heuristic to reorder the execution sequence of jobs within the equilibrium window for increased utility accrual. This heuristic is based on the utility density of jobs, and does not compromise the feasibility of the original schedule. We illustrated the heuristic with a scheduling example.

Simulation results in section V.4 showed that EDF-swap accrues more utility and has higher acceptance ratio than any other scheduling algorithm for the gravitational task model. Moreover, varying the target points within the execution window of jobs does not have a negative impact on the performance of EDF-swap. This superiority is also followed by a significant reduction of the computational complexity.

Chapter VI

In this chapter, entitled *Scheduling tasks for increased system utility under scarce resource availability*, we showed that executing as many tasks as possible does not imply in maximized system utility. In fact, rejecting some tasks frees resources that other tasks can use to accrue more utility. For instance, tasks that do not meet their starttime-

deadline requirements do not accrue utility to the system, yet consume resources that other tasks can use to increase the utility accrual. The overload situation can be overcome by either delaying or dropping jobs, and the choice must account for final increased utility accrual. Dropping jobs is a combinatorial problem.

In section VI.2, we proposed a heuristic overload handling mechanism for target sensitive RT applications. This mechanism increases the system utility by both aborting jobs and computing the compromise among the deviation of the ones scheduled for execution. The abortion heuristic is based on the utility density of jobs, which is the ratio between utility accrual and execution time. This mechanism has quadratic complexity, and is application independent. Hence, any application can benefit from it by modeling its requirements into the parameters of the gravitational task model.

Section VI.3 described a multimedia case study where we use this mechanism for reduced frame display jitter and improved resource usage under scarce resource availability. The frame decoding timing constraint assignment is based on the frame skipping algorithm [Isovic 04] for quality aware video adaptation. Simulation results showed the efficacy of the overload handling mechanism.

Chapter VII

In this chapter, entitled *Applications of the gravitational task model*, we presented 3 applications enhanced with the gravitational task model in order to demonstrate the validity of our work. Section VII.1 presented a multimedia application where we use the gravitational task model to reduce the degradation in the perceived quality of video (PQV) during overload. The basic idea is to relax the temporal constraints of the frame display. This way, the decoder may prepone/postpone the frame display in order to cope with the longer decoding times without the need to drop frames. Of course, without further consideration, this approach leads to large variations of inter-display time, which degrades the PQV. We use the gravitational task model to schedule the completion of the decoding task as close as possible to the optimum display instant, hence accounting for both optimum PQV under low system workload, and adaptivity under overload with decreased degradation of the PQV.

We measured the frame display jitter and the frame drop rate, as there is no precise evaluation metric for the PQV. Results showed that with the gravitational task model the jitter distribution is closer to 0, and the frame drop rate is 0. We also produced the resulting video output to visualize the effect of jitter and frame drop, concluding that the gravitational task model provides for better PQV than other approaches.

Section VII.2 presented an algorithm for adaptive resource management under scarce resource availability. We considered a multicore system where applications may contain parallelizable execution flows, and each flow has a resource requirement. Applications may also offer different service levels, depending on the resource availability. Our algorithm uses a gravitational task model based compression method which has linear complexity for reallocation of resources among applications, and service level assignment. The pendulum analogy provides for the intuition of the solution. This algorithm has been implemented in the ACTORS Framework, which is a project of the 7^{th} Frame-

work Programme for Research and Technological Development.

Finally, section VII.3 described an opportunistic packet scheduling strategy which aims at reduced retransmission ratio of packets in BANs (refer to section II.3 for a detailed description of packet scheduling issues in BANs). The basic idea is that sending packets when the Reveived Signal Strength Indicator (RSSI) value is high reduces the retransmission ratio, which increases communication reliability and reduces energy consumption. This application has been entirely developed by researchers not involved with studies on the gravitational task model, and evidences the contribution of the gravitational task model's intuition to ease the cooperation among different research fields.

This section described a set of experiments for the characterization of RSSI fluctuations in BANs. The results of this characterization were used to define a transmission window, within which packets are more likely to be successfully transmitted. As all packets share the same target point for transmission, they are scheduled using the equilibrium of the gravitational task model.

Chapter VIII

In this chapter, entitled *Discussion*, we covered a few scheduling related topics that may impact on the utility accrual of a schedule, and thus, we consider that they should not be ignored. However, due to the complexity to address these topics, we just present a discussion of their impact on scheduling, and potential research directions. These topics are early completion times, preemptive scheduling, and multicore scheduling.

Although we have considered the actual execution times of jobs are equal to the worst case execution times (WCET)) throughout this work, most likely execution times are variable. For example, multimedia applications have target sensitive constraints, and highly variable execution times. We showed in section VIII.1 how can this alter the utility accrual of a schedule, and discussed pros and cons of possible solutions.

In section VIII.2, we discussed the expressiveness that preemptive scheduling requires from the task model. We used the control application described in section II.5 as example to illustrate the impact of preemption on utility accrual, and to identify which the support that task models have to supply.

Finally, in section VIII.3, we briefly discussed the issues involved in scheduling of target sensitive applications on multicore systems.

Chapter IX

This chapter summarized the main contributions of this thesis, and brought the concluding remarks.

Appendix A

This appendix brings extended experimental results from the applications presented in chapter VII. These results were not presented in their respective section due to their similarities with other already discussed results.

Zusammenfassung

Echtzeitanwendungen haben zeitliche Begrenzung zusätzlich zu ihren logischen Outputs für richtiges funktionales Systemverhalten. Regelungsanwendungen, die die Ursache für die Entstehung der zeitlichen Begrenzungen der Echtzeitsysteme sind, wirken auf die Regelstrecke um diese entsprechend der Führungsgröße zu steuern. Regelungsysteme beinhalten sicherheitskritische und betriebsnotwendige Anwendungen, welche strikte zeitliche Begrenzungen haben. Nichteinhaltung eines einzigen Fristablaufs kann den Betrieb des Systems zum Absturz führen und sogar katastrophale Konsequenzen herbeiführen.

Üblicherweise bestehen Echtzeitsysteme aus mehreren Tasks, die gleichzeitig um Systemressourcen konkurieren. Daher erfordern Echtzeitanwendungen spezielle Zeitplanungsalgorithmen (sogenannte Echtzeitplanungsalgorithmen), die zeitliche Begrenzungen beachten. Diese Algorithmen legen eine Anzahl von Regeln fest um die Reihenfolge der Task-Ausführung zu planen, damit die zeitlichen Begrenzungen eingehalten werden. Zum Beispiel haben einige Tasks Einschränkungen bezüglich ihres frühestmöglichen Starts uns ihrer Abarbeitungsfrist. Diese beiden Zeitpunkte definieren das *execution window* des Tasks. Das execution window ist ein Zeitraum in welchem das logische Ergebnis in korrektem Systemverhalten resultiert. Ein Echtzeitplanungsalgorithmus muss garantieren, dass alle Tasks vollständig innerhalb ihres execution windows ausgeführt werden.

Manche Echtzeitanwendungen haben sehr enge optimale execution windows für einen maximalen Systemnutzen (*system utility*), akzeptieren aber aus Gründen der Realisierbarkeit (*feasibility*) eine gewisse Flexibilität durch Ausdehnung der execution windows. Um die Flexibilität aufrecht zu erhalten, werden Verluste der maximalen system utility in Kauf genommen. Zum Beispiel Tasks sogenannter *target sensitive applications* sollten bevorzugt an ihrem Zielzeitpunkt (*target point*) ausgeführt werden. Auch die Ausführung eines Task in zeitlicher Nähe seines target points ist möglich, reduziert aber die system utility. Diese Reduktion ist proportional zur Wichtigkeit der Anwendung für das System. Idealerweise werden alle Tasks an ihrem target point ausgeführt, was wegen Überlappung der Ausführungen nicht möglich ist. Unter dieser Bedingung müssen die Ausführungen zeitlich geplant werden, so dass alle zeitlichen Begrenzungen eingehalten werden und die system utility maximiert wird. Wichtige Anwendungen sind weniger tolerant gegenüber der Abweichung zum target point.

Neben der notwendigen Ausdrucksfähigkeit, müssen die Task-Modelle auch simple

Abstraktionen bereitstellen, damit die Anwendungsentwickler sie einfacher verstehen können. Darüber hinaus erfordert Anpassbarkeit zur Laufzeit Zeitplanungsalgorithmen mit geringem Overhead. Ein Kompromiss zwischen den Tasks mit der Ausführungsüberlappung schließt eine Ausführungsumstellung des Jobs, seine Verschiebung und möglicherweise seinen Abbruch ein. Eine Schwierigkeit besteht darin auszudrücken ob zwei weniger wichtige Tasks wichtiger sind als ein sehr wichtiger Task. Eine andere Schwierigkeit besteht darin, dass eine Änderung der Ausführungsreihenfolge ein NP-vollständiges Problem darstellt. Deswegen ist die Entwicklung von Zeitplanungsalgorithmen mit geringem Overhead eine besondere Herausforderung.

In dieser Dissertation wird ein neues Echtzeit-Task-Modell vorgestellt, welches simple Abstraktionen zum Ausdruck von zeitliche Begrenzungen der target sensitive applications berücksichtigt: das *Gravitational Task-Modell*. Dieses Task-Modell verwendet Pendel als Analogie um das Verständnis seiner zeitlichen Abstraktionen zu vereinfachen. Ebenso werden einige Zeitplanungsalgorithmen für das Gravitational Task-Modell vorgestellt. Diese Algorithmen verwenden auch die Analogie des Pendels auf Grund der Deutlichkeit. Zum Schluss werden drei Anwendungen vorgestellt, die von dem Gravitational Task-Model unterstützt werden und die Verbesserungen hervorgehoben.

Im Folgenden wird auf den Inhalt der einzelnen Kapitel eingegangen.

Kapitel I

Dieses Kapitel stellt die Einführung dieser Doktorarbeit dar. Im Abschnitt I.1, wird das Basiskonzept der Echtzeitplanung beschrieben. Im Abschnitt I.2 wird der Fokus auf Eigenschaften und Anforderungen von Echtzeitsysteme gelegt. Im Abschnitt I.3 werden die target sensitive Echtzeitanwendungen und die dazugehörigen Zeitplanungsprobleme erklärt, gefolgt von verwandten Arbeiten in diesem Bereich im Abschnitt I.4. Im Abschnitt I.5 werden die Hauptbeiträge dieser Dissertation für den state-of-the-art der Echtzeitsplanungs-Theorie zusammengefasst. Zum Schluss wird im Abschnitt I.6 der inhaltliche Überblick der Dissertation gegeben.

Kapitel II

In diesem Kapitel, mit dem Titel *Examples of target sensitive real-time aplications*, werden fünf Beispiele für target sensitive applications beschrieben, um die Notwendigkeit eines Task-Modells zu bestätigen, welches die zeitliche Begrenzungen solcher Anwendungen berücksichtigt.

Im Abschnitt II.1, wird eine Multimediaanwendung vorgestellt. Multimediasysteme sind in unserem Alltag integriert, mit Anwendungen in verschiedensten Lebensbereichen, z.B. Gesundheitssysteme, Computerspiele, Automobilindustrie, Werbeindustrie, etc. Der MPEG-2 Videokodierungsstandard wird verwendet, um die Vorgänge des Videoabspielens, sowie die Ressourcenanforderungen und zeitliche Begrenzungen dieser Vorgänge zu erläutern.

Im Abschnitt II.2 wird die adaptive Ressourcenverwaltung von Echzeitanwendungen beschrieben. Werden adaptive Anwendungen auf Systemen ausgeführt, deren Ressourcen

nicht für den worst case konzipiert wurden, so erfordern sie eine solche Ressourcenverwaltung, damit der Schwund des *Quality of Service (QoS)* reduziert wird.

Im Abschnitt II.3 wird eine Anwendung der drahtlosen Sensorsysteme in der medizinischen Technik vorgestellt, sogenannte *Body Area Networks (BAN)*. BANs verfügen über sehr geringe Ressourcen und hohe QoS-Anforderung. Dieser Abschnitt zeigt auch die zeitlichen Begrenzungen und Ressourcenanforderung des Scheduling von Netzwerkpaketen, sowie die Gründe für die Bezeichnung dieser Anwendung als target sensitive application auf.

Im Abschnitt II.4 wird die Relevanz des Datenflussprogrammierungparadigmas vorgestellt, welches es erlaubt das Abstraktionsniveau in der Programmierung für Multicoresysteme zu erhöhen. Danach werden Probleme des Scheduling von Datenflussgraphen beschrieben.

Zum Schluss, im Abschnitt II.5, werden die Ressourcenanforderungen und zeitliche Begrenzungen von Regelungssystemen vorgestellt. Es wird auch gezeigt, dass die zeitlichen Begrenzungen nicht so strikt sind wie üblicherweise angenommen. Solche Systeme akzeptieren etwa Flexibilität in den zeitlichen Begrenzungen auf Grund der Realisierbarkeit der Zeitplanung des involvierten Tasks. Deswegen können sich Regelungssysteme von Task-Modellen und Zeitplanungsalgorithmen, die für target sensitive applications entwickelt wurden, positiv auswirken.

Kapitel III

In diesem Kapitel, mit dem Titel *Gravitational Task Model: a bob pendulum based approach to express trade-offs*, wird das Gravitational Task-Modell für target sensitive applications vorgestellt. Im Abschnitt III.2 werden Terminologie und Vermutungen präsentiert und im Abschnitt III.3 wird aufgezeigt, dass die Zeitplanung von target sensitive applications eine Kompromissberechnung einschließt. Ebenso wird dargelegt, dass klassische Task-Modelle entweder die Realisierbarkeit auf Grund der erhöhten angefallen system utility beeinträchtigen oder bei effizienter Ressourcennutzung versagen.

Im Abschnitt III.4 wird das Pendel als Visualisierungsmodel für den Kompromiss zwischen allen target sensitive applications vorgestellt. Ein Pendel besteht aus einer Masse am Ende eines Seiles, welche von der Erdanziehungskraft angezogen wird. Befindet sich der Masseschwerpunkt in dem zentralen, tiefsten Punkt des Pendeltrajektorie (der Ruheposition) so schwingt die Masse nicht. Wenn mehrere Pendel aufeinander treffen, die aus verschiedenen Massen und Größen bestehen können, so ist die ursprüngliche Ruheposition eines jeden Pendels versetzt. Die neue Ruheposition (das sogenannte Equilibrium) entspricht einem Kompromiss zwischen allen Pendel bezüglich ihrer Größen und Massen.

Im Abschnitt III.5 wird die Analogie zwischen Pendel und target sensitive application beschrieben. Durch diese Analogie und das Equilibrium der Pendel ist eine einfache Berechnung des Kompromisses zwischen mehreren target sensitive applications möglich.

Zum Schluss im Abschnitt III.6 wird die Berechnung des Equilibriums der Pendel behandelt. Es werden sowohl eine Näherungslösung als auch eine exakte Lösung vorgestellt.

Kapitel IV

In diesem Kapitel, mit dem Titel *Scheduling target sensitive real-time tasks*, werden verschiedenen Zeitplanungsalgorithmen für das Gravitational Task-Modell vorgestellt. Im Abschnitt IV.2 wird eine Methode vorgestellt, welche das Equilibriumkonzept des Pendels verwendet um die Ausführung der Jobs in einer busy period zu planen, damit die angefallene system utility erhöht wird. Im Abschnitt IV.2.1 wird eine weitere Methode beschrieben, welche die busy periods in einer Zeitplanung aufzeigt und ebenso auf der Pendelanalogie basiert. Beide Methoden haben eine lineare Komplexität.

Im Abschnitt IV.2.2 wird ein Algorithmus für die Berechnung einer Näherung des Equilibriums von zwei busy periods mit Ausführungsüberlappung vorgestellt. Es wird auch gezeigt, dass es in einer Zeitplanung nicht mehr als N Ausführungsüberlappungen geben kann, wobei N für die Anzahl der Jobs steht. Da dieser Algorithmus eine konstante Komplexität hat, ist die Komplexität der Zeitplanung aller Jobs linear.

Im Abschnitt IV.2.3 wird ein Verfahren für die Job-Zulassung zur Laufzeit präsentiert und in Abschnitten IV.2.4 und IV.2.5 werden die Zeitplanungsalgorithmen mit Beispielen illustriert.

Im Abschnitt IV.3 werden die Wichtigkeit der Reihenfolge der Jobausführungen und verschiedene Sortierungsheuristiken präsentiert. Das finden der optimalen Ausführungsreihenfolge für Realisierbarkeit und maximale angefallene system utility der Zeitplannung ist ein NP-vollständiges Problem. Deswegen benötigt die Berechnung optimaler Lösungen erhöhter Rechenaufwand, welche die Verwendung zur Laufzeit solcher Algorithmen verhindert. Sortierungsheuristiken finden einen Kompromiss zwischen der Qualität der Lösung und des Overheads der Berechnung.

Im Abschnitt IV.3.1 wird die Heuristik DST-1 präsentiert, welche die Komplexität $O(N \times log(N))$ hat. Diese Heuristik berücksichtigt die execution windows der Jobs nicht für das Sortierungsverfahren und beeinträchtigt daher die Realisierbarkeit. Im Abschnitt IV.3.2 wird die Heuristik DST-2 vorgestellt, welche eine Modifikation zu DST-1 ist, die die execution windows der Jobs berücksichtigt. Diese Heuristik hat ebenso wie DST-1 die Komplexität $O(N \times log(N))$. Im Abschnitt IV.3.3 wird auf die Heuristik DST-3 eingegangen, welches eine erneute Sortierung der Ausführungsreihenfolge zur Zeit der Ankunft eines Jobs verhindert. Diese Modifizierung ermöglicht eine weitere Reduzierung der Komplexität, was eine lineare Zeitkomplexität zur Folge hat.

Zum Schluss im Abschnitt IV.4 werden die Simulationsergebnisse präsentiert, welche aus den oben genannten Heuristiken resultieren und sich als erfolgreich erwiesen haben. Es wurde festgestellt, dass DST-2 eine höhere angefallene system utility erreicht. Außerdem wurde beobachtet, dass DST-3 gegenüber von DST-2 bei signifikanter Reduzierung der Komplexität nur eine geringfügig schlechteres Endergebnis liefert. Die Ergebnisse der Simulationen zeigen auch Vorteile von dem Gravitational Task-Modell gegenüber den anderen klassischen Task-Modellen und deren Zeitplanungsalgorithmen. Durch das Gravitational Task-Modell wird die angefallene system utility erhöht ohne die Realisierbarkeit der Zeitplanung zu beeinträchtigen.

Kapitel V

In diesem Kapitel mit dem Titel *Reducing the complexity of periodic tasks' scheduling*, geht es um einen Zeitplanungsalgorithmus für periodische Tasks, der auf dem gravitational task model basiert. Dieser Algorithmus ist eine Mischung aus EDF under der Heuristik DST-3 und heißt EDF-swap. EDF-swap zieht erhöhte Realisierbarkeit und erhöhte angefallene system utility in Betracht. Dazu reduziert er die Komplexität von $n!$ zu n^2.

Im Abschnitt V.2 wird gezeigt, dass das Equilibrium der Jobs nicht ohne weitere Berücksichtigung mit EDF kombiniert werden kann. Die Schwierigkeit liegt darin, die Ausführung der Jobs zu verschieben ohne dabei die ursprüngliche Ausführungsreihenfolge umzustellen. Dieses Problem wurde gelöst durch die Einführung eines Zeitraums (*equilibrium window*), in dem das Equilibrium die Jobausführungen verschieben darf. Das equilibrium window garantiert die ursprüngliche Ausführungsreihenfolge bei einer möglichen Verschiebung der Jobausführungen.

Im Abschnitt V.3 wird eine Heuristik vorgeschlagen, die eine erhöhte angefallene system utility durch die Umstellung der Ausführungsreihenfolge erreicht. Diese Heuristik beeinträchtigt nicht die Realisierbarkeit der Zeitplanung. Anhand eines Beispiels wird diese Heuristik illustriert.

Ergebnisse von Simulationen in Abschnitt V.4 zeigen, dass EDF-swap eine reduzierte Komplexität, ohne die Beeinträchtigung sowohl der Realisierbarkeit als auch der system utility Ansammlung, erreicht.

Kapitel VI

Im diesem Kapitel mit dem Titel *Scheduling tasks for increased system utility under scarce resource availability* wird gezeigt, dass der Abbruch eines oder mehrerer Tasks eine endliche erhöhte angefallene system utility ermöglichen kann. Es geht darum, dass die Abbrüche einiger Tasks es ermöglichen, dass die übrigen Tasks näher an ihren entsprechenden target points ausgeführt werden. Bei der Entscheidung über eine Verschiebung oder ein Abbruch einer Jobausführung, wird die angefallene system utility berücksichtigt. Diese Entscheidung, welche ein NP-vollständiges Problem ist, wurde im Abschnitt VI.2 durch eine Methode mit $O(N^2)$-Komplexität gelöst. Die Vorteile dieser Methode werden im Abschnitt VI.3 mit einer Multimedia-Anwendung für qualitätsbewusste Videoanpassung unter mangelnder Ressourcenverfügbarkeit. Diese Anpassungsstrategie verwendet die oben genannte Lösungsmethode um zu entscheiden, ob die Dekodierung eines Einzelbildes verschoben oder abgebrochen werden soll.

Kapitel VII

In diesem Kapitel, mit dem Titel *Applications of the gravitational task model*, werden drei Anwendungen vorgestellt, die von dem gravitational task model unterstützt werden, und die Verbesserungen hervorgehoben. Die Anwendungen sind die folgenden: Multimedia-Anwendung für qualitätsbewusste Videoanpassung, Ressourcenverwaltung

der adaptiven Echtzeitanwendungen und Zeitplanung der Netzwerkpaketensendungen in einem Body Area Network.

Chapter VIII

Im diesem Kapitel, mit dem Titel *Discussion*, werden die Wichtigkeit 3 Themen im Kontext des Scheduling der target sensitive applications diskutiert, und wie sie die angefallene system utility beeinflussen können: frühstens completion time, preemption, und multicore scheduling.

Chapter IX

Dieses Kapitel fasst die Hauptbeiträge dieser Dissertation zusammen und endet mit dem Schlussbemerkungent.

Anhang A

Dieser Anhang enthält zusätzliche Versuchsergebnisse der Anwendungen von Kapitel VII. Aufgrund ihrer Gleichartigkeit werden diese Ergebnisse nicht in den entsprechenden Abschnitten erwähnt.

Curriculum Vitae – Raphael Pereira de Oliveira Guerra

Contact Information

Filiation: TU Kaiserslautern
Position: Research Assistant
Address: Mühl-str 41, 67659
City: Kaiserslautern
State: Rhineland-Palatinate
Country: Germany
Email: guerra@eit.uni-kl.de
Phone: +49 (0) 631 205 3674
Mobile: +49 (0) 176 6339 1952
Fax: +49 (0) 631 205 4199

Personal Information

Gender: Male
Date of birth: 19/05/1983
Place of birth: São Gonçalo, Rio de Janeiro, Brazil
Citizenship: Brazilian
Marital status: Single

Research Interests

Flexible and adaptive real time systems, power and energy aware computing, resource management, multimedia, distributed systems, and network.

Unviversity Education

Nov 2006 – Jun 2011, *Doctorate in Electrical Engineering with honors*
- ➢ Technische Universität Kaiserslautern, Germany
- ➢ Thesis: A Gravitational Task Model for Target Sensitive Real Time Applications
- ➢ Advisor: Prof. Gerhard Fohler

Feb 2005 – Sept 2006, *Master of Science in Parallel and Distributed Computing*
- ➢ Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil
- ➢ Thesis: Improving Response Time and Energy Efficiency in Server Clusters
- ➢ Advisor: Prof. Julius Leite

Feb 2001 – Dec 2004, *Bachelor of Science in Computer Science with honors*
- ➢ Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil
- ➢ Thesis: Sistema de Segurança para a Plataforma FIPA-OS (*translation: Security system for FIPA-OS platform*)
- ➢ Advisor: Prof. Julius Leite

Other Studies

Feb 2000 – Dec 2000, *High school*
- ➢ Centro Educacional Farias Ribeiro, Maricá, Rio de Janeiro, Brazil

Feb 1998 – Dec 1999, *High school*
- ➢ C.E.M. Joana Benedicta Rangel, Maricá, Rio de Janeiro, Brazil

Feb 1996 – Dec 1997, *Middle school*
- ➢ Colégio Cenecista Maricá, Maricá, Rio de Janeiro, Brazil

Feb 1994 – Dec 1995, *Middle school*
- ➢ Colégio Albert Sabin, Niterói, Rio de Janeiro, Brazil

Feb 1992 – Dec 1993, *Elementary school*
- ➢ Centro Educacional de Inoã, Niterói, Rio de Janeiro, Brazil

Feb 1990 – Dec 1991, *Elementary school*
- ➢ Instituto Cultural Amendoeira, São Gonçalo, Rio de Janeiro, Brazil

Academic Experience

Jun 2009 – present, *EU ICT Project ACTORS*
- ➢ Technische Universität Kaiserslautern, Germany
- ➢ Position: Research assistant
- ➢ Advisor: Prof. Gerhard Fohler
- ➢ Description: Research on adaptive resource management for parallel and multicore real-time systems.

Nov 2006 – May 2009, *EU IST Project FRESCOR*
- ➢ Technische Universität Kaiserslautern, Germany
- ➢ Position: Research assistant
- ➢ Advisor: Prof. Gerhard Fohler
- ➢ Description: Management of project requirements, and research on adaptive resource management.

Apr 2003 – Dec 2004, *Lab Administrator*
- ➢ Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil
- ➢ Instituto de Computação e Pós-Graduação
- ➢ Description: Unix network administration and hardware maintenance.

July 2001 – July 2002, *Professor Assistant in Linear Algebra*
- ➢ Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil
- ➢ Advisor: Prof. Ricardo Eleodoro Fuentes Apolaya
- ➢ Description: Assist students on homework.

Academic Development

Exchange programs and scholarships

Sept 2009 – Nov 2009, *External research program*
- ➢ University of North Carolina at Chapel Hill, NC, USA
- ➢ Research field: Adaptive and flexible real-time scheduling theory
- ➢ Advisors: Prof. James H. Anderson and Prof. Sanjoy Baruah

Feb 2005 – Sept 2006, *CAPES scholarship to support M.Sc. studies*
- ➢ Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil
- ➢ Research field: Energy-aware distributed real-time computing systems
- ➢ Advisor: Prof. Julius Leite

May 2004 – July 2004, *Exchange program sponsored by Linnaeus-Palme*
- ➢ Mälardalens Högskola, Västerås, Sweden
- ➢ Research field: Available Bandwidth Estimation Over Wireless Network
- ➢ Advisor: Prof. Gerhard Fohler

Apr 2003 – Dec 2004, *CAPES scholarship to support junior research*
- ➢ Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil
- ➢ Research field: Mobile Agent Security
- ➢ Advisor: Prof. Julius Leite

Student supervision

Sept 2009 – Aug 2010 — Supervised Ardo Schäfer during his *Diplomarbeit*
- ➢ Technische Universität Kaiserslautern, Germany
- ➢ Thesis: Determination the largest busy period of periodic real-time tasks

Sept 2008 – Mar 2009 — Supervised Ardo Schäfer during his *Studienarbeit*
- ➢ Technische Universität Kaiserslautern, Germany
- ➢ Thesis: Control-flow-aware Scheduling of Soft Aperiodic Tasks in Hard Real Time Systems

Activities and services

Program committee member of the 1^{st} and 2^{nd} Junior Researcher Workshop on Real-Time Computing (JRWRTC).

Reviewer in some computer science conferences and journals.

Held a few lectures and exercise-classes at Technische Universität Kaiserslautern in Germany as professor assistant.

In charge of the Real-Time Systems Seminar at Technische Universität Kaiserslautern.

Honors and Awards

Honors Doctorate, *Technische Universität Kaiserslautern, Germany*

Computer Science Departmental Award, *Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil*
In recognition for being the second best student ever graduated in the department until 2004.

Best Paper Award, *20th Euromicro Conference on Real-Time Systems*
Paper entitled *A Gravitational Task Model for Target Sensitive Real-Time Applications.*

3^{rd} Best Exercise-class Award, *Exercise-class of Real Time Systems II*
Award granted by the Department of Electrical Engineering at Technische Universität Kaiserslautern in Germany, winter semester 2009.

Selected Publications

Journal papers

1. Enrico Bini, G. Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Vanessa Romero, Claudio Scordino, Karl-Erik Arzen, Resource Management on Multi-core Systems: the ACTORS approach, IEEE Micro, December 2010.
2. Raphael Guerra, Gerhard Fohler, A Gravitational Task Model with Arbitrary Anchor Points for Target Sensitive Real-Time Applications, Real-Time Systems Journal, September 2009.

Conference/workshop papers

3. Raphael Guerra, Gerhard Fohler, What is the meaning of preemption in utility-based real-time scheduling?, Proceedings of 2nd International Real-Time Scheduling Open Problems Seminar, July 2011.
4. Raphael Guerra, Gerhard Fohler, Handling overload of target sensitive real-time applications for increased system utility and improved resource usage, SAC'11: Proceedings of 26th ACM Symposium on Applied Computing, ACM Publisher, Taiwan, March 2011.
5. Raphael Guerra, Gerhard Fohler, An optimum generalized equilibrium solution for the gravitational task model, RTNS'10: Proceedings of the 18^{th} International Conference on Real Time and Network Systems, IRIT lab, Toulouse, France, November 2010.
6. Raphael Guerra, Stefan Schorr, Gerhard Fohler, Adaptive Resource Management for Mobile Terminals - The ACTORS approach, WARM'10: Proceedings of the 1^{st} Workshop on Adaptive Resource Management, Stockholm, Sweden, April 2010.
7. Raphael Guerra, Gerhard Fohler, On-line Scheduling Algorithm for the Gravitational Task Model, ECRTS'09 - Proceedings of the 21^{st} Euromicro Conference on Real-Time Systems, IEEE Computer Society, Dublin, Ireland, July 2009.
8. Raphael Guerra, Gerhard Fohler, Video decoding for reduced jitter and improved resource usage, WRT'09: XI Workshop de Tempo Real, Recife, PE, Brazil, May 2009.
9. Raphael Guerra, Gerhard Fohler, A Gravitational Task Model for Target Sensitive Real-Time Applications, ECRTS08 - Proceedings of the 20^{th} Euromicro Conference on Real-Time Systems, IEEE Computer Society, vol. 1, pag. 309 – 317, Prague, Czech Republic, July 2008.
10. Raphael Guerra, J.C.B. Leite, Gerhard Fohler, Attaining soft real-time constraint and energy-efficiency in web servers, ACM Symposium on Applied Computing, pag. 2085-2089, Fortaleza, Brazil, March 2008.
11. Raphael Guerra, Gerhard Fohler, A gravitational task model for target sensitive real-time applications, RTSS'07: Work-in-Progress Proceedings of the 28^{th} IEEE Real-Time Systems Symposium, IEEE Computer Society, pag. 49-52, Tucson, AZ, USA, December 2007.
12. Raphael Guerra, Luciano Bertini, J.C.B. Leite, Managing Energy and Quality of Service in Heterogeneous Server Clusters, CLEI'06 - Proceedings of the 32^nd Latin-American Conference on Informatics, Santiago, Chile, August 2006.
13. Raphael Guerra, Luciano Bertini, J.C.B. Leite, Improving Response Time and Energy Efficiency in Server Clusters, WRT'06: VIII Workshop de Tempo Real, Curitiba, PR, Brazil, May 2006.

Skills

Technical skills

- ➢ Theoretical real-time scheduling
- ➢ Resource management for adaptive systems
- ➢ Scientific and technical writing and presentation
- ➢ Experience teaching

IT skills

- ➢ *Programming languages*: C, C++, Java, script languages
- ➢ *Scientific software*: R, gnuplot, LaTeX
- ➢ Low-level programming
- ➢ Hardware configuration and maintenance
- ➢ Unix network configuration and maintenance

LANGUAGES

Portuguese — Native

English — Fluent

Spanish — Advanced

German — Intermediate/Advanced

REFEREES

Available on request.