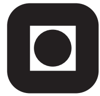

Alternatives to Classic Real Time

A Literature Study

Nicholas Stangeland

Master of Science in Cybernetics and Robotics
Submission date: June 2015
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Objective

A study of alternative ways to make real-time systems.

Liu & Layland introduced what is now the classic way to create real time systems: We divide the system into threads responsible for each time requirement, choosing a predictable scheduler, and assuming we know the worst-case run time of each thread [1]. This enables us to conduct a schedulability proof which argues that all deadlines will be reached.

The student is to study (all) alternative ways to create real time systems. A literature search will be the main part of the work, but should contain more or less of collecting and generating ideas to span the space of alternatives.

Examples of non-schedulability proof-based ways to create real-time systems may be e.g.:

- Can we detect and handle time-domain errors like all other errors in the system?
- Can we predict (eg. by online execution time analysis) deadline-misses before they happen?
- Can we interact with applications to adjust their needs for computing power when the system is busy?
- Can we have adaptive systems that reduce the probability of timing errors to acceptable values?
- Can we redefine "real time" to mean something other than the traditional "hard deadlines," while still making sense?
- Can we allow stochastic models etc. for real-time?
- ...

If relevant, one or more exciting alternatives can be investigated more thoroughly.

The thesis may conclude with an analysis of the state of the art, identification of trends, predictions about the future etc.

Abstract

This paper will take a closer look at some of the research in the field of real time systems. The classic approach to real time, the Liu and Layland method is thoroughly investigated and compared to other approaches to real time. The main focus is the task model and variants of the task model. Topics relevant for the task model such as scheduling, dynamic voltage scaling, WCET and time/utility functions are discussed. Control theory and its implications in real time are investigated as well as the science of making safety-critical systems.

Sammendrag

Denne oppgaven tar for seg forskning innen sanntidsprogrammering. Den klassiske tilnærmingen til sanntid, Liu og Layland-metoden er gjennomgått og sammenlignet med andre tilnærminger til sanntid. Hovedvekten av oppgaven baseres på ”task”-modellen og varianter av denne. Relevante temaer innen ”task”-modellen som ”scheduling,” dynamisk spenningsskalering, verste kjøretid (WCET) og tid/nytte-funksjoner blir gjennomgått. Reguleringsteknikk og forholdet til sanntid blir beskrevet, i tillegg til vitenskapen bak sikkerhetskritiske systemer.

Content

Objective 3
Abstract 4
Sammendrag 5
Introduction 10
The Definition of Real Time Systems 10
Types of Real Time Systems 10
Task Based approaches 10
Classic Real Time: Liu & Layland 10
Schedulability Proofs 11
Liu and Layland Classification of Real Time Systems 12
Hard 12
Firm 12
Soft 13
Stochastic Real Time 13
Petri Net and FPGA 13
Scope of Work 14
Outlines to the Alternatives of Classic Real Time 14
Method 16
Preliminaries 18
Historical background 18
Today 18
Safety-Critical Systems 19
Hardware and Software 19
Achieving Safety Criticality 19
The Answer is Real Time 20
Airplanes and Other Safety-Critical Systems 20
Approaches to Reactive Systems 22
Preliminaries 22
Synchronous Approaches to Reactive Programming: E.G.: Lustre and Esterel 22
The Task Based Model 23
Communicating Sequential Processes (E.G.: Ada and Occam) 23

The Petri Net-Based Model 24
Scheduling 25
Preliminaries 25
Scheduling Strategies 25
Preemption and Overhead 25
“Open Loop” 26
Static 26
Dynamic 26
Closed Loop 26
Feedback Scheduling (FBS) 26
Summary 27
Dynamic Voltage Scaling (DVS) 28
Background 28
Hard Real Time 28
Soft and Firm Real Time 29
Battery Powered Systems 29
Overclocking 30
Concluding remarks on DVS 30
Real Time and Control Systems 32
Background 32
Timing in Control Systems 32
Jitter 32
Sampling Jitter 32
Sampling-Actuation Delays 32
Sampling Jitter and Sampling-Actuation Delays 33
Jitter Compensation 33
Model of Computation: Continuous Stream Task Model 34
Verification 34
Simplex Design 34
Concluding Remarks on Control Systems 35
Scheduler Interaction with Controller 35
Worst Case Execution Time (WCET) 36

Preliminaries ... 36
Average Case Execution Time (AET) ... 38
Preliminaries ... 38
Computing AET ... 38
Summary ... 39
Predictability and Determinism ... 40
Preliminaries ... 40
Statistics: The use of average to achieve predictability in uncertain environments ... 40
Online Estimation of Run-Time ... 41
Time/Utility Functions ... 43
Preliminaries ... 43
Use of TUFs ... 43
Suggestion: Reclassifying the Real Time Task Model ... 44
The Task Model Classification ... 44
Hard Real Time ... 44
Classic Hard Real Time ... 44
Non-Strict Hard Real Time ... 44
Firm Real Time ... 45
Soft Real Time ... 46
Scheduling Real Time Built upon the Task Model ... 48
Stochastic Real Time ... 48
Preliminaries ... 48
Soft and Firm Schedulers (Stochastic) ... 49
Hard Schedulers (Stochastic and deterministic) ... 49
Non-strict ... 49
Strict (Deterministic) ... 49
Designing Real Time Systems ... 50
The Three Dimensions of a System ... 50
Criticality ... 51
Result ... 51
Time ... 51
Choosing the Right Model ... 51

Current Trend in Real Time Research.. 53
Answering the Questions 55
Can we detect and handle time-domain errors like all other errors in the system?........... 55
Can we predict (eg. by online execution time analysis) deadline-misses before they happen?.. 55
Can we interact with applications to adjust their needs for computing power when the system is busy?.. 55
Can we have adaptive systems that reduce the probability of timing errors to acceptable values?.. 55
Can we redefine "real time" to mean something other than the traditional "hard deadlines," while still making sense?.. 56
Can we allow stochastic models etc. for real-time? 56
Glossary 57
Bibliography/References 58

Introduction

The Definition of Real Time Systems

What does it mean for a system to be real time? In computer systems it usually refers to either a simulation with a system clock running 1:1 with the real clock or a system under real time constraint. In this text real time refers to the latter. For a system under real time constraints a right answer is only right if it is correct and is given at the right time, in contrast to non-real time. But when is the right time? Clearly a system which responds too late is not on the right time, but what if it is a little early? This will differ from system to system. Imagine a completely electronic car: When you hit the accelerator you expect the car to accelerate. In this case there is impossible for the system to be too early. Accelerating before you hit the gas is simply a wrong answer. For a system that can be too early: When listening to music the application needs to process the sound data before it can be played. During this process data is usually not processed in the exact right order meaning the application can have a note ready before it should be played. The problem? Playing a correct note too early is as bad as playing a wrong note.

To avoid an early answer the system can simply hold the answer until the correct time approaches, but this is clearly not possible with a late answer. Therefore it is easily deductible that two solutions exist: Ensure that the answer isn't late or fix the fault that occurs because the system is late. However, according to Liu & Layland [1], the latter is not possible for a hard deadline system. In this text that assumption will be challenged.

Types of Real Time Systems

Task Based approaches

In this approach every job is divided into tasks which are responsible for part of a job. For instance is a controller a job consisting of a sampling task, a controller algorithm task and an actuator task.

Classic Real Time: Liu & Layland

In 1973 Liu and Layland published their famous paper "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment" [1] in which most of the real time science is based upon and it has been cited several thousand times according to citeseerx and Google Scholar, listing it with nearly 10 000 citations [9]. In their paper Liu and Layland list the following five assumptions which are necessary for a hard real time system:

1. The requests for all tasks for which hard deadlines exist are periodic, with constant interval between requests.
2. Deadlines consist of run-ability constraints only--i.e. each task must be completed before the next request for it occurs.

3. The tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks.
4. Run-time for each task is constant for that task and does not vary with time. Run-time here refers to the time which is taken by a processor to execute the task without interruption.
5. Any nonperiodic tasks in the system are special; they are initialization or failure-recovery routines; they displace periodic tasks while they themselves are being run, and do not themselves have hard, critical deadlines.

In fact it is hard to defend any other assumption than assumption 2. In the conclusion of their paper they do admit to the following: “Perhaps the most important and least defensible of these are (A1), that all tasks have periodic requests, and (A4), that run-times are constant.” Even so assumption 3 will usually not hold because of interdependence of the system and assumption 5 requires any other non-periodic task to be supervised by a periodic task [13] which is sub-optimal (polling as opposed to interrupts). It may even be impossible to do this if the frequency of this supervisor task must be so high that some other task will not reach its deadline.
In other words, an important part of the foundation of the hard real time science rests on several major flaws. However, this only proves that it is impossible to make a perfect system. What is important though is that this way of thinking hard real time cannot be the only way. This paper will list some viable alternatives that still are safety-critical as classic hard real time, but uses other assumptions and therefore do not have its flaws.

Worryingly enough is the fact that, according to Devillers and Goossens in their paper entitled “Liu and Layland's schedulability test revisited” [27], some of the proofs in Liu and Layland [1] were based on wrongful assumptions. Although the conclusion based on the proofs were more or less correct, a wrong proof should be alarming. Taken into account the number of years before the proof was corrected, this may be a bad sign for the classic hard real time science. Nonetheless it is only cited 8 times according to [29]; therefore its impact on classic real time science is negligible. However, Google Scholar lists 39 citations, but it is still an insignificant number compared to revolutionary papers.

Schedulability Proofs

Associated with classic real time is the necessity of performing a schedulability proof of the worst case execution time. In essence this means that for a given system one first has to split all work into tasks and further calculate the worst case execution time for every task. This may be difficult if even possible due to the following: Modern processors are being design to prioritize average over worst case, according to Nilsen and Rygg in their paper from 1995 and nothing seems to have changed this trend [14]. Thus, with unpredictable interdependence among the tasks, partly because of the processor, the WCET computation really is a huge problem.

If it can be proven, apart from simplifications or other assumptions, that the tasks can all meet their deadlines by some scheduler then the system is said to be schedulable. However, a dynamic scheduler may fail to ensure that all tasks meet their deadlines even with a schedulability proof. For instance the scheduler, earliest deadline first (EDF), suffers from a phenomenon referred to as "Thrashing" [17, 18]. This is when the scheduler always selects the task with the earliest deadline among two or more tasks when they both have similar or nearly similar deadlines, preempting the currently running task. Due to overhead from switching between the tasks, this causes the processor to spend more time alternating between tasks than doing useful work. The simple solution is to not pre-empt and the harder solution is to not make wrong assumptions about a preempting with no overhead when switching tasks.

Liu and Layland Classification of Real Time Systems

Usually, real time is being classified into the three main branches hard, firm and soft [10]. (Sometimes with firm considered as a subclass of soft, as noted by Bøgholm in his thesis [85]. Or a hybrid between hard and soft, as noted by Kaldewey et al., in their paper on Firm Real-Time Processing [12].)

Hard

According to Liu and Layland a hard real time implies that a single missed deadline results in failure of the entire system [1]. Arguably, this is rarely the case for most systems. Many tasks can be separated into a mandatory part and an optional part in which only the mandatory part has to be executed in order to ensure the minimum functionality of the entire system [2, 3]. Another case where this assumption fails to hold can be seen in a simple controller system. The controller consists of three systems: Input data (e.g. sensors), calculations (e.g. MPC) and output data (e.g. engine). If one part is cut out of the equation the entire system will fail, thus the systems consist only of mandatory tasks. Nonetheless, the system will still work if one misses a single sample from the input data or previous output data is used instead. The system may be indifferent whether this is caused by calculation in the controller being unfinished or output data being at fault. In sum, the conclusion is that the system is "somewhat" hard real time because a deadline miss at the wrong time causes the system to fail, but a deadline miss will not necessarily bring down the entire system. In fact most of the deadlines are firm, but the next deadline for each task essentially becomes "harder" for every missed deadline.

Firm

Although some authors, such as T. Kaldewey et al. and T. Bøgholm, list firm real time as a subclass of soft real time it may very well be thought of as an independent class of real time systems [12, 85]. It differs from the hard type due to the possibility of missing a deadline without fatal consequences and it differs from the soft type because the results are always worthless after the deadline has passed. The best known use of firm real time is in sound systems where missing a note is, sometimes, noticeable, but is no disaster and where a note that isn't on time should never be played at all.

Another notion from firm real time is the use of (m,k)-firm deadlines, where specifying the two constants k and m such that the quality of service (QoS) is acceptable. That is, as long as at least m instances in any window of k consecutive tasks meet their deadlines, then the QoS is acceptable [3, 63].

Soft

Soft real time is the class of real time that can tolerate the greatest number of deadline misses. Although the QoS will usually diminish as more and more deadlines are missed, it is not always the case. What is important is how they are missed. Simply put, if a video misses every 25 frame nobody is likely to notice, but if 25 frames are missed in a row then the result is clearly visible. In other words, deadline priorities might be adjusted dynamically to enhance the performance of the system and this will, if executed correctly, result in a higher QoS.

The value of a result in a soft real time may diminish with time as deadline is passed or it may simply be useless, but this varies from system to system. One solution is to make the system such that it has bounded lateness, as proposed by Valente and Lipari in their paper, which makes late results nearly as valuable as if they were on time [11]. And finally, if some deadlines cannot be within that time, only then are they useless.

Stochastic Real Time

A new "revolution" came when Jensen published his paper on time driven scheduling in 1985 [84]. By considering the value of a task's completion this method allows one to design systems in which priority is based on the actual value of a task. This is in contrast to the classic approach in which the priority is more or less static determined before execution. Further work, in particular by Jensen, introduced the Time/Utility function in which the stochastic real time approach is based on [39].

Rather than requiring that all deadlines are met, the stochastic approach requires that at least x % of all deadlines are met. The complication here is that it is not indifferent what deadlines are missed. If one task misses all its deadlines, but the overall system still performs at x % of all deadlines then, although depending on the criticality of that task, failure is the result. This also led to more research in scheduling algorithms and one proposed solution is to use feedback scheduling.

Petri Net and FPGA

The Petri Net method is a modeling system based on graphical/mathematical reasoning. This is in many ways one layer above the task model as the task model can be used to implement a Petri net, but it is rarely practical. Usually the Petri net results in FPGA based systems which in many ways are more predictive than the classic approach of buying "off-the-shelf" hardware and make the most of it. In theory all operations of a FPGA can be traced and predicted thus resulting in a more predictable system than classic real time.

Scope of Work

The main focus of the paper will be about the task model, but will still cover other approaches to real time. Safety-critical systems will also be covered in greater depths than systems that can be designed as best-effort, which is not real time. This includes several of the approaches to media-streaming in which the soft real time approach is too similar to the best-effort approach. A typical example is streaming of television programs on tablets in which the live coverage appearing on the tablet is behind what is happening on the television.

Topics that will be covered in this paper include work that is relevant to nearly all approaches to real time such as control systems, WCET, Safety-critical systems and stochastic methods. Further topics consist of various approaches to real time and how to achieve the wanted results. Topics such as FPGA, belonging to electronics, are not covered in such depths as main topics of this field.

Outlines to the Alternatives of Classic Real Time

If a hard real time system fails to deliver a correct result on time it is considered to be a full system failure which cannot be recovered, but if it delivers a wrong result on time then this is just a fault that can be recovered with ordinary fault mechanism. There is however no literature that supports this reasoning, both are errors that must be dealt with nonetheless. The assumption does not hold, as can be seen in the paper by Fontanelli et al. [72] where they refer to the work done by Kopetz et al., on the Time Triggered Model of Computation [86] concluding that a job/task can be cancelled and if necessary use the previous value instead.

Because there is no difference between the faults and both may lead to errors, the solution is nevertheless to fix them. For a fault consisting of a wrong result within the deadline standard measures are undertaken. However, if the fault is of a missed deadline there exist two options: One is to extend the deadline temporarily [11] and let the result arrive at the new deadline, the other option is to recover the fault as if it was a wrong answer within the deadline. The fault recovery should adjust to ensure that the fault is recovered in the best way. If the deadline is missed it is not always possible to just rerun the calculation and get it done within the deadline the second time. This will be dependent on whether the load of the system was at fault or if it was the load of this single application. Therefore it is sometimes better to extend the deadline instead of rerunning the calculation as this is essentially the same solution, but with a smaller run time, although it will fail if the late calculation is wrong in addition to being late.

Prediction of running times should be possible if an application is built to support this, as advocated[1] by Haugli and Hendseth in a thesis and presentation, respectively [25, 26]. By utilizing this possibility it is thus possible to predict whenever a deadline won't be met. It is

[1] The proposal was estimation of WCET, but the principle is the same.

therefore possible to recover from the fault before it actually occurs and in fact prevent it from ever happening. A viable option could for instance be overclocking the processor or extend the deadline if possible. Nonetheless, the important thing is the overhead of the predictor.

Method

The first part of this project consisted of reading trough articles published with the keywords "Real Time." Further reading included symposiums and other papers that provided an overview on the topic of real time. As there are many more articles on the classic approach to real time than on the other fields within real time, the classic approach became the main reading at first. Then to understand how to deviate from classic real time it was even more important to fully understand classic real time to explain its strengths and weaknesses. Papers describing general concepts on real time were read simply for depths on the subject and some of the topics are omitted in this paper.

In order to ensure that all relevant data on a subject is found there are a few methods that have proven useful. Google (Scholar) search of relevant keywords is the easiest when it comes to mainstream or other well known articles, similar with Microsoft Academic Search. Google Books has been used to skim through books with titles that suggest they are relevant to the subject at hand. Publishing cites such as IEEE and dl.acm.org are thoroughly searched. Several symposiums have been particularly useful to get an overview of certain topics, this includes RTSS and RTAS. Further findings have come from utilizing citeseerx.com to find which articles that has cited the paper which proved to be relevant. Finally all papers do have their references listed which in turn may have more information about the topic and therefore their abstracts are read through in order to find any more relevant information and, if it is relevant, the entire paper.

Also, many articles contain a line of keywords. This in particular makes searches for similar papers easier. Further a dictionary is used to find synonyms in case authors use different words to mean the same thing. Which isn't that uncommon as Oxford English differs from American English and especially authors that do not have English as their mother tongue. This means that a search centered on "online prediction of running time" also must be run as "online estimation of running time," "online estimation of execution time" and "online prediction of execution time" just to be sure.

Whenever a claim is made as to whether something is new and/or revolutionary it can usually be put to test by simply counting references to the article. Therefore there is no doubt that Liu and Layland really revolutionized the field of real time when they published their paper. [1] Articles that are several years old and barely cited usually oversell their publication when claiming that it is great (and the world couldn't live without it). The tricky part is new publications, but comparing it to previous written articles found by using the same keywords in the search usually reveals whether or not it is brand new.

To give some insight into whether a paper is "famous" or not, the citation count of every paper, if found, is given in the bibliography. (For introductory papers this tells a lot about whether it is well written and easy to understand or not.) Newer papers usually have fewer

references than old ones, meaning that old papers with only a handful citations are probably less useful. Nonetheless, that does not necessarily mean that the idea in the paper is not useful. Therefore papers with similar ideas or papers that refute the idea must be found and compared as to whether the idea has any merit or not. Also, the total number of citation may not necessarily be accurate as IEEE only lists citing by other IEEE publications whereas Google Scholar list several more. I.E.: [39] is listed with 18 citations on IEEE, but with 81 on Scholar. Thus, any citation listings may be somewhat inaccurate, but close enough to get a fair picture. In fact, several of the most cited papers in the reference list have received more than a handful new citations during the writing of this paper.

Another useful cue is the author(s) behind the article. Famous authors usually rely on their reputation alone without overselling their ideas. However, being famous usually helps you getting published in the first place and therefore it isn’t necessary a good paper just because of the author. Nonetheless, they are not likely to put their reputation at stake for some ludicrous paper, which means that its content probably has roots in the real world. Finally, some departments do have the same author listed as the last name on virtually every paper for funding reasons, which means that the author(s) famousness is not necessarily the best way to measure the value of a paper.

One challenging task is to judge whether a paper present useful information for this paper or not. Much information will only prove to be relevant after several papers have been collected to give new insight as to what the method could imply. For instance was the paper on simplex design [22] not that relevant until coupled with the paper on the continuous stream task model [72]. This is the relevance paradox [74].

A part of the methodology is dividing real time into the correct classification. Is the classic approach of hard, soft and firm sufficient or should it rather be classified differently? An approach in which also the best-effort method is included may be more viable as many soft real time systems don’t differ that much from best-effort.

Google Scholar also has the authors listed when you search for an article and you can therefore look them up and see all publications listed by Google Scholar for each author. There, every article listed is sorted by the number of citations meaning you’ll get a fair impression of their famousness within their field.

Keywords are listed in the beginning of every chapter and are only listed for background chapters or result chapters that are not covered in the background chapters.

Preliminaries

Historical background

When Liu and Layland presented their infamous paper in 1973 this represented a revolution in the real time science [1]. Earlier, the real time systems were scheduled by cyclic executives; created in a more or less ad hoc based manner. Thus neither schedulability nor safety-criticality could be assured. However, the static scheduling paradigm was not without its flaws and the assumption that all tasks are independent does not hold, same goes for the assumption that all tasks are periodic. Still, further work introduced fixed priority scheduling which solved the initial problem of priority inversion by defining the priority inheritance protocol (PIP), setting a blocking task's priority to that of the blocked task. Nonetheless, this did not solve the deadlock problem [83].

The main problem with the Liu and Layland method is the lack of support to non-periodic tasks. Several methods have been proposed, but they are rather ad hoc, resulting in history repeating itself, as this was already the method of choice for real time systems pre Liu and Layland. All methods proposed to solve this problem suffer from wrongful assumptions such as release time can be known in advance, WCET can be known in advance etc. Thus other approaches are necessary.

In 1985 Jensen introduced the time driven approach which culminated in Time/Utility Function (TUF). This represents the first main deviation from Liu and Layland. The paradigm of scheduling proofs still lives on, but as more research are put into TUFs it has been shown that the use of stochastic real time systems provide a feasible alternative to the classic approach.

Today

Still, most research is put into the classic Liu and Layland approach. The schedulability proofs are being adjusted so that they take into account the overhead of the scheduler when computing schedulability, WCET are still pursued and alternatives are not pursued in greater depths as is evident from the lower citation count of such articles.

Safety-Critical Systems

Keywords: Safety, criticality, hardware, software, airplanes, nuclear plants.

Hardware and Software

Hardware-based and software-based systems have traditionally been about making the choice of the lesser evils as Bate put it in his doctoral thesis [4]. While hardware is somewhat safer than software, but more costly; the software-based approach is much more flexible, but at the cost of less predictability. With classic real time and the use of schedulability proofs this is becoming increasingly more difficult as modern processors are focused on decreasing average execution time at the cost of predictability and time of the worst case scenario [14]. For simple systems such as anti-lock braking system (ABS) there is four components: Speed sensors, valves, a pump and a controller which through design is a hardware-based system. However, with more complexity such as autopilot and other crucial parts of an airplane, it might be necessary to use software in order to manage the system as a whole. Nonetheless, as noted by I. J. Bate: “there are a number of problems with hardware solution including; the cost of producing systems is high, the hardware that provides the control is large and heavy and the hardware has a finite slew rate performance that limits the system’s responsiveness.” [4].

Thus we turn to software for complex safety-critical systems that are likely to be upgraded or otherwise altered during their lifetime. However, this does not solve the issue associated with classic hard real time. The schedulability proof would still be necessary for every single change done to the system. That is, as long as it is not performed on an isolated part.

One notable issue with most approaches is that the scheduler has to check whether or not the deadline is missed. The solution suggested by the Sloth-project, currently running in Germany, is to use hardware interrupts only when the deadline is missed [73]. That is, for any task that runs to its completion, the deadline interrupt is disabled.

Achieving Safety Criticality

The “simple” method is to examine every possible way the system can fail and prevent it from happening. In practice this is rarely possible apart from really simple systems. Then again, nothing really is foolproof[2]. However, this is the approach used in classic real time. By giving a schedulability proof of the task set then it is proven that all tasks will execute at the right time without failing a deadline. Nonetheless, this is more theoretical than practical as uncertainty in the environments can give unexpected situations. Should for instance an overload occur, then the EDF- and RM-scheduler will fail, resulting in total system failure.

[2] ”A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools." (Douglas Adams.)

To achieve criticality safety there are other approaches to real time apart from the Liu and Layland task model. For instance are Lustre [15] and Esterel [16, 64] programming languages that solves this in a different way than Ada or Real-Time Java, yet all languages could, in theory, be used (see next section). The object is to find the right language for the right task, but the overall goal is to find one method for all systems. Seeing as reactive systems[3] are the most efficient way to achieve critical safety, in particular when it comes to systems with some degree of uncertainty this approach will be the main focus of this paper. Only if the environment is entirely predictable can one design a system that is proactive rather than reactive.

The Answer is Real Time

The overall goal is to create a system that interacts with its environment in a timely manner, whenever the environments changes the system has to respond. This applies to everything from an airplane to more simple systems such as a simple controller. Sometimes the environment is nearly static and the system is not made to deal with the environments, but only its own functionality. Examples include media players and similar simple systems. These systems are built to deliver a QoS and do not need to be real time, but are usually designed this way nonetheless. However, they could easily be designed as best effort. The overall goal is to deliver frames and audio without any observable loss.

For safety-critical systems the solution of best effort is not a viable solution. “As fast as possible” is not deterministic enough for use in practical applications when it can be hazardous to fail. The solution is to change the constraint into “within timely manner,” where the difference is that; as soon as possible is not optimal, whereas the timely manner is. An example can reveal this: 2 jobs must be executed, but job 1 takes longer time if executed first. Hence, the as soon as possible approach will execute 2 then 1, but if job 1 is critical it must be executed first, therefore the timely manner approach is the only viable option.

Airplanes and Other Safety-Critical Systems

Nothing serves as clearer examples than airplanes and nuclear plants when it comes to safety-critical systems. One small fault will inevitably occur, but if that tiny fault manages to propagate throughout the system and bring it down, the results would be disastrous. The popular show from National Geographic “Mayday” or “Air Crash Investigations” often point out one failure in equipment, but also point out how many other things that have failed for the faulty equipment to bring down the plane. Of course, there are exceptions, notably the incident where a plane was loaded with oxygen tanks that exploded mid flight and the resulting fire brought down the plane.

[3] A reactive system is a system that reacts to events, as opposed to a proactive system that reacts in advance.

In airplanes the system is built up by constructing several smaller modules that should be capable of operation whether or not the other modules are functioning. However, this is not easy to achieve in practice, as is evident from the recent order from Federal Aviation Administration (FAA) in which the Boeing 787 computer can overflow and cut down the entire electric system [88]. This illustrates the current problem with real time systems, although the Liu and Layland approach is better than the previous technique (ad hoc) it is not a viable approach for practical purposes.

Guaranteeing safety-criticality is practically impossible, but by proper design of error handling and recovery it is possible to make a system that is very unlikely to fail. This will also require maintenance plans as all systems are subjected to aging and the resulting wear and tear. And as for safety-critical systems, the technique of smaller less complex modules is currently the approach used in airplane design [89, 90].

Approaches to Reactive Systems

Keywords: Reactive system, synchronous system, Lustre, Esterel, Liu and Layland, the task model, communicating sequential processes, petri-net, FPGA, Ada.

Preliminaries

A reactive system in computing science refers to a system that reacts to events, both internal and external. However a search on Google Scholar reveals no precise definition. A suggestion by A. Benveniste and G. Berry is that a reactive system is defined as “a system that maintains a permanent interaction with its environment” [101]. This appears to be a fitting definition for computer science on reactive systems.

In addition to the task based model there is also a few other approaches to reactive systems. Many of them fall within the realm of real time. Operating systems (OS) are not real time, and are beyond the scope of this paper.

The commonalities between these approaches are that they include concurrency, they are submitted to strict timing requirements, they are in general deterministic, their reliability is particularly important and they are made partly out of soft- and hardware [65].

Classic approaches include petri-net-based models, the task model and communicating processes. The synchronous approach is to handle events with respect to order and is designed for complete determinism [65].

Synchronous Approaches to Reactive Programming: E.G.: Lustre and Esterel

The basic model of Esterel/Lustre and other synchronous languages is the reactive model in which we consider systems that interact with their environment continuously[4] [16, 64]. This is based on the basic assumption of synchronous languages: The “perfect synchrony hypothesis” which states that reactions are instantaneous so that activations and productions of output are synchronous. This is the main idea of the synchronous approach. However, this idealized hypothesis can be rendered more practical by assuming reactions are atomic, thereby moving from theory into the world of practical applications. This is also a necessary assumption for synchronous languages to have deterministic parallelism [64]. Nonetheless, the assumption that all reactions are instantaneous is just a theory. Yes, it does seem to have the same issue as classic real time with theory vs. practice, but it is possible to move the theory into practice by loosen the assumption of perfect synchronization a little.

[4] In theory. In practice, all discrete systems are, by definition, not continuous.

In synchronous languages time is considered an external event, as opposed to other parallel languages such as Ada, where time plays a specific role [65]. Nonetheless, timing requirements are still met.

One drawback with the synchronous approach is that where asynchronous languages only have deadlocks, the synchronous languages have causality problems too. Although most of them are noticed by compilers they may still be an issue. Causality problems can be seen as instantaneous deadlocks [64].

The Task Based Model

The task based model which was revolutionized by Liu and Layland in [1] is the most known method for thinking and programming real time. The method consists of dividing job into smaller part called tasks, giving them priority and schedule them on the processor. For classic hard real time systems this includes a schedulability proof [1], whereas for soft and firm real time systems this usually only requires a rough estimate. As long as it runs with a decent QoS everybody is happy. In other words, if a soft or firm system appears flawless for the end user, it is a good system. This is not the case for a classic hard real time system in which the end user also will want a guarantee that it will continue to operate flawlessly for the future. (If a TV breaks you send it back to the store, if your ABS brakes fail on your car …)

Communicating Sequential Processes (E.G.: Ada and Occam)

Another type of reactive systems is Communicating Sequential Processes (CSP). This is a type of reactive systems that also reacts with its environment, but also interacts with itself.

Rather than a shared memory this method consists of using synchronization, such as rendezvous, and communication[5] [64]. A communicating process communicates with it environments in some alphabet of atomic communications or events [67]. Several of them use the task model in combination with communicating processes [68]. What separates Ada from the classic task model is the departure from the cyclical executive model [69]. The dynamic preemption at runtime generates non-deterministic timelines that are contrary to the idea of fixed execution timeline. However, this non-determinism does not result in impossibility to schedule and predict if the scheduling holds. Keeping the utility bound below a given threshold bound will ensure that all tasks meet their deadlines [69]. The main difference between CSP and the task based model can be illustrated as follows: In Posix you use permanent marker on the board and hope no one erases it before everyone has seen it and in CSP you gather everyone and tell them what you would have written on the board.

For more on CSP refer to the book by C. A. R. Hoare [70].

[5] Thus it is not possible to have one variable changed between the access of the first task and the last, in contrast to shared memory.

The Petri Net-Based Model

Invented by Carl Adam Petri in 1962 as a model for describing information flow, this method has proven versatile in visualization and analyzing behavior in concurrent asynchronous systems [94]. By describing all reactions of a system, this is a versatile and powerful model for creating reactive systems.

This method is a mathematical and graphical modeling tool which is capable of handling both deterministic and stochastic environments [66]. It is designed to be a tool both for practitioners and theoreticians which will enhance the learning between them. However, Petri nets still need a scheduler [71], which gives them much of the same problems as the classic task model. Although, for modeling finite state machines (FSM) the Petri net-based model has proven to be very useful [66]. The usefulness on very complex systems diminishes as the graphical representation is hard to follow as complexity increases [66].

For implementing a Petri net the simplest way is using FPGA, especially if the implementation should be on a small chip [76, 77]. However, the implementation is not as straightforward as some other methods [78]. This is the main drawback with the method and is why other methods still exist. In addition to FPGA there also exist methods for generating C- or Java-code from Petri nets [78, 79].

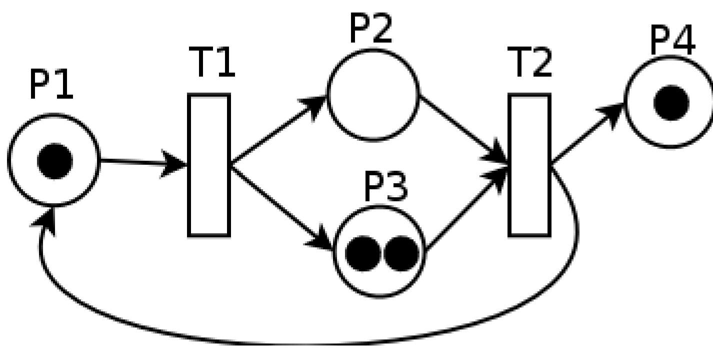

Figure 1: A Petri net is modeled as a directed bipartite graph in which the circular nodes are states; the squares are transitions and arrows signals direction of flow.

For more on Petri nets refer to these papers: [66, 71, 76, 77 and 78]

Scheduling

Keywords: Scheduling, earliest deadline first, EDF, RM, rate monotonic, task, Liu and Layland, preemption, processor, overhead, static scheduler, dynamic scheduler.

Preliminaries

The most common practice in real time system design is to divide jobs into smaller part called tasks [1]. Some tasks are divided further into necessary and optional parts [2, 3]. For a system to execute effectively a scheduler is employed to sort the execution of tasks into an as efficient manner as possible.

Real time scheduling falls into two categories, static and dynamic. Further, the dynamic scheduler is divided into two domains, resource sufficient and resource insufficient environments. Schedulers such as the EDF is optimal in resource sufficient environments, but not in resource insufficient environments [31]. Using admission control, then it is possible to ensure sufficient resources. However, the cost is leaving tasks, not granted admission, starving. That is, as long as there isn't another processor where they can execute.

Scheduling Strategies

Schedulers such as rate monotonic (RM) and earliest deadline first (EDF) are "open loop", meaning they are not adjusted once they have been implemented. Although "open loop" schedulers work well in predictable environments they are inferior to "closed loop" schedulers, especially in unpredictable environments [31, 62]. Thus "closing" the loop creates the class of adaptive schedulers which can be utilized where the "open loop" schedulers are insufficient.

Preemption and Overhead

When a task is stopped during execution to let another task run the currently running task is preempted. Saving information about the task in the cache or other memory banks lets the task continue as nothing has happened when it is given runtime on the processor again. However, time still passes and its data may have changed in the meantime. This is the main setback of preemption. Another issue is the overhead. When interrupted, the task has to clean up states and save its data to the memory bank. Further, it has to fetch the data from the memory after it regains processor time. This takes time and is called overhead. Another source of overhead in schedulers is the time the scheduler takes to compute priority of the task. Schedulers such as Round Robin will have negligible overhead for priority computation as it is fixed, but suffers from large overheads by preempting very often.

"Open Loop"

In predictable environments the "open loop" schedulers are usually more efficient as they have a lower overhead compared to the "closed loop" [31]. However, things are far more complicated when preemption is present [60].

Static

Static schedulers are characterized by being deterministic. All scheduling is computed offline and then implemented on the scheduler.

The RM scheduler is optimal, meaning that if any static scheduler can schedule a given set of task then the RM can too. Further it is predictive which may be utilized to give a schedulability proof. This allows the RM scheduler to be used in safety-critical environments as its deadlines can be guaranteed [1]. The drawbacks are that every task must be periodic, known in advance and independent of all other task. Finally, its utilization is below ln 2 for an infinite number of processors and 0.8284 for two processors, but this is only a sufficient bound, not a necessary [1].

Dynamic

Dynamic schedulers differ from static by being able to adapt to the situation. If a task completes faster (or slower) than expected, then the scheduler can schedule another job to the processor where the static scheduler would just have been idle.

The most well known dynamic scheduler is EDF. It is simple and also optimal, meaning any task set schedulable for a dynamic scheduler is also schedulable on EDF. It is however not optimal on a multiprocessor system [61].

The overhead of dynamic schedulers may be less than static schedulers because of the optimization, which gives less preempting, thus resulting in the EDF scheduler having a smaller overhead than the RM scheduler [60].

Closed Loop

Feedback Scheduling (FBS)

This class of schedulers opens a whole new range of possibilities including the use of stochastic real time systems, which will be examined in detail in a later section. By utilizing the possibility of using a dynamic closed loop controller it is possible for the scheduler and the controller to interact. If a scheduler has excessive computation time available, this can be given to the controller, giving it more computation power for the controller algorithm and similar when the scheduler has too little time to give to the tasks. The controller adjusts its parameters in order to compensate the lack of computation time so that it prevents system failure. By letting the scheduler tell the application, in this case the controller, how much it computation time it may expect, the possibilities for system adjustments becomes wider than before.

FBS makes it possible to handle unpredictable workloads and environments. Further, the use of Feedback Control Real Time Scheduling (FCS) can provide deadline miss ratio guarantees and CPU utilization guarantees [62]. Still, with the addition of online prediction of running times, this can be utilized for even further optimization.

“The core of any FCS algorithm is a feedback control loop that periodically monitors and controls its controlled variables by adjusting a QoS parameter (e.g., task rate)“ [62]. Controlled variables may vary from system, but preferable candidates are deadline miss ratio and CPU utilization [62]. This has resulted in the algorithms FC-U, FC-M and FC-UM for the control of utilization, deadline misses and the combination of utilization and deadline misses, respectively. In her paper, S. Lin suggested that FBS can be mixed with (m, k)-firm systems to further optimize such a system and shows that the QoS gain is significant [63].

Summary

Static schedulers are useful for deterministic classic real time systems, but are far inferior whenever there is a chance of randomness. Dynamic schedulers are required when the environment is somewhat unpredictable and closed loop controllers are necessary if there are insufficient resources.

If more information about scheduling is required, refer to this paper by Davis and Burns. [59]

Dynamic Voltage Scaling (DVS)

Keywords: Dynamic voltage scaling, DVS, processor, CMOS, overclocking, battery powered systems, supply voltage.

Background

Dynamic voltage scaling has become a major research area in the last decades [5]. This is because the processor can use, from an average of, 18-30% [6] of the power consumption and even exceed 50% [7, 8] for CPU intensive workloads. According to H. Aydin's paper, the power consumption of an on-chip system is a strictly increasing convex function of the supply voltage (V_{dd}) [5]. For the commonly used Complementary Metal–Oxide–Semiconductor (CMOS) technology, energy consumption is dominated by the dynamic power dissipation P_d, given by[6]:

$$P_d = C_{eff} * V_{dd}^2 * f$$

The gate delay (D), which dictates the speed of the system, is inversely related to the supply voltage (V_{dd}) by the following formula:

$$D = k * \frac{\boldsymbol{V}dd}{(\boldsymbol{V}dd - \boldsymbol{V}t)^2}$$

Here, k is a constant based on the electric characteristics of the chip and V_t is the threshold voltage [5]. From these two equations it is deducible that one can save power by reducing the supply voltage and the clock frequency simultaneously. This is dynamic voltage scaling.

Finally it should be noted that there is a threshold as to how much one can scale the processor down in systems with non-negligible static power. This is called the critical frequency and occurs because the static energy consumption grows with longer execution times when running at lower frequencies, dominating the energy consumption [34].

Hard Real Time

For all hard real time system designed in compliance with Liu and Layland [1] there is likely to be a gap between the WCET and the actual execution time. This time can be reclaimed in several ways, usually running best effort tasks or soft tasks [32]. Another option is to simply adjust the processor such that it does not run unnecessarily. All the tasks' periods are supposedly known in advance in a hard real time system based on Liu and Layland. Thus it is also known how much slack[7] the given task has after its completion, making it far easier to

[6] C_{eff} is the effective switched capacitance and f is the clock frequency

[7] Slack time: $s = d - t$, where s is the slack, d is the deadline and t is the actual time elapsed when the computation is done. In other words, the time allocated to a task that is not used by the task.

reclaim this slack trough DVS. However, even if the hard real time system is not in compliance with the Liu and Layland method, there is a possibility for slack in the system.

As noted by H. Tang et al., many hard real time systems are of a mixed criticality and therefore it is particularly useful to let the soft (aperiodic) tasks use the processor only when the hard tasks let the processor stay idle [33].

Another option is simply to shut down the processor whenever a hard task completes, or clock it down when it is about to complete before its deadline, thus saving energy [34, 35, 36].

Soft and Firm Real Time

Anything that works on hard real time also works on firm and soft systems. Similarly, anything that works on a firm system will work on a soft system. However, for any system with utility larger than one, there is not possible to use only classic hard real time. Mixed criticality is possible though, as long as the "hard" part is below the utility constraint.

For any soft or firm system with utility above one, there is not necessarily any gain in reducing the processor clock as there always is a task ready to execute. Obviously there is a difference here between battery powered systems and those directly connected to an "everlasting" power source. And to further complicate things, there are even rechargeable systems. These systems pose an interesting dilemma: Should they be power neutral and only consume the energy whenever they accumulate it, optimize with respect to deadlines at the risk of running out of power from time to time, or always have enough power leftover such that any worst case scenario is covered. M. Chetto and H. Ghor introduced a scheduling algorithm that covers partly this dilemma [37]. Finally, resolving this dilemma is not possible without knowing anything about the application. For a stabilizing controller, the important thing is to always run, thus preventing instability. For a defibrillator the important thing is to provide a good jolt within the first minutes [38]. For a cell phone or tablet, the main task is to provide good service and last as long as possible. With three different goals it is up to the system designer to solve this problem as there is currently no framework covering all three, somewhat contradictory, goals.

One solution for a framework is to adapt the existing time/utility functions (TUF) so that it also includes power consumption/preservation [39].

Battery Powered Systems

Battery powered system can be separated into two groups. Those with more or less continuous recharging (solar panels etc.) and those who can only be recharged when connected to a power outlet (tablets etc.).

For devices only rechargeable by wire there is not much besides making most of the power available at all times and thus it is essentially only an energy preservation problem. However, for devices with more or less continuous recharging there is also the question of predicting how much power can be gained and when. At times this is fairly easy (solar panel gain on a roof can be roughly estimated by reading the weather forecast) and sometimes it is significantly harder.

Overclocking

Overclocking is a procedure in which the operating frequency is increased beyond manufacturer specified frequency limits for reliable operation, without changing the system supply voltage [40, 41]. The main issue with overclocking is overheating and the resulting hardware failure. A simple way to resolve this is additional cooling, but this requires more hardware and for many embedded systems in particular, it is not a viable alternative.

Hardware manufacturers have introduced technology that allows overclocking [42], but still it is safer and more reliable to run the clock at normal and subnormal speeds [41]. As V. Subramanian et al. suggests in their paper, it is important to overclock the system reliably, thereby making the average case faster [43].

A common method used in overclocking is "Timing Speculation" which is to increase frequency (at constant voltage) and correct the resulting faults [44]. At best there are no timing errors, but to make the system reliable there has to be fault correction to detect and recover from timing errors [45].

The major setback with overclocking is that it is unreliable, both in the short run and the long run. This is a solvable problem though, as it is possible to run the processor at lower or ordinary speed to let it cool down and thus avoid the long term issues. As for the short term issues, these can be avoided by use of the fault correction methods mentioned in [41, 43, 45 and 46]. Therefore, the use of overclocking is a viable alternative for shorter periods of time and in particular whenever the processor is overloaded.

Concluding remarks on DVS

The dynamic voltage scaling technique is useful for all software systems. For soft and firm real time systems the use of DVS is clearly advantageous as it can be utilized without missing any more deadlines than without DVS. Nonetheless, it should be noted that when using a stochastic approach where the deadlines should be met by a given percentage, then it is possible to use DVS, thus getting closer to the bound. Consider a system with 90% successful deadlines as a minimum requirement: It is possible to save much power if it is actually averaging 95%.

Under hard real time system with proof of schedulability it is easy to use DVS, because the entire system is more or less deterministic. As indicated in [5] it is possible to use DVS to

save energy by reducing any unused processor time whenever the task completes before its worst case scenario.

Overclocking is also viable, but mainly for shorter periods of time. For a hard real time system the main point is to still keep the system reliable such that no errors occur and to counter the eventual design flaw where the processor suddenly is overloaded. In theory this should never happen [1], but this is only true when *everything* is known about the system *a priori*.

With soft and firm real time systems the use of overclocking poses the same problems as with hard real time systems. However, it is possible to adapt a time utility function so that the system only runs overclocked when it is necessary or beneficial. Nonetheless, extensive knowledge of the risk associated with overclocking is necessary to construct this TUF with overclocking. In sum, the objective of overclocking is to balance the trade-offs between reliability and speed without compromising the system's overall performance [46].

Real Time and Control Systems

Keywords: Control systems, real time control, jitter, timing, sampling, delay, continuous stream task model, simplex design.

Background

Control systems are used everywhere in today's modern world and in particular in process plants. Designing and verifying the controller is a, usually, costly and rigorous task. Several problems arise when the complexity of the controller exceeds a certain threshold. Time delay is one of the biggest issues:

"It is well-known that the manifestation of time delays in a system can lead to performance degradation and even destabilization of the system" [19]. Clearly time delays are not wanted in control systems, but it is hard to avoid them altogether [47]. Thus several attempts to fix or eliminate the problem have been tried out with varying success.

Another crucial point in control systems is the interaction between time and control. When designing a control system the decisions made with respect to timing will afflict control and *vice versa*. In their paper, A. Cervin et al. suggest using mathematical modeling tools, such as jitterbug, to analyze the sensivity of jitter and other timing problems [99].

Timing in Control Systems

"In control theory, sampling and actuation are generally considered synchronous and periodic, and a highly deterministic timing in task execution is assumed" [20]. However, this assumption does not hold in practice [19, 20, 23].

Jitter

"Jitter is the deviation from true periodicity of a presumed periodic signal" [21]. Or in other words; the timing error. There are only two options here: Either remove the jitter or compensate for it. Finally, when the scheduling algorithm is fulfilling the stringent timing constraints control theory mandates, the result is often poor schedulability. [20]

Jitter mainly arises because of the following reasons [20]:

Sampling Jitter

"Time intervals between consecutive sampling points may not be constant."

Sampling-Actuation Delays

"Even if sampling occurs at regular intervals, there could be a delay between when a sample arrives and when the actuation response occurs after the completion of the control computation. This can be due to start-time delays in the control computations."

Sampling Jitter and Sampling-Actuation Delays

"This is the combination of the previous two problems and is caused by varying sampling intervals, delays in the start of control computations, non-negligible execution times, and preemptions during the control computations, which in turn can lead to variable actuation times."

Jitter Compensation

Seeing as jitter is virtually impossible to eliminate [19, 20, 23] it is necessary to compensate for it in another way. One possible option is to model the jitter as a stochastic variable.

L. Hetel et al. proposed that: "… continuous time systems with uncertain time varying feedback delays can be expressed as a problem of stabilizability for uncertain systems with polytopic[8] uncertainties." [23] Thereby compensating jitter by modeling it as a stochastic variable rather than requiring it to be known. In this case the controller can be obtained via linear matrix inequalities. [23] If the delay is known at all instances then there exists simpler ways to compensate for it. [24] Although the model in [23] is a switched model it can be reduced to a LTI[9] system and thus cover most control systems. The weakness of this method is the requirement that the delay is shorter than the sampling frequency. This will require that the worst case execution time is known.

"Time-delay robustness is often studied for situations in which the delay is uncertain but remains constant throughout time" [19]. However, it is rarely the case that the delay, and hence jitter, is constant. Systems with time-varying delays are also less stable to than those with constant delays, requiring more from the controller stabilizing the former system. Unfortunately, the delay parameter proposed in [19] is bounded, but in any real system there is the possibility of having a long, unexpected delay. In practice this does happen from time to time, but usually in opera house constructions[10] rather than computer systems. By using the Integral Quadratic Constraint (ICQ) or equivalently Semi-Definite Programs (SDP), Chung-Yao Kao and Anders Rantzer proved that a stochastic modeling of the jitter is feasible [19].

According to the paper by T. Phatrapornnant it is necessary to have hardware support whenever minimal jitter is required [28]. For instance is the hardware systems in Sloth a viable option [73]. The question is then dependent on the application at hand, whether jitter must be controlled and compensated or minimized altogether. One example: If one uses the successive-approximation ADC then it is not surprising to find that different input values have different processing times. Take for instance an 8V signal into a 0-16V ADC: One step is

---

[8] Polytopic: Of or pertaining to a polytope.
In elementary geometry, a polytope is a geometric object with flat sides, and may exist in any general number of dimensions n as an n-dimensional polytope or n-polytope. For example a two-dimensional polygon is a 2-polytope and a three-dimensional polyhedron is a 3-polytope. [48]

[9] Linear Time Invariant

[10] http://www.eoi.es/blogs/cristinagarcia-ochoa/2012/01/14/the-sidney-opera-house-construction-a-case-of-project-management-failure/

required as the comparator starts at the midpoint, thereby requiring shorter time than a 10V signal. Thus the hardware and software must be designed specifically to prevent this type of jitter. One solution is to mark the start of the conversation and read the output after a given time, which ensures synchronization for any given input signal. However, for many actuators this approach may be impossible. A motor for instance, will not go from 1000 rpm to 5000 rpm as fast as from 4000 rpm to 5000 rpm. We then, again, have a case where the time delay is not constant and may vary in a somewhat unpredictable manner.

The upper bound, Ji, on the relative jitter in the start time of two successive jobs when Ti=Di is given as:

$$Ji = 2(1 - \frac{Ci}{Ti})$$

Resulting in an upper bound of 200% when Ci<<Ti. Therefore jitter could pose a serious issue if the controller is not robust. Finally, the jitter will be lowest for the highest priority task [98].

Model of Computation: Continuous Stream Task Model

One approach to real time control systems is the continuous stream task model. The key advantage to this model is the capability of analyzing probabilistic evolution of delays. For many tasks the input data, which is not necessarily known *a priori*, fluctuates a lot. The difference between WCET and best case may be in magnitude of several seconds. In order to maintain the periodic sampling necessary to ensure minimum operation the time-triggered approach cannot be dropped altogether. One solution is thus to implement classic controller design, but if the task does not meet its deadline then it is simply cancelled. Another solution is to use the continuous stream task model: “The continuous stream model of computation is time–triggered but in a flexible way (there is not a fixed point in time where I/O operations have to be performed, but a discrete number of possible of points). What is more, the activation of a job is not triggered by an absolute time, but by the completion of the previous one. This way, the delay introduced by a job is independent by the delays experienced in the previous ones” [72]. The overall result is that less CPU time is required to control the system.

Verification

Simplex Design

For many complex systems it is the design and verification of the controller that is the most challenging task. To make this job easier, a technique called Simplex Architecture is developed to make a simple, but verified controller that takes over whenever the more complex controller operates outside its safe range [22]. By using this method the simple controller act as a supervisor and therefore the complex controller can optimize within its range without taking into consideration what happens outside its range.

Concluding Remarks on Control Systems

Scheduler Interaction with Controller

By combining the methods of the continuous stream task model with the simplex design it possible to design a control system such that whenever the scheduler sees that the system is overloaded, and then the controller can be tuned down to a lower operation. This ensures stability at a lower CPU usage, but at the cost of lower performance.

Further this method could be used at any real time system if the application can operate at a lower speed. One typical example is streaming in which the quality of frames or the number of frames per second could be reduced to compensate for an overloaded processor.

In their paper, J. Eker et al. uses a feedback scheduler along with a controller where the controller feedbacks the scheduler and *vice versa* [97]. They show that scheduling a set of controller loops for optimal performance will require the feedback scheduler and also that it is possible for the scheduler to successfully interact with the controller. The scheduling policy allows for some deadlines to be missed, but adjusts the sampling rate to ensure it will happen less frequent every time. However, if an overload situation can be predicted in advance, then the scheduler can adjust the sampling rate beforehand. Another solution is admission control in which the controller cannot switch mode before given clearance from the scheduler.

Worst Case Execution Time (WCET)

Keywords: Worst case execution time, WCET, predicting WCET, estimating WCET, real time calculus, computing WCET, Liu and Layland.

Preliminaries

In classic hard real time, the main problem is calculating the WCET. Also, it is not always possible to determine the worst case simply because it may be impossible to know all the relevant parameters in advance. Take for instance a radar system, the number of objects that may appear at any given time is unknown and the maximum of objects the radar can detect depends on several factors including speed, direction and overlap of objects. Thus one can only give a decent WCET estimate by imagining the entire area filled with the smallest detectable objects at the worst velocity. For most radar systems this is not a good solution as it will never operate near its worst case and the result is a way too pessimistic WCET for practical use.

Another problem associated with WCET is blocking [49]. To determine the WCET of a task that first needs a locked resource from another task before it can run is even harder than the estimation of WCET for a single task. This results in a NP-hard problem and although priority ceiling and similar techniques have been proposed, there is no simple solution, except never to use preempting [49].

Several attempts have been made to simplify the methods of calculating WCET, including real time calculus (RTC) and the use of harmonic periods [50, 51]. Both giving better estimates of the WCET, but both methods are still pessimistic.

The pessimistic approach necessary to estimate the WCET is the fundamental issue. Clearly one cannot use an optimistic approach as the consequences are dire if a mistake is made [1, 4]. In sum, the methods of estimating WCET are tedious, imperfect and do not give exact time for several cases.

However, there are still cases when the WCET is necessary. For the cases of fail-safe and recovery mechanisms there must necessarily be enough computing power and resources to complete the given action. A nuclear plant must be able to shut down before it explodes and a plane must be able to pull up before it hits the mountain. Thus, this estimate is a lower bound for which the minimum requirements of the crucial system parts should be set. In practice, these minimum requirements are usually satisfied by hardware design alone or with minimum software. Cars with ABS do not have the complexity that could cause the ABS to miss a deadline and in many cases there is also a manual override that shifts the responsibility to the operator. So, design that makes the WCET marginally worse than the Average execution time (AET) is preferable for safety-critical systems, particularly if it is

easily obtained. However, as is evident from modern processor design, the trend is to shorten the AET at the risk of increasing the WCET [14].

In addition to the WCET, a probability for its occurrence should also be included for it to have meaningfulness in firm and soft real time systems (SRT). Because if in a SRTS a task has a probability of its deadline being missed once every thousand year, then one could easily ignore its negligible effect on the system. In practice the WCET is unlikely to occur [75].

In sum, WCET is not that useful for most system and in fact, one should rather shift focus over from WCET to AET. Thus one should reserve WCET analysis only to systems which must comply with classic hard real time design and consider the use of stochastic methods to allow AET to be utilized.

Average Case Execution Time (AET)

Keywords: Average case execution time, AET, predicting AET, computing AET, AET for real time.

Preliminaries

Modern processors and chips are optimized towards a better AET [14]. This is something RTSs should take advantage of instead of letting it be a hinder, as is the case with WCET estimation. Therefore, the use of stochastic systems should be further researched.

One important thing with the AET is the question of what average. If one has a system that runs for 3 hours with 10 tasks with AET of 1 sec and one task that has a possible WCET of 10 hours, what should the AET be? Obviously, the 10 hour task will never finish as its WCET (and, in this case, its deadline) is longer than the operation time. Thus, one should instead focus on the AET of task for as long as they actually run. Further, one should take into account whether the given task has a probable chance of running till completion. If the AET exceeds the deadline then the task probably won't. The implications of this will be discussed under the section on stochastic RTSs.

For a numerical example consider the following three tasks A, B and C, with AET of 2, 2 and 4 time units; WCET of 8, 3 and 4 time units and deadlines of 3, 7 and 6. Notice that task A might run for 8 time units, but has an AET of 2, whereas task C has the same AET and WCET. Now, as task A has a deadline of 3 time units it is likely to succeed on most occasions, it will however fail to meet its deadline if the worst case scenario occurs. This is the case of most SRTSs [53, 54]. However, without a confidence interval the average estimations are not particularly useful. In the example above, task C had a 100% confidence interval as the AET=WCET ➔ the WCET always occurs. Task A might complete in 0.1 very often, but exceed the average too often for the system to be stable or operational. Therefore many systems use confidence intervals especially for predictions [95].

Computing AET

As mentioned earlier, the WCET is hard to estimate. For the AET this is easier, one could either run simulations (some authors suggest the use of a Monte Carlo Engine [57]) and compute the average, or use techniques such as counting floating points operations to gain a fair estimate of the average. Further one should compute the probability of completing the task within the average and also its standard deviation, assuming that it is Gaussian. Thus predicting whether any scheduler can schedule the tasks and the expected number of deadline misses. If the simulations give a distribution deviating from the normal distribution then a simple analysis will be executed to find the closest distribution model.

One problem associated with this method occurs when a scheduler is preemptive. Whenever a task is preempted it can finish within its AET, but still miss its deadline. Solving this issue requires either computation of the probability of preempting or avoiding it altogether. Another solution is also possible, by letting only critical task preempt another task and never let a critical task to be preempted. Then one can safely dismiss the currently running task at the cost of one deadline miss, thereby simplifying the AET computation, counting an event of preempting as a deadline miss.

Summary

The use of average case estimation time is an important part of stochastic real time [58]. Further, it is in general much easier to compute than WCET. By giving a probabilistic measure of deadline misses the scheduler can be designed to meet the necessary amount of deadlines and similarly for the hardware design.

Predictability and Determinism

Keywords: Determinism, predictability, statistics.

Preliminaries

Classic real time is built on the assumption that the world is deterministic as is evident from the naïve approach: One assumes that any event can be predicted in advance and further that every task is periodic [1]. Therefore, WCET can be predicted. According to several theories, in particular quantum mechanics, there is evidence to support truly random events, or further, a non-deterministic universe[11]. Even so, chaos theory depicts a world that cannot be predicted whatsoever. Alas, one should look at the universe as a stochastic, rather than a deterministic system.

As RTS are designed to operate in fairly unpredictable environments one would certainly strive to make the system itself predictable. How else could one assure reliability? Predicting the unpredictable is an impossible task, but one may succeed in predicting the improbable. Therefore any system should be robust to unpredictable events as to minimize the consequences of Murphy's Law.

Synchronous languages like Esterel have been designed to be deterministic and in particular when it comes to parallelism [64].

Statistics: The use of average to achieve predictability in uncertain environments

In thermodynamics temperature is given as an average distribution of speed among particles. Although the particles may vary at speeds ranging from 0 to c[12], the average is more meaningful than every single measurement. It is worth noting that there is an upper and a lower limit of the speeds in this case, which may not necessarily be the case in all systems. However, for practical purposes, and at least for RTSs, the assumption that there is an upper limit to run time, is safe to make as no system is designed to run forever. (And also, several scientist and religions predicts an ultimate end to the universe.) It is therefore possible to use statistics and further assume that the AET, including its probability of occurring, of any task can be calculated/simulated.

Should a single task run way beyond its estimated WCET then the average of the total tasks will barely be affected. However, the predicted WCET have, in such a case, proven to be worthless.

[11] I.E. Bell's Theorem and Heisenberg's uncertainty principle. [55, 56]

[12] Only massless particles can achieve a speed of c, so this is to be read as "up to, not including c." (And certainly not as c^12)

Online Estimation of Run-Time

Keywords: Online estimation of run-time, prediction of run-time, run-time, online prediction of WCET.

Effectiveness of real time systems relies mainly on the effectiveness of the scheduler. However, the scheduler is not clairvoyant and hence the scheduler does not make optimal decisions as to what task should be scheduled when. Only in retrospect is it possible to tell where the mistakes were made in the scheduling. By predicting the execution time of all tasks, then the scheduler essentially becomes clairvoyant, but this is not simple. Several methods have been proposed, but there is currently no exact method available. This also includes WCET predictions [25, 80].

One step along the way would be to predict WCET since it is somewhat easier[13] and gives valuable information as to whether a task may be schedulable or not. But as noted in 2014 by the revision committee of WCET Workshop in Madrid in [25] there is currently no way to do this: “The assumption that a WCET analysis can be done online is unrealistic.” Further searches on Google Scholar, IEEE and dl.acm reveal no articles written after 2014 in which this has changed.

Many execution models are based on off-line methods which consists of code analysis, analytic benchmarking/code profiling and statistical prediction. [81] They are however not as precise as the online methods can be, unless the implementation is straightforward on predictable hardware, which is practically never the case.

There exist methods to give fair online predictions of execution time though. Online methods include simulation data, in which the scheduler can compare the actual results and use curve fitting or similar approaches; extrapolation; statistical methods and input data [81]. Input data can, for instance, be frames of a video in which the number of pixels that differ from the last frame is given as part of the frame information. Hence the run-time can be predicted by using the time it takes to change one pixel and multiply it by the total number of pixels.

A method for estimating the rendering of 3D objects is proposed in [82]. Although the focus of that paper is the GPU, the method could still be used by any scheduler responsible for a 3D rendering application. Nonetheless, the method is not exact.

Combining methods from offline and online into statistical methods will give fair estimates, but, depending on the application, may not be accurate enough. In one of his papers, Peter A. Dinda presents a system that, if the resource requirements are known, will give the

[13] Exact prediction requires lower and upper bound whereas WCET only requires an upper bound.

application an estimate of the running time within a confidence interval [95]. As he notes in his paper, there is surprisingly little work and the subject has a short history.

In sum there is no exact method available for determining the running time of a task which does not involve a lot of work. This can be seen by the fact that WCET science is still in business. The methods improve, but they have not completed their task.

Time/Utility Functions

Keywords: Time utility function, TUF, time value, task value.

Preliminaries

Rather than just setting a task's priority based on its periodicity it has been proposed by Ravindran et al., L. D. Brinceño et al. and Jensen et al., to take into account the actual value of a task's completion [39, 84, 87]. Take for instance a controller, it will usually work just fine if one sample is missed, but applying the output signal too late will be of more severe consequence. Thus the periodicity may not be an optimal way of designing priority. In their survey Ravindran et al. argue that TUFs are superior to the classic approach of setting priority based on deadlines alone [39]. Clearly, a job consisting of several tasks in which all must be complete for the job to have any utility is worthless once it is clear that one of the tasks will fail. Therefore a TUF will be superior as it can dismiss the job entirely, giving room for more valuable tasks.

With the assumptions of classic hard real time a TUF is generally useless as all tasks must meet their deadline, but, as advocated earlier, the assumptions are flawed as some tasks can miss their deadlines [72]. Therefore it is possible to use TUFs for all classes of real time systems, ranging from hard to soft.

The notion of a task's value as a function of time was proposed by Jensen et al., as a generalization of deadlines [84]. It was proposed in 1985 as a method for letting the OS be both more predictable during transient overloads and maximize the collective value of the tasks. By focusing on dynamic scheduling compared to offline-scheduling they realized that although the scheduling problem is NP-complete, much data is only available after the system is in operation and therefore the scheduling must be dynamic.

Use of TUFs

By implementing TUFs in the scheduler while using a feedback scheduler it is possible to make better use of stochastic systems as any off-line scheduling will be inefficient. Due to possible overload conditions it is also difficult to use EDF or variants of EDF. Even for classic hard real time systems there might be situations where two or more tasks may have insufficient time to complete their deadlines and the scheduler cannot decide which one of them is more important. Using TUFs the solution might even be that none of the tasks should run, but an error handling task should run instead.

Suggestion: Reclassifying the Real Time Task Model

The definitions listed in the introduction, hard, firm, soft, are severely limited and the definition of hard real time in particular. As has been pointed out it is necessary to redefine hard real time in a way that makes it useful in practice and not just a theoretical concept. The line between soft and firm is also vague or even dubious. For is a multimedia system firm or soft, or is this simply a designer issue? The same question can be asked again and again for several real time systems and this suggests a redefinition.

The Task Model Classification

Hard Real Time

Hard real time should be divided into two subcategories: The classic hard real time and the non-strict hard real time.

Classic Hard Real Time

Classic hard real time, Liu and Layland or, henceforth, strictly hard real time is the designation given to real time systems where no single deadline can be missed. This is more of a theoretical concept than a practical application. For mixed criticality systems the definition is useful, particularly for the fail-safe or shutdown of a process. Thus, for this reason, the definition of a strict real time is a system of tasks where all deadlines must be met in order to ensure the full operation of a system without failing. And, a wrong operation at the correct time is just as bad as a deadline miss. This is to complement the definition and it explains why it impossible to recover from deadline misses. If it was not the case, then all system could recover from a deadline miss the same way they recover from a calculation fault, simply by redoing the operation. Therefore the assumption is that there is no time for (any) recovery, because if it was, then the deadline would not be strictly hard.

The typical strictly hard real time system is implemented as hardware rather than software, underlining that this is more useful in theory than in actual applications. Although there have been systems that supposedly are truly hard in the strict sense: “Early video game systems such as the Atari 2600 and Cinematronics vector graphics had hard real-time requirements because of the nature of the graphics and timing hardware” [96][14].

Non-Strict Hard Real Time

For this new definition of hard real time it is important to build it on assumptions that will hold in general, contrary to the assumptions of classic real time. Assumption 1 is that the system will fail only when a crucial deadline is missed. Assumption 2 is that it is possible to

[14] There is no valid citation for this claim apart from the Wikipedia article.

identify a crucial deadline in advance, but not necessarily exactly. Assumption 3 is that the system can only guarantee its performance when all non-redundant, vital hardware is functional.

Assumption 1 dictates that not every deadline is crucial. For instance a ship navigation system will have a crucial deadline for braking only when the ship is sufficiently close to land and at sufficient speed. In other words, if the ship has to brake when changing course and there are no obstacles in the way, the effect will only be a slight deviation from course which is only a deviation from its full QoS. However, if it is on collision course with land, then it has to brake or steer away before it is too late. Therefore the crucial deadline is the latest point in time where the ship can avoid crashing. Identifying the crucial deadline is not necessarily trivial and will probably be the hardest design problem with this system.

Assumption 2 is a little looser and needs only to hold in a general sense. It can, for the example above, be defined as the safety point where the ship has to brake under given conditions, but if the ship has a lower speed then the safety point is a not the crucial point. Therefore the definition of the crucial point is left to the designer and it should be defined as the safety point.

Assumption 3 is self-explanatory. If the ship in the example above lacks braking then the software is of limited or no use.

This definition of hard real time is not new as similar relaxing condition has been proposed earlier [52], although this definition is much more relaxed in comparison.

Firm Real Time

Although firm real time is quite similar to hard real time, there is a crucial difference when a deadline is missed. Whenever a deadline is missed, the quality of the system will degrade, observable or not, but the system will still be operational. Any result delivered after a deadline is useless and therefore should the scheduler terminate the task and reschedule any remaining tasks. In other words; when a deadline is passed without the task running to completion the TUF is 0.

For most firm systems there exists no reason to run recovery procedures when a deadline is missed. However, the system should take any necessary precautions to prevent them from happening in the first place. That is, unless the system is stochastic and it has been designed to meet less than 100% of all deadlines.

If a system is designed in compliance with bounded lateness then it is a borderline between soft and firm, but its deadlines are nonetheless firm with respect to the bound, and thus it is classified as firm [11].

In conclusion the definition of firm real time is fine as it already was. No changes are necessary, but, depending on the application at hand, it may be useful to optimize the scheduler to a firm type as opposed to a hard type.

Soft Real Time

The main difference between a firm and a soft system is the usefulness of a result after the deadline has passed. For the soft system the result is still useful, but its usefulness degrades with time. Though not necessarily linearly. (If it does not degrade at all as time passes then the system is not real time.) The end result for missing deadlines is reduced QoS, but at a lower rate than firm systems. When a deadline is missed the TUF is not 0 as for firm systems, but it will gradually decrease to 0.

Soft systems are in particular useful, along with firm systems, for stochastic real time. Because the predictability and the lack of determinism it is possible to design the system with a given quality of service. However, it is not necessary, nor possible, to perform any scheduling proof.

The scheduler algorithm should be adapted for a soft system in order to be of any use. As mentioned previously, the EDF scheduler performs poorly when a system is operating with a load exceeding 100%. In conclusion, a scheduler for a soft system should be dynamic in order to rectify any case of deadline misses whenever a following deadline miss will further reduce the quality of service. Consider a part of a radar image: If it fails to render once it is probably not noticeable, but every succeeding deadline miss will reduce the quality of the radar service. Thus, the task that is missed should be assigned a higher priority for every miss to prevent this from happening.

Ultimately the question is whether or not there is such a thing as a soft real time system. If the deadline is important then the system is firm, but if not, then it could just as well be best effort? The distinction is probably best justified with an example: After you have just had lunch you would think the waiter is a bit off if he brings you dinner. This is an example of best effort; it brings you things as soon as possible. The soft system delivers you dinner, possibly a bit late, but not too early. Soft real time implies a deadline and usually the implication is that the system cannot be too early, rather than; do not be late. Another example should clarify some more: A Pizza delivery says the pizza comes within 15 minutes or it is free. One might argue that this is a firm system, but the TUF is not 0! The customer will be happy to receive free pizza and is likely to order more on a later occasion. The TUF does not drop to 0 before the pizza is cold; when the customer does not want it anymore. And finally, a best effort doesn't fully describe the system. Time is a factor, even though the pizza should be delivered as fast as possible if not the customer indeed ordered his pizza to be delivered at a specific time.

In sum, the soft real time definition differs from firm systems by being able to repair some of the damage resulting from a missed deadline. It differs from the best effort by making time

an important factor. So if the goal could be achieved by using best effort, but fails to describe the system in full, one should classify the system as soft real time.

Scheduling Real Time Built upon the Task Model

The RM scheduler is static and therefore is not usable in systems with some unpredictability. The EDF is more dynamic, but is still too static when dealing with overloads and is in general incapable of adapting. The solution is feedback schedulers and using estimates of run-time of tasks, preferably off-line estimates, but if they are too inaccurate it may be better to use some processing time to get better estimates. Stochastic schedulers can utilize TUFs to further improve QoS.

Stochastic Real Time

Keywords: Stochastic real time, random systems in real time.

Preliminaries

A stochastic real time system is both non-deterministic and predictable. This implies that any decision by the system cannot be predicted in advance, but the final outcome can be, to some extent. The system must meet a certain amount of its deadlines, usually achieved by FCS. The only variable that has to be stochastic is the deadline (met/not met), but several more variables can be stochastic based on the implementation.

Many applications of stochastic real time are just variations over Liu and Layland where the unknown execution times are modeled as stochastic variables [91]. The weaknesses of the classic approach are thus not eliminated.

Instead of analyzing every task's execution time, a probabilistic approach can be used. As is evident from years of research it is not possible to design a flawless system and many approaches are built upon making any failures extremely unlikely. This design is found for instance in airplanes, in which any fault is unlikely, the propagation of the fault is unlikely and the consequence of the fault to be of severe consequences is extremely unlikely. Only if "everything" goes wrong at once the plane will crash, or the operator (read: pilot) chooses to crash the plane.

By using FCS it is possible to design a system such that a specific number of deadlines can be guaranteed to be met and further balance which ones. That is, ensuring that any worst case scenario, i.e. starvation etc. does not occur. Most of the research in this field is conducted by C. Lu et al. [31, 62]. Further research will probably utilize this method and make way for real time systems that are even more stochastic; meaning that once the specifications for the system have been given, the system is predictable, but non-deterministic.

A paper written in 1993 suggested using Artificial intelligence (AI) as a way to optimize FBS to give even better results than just a standard FBS [100]. However, the idea seems

abandoned as no progress is made and AI is barely, if ever, mentioned in scheduling papers. A search for papers, with the keywords “AI scheduling real time” or “CIRCA: The Cooperative Intelligent Real-Time Control Architecture” written after 2011, returned no relevant results.

Soft and Firm Schedulers (Stochastic)

Using feedback schedulers it is possible to design systems such that a specific number of deadlines must be met and from there it is possible to, for instance, create schedulers for multimedia applications. It is trivial to increase the frame data to include the number of pixels that must be altered from the previous frame and thus the system has a very good estimate of the running time of that task. If there are too many pixels that cannot be rendered in time it is simple to just skip the frame and run the next one while also sending feedback to the scheduler that this next frame is more important to maintain decent QoS.

Hard Schedulers (Stochastic and deterministic)

Non-strict

Typically, control system can tolerate some missed deadlines without system failure [30, 72], and therefore belong in the non-strict classification. The design of non-strict systems should focus on utilizing TUFs to be able to determine what deadlines could be missed when necessary and thus perform at its best. As it is possible to have constraints such as 99% of all deadlines must be met by the tasks, a feedback scheduler can be used to ensure that the “correct” 1% is missed. The issue here is that it is not indifferent as to what deadlines are missed and the statement: 99% of all tasks must meet their deadline is not equivalent with 99% of all deadlines must be met.

Strict (Deterministic)

For systems were all deadlines must be met the Liu and Layland approach is quite possible the best. However, this put many limits on the system and therefore the complexity of the system cannot go above a given threshold. Further, small, non-complex systems are sometimes implemented as just hardware. The advantage of this approach lies in the schedulability proof; that you can ensure with certainty that all deadlines will be met.

Designing Real Time Systems

The first question one should ask when designing a system is what purpose it will serve. Next question is whether or not it is safety-critical. In fact it is useful to consider a three dimensional model for designing systems based on what purpose it should serve.

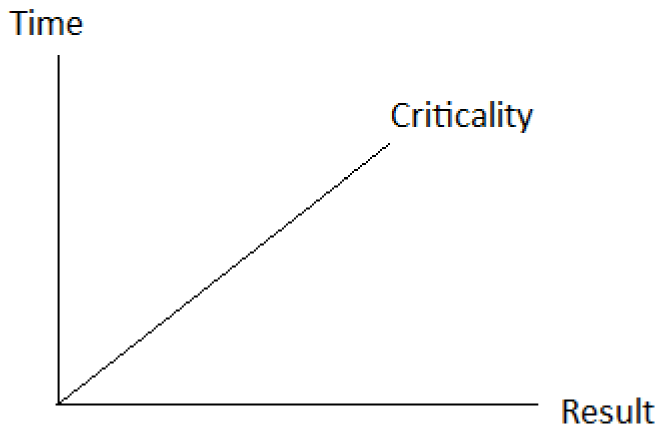

Figure 2: Simple model for system design, where the factors are time, criticality and result.

From this model (Figure 2) it can be argued that although best effort is not real time, time may still be an important factor. Therefore, when designing a system one should not limit oneself to only real time. ABS-brakes are one example of a best effort system that could also be a real time system. The specification says that once the brake blocks the wheel it should release. Thus, from the specification it does not constitute a real time system as the as soon as possible/immediately is not a real time specification. However, this can be changed by saying that the brake should release within x milliseconds. Nonetheless, this is an example of where a best effort system will fulfill the functionality that can also be modeled as real time.

Now, imagine a controller that should control an oscillating process. Here, the measurements must be conducted on time and periodically, to ensure correct control of the system. If a best effort system were used, then the measurements would not be conducted properly as the timing requirement is vastly different from as soon as possible. In fact, too tight regulation of a system might result in instability or breakdown of a system, as Maxwell discovered back in 1867 [93]. Therefore it is not always feasible to use best effort and thus real time is necessary, not because of the specification, but because of the system itself.

The Three Dimensions of a System

Both real time and best effort can be placed all over this three dimensional model, therefore it is necessary to explain what the dimensions actually mean.

Criticality

Is there a risk to lives or environment? This question is the key here. For a nuclear plant the criticality is high and for a simple calculator the criticality is low.

Result

The importance of a correct result, but also the complexity of the result is given by this dimension. CERN for instance will, in most cases, rather spend one year of computation to have an exact result, than five minutes to have a rough estimate, because exact results are necessary in particle physics.

In some systems the complexity is so low that time is the only factor. Take for instance a saw with skin detection. A current is altered when a conductive material such as skin is present and the system shuts down. This would be a system with high criticality and low result also resulting in a high time factor.

Time

Time, in this case, actually has two distinct meanings. One is the timing between two actions and one is the speed of which things happens. The distinction is obvious when looking at a controller. Here, the timing means that it should sample regularly and have a constant time between each sample. The speed means that its actuators should be as fast as possible. It is when timing is an issue that real time must be used; otherwise a best effort could perform just as well. Therefore, the time dimension in this model refers to timing.

Choosing the Right Model

When the specifications for the system have been determined and assuming it falls within the realm of real time, the next step is determining which approach to use. If the system is simple then a classic approach can be used as the WCET estimation and scheduling proofs are simple. However, this is rarely the case and this calls for more sophisticated methods. Unfortunately, there are not many approaches better than ad hoc. Design methods should focus on module based approaches as it is then possible to construct simpler systems before combining them into a more complex system. This also requires sufficient fault detection to ensure that faults do not propagate through the system. Designing simple modules with the factors; criticality, time and result will result in a robust system when the modules are combined. It may be a challenging task to determine how much processing power the entire system needs, but it is easier to determine sufficient processing power for a small module. This is the case for all hardware and which is why a module based approach is, in most cases, superior to other approaches. Unfortunately, it is not necessarily easy to determine when the module based approach may be inferior. The interdependence of software and hardware makes the design process challenging if there is no similar system developed before that can function as a template.

In sum, the real time science requires more research as the classic approach is just theoretical and there is no unified theory about how to design a real time system. There are suggestions and rules of thumb, but no foolproof way to ensure that the design process is correct. Modeling languages works for simple systems, but when the complexity increases, especially in non-linear systems, the models may be too inaccurate to be of any use. Finally, any search for keywords associated with design(ing) real time systems reveals that there are a lot of suggestions, but no blueprint.

Current Trend in Real Time Research

Liu and Layland's method of using scheduling proofs is still the dominant field within real time. Because the theory is solid, people will not willingly abandon it even though the practical implementations are hard to base on the theory. Research in psychology conducted by A. Tversky and D. Kahneman suggests that people prefer a wrong map over no map at all.[15] Therefore, the trend of fitting the terrain to the map, as opposed to the reverse, will continue.

Classic real time will undoubtedly be the focus of many papers still. The theory is sound and if it could be used in practice the gains would be significant. This will require that the hardware is adapted to suit the classic approach to real time instead of focusing on a better average case.

WCET are probably the most studied subject within the field of real time returning 12 000 results on Google Scholar. (Search: "WCET real time") However, all methods of WCET estimation are necessarily pessimistic to ensure that all hard deadlines are met. Therefore, the methods of WCET estimation are flawed until a perfect method can be found. Based on several years of research and the increased complexity of processors, interdependence and similar issues, it is not apparent that this problem will ever be resolved.

In control theory, the ongoing research of reducing jitter will undoubtedly continue. Designing robust control system is not an easy task, but real time is one way of designing such systems. Thus, real time approaches to control theory will probably continue to be a main focus of real time research. The RTSS symposium of 2014 illustrates this by having several papers dedicated to the subject.

Scheduling theory will continue to develop. As several of the scheduling problems are NP-hard the research in the subject will never be complete. Heuristics will be further developed to allow for better scheduling strategies and hopefully TUFs will be implemented more often to ensure better scheduling. Feedback scheduling shows promising result along with TUF and more papers should pursue this idea further.

Research in dynamic voltage scaling will undoubtedly continue. Saving power and thus providing more efficient systems are particularly important in battery powered systems, but by also reducing heat developed in a system, it has its use in most systems.

In addition to offline WCET estimation, there should be more focus in the online prediction. Not only WCET, but the expected run time. Predicting the run time will allow for better

[15] DLD09 conference: https://www.youtube.com/watch?v=LjGl6bZF6zs#t=3386 (55:48) (Although he was pointing his finger to econometrics, the point still applies elsewhere.)

scheduling which in turn will allow for less power use. By using confidence intervals and TUFs there is a potential for a gain in both power use and QoS.

Answering the Questions

This section serves as a quick reference to all the questions asked in the "Objective" section. These are the conclusions made after writing this paper.

Can we detect and handle time-domain errors like all other errors in the system?

Yes, in fact the Sloth research project does make the hardware responsible for checking for deadline misses [73]. By letting the hardware interrupt tell the processor whenever a deadline is missed, it is possible to handle and recover this error just like any other error. There are however some exceptions, in particular jitter and other faults occurring due to errors in timing rather than deadline misses. These faults must be corrected in other ways and jitter must be reduced by adapting hardware or other techniques listed in the section under "Real Time and Control Systems." Finally, these problems are also present in classic approaches to real time.

Can we predict (eg. by online execution time analysis) deadline-misses before they happen?

Deadline misses can be predicted, but current methods for predictions perform poorly. (See section titled "Online Estimation of Run-Time.")

Can we interact with applications to adjust their needs for computing power when the system is busy?

This is done for several control systems [97]. It has proven to be an efficient method for control systems without requiring a scheduling proof. (See section "Timing in Control Systems," subsection "Concluding Remarks".)

Can we have adaptive systems that reduce the probability of timing errors to acceptable values?

This is one of the main purposes of feedback schedulers. However, the research done on this subject is very small. See these papers [31, 62] or read the section titled "Scheduling Real Time Built upon the Task Model."

Can we redefine "real time" to mean something other than the traditional "hard deadlines," while still making sense?

Apart from the obvious soft and firm real time, there is also the use of other reactive systems listed under “Approaches to Reactive Systems.” Notably, most of this is out of scope for this paper and therefore just briefly mentioned.

Can we allow stochastic models etc. for real-time?

Yes, this is essentially the same as the question about reducing the probability of timing errors. See the section “Scheduling Real Time Built upon the Task Model.”

Glossary

A(C)ET: Average (Case) Execution Time

ADC: Analog Digital Converter

CMOS: Complementary Metal–Oxide–Semiconductor

CPU: Central Processing Unit

CSP: Communicating Sequential Processes

DV(F)S: Dynamic Voltage (and Frequency) Scaling

EDF: Earliest Deadline First

FBS: FeedBack Scheduler

FCS: Feedback Control Real Time Scheduling

FPGA: Field Programmable Gate Array

FSM: Finite State Machine

GPU: Graphics Processing Unit

MPC: Model Predictive Control

OS: Operating System

QoS: Quality of Service

RM: Rate Monotonic

RT: Real Time

RTC: Real Time Calculus

RTS(s): Real Time System(s)

SRTS(s): Soft Real Time System(s)

TUF: Time/Utility Function

WCET: Worst Case Execution Time

Bibliography/References

1. C.L. Liu, James Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.” 1973. (Cited By.: 9000+)
2. linwei Niu, “Leakage-Aware Scheduling for Real-Time Embedded Systems with QoS Guarantee.” 2009. (cb.: 1)
3. P. Ramanathan, “Overload Management in Real-Time Control Applications Using (m,k)-Firm Guarantee.” 2000. (cb.: 209)
4. Iain John Bate, “Scheduling and Timing Analysis for Safety Critical Real-Time Systems.” 1998. (cb.: 65)
5. Hakan Aydin, et al., ”Power-Aware Scheduling for Periodic Real-Time Tasks.” 2004. (cb.: 413)
6. J. R. Lorch, A complete picture of the energy consumption of a portable computer.” 1995. (cb.: 63)
7. J. Pouwelse, et al., ”Dynamic Voltage Scaling on a Low-Power Microprocessor.” 2001. (cb.: 401)
8. H. Zeng et al., ”ECOSystem: Managing Energy as a First Class Operating System Resource.” 2002. (cb.: 411)
9. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.8216&rank=1
10. Phillip A. Laplante, ”Real-Time Systems Design and Analysis.” 2004. (cb.: 593)
11. Paolo Valente and Giuseppe Lipari, “An Upper Bound to the Lateness of Soft Real-time Tasks Scheduled by EDF on Multiprocessors.” 2005. (cb.: 28)
12. Tim Kaldewey et al., ”Firm Real-Time Processing in an Integrated Real-Time System.” 2006. (citations not found)
13. Tein-Hsiang Lin and Wernhuar Tarng, “Scheduling periodic and aperiodic tasks in hard real-time computing systems.” 1991. (cb.: 49)
14. Kelvin D. Nilsen and Bernt Rygg, ”Worst-Case Execution Time Analysis on Modern Processors.” 1995. (cb.: 68)
15. P. Caspi et al., ”LUSTRE: A declarative language for programming synchronous systems.” 1987. (cb.: 656)
16. Gerard Berry and Georges Gonthier, ”The Esterel Synchronous Programming Language: Design, Semantics, Implementation.” 1992. (cb.: 2036)
17. Hermann Kopetz, “Real-Time Systems: Design Principles for Distributed Embedded Applications.” 2011. (cb.: 2203)
18. Jens Hildebrandt et al. “Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems.” 1999. (cb.: 51)
19. Chung-Yao Kao and Anders Rantzer, “Stability Analysis of Systems with Uncertain Time-Varying Delays.” 2006. (cb.: 144)
20. P. Marti et al., “Jitter Compensation for Real-Time Control Systems.” 2001. (cb.: 169)
21. http://en.wikipedia.org/wiki/Jitter
22. S. Bak et al., “Real-Time Reachability for Verified Simplex Design.” 2014. (citations not found)
23. L. Hetel et al.,, “Stabilization of Arbitrary Switched Linear Systems with Unknown Time Varying Delays.” 2006. (cb.: 235)
24. W.Zhang et al., ”Stability of networked control systems.” 2001. (cb.: 3433)

25. Fredrik Bakkevig Haugli, ”Using online worst-case execution time analysis and alternative tasks in real time systems.” 2014. (Citations not found)
26. S. Hendseth, ”Exploiting online WCET estimates.” 2012. (Presentation. cb.: 1)
27. R. Devillers and J. Goossens, “Liu and Laylands schedulability test revisited.” 1999. (cb.: 39)
28. Teera Phatrapornnant, “Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling.” 2007. (cb.: 45)
29. http://citeseerx.ist.psu.edu/showciting?doi=10.1.1.28.7640
30. Z.Sahraoui et al., “Antinomy between schedulability and quality of control using a feedback scheduler.” 2014. (cb.: 0)
31. C. Lu et al., “Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms.” 2002. (cb.: 623)
32. Scott A. Brandt et al., “Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time and Non-Real-Time Processes.” 2003. (cb.: 192)
33. H. Tang et al., “Combining Hard Periodic and Soft Aperiodic Real-Time Task Scheduling on Heterogeneous Compute Resources.” 2011. (cb.: 18)
34. Santiago Pagani and Jian-Jia Chen, “Energy Efficient Task Partitioning based on the Single Frequency Approximation Scheme.” 2013. (cb.: 5)
35. S. Albers and A. Antoniadis, “Race to idle: new algorithms for speed scaling with a sleepstate.” 2012. (cb.: 42)
36. S. Irani et al., “Algorithms for power savings.” 2003. (cb.: 285)
37. M. Chetto and H. Ghor, “Real-time Scheduling of periodic tasks in a monoprocessor system with rechargeable energy storage.” 2009. (cb.: 7)
38. http://www.heart.org/HEARTORG/Conditions/More/CardiacArrest/About-Cardiac-Arrest_UCM_307905_Article.jsp
39. B. Ravindran et al., ”On Recent Advances in Time/Utility Function Real-Time Scheduling and Resource Management.” 2005. (cb.: 81)
40. B. Colwell, ”The Zen of overclocking.” 2004. (cb.: 24)
41. V. Subramanian, “Timing speculation and adaptive reliable overclocking techniques for aggressive computer systems.” 2009. (citations not found)
42. AMD, “AMD “Dragon” Platform Technology Performance Tuning Guide.” 2009. (User manual)
43. V. Subramanian et al., “Superscalar processor performance enhancement through reliable dynamic clock frequency tuning.” 2007. (cb.: 20)
44. J. Torrellas and B. Greskamp, ”Timing Speculation: Designing Multi-Cores for Single-Thread Performance.” 2008. (Presentation. Citations not found)
45. M. Bezdek. ”Utilizing timing error detection and recovery to dynamically improve superscalar processor performance.” 2006. (Citations not found.)
46. G. Memik et al., “Engineering Over-Clocking: Reliability-Performance Trade-Offs for High-Performance Register Files.” 2005. (cb.: 29)
47. Y. He et al., “Parameter-Dependent Lyapunov Functional for Stability of Time-Delay Systems with Polytopic-Type Uncertainties.” 2002. (cb.: 696)
48. http://en.wikipedia.org/wiki/Polytope
49. A. Wieder and B. Brandenburg, “On the Complexity of Worst-Case Blocking Analysis of Nested Critical Sections.” 2014. (cb.: 1)
50. V. Bonifaci et al., “Polynomial-Time Exact Schedulability Tests for Harmonic Real-Time Tasks.” 2013. (cb.: 4)

51. N. Guan and W. Yi, “Finitary Real-Time Calculus: Efficient Performance Analysis of Distributed Embedded Systems.” 2013. (cb.: 1)
52. P. Kumas and L. Thiele, ”Quantifying the Effect of Rare Timing Events with Settling-Time and Overshoot.” 2012. (cb.: 4)
53. H.Alhussian et al., ”A Semi Greedy Soft Real-Time Multiprocessor Scheduling Algorithm.” 2014. (Citations not found.)
54. L. Burdalo et al., ”Analyzing the Effect of Gain Time on Soft-Task Scheduling Policies in Real-Time Systems.” 2012. (Citations not found.)
55. P. Busch et al., ”Heisenberg's Uncertainty Principle.” 2007. (cb.: 157)
56. J. F. Clauser and A. Shimony, ”Bell's theorem. Experimental tests and implications.” 1978. (cb.: 1501)
57. N. Satish et al., ”Scheduling Task Dependence Graphs with Variable Task Execution Times onto Heterogeneous Multiprocessors.” 2008. (cb.: 22)
58. A. F. Mills and J. H. Anderson, ”A Stochastic Framework for Multiprocessor Soft Real-Time Scheduling.” 2010. (cb.: 28)
59. R. I. Davis and A. Burns, ”A survey of hard real-time scheduling for multiprocessor systems.” 2011. (cb.: 394)
60. G. C. Buttazzo, ”Rate Monotonic vs. EDF: Judgment Day.” 2005. (cb.: 330)
61. M. Bertogna and S. Baruah, ” Tests for global EDF schedulability analysis.” 2010. (cb.: 44)
62. C. Lu et al., ” Feedback Control Real-Time Scheduling in ORB Middleware.” 2008. (cb.: 63)
63. S. Lin, ” Feedback-based task scheduling in real-time systems.” 2005. (citations not found.)
64. F. Boussinot and R. D. Simone, ”The Esterel Language.” 1991. (cb.: 578)
65. N. Halbwachs, ” Synchronous Programming of Reactive Systems.” 1993. (cb.: 1093)
66. T. Murata, ” Petri Nets: Properties, Analysis and Applications.” 1988. (cb.: 9799)
67. G. M. Reed and A. W. Roscoe, ”A Timed Model for Communicating Sequential Processes.” 1988. (cb.: 367)
68. http://www.ada-auth.org/standards/ada12.html
69. L. Sha and J.B Goodenough, ”Real Time Scheduling Theory and Ada.” 1989. (cb.: 411)
70. C. A. R. Hoare, Communicating Sequential Processes.” 2004. (cb.: 11816)
71. M. V. Iordache and P. J. Antsaklis, ” Petri Nets and Programming: A Survey.” 2009. (cb.: 25)
72. D. Fontanelli et al., ”The Continuous Stream Model of Computation for Real–Time Control.” 2013. (citations not found.)
73. W. Hofer et al., ”SLOTH ON TIME: Efficient Hardware-Based Scheduling for Time-Triggered RTOS.” 2012. (cb.: 5) (Part of research project: https://www4.cs.fau.de/Research/Sloth/)
74. http://www.claverton-energy.com/energy-experts-online/relevance-paradox
75. N. C. Cox et al., ” A Refined Approach for Stochastic Timing Analysis.” 2014. (citations not found.)
76. G. A. Bundell, ” An FPGA Implementation of the Petri net Firing Algorithm.” 1997. (cb.: 13)
77. E. Soto and M. Pereira, ” Implementing a Petri Net Specification in a Fpga Using Vhdl.” 2001. (cb.: 19)

78. P. Jakubco and S. Simonak, ”Petri Net Approach for Algorithms Design and Implementation.” 2011. (cb.: 1)
79. C. Conway et al., ”Pencil: A Petri Net Speci_cation Language for Java.” 2002. (“This was a one-term project in a graduate compilers class and as such exists somewhere in the zone between "good enough to get an A" and "good enough to actually use."” https://www.cs.nyu.edu/~cconway/pencil/)
80. M. Chtepen et al., ” Online execution time prediction for computationally intensive applications with periodic progress updates.” 2012. (cb.: 2)
81. M. A. Iverson et al., ” Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment.” 1999. (cb.: 179)
82. S. Schnitzer et al., ” Concepts for execution time prediction of 3D GPU rendering.” 2014. (cb.: 1)
83. L. Sha et al., “Real Time Scheduling Theory: A Historical Perspective.” 2004. (cb.: 474)
84. E.D. Jensen et al., ”A Time-Driven Scheduling Model for Real-Time Operating Systems.” 1985. (cb.: 639)
85. T. Bøgholm, “Hard Real-Time Java.” 2012. (citations not found, download statistics can be found here: http://vbn.aau.dk/en/publications/hard-realtime-java(3f0f4a22-4f04-4f0f-b3d8-309ecdd738ad).html)
86. H. Kopetz et al., “The Time Triggered Model of Computation.” 2003. (cb.: 136)
87. L. D. Brinceño et al., “Time Utility Functions for Modeling and Evaluating Resource Allocations in a Heterogeneous Computing System.” 2011. (cb.: 13)
88. http://www.nytimes.com/2015/05/01/business/faa-orders-fix-for-possible-power-loss-in-boeing-787.html
89. R. L. C Eveleens, ”Integrated Modular Avionics Development Guidance and Certification Considerations.” 2006 (c.b.: 9)
90. http://en.wikipedia.org/wiki/Integrated_modular_avionics
91. J. Nilsson et al., “Stochastic analysis and control of real-time systems with random time delays.” 1998. (cb.: 971)
92. J. L. Diaz et al., “Stochastic analysis of periodic real-time systems.” 2002. (cb.: 148)
93. J.C. Maxwell, “On Governors.” 1876. (cb.: 416)
94. M. Zhu and R. R. Brooks, “Comparison of Petri Net and Finite State Machine Discrete Event Control of Distributed Surveillance Networks.” 2009 (cb.: 1)
95. P. A. Dinda, “Online Prediction of the Running Time of Tasks.” 2001. (cb.: 110)
96. http://en.wikipedia.org/wiki/Real-time_computing
97. J. Eker et al., ”A Feedback Scheduler for Realtime Controller Tasks.” 2000 (cb.: 141)
98. KE. Årzén et al., ”Integrated Control and Scheduling.” 1999. (cb.: 57)
99. A. Cervin et al., ”How Does Control Timing Affect Performance?” 2003 (cb.: 436)
100. D. J. Musliner, ”Circa: The Cooperative Intelligent Real-Time Control Architecture.” 1993 (cb.: 6)
101. A. Benveniste and G. Berry, ”The Synchronous approach to reactive and real-time systems.” 1991 (cb.: 897)