Three Obstacles to Flexible Scheduling

Guillem Bernat and Alan Burns
Real-Time Systems Research Group
Department of Computer Science
University of York
Heslington, York YO10 5DD - England
{bernat,burns}@cs.york.ac.uk

Abstract

The key to the next generation real-time systems is flexible scheduling mechanisms that guarantee hard deadlines and use available spare resources to maximise total system utility. This is a multicriteria scheduling problem. It is argued that common approaches like eager slack usage and mandatory first schemes are not only not optimal but not adequate for a wide class of process models. It is also shown that a late acceptance test model is preferable to an early acceptance test model due to the uncertainty of future behaviour of the system. The discussion is complemented with simulation results.

1 Introduction

The challenges of flexible scheduling are easy to (informally) define: hard deadlines must always be met, and any spare capacity (typically CPU resource) must be used to maximise the utility of the application. The 'extra' work that could be undertaken typically exceeds the spare capacity available, and hence the scheduler must decide which non-hard task to execute an any time. As processors become more complex and the analysis of worst-case execution time more pessimistic, the amount of spare capacity will increase. Indeed the utilisation of the hard tasks (in the worst case) will often be less than 50% (leaving even more capacity available at run-time due to tasks not running for the worst case execution time at every invocation). There is always pressure on application to minimise their use of hard tasks and to exploit flexible scheduling to deliver dynamic behaviour.

In this paper we review three problems of (or obstacles to) on structuring real-time systems for flexible scheduling. These three problems are concerned with when to undertake key elements of the scheduling process:

- When to use slack
- When to run mandatory components, and
- When to apply acceptance tests.

1.1 Components of flexible scheduling frameworks

It is clear that any scheduling framework must address the following key issues:

- Computational model: how to structure application tasks so that they can make effective use of flexible scheduling
- Resource management: How to ensure that hard tasks have their required resources (when they need them, so that no hard deadline is missed) and how to maximise the available resources for the non-hard tasks.
- Load management: how to schedule the current load of non-hard tasks to maximise the total system utility

The first issue is generally modeled by dividing the tasks as a mandatory part followed by an optional part. In this way a minimum level of service is attained and if there are spare resources the quality of the result can be increased by the optional component.

The second issue is addressed by:

- Slack usage: Slack management techniques are generally used to delay the mandatory parts as late as possible so that optional components can be run "as soon as possible". This is the first obstacle. It is shown later in section 3 that making slack immediately available for optional components is not only not optimal but it can lead to very low performance. We call this first problem: *'Eager slack usage is too eager'*.

- To prevent the negative effects that eager slack usage techniques induce, some authors advocate a "mandatory first approach" [24]. In this approach mandatory components are run as soon as possible so that optional components are released as soon as possible and therefore can compete for resources. We later show in section 4 that this option does not lead to good results either. We call this problem: *'Lazy slack usage is too lazy'*.

The third issue is addressed by an acceptance test that throws away jobs of low value to keep the system from becoming overloaded. The decision is generally taken as soon as the task is released. The test can never be optimal as there is always a level of uncertainty on the information of the current state of the system. This is due to:

- Tasks do not always run for their WCET, it is difficult to know how much time the tasks have left to execute and therefore how much time is available for optional components. An acceptance test which only relies on WCET values is unnecessarily pessimistic.
- Actual arrival times are uncertain. In real scenarios, not all tasks rearrive periodically, and the actual arrival instants of the tasks are unknown.
- The number of iterations in incremental computations is not known. Optional components are not necessarily released after every mandatory computation.

Taking an early decision on whether to accept a task is less accurate that taking the decision later as by doing it sooner the test has to account for more uncertain events of the future (of course taking the decision too late, for instance at the deadline, is not good either). This uncertainty leads to more wrong decisions and therefore to lower value achieved. This is the third obstacle and it is named *'Early acceptance test is too early'* and is explained in section 5.

In the rest of the paper we describe these three problems in detail, and show some evaluation results based on simulation studies.

2 Computational Model

Little of the published material on flexible scheduling identifies how an application can exploit any proposed scheme in order to build more effective systems. Indeed many papers deal solely with independent aperiodic activities that seem to arrive in the system in a completely random fashion. However, a real-time system is not a general purpose computing resource; it is a single system in support of a single application, albeit a potentially multi-faceted complex one. Once execution has commenced all computing activities are known. In this paper we are concerned only with activities that have associated deadlines, i.e. real-time jobs. Such jobs can be classified via a number of distinct characteristics:

criticality - jobs that must always complete by their deadline are known as *hard*, others are non-hard; jobs may have components that are of mixed criticality i.e. some hard, but not all.

regularity - time-triggerered jobs that have a regular cycle time are known as *periodic*, event-triggered jobs that have a minimum inter-event separation are *sporadic*, other event-triggered jobs are called *aperiodic* but even these have a bound on their arrival patterns (which may be expressed probabilistically).

predictability - jobs that are not, temporally, highly dependent on input data (or internal state) can have a known worst-case execution time (*WCET*), other jobs will have more dynamic behaviour, but they are never unpredictable.

In order to statically guarantee a hard job (or hard component) it must be either periodic or sporadic and have a known WCET.

The complete set of jobs in an application will have a number of inter-job relationships including data sharing, producer/consumer interaction and various forms of precedence orderings. Of course a hard activity must never have a synchronous dependency on a non-hard job. Similarly a periodic activity cannot rely on the arrival of sporadic or aperiodic jobs.

In general, non-hard tasks are classified as either *firm* or *soft*. A firm task is one that should complete by its deadline - it is not critical but there is no value in it continuing to execute after its deadline. The definition of soft usually implies that the task has a deadline but that execution after the deadline is still useful. This is a very poorly defined notion. To exploit this property at run-time requires knowledge of how the significance of the task decays after the deadline. It is unclear that applications really possess this property or can represent the decay with any meaningful precision. Hence we conclude that the computational model should contain only hard and firm tasks. If a hard task misses a deadline it is an error condition. A firm task can either be aborted at its deadline or be allowed to complete.

The restriction that a hard job must have a known WCET limits the functionality of the hard activity. To counter this, a generic structure can be used that implements a hard job as three tasks (two hard and one firm). Figure 1 illustrates this structure. The complete job can be either periodic or sporadic. The firm task F is released by the first hard task H^1; the deadline of F coincides with the release of H^2. In previous publications [2, 1] we have shown how this combination of tasks can implement a wide class of imprecise computations [23, 17, 21], imprecise query systems [6, 26], multiple method [11, 12], heuristic search [16, 19, 15] and anytime algorithms [10, 27, 22, 28]. For many of these applications the final hard task H^2 is omitted, and in some approaches there is more than one firm task executing in the intervening interval (or the same firm task executes a number of times before the deadline).

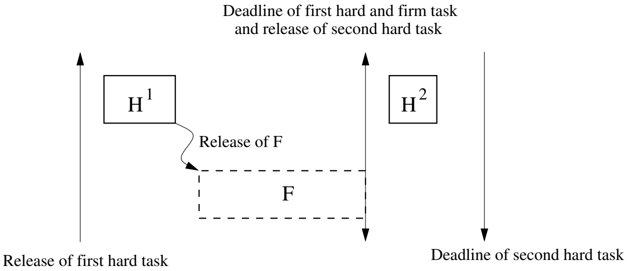

Figure 1. Hard job as hard and firm tasks

An application with a number of firm tasks that may have dynamic execution times cannot statically guarantee all computations. At run-time they may be an overload and a decision must be made as to which jobs to execute and which to pass over. To make this decision, some notion of *utility* must be assigned to all firm tasks. The usual approach is to assign a 'value' to the task [18, 5, 4, 8, 13]; if it completes by its deadline it delivers this value to the application, otherwise the value achieved is zero.

In other work we have addressed the issue of assigning value to tasks [7]. We showed that the construction of a multi-criteria value system is non-trivial. Hence the use of complex time-value functions and very precise measurements of value are not usable or required by applications. We conclude that a relatively small range of values be used [20] and that within the computational model firm tasks have a static value assigned when they are released [14]. For example, if hard task H^1 releases firm task F it will inform the scheduler of the value of F if it completes by its deadline. Note also that the deadline is known statically and that at least the average execution time of F is known. This can be obtained during the testing phase.

One interesting result is that in a purely value-based scheduling framework, scheduling the task with the highest value-density (defined as V/C) is optimal, because the scheduler executes the task that yields the highest value per unit of time.

However, for HF task models this result can not be applied in a direct way. One particular case is when the value of all tasks is equal to their computation time which effectively assumes an identical value-density (V/C) for all tasks. This model is too constrained, we assume a more general model in which tasks have different values and varying computation times (and consequently different value-densities).

Using this scheme for values requires that a classic anytime algorithm is implemented. In this case a firm task must structure itself as a series of invocations each with a similar execution time but monotonically decreasing value. A run-time decision can be made as to how many invocations to release at any one time.

In summary, we consider a computational model which allows an application to use periodic or sporadic hard tasks with known inter-arrival intervals of T, deadline D and worst-case execution times C^w and firm tasks (periodic, sporadic or aperiodic) with bounded arrival characteristics and known average computation time C^a; and characteristics of deadline D and value V - these being known either statically or when the task arrives in the system. For convenience we also write $W = V/C$ as the value-density of the task.

3 'Eager slack usage is too eager'

One of the most common approaches for flexible scheduling is to make slack available for running non-hard tasks with the objective of minimising their average response time. These slack mechanisms differ in the way slack is extracted and by how much they can extract, but in all cases they are designed so that they delay the execution of hard tasks as much as possible.

This eager slack usage has been found not to be optimal [25] (even for independent non-hard tasks) in the sense that in some circumstances not using the slack as soon as possible results in more slack available in the near future. Note that finding an optimal schedule (i.e. determining when to use the slack or when not to use it, even though there is some available) is an NP-Hard problem. Moreover, in the context of the HFH task model we have identified a much worse scenario. This happens when a hard task that may release a high value optional task is delayed because the system is busy running low value optional tasks because there is slack available.

This is shown in figure 2. At time t_0, H_1 is released which completes at time t_1 and releases its firm component F_1 with low value V_1. At a time instant $t_2 > t_1$ a hard task H_2 is released that when it completes will release a high value task F_2. Any eager slack mechanism delays the execution of task H_2 in favor of F_1 until a time t_3 that will guarantee that H_2 will finish by its deadline. At this time instant, task F_2 is released but is rejected immediately as it has no time to finish by the deadline. At time $t4$, F1 finishes. An optimal scheduler would have run H_2 as soon as it was released so that F_2 would be released and therefore would be able to compete with F_1.

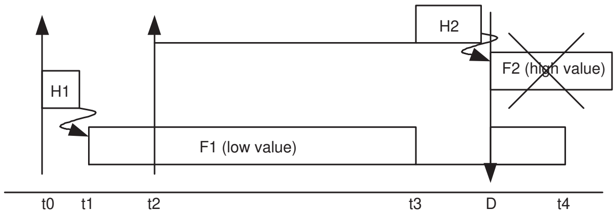

Figure 2. Effects of eager slack usage. High value task F2 can not run due to F1

The competitive factor α of a scheduling algorithm S is the ratio between the value that an optimal scheduler can achieve over the value achieved by S. Any scheduler which tries to make the maximum slack and use it immediately (including eager slack managers and the [25] model) can have a very low competitive factor. If $V_{\mathtt{max}}$ and $V_{\mathtt{min}}$ are the maximum value-density and minimum value-density of the optional components in the system respectively, in the worst case an eager on-line scheduling algorithm under overload may achieve as low as a $\frac{V_{\mathtt{min}}}{V_{\mathtt{max}}}$ competitive factor. As a non-clairvoyant scheduler could always select a low value-density optional component instead of a high value-density one, the ratio between the performance of the non-clairvoyant algorithm vs. the clairvoyant one is $\frac{V_{\mathtt{min}}}{V_{\mathtt{max}}}$.

The underlying problem is common in all "two-phase" schedulers in which they first make slack, and then, try to use it in the best way. Whereas any optimal scheduler should consider the value of the optional components potentially released by the hard tasks.

The conclusion is that trying to make the most slack available as soon as possible is not optimal for process models with dependent hard and non-hard tasks. Moreover, any approach which is two-phased can have the same potential problem. The fact that most slack stealing algorithms are computationally expensive (in time or size) does not help when considering them as an option for a flexible scheduling framework. These views are supported by simulation results shown in section 6.

4 'Lazy slack usage is too lazy'

The previous problem suggests an approach for scheduling. In order to obtain the maximum value, all optional components have to be ready for execution as soon as possible. Therefore an approach that runs all mandatory components first is sometimes advocated [9]. This approach effectively does not use slack at all. However, we already know that this approach is not optimal in the sense of maximising the total value achieved, and also it has a poor average case behaviour [3]. This is because it actually schedules any non-mandatory component as a background job.

This is very easy to see with an example. Consider a two task system with $T_1 = D_1 = 3$, and $T_2 = D_2 = 6$, with mandatory execution times $C_1 = 1$ and $C_2 = 2$. And Assume task 1 has also an optional component with $C_1 = 1$. Under a mandatory first approach, the total value achieved is 1/2 of the maximum achievable value because the optional component is discarded every other invocation. However the maximum possible value can be easily obtained by scheduling all components under EDF. A simulation of this problem is shown in section 6.

Actually, the process is not as dichotomic as it look like. Hard tasks do not necessarily have to be scheduled as"mandatory first" or "as late as possible". We can incorporate the value of the F tasks of the HF pair in the consideration of the scheduling decision. Of course, in a way that hard tasks are still always guaranteed.

In summary, neither aggressive slack techniques that push hard tasks as late as possible, neither mandatory first only approaches are adequate for scheduling flexible real-time systems. A more value-based oriented approach for the hard tasks as well with a mechanisms that guarantees the deadlines of the hard tasks is advocated.

5 'Early acceptance test is too early'

As we consider that the system is potentially overloaded (otherwise all components could meet their deadlines and maximum value could be easily achievable) there is the requirement for an acceptance test to keep the system underloaded.

Some scheduling frameworks do not use acceptance tests at all and allow all components in the system. However, running an optional task that fails to complete by the deadline leads to wasted computation time. By rejecting the tasks that will not finish on time the total value achieved is significantly increased. Also, being able to reject low value tasks also ensures a high total value achieved.

Several acceptance test models have been developed. The acceptance tests may consider, in their decision to accept or reject a task:

- Hard components already in the system.
- Hard components that will arrive in the future.
- Accepted components in the past (and that they have not finished yet).
- Optional components that will arrive in the future.
- Some general knowledge of the state of the system. (for instance, a fairness criteria for not penalising always the same task)

These tests employ schedulability tests (either response time based or utilisation based) and may be different when the system is underloaded, potentially overloaded and overloaded.

One issue of major importance is the cost of performing the acceptance test. The more accurate the test, the more computationally complex it will be. However, any accurate acceptance test needs to make a guess on the potential evolution of the system in the future. As the future is uncertain there is always some level of unpredictability on whether the decision of accepta nce/rejection of a given component is adequate.

5.1 Cone of uncertainty

One common approach for acceptance test is to assume a predictable execution of the components in the system (i.e. constant execution times equal to their worst case execution time, periodic arrivals, etc). However, as has been mentioned in the introduction, these assumptions although predictable are very unrealistic.

Our main hypothesis is that the future is uncertain. We can make a simple argument to roughly quantify the level of uncertainty in the actual execution of a real-time system:

- WCET: some authors report differences between average case behaviour and WCET of up to one order of magnitude. This can be attributed to high variations on execution times based on input data and difficulty of modeling real execution processors (i.e combined cache and pipeline analysis).
- In some systems the main contribution of the WCET of a task are their error recovery functions and/or initialisation code. Under normal operation these functions are seldom called.
- Sporadic tasks may arrive, in average, up to one order of magnitude slower than worst case assumptions would consider.
- Conditions may change drastically in the future, for example due to changes in the environment or mode changes.
- The execution time of the optional components and the number of iterations or sequences of optional components releases is not predictable (the whole idea of flexible scheduling is being able to include non-predictable components in the system!).

This shows that the longer we look in the future, the less predictable it is. We can depict this level of prediction of the behaviour of the system in the future as the *Cone of uncertainty*. We can visualise this cone of uncertainty graphically as shown in figure 3. Assume that we need to apply a decision on rejecting or accepting a task at time instant t_1 which has a deadline at D. This decision depends on the current state of the system, the operations that have just performed, but more significantly on what is left to execute. The remaining execution time of the current ready tasks and their deadlines, the tasks that will arrive until time D and their execution times. The longer D is in the future, the more events that can occur and therefore the less accurate the prediction is.

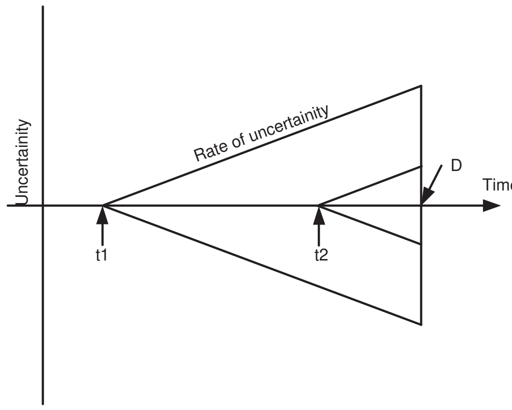

Figure 3. Uncertainity of the future. A decision taken at t_1 on the evolution of the system up to time D is less accurate than a decision taken at time t_2

5.2 Early vs. Late acceptance tests

An early acceptance test is the third obstacle to flexible scheduling. A late acceptance test is more adequate. By delaying the acceptance test as late as possible, the time window within which the schedulability is made is smaller and therefore the system is more predictable and the test more accurate. However, it is obvious that making the test too late (for instance at the deadline) is too late.

Ideally, it is not interesting to decide to reject a task after it has started running (using resources) as it leads to wasted computation time. Therefore, the latest time at which this test may be performed is actually at the time the task receives its first tick. We show this effect with some simulations in section 6.

Table 1. Example task set. Bounded exponential computation times [e:min,avg,max].

ID	T	C_{hard}	Y_i	$PLow$	$PHigh$	C_{Firm}	V_{Firm}
H1	60	3	57	14	1	N/A	N/A
H2	120	[e:5,8,10]	114	15	2	[e:10,50,60]	1000
H3	190	[e:5,10,14]	163	16	3	[e:10,50,60]	10
H4	200	[e:3,7,8]	165	17	4	[e:20,80,100]	5
H5	400	[e:5,7,12]	353	18	5	[e:20,80,100]	9
H6	500	[e:5,5,9]	430	19	6	[e:20,80,100]	5
H7	500	[e:5,5,8]	422	20	7	[e:20,80,100]	1
H8	600	[e:5,8,10]	512	21	8	[e:20,80,100]	15
H9	700	[e:5,9,12]	600	22	9	[e:20,80,100]	23
H10	3000	[e:5,9,11]	2889	23	10	[e:30,80,100]	19
H11	4000	[e:5,9,11]	3865	24	11	[e:30,80,100]	6

6 Evaluation

6.1 Simulation environment description

In order to illustrate and quantify the potentially negative effects of the three obstacles presented above, we have performed some simulations to address each of the issues in turn.

Table 1 shows the task set used for the simulations. It is made up of 11 hard tasks, 10 of which are HF tasks. The computation times follow a upper and lower bounded exponential distribution denoted by $[e : min, avg, max]$. The dual priority mechanism is used to guarantee the hard tasks deadlines with promotion times denoted by Y_i. Promotion times are computed assuming worst case execution times. Upon completion, the hard tasks release a Firm instance with its associated value. All firm tasks run at priority level 13 and are scheduled in EDF order (at that priority), higher priority levels run in fixed priority. The same task set is used for the different simulations with some slight variations. The average hard utilisation is 27%. The average load of the firm components is 190% which leads to a total utilisation of 217%. The graphs show the results for different total load varying from 90% to 350%. This is achieved by scaling the computation time of the firm components only. All simulations were performed for 1 million ticks, which showed stable results.

6.2 Slack usage

The eager slack usage is compared with the simple mandatory first approach. This is achieved by changing the scheduling policy of task HF_2 only. The results are shown in figure 4.

The task set uses dual priority scheduling as the slack management technique. The bottom line shows the percentage of the value achieved for different total loads. Both task sets are identical except for task HF_2. The bottom line uses

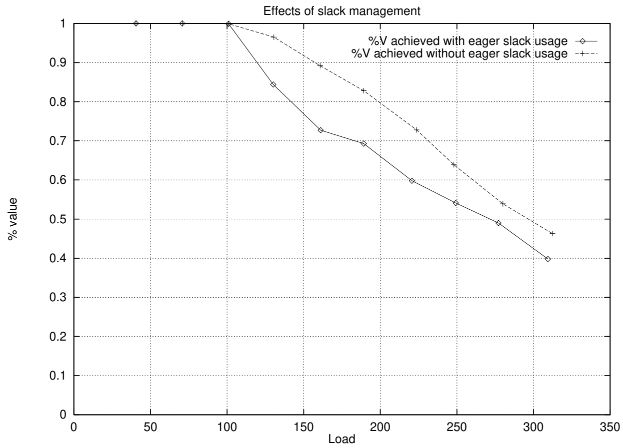

Figure 4. Eager vs. lazy slack usage effects

dual priority for all the tasks. The top line has disabled dual priority scheduling for the hard task $H2$ only. This is simply done by setting a priority promotion time of zero. No acceptance test is performed. This single operation leads to an increase in around 10% the total value achieved. This is amplified due to the high value assigned to F_2. However, running all tasks as mandatory first isn't good either as it is shown in the next experiment.

6.3 Mandatory first

The same task set is used to show how an "all mandatory first" only strategy is not optimal.

In figure 5, the bottom line represents the same task set with all tasks running as "mandatory first". The top line represents the task set scheduled with all tasks under dual priority. In this case, in the long term the aggressive slack usage technique provides a much better average response time for the firm tasks, and therefore more of them finish by

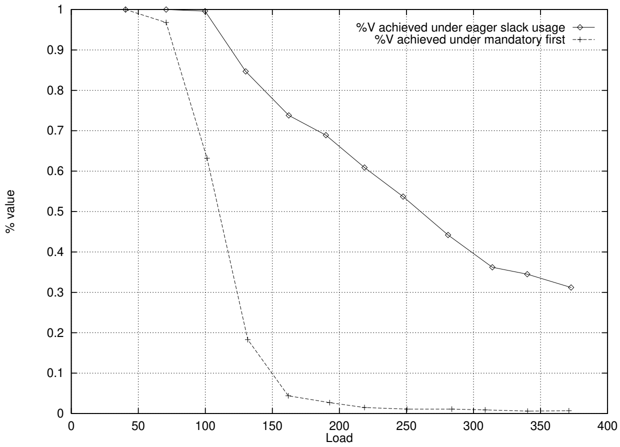

Figure 5. Mandatory vs. eager slack usage

the deadline. Thus achieving a higher value.

6.4 Early/late acceptance tests

Finally, we show in figure 6 the effects of the instant when the acceptance test is actually performed. This simulation uses the same task set as before with all tasks under dual priority, the only difference is that an acceptance test is used and that all hard tasks start executing at priority level 13, and therefore, initially scheduled under EDF. Note that hard tasks deadlines are guaranteed by the operation of the dual priority mechanism.

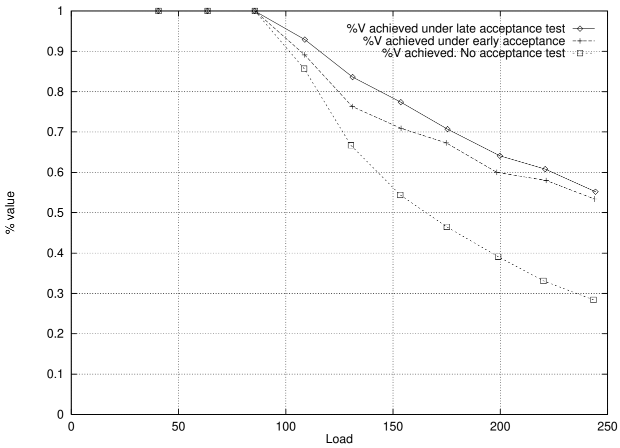

Figure 6. Early vs. Late acceptance tests

The acceptance test is an upgraded version of the best-effort queue reordering technique. At each acceptance test, all pending elements in the queue are tested to see whether they are schedulable assuming a worst case behaviour. If any of them is not schedulable, then all tasks are taken out of the queue and incorporated again one by one in value density order. The tasks which can not be scheduled are thrown out.

The only difference between the two top lines is when the test is actually performed, the top line represents the percentage of value achieved when the test is performed when the firm tasks actually get their first tick. The second lines corresponds to an acceptance test performed upon task release (i.e. completion of the hard task). For comparison purposes we also show the value achieved when no acceptance test is used. The performance of the late acceptance test is better than the early acceptance test for this particular configuration result.

7 Conclusion

The paper has described three common approaches in the context of flexible scheduling, namely, eager slack usage, mandatory first approach and early acceptance test which are shown to be inadequate.

Neither eager slack usage, nor mandatory first are optimal in the context of value based scheduling under the HF model. We have shown that these approaches are an obstacle to flexible scheduling and we argue that a unified model of scheduling hard and firm tasks needs to be used.

Flexible scheduling approaches are also characterised by their inherent unpredictability of computation times, arrival of events and environment changes. These sources of unpredictability result in early acceptance tests being less accurate that late acceptance test.

Some simulations have been shown that quantify the effects of such scheduling approaches.

References

[1] N. C. Audsley, A. Burns, R.I. Davis, and A. J. Wellings. Integrating unbounded software components into hard real-time systems. In S. Natarajan, editor, *Imprecise and Approximate Computation*, pages 63–86. Kluwer Academic Publishers, 1995.

[2] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Incorporating unbounded algorithms into predictable real-time systems. *Computer Systems Science and Engineering*, 8(2):80–89, 1993.

[3] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Optimal reward-based scheduling of periodic real-time tasks. In *20th IEEE Real-Time Systems Symposium. Phoenix. USA*, December 1999.

[4] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the

competitiveness of online real-time task scheduling. *Real-Time Systems*, 4(2):124–144, 1992.

[5] S. Biyabani, J.A. Stankovic, and K. Ramamritham. The integration of criticalness and deadlines in scheduling real-time tasks. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pages 152–169, 1988.

[6] P. Buneman, S.B. Davidson, and A. Watters. A semantics for complex objects and approximate queries. In *Proceedings of the 7th Symposium on the Principles of Database Systems*, pages 305–314, 1988.

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

[8] G. Buttazzo and J. Stankovic. RED : A robust earliest deadline scheduling algorithm. In *Proceedings of the 3rd International Workshop on Responsive Computer Systems, Austin*, 1993.

[9] Jen-Yao Chung, Jane W. S. Liu, and Kwei-Jay Lin. Scheuduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 39(9):1156–1173, 1990.

[10] T.L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

[11] A. Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, 1993.

[12] A.J. Garvey and V.R. Lesser. A survey of research in deliberative real-time artificial intelligence. *Real-Time Systems*, 6(3):317–47, 1994.

[13] E.D. Jensen, C.D. Locke, and H. Tokuda. A time driven scheduling model for real-time operating systems. In *Proceedings IEEE Real-Time Sytems Symposium*, pages 112–122, 1985.

[14] Chen K and P. Muhlethaler. A scheduling algorithm for tasks described by time value function. *Real-Time Systems*, 10(3):293–312, 1996.

[15] S. Koenig and R. Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms. *Machine Learning Journal*, 22:227–250, 1996.

[16] R. Korf. Real-time heuristic search: First results. In *Proceedings of the AAAI*, pages 133–138, 1987.

[17] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C.S. Yu, J.Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, pages 58–68, 1991.

[18] C.D. Locke. Best-effort decision making for real-time scheduling. CMU-CS-86-134 (PhD Thesis), Computer Science Department, CMU, 1986.

[19] S. Matsubara and T. Ishida. Real-time planning by interleaving real-time search with subgoaling. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 122–127, 1994.

[20] C. McElhone and A. Burns. Scheduling optional computations for adaptive real-time systems. *Journal of Systems Architectures*, 46:46–77, 2000.

[21] R. Oliveira and J. Fraga. Scheduling imprecise computation tasks with intra-task/ inter-task dependence. In *Proceedings of the 21st IFAC/IFIP Workshop on Real-Time Programming, WRTP'96*, pages 51–56, 1996.

[22] S.J. Russell and S. Zilberstein. Composinng real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217, 1991.

[23] W. K. Shih, J. W. S. Liu, and J. Y. Chung. Algorithms for scheduling imprecise computations with timing constraints. In *Proc. IEEE Real-Time Systems Symposium*, 1989.

[24] W. K. Shih, J. W. S. Liu, J. Y. Chung, and D. W. Gillies. Scheduling tasks with ready times and deadlines to minimise average error. *ACM Operating Systems Review*, 23(3):14–28, 1989.

[25] Too-Seng Tia, Jane Liu, and M. Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Journal of Real-Time Systems*, 1(21), 1996.

[26] S.V. Vrbsky and J.W.S. Liu. Approximate: A query processor that produces monotonically improving approximate answers. *IEEE Trans. on Knowledge and Data Engineering*, pages 1056–1068, 1993.

[27] S. Zilberstein. Resource-bounded sensing and planning in autonomous systems. *Autonomous Robots*, 3:31–48, 1996.

[28] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.