Scheduling Issues Arising from Automated Real-Time System Design

David J. Musliner
Institute for Advanced Computer Studies
The University of Maryland
College Park, MD 20742

musliner@umiacs.umd.edu
(301) 405–6761
FAX: (301) 405–6707

Revision : 4.1

ABSTRACT

We have developed CIRCA, the Cooperative Intelligent Real-time Control Architecture, as a mechanism for automating the on-line, adaptive design of real-time systems. The real-time control tasks generated by CIRCA do not meet many of the simplifying assumptions made to develop traditional scheduling algorithms. We examine several problematic issues that arise in scheduling these automatically-generated real-time monitoring tasks, and describe two solution approaches. In the first approach, a drastic reformulation of the scheduling problem leads to a new, highly effective scheduling algorithm, at the cost of synchronous periodic behavior in scheduled tasks. In the second approach, measured alterations are made to task periods by a band-limited search algorithm, seeking to find nearby periods with a tractable LCM, so that traditional schedulers may be applied. We compare and contrast these approaches using experimental results from both randomly-generated task sets and task sets generated automatically by CIRCA. Our results reveal significant advantages for the modified scheduling algorithm, which is based on specifying *invocation separations* (or "distance constraints") rather than periods.

Available as Technical Report CS-TR-3364, UMIACS-TR-94-118
University of Maryland Department of Computer Science

The work reported in this paper was supported in part by the National Science Foundation under Grants IRI-9209031 and IRI-9158473, by a NSF Graduate Fellowship, and by the Arpa/Rome Laboratory Planning Initiative (F30602-93-C-0039). The opinions, findings, and recommendations expressed in this publication are those of the author, and do not necessarily reflect the views of the funding agencies. David Musliner is also affiliated with the UM Institute for Systems Research (NSF Grant NSFD CDR-88003012).

1 Introduction

Traditional real-time scheduling is based on the assumption that task specifications will include at least two numbers: the worst-case execution time (WCET) of the task and the period with which the task must be executed. Recently, researchers have been flexing these requirements in a number of ways to accommodate the complexity, uncertainty, and dynamicity of real-world problems. For example, the requirement of a known WCET has been relaxed for imprecise computations [6].

In this paper, we focus new attention on the second numeric task specification, the task period. We show how, for a large class of real-world tasks, periods are not optimal specifications, and can be replaced with a less-constraining *maximum invocation separation* (or "distance constraint" [2]). We explore this observation in the context of the Cooperative Intelligent Real-time Control Architecture [7, 8]. CIRCA is a real-time AI architecture that combines positive aspects of both search-based AI planning technologies and hard-real-time scheduling and execution methods. In the next section, we present a brief overview of CIRCA, viewing the system as a mechanism for automating the process of designing real-time reactive systems. We then focus on the scheduling difficulties presented by CIRCA's automatically-generated control plans (Section 3), and discuss the techniques we have developed to address these problems. In addition to designing novel scheduling methods for these tasks (Section 4), we have also built tools that automatically adapt the tasks to work with existing schedulers (Section 5). In Section 6, we present experimental results showing that, even with these adaptations, traditional scheduling methods are significantly less effective on the tasks generated by CIRCA.

2 Overview of CIRCA

The increasing complexity of real-time computer control systems has motivated a growing interest in applying mature AI techniques to the control of real-time systems, in an effort to develop more intelligent, flexible automated systems [3, 9, 10]. One way to achieve intelligent real-time control is to use AI methods to automatically design real-time control methods on-line, dynamically altering the executing control system in response to changes in the system's goals or its environment [8]. Illustrated in Figure 1, CIRCA automates the entire process of building a real-time system, from planning tasks, to deriving their constraints, to scheduling them, and finally to executing them predictably. By automating the design and implementation process, CIRCA can be "intelligent *about* real-time." That is, CIRCA uses AI methods to dynamically and flexibly develop and modify its real-time behavior in the face of changing goals, capabilities, and/or domains.

CIRCA is initially given a specification of the system to be controlled, including: a set of initial world states, a set of state transitions that describe how the world can change, and a set of agent capabilities, describing how the agent can change the world. CIRCA is also given a description of the desired system behavior, which may include both goals of avoidance and goals of achievement.

Given these specifications, CIRCA's AI subsystem builds a reactive control plan using lookahead

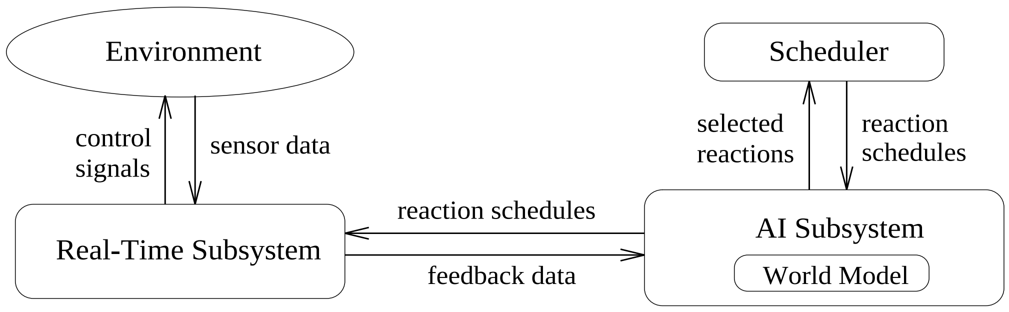

Figure 1: CIRCA, the Cooperative Intelligent Real-Time Control Architecture.

planning methods described fully in [8]. The planned reactions are cast in the form of simple Test-Action Pairs (TAPs) that specify the appropriate control actions for various possible future states of the world. Deadlines defined by the transitions in the world model are translated into response-time requirements for each TAP that is critical to system safety. Thus a TAP that responds to an emergency situation would have to be tested (and possibly activated) frequently enough to ensure that the emergency is recognized and dealt with before failure can occur. The Scheduler module then verifies that the planned TAPs can be executed successfully by the RTS, meeting all the timing requirements. Assuming that the Scheduler is able to produce a feasible schedule that meets all the timing requirements, the AIS can send the schedule to the RTS, which will then execute the TAPs in a fully predictable manner, enforcing the safety guarantees and behaviors specified by the planning system. It is also possible for the TAP design or scheduling phase to fail, indicating that resources are overconstrained, and some modifications must be made to the initial design or the specifications. Such modifications are essential to automating the overall design process and having CIRCA adapt to dynamic, unpredictable environments.

Space limitations preclude detailed descriptions of CIRCA's application domains, which have included an autonomous mobile robot and a simulated Puma arm [7]. This paper focuses on the scheduling phase: in what order should the RTS execute a set of automatically-generated TAPs?

3 The Scheduling Problem

CIRCA's AI planning system generates a set of non-interruptible TAPs which must be executed sequentially on a single processor. Each TAP has a known worst-case execution time requirement. All TAPs are known at the start of the scheduling session. TAPs fall into two classes; critical and not. All critical TAPs are responsible for meeting hard deadlines, so they *must* be included in any feasible schedule. Non-critical TAPs are considered optional, and may be executed periodically, sporadically, or not at all. The RTS can run these non-critical TAPs in an "if-time" manner, executing them only when slack time becomes available because some scheduled task has used less than its allotted time.

The CIRCA scheduling problem is not as complex as the problems addressed by some modern scheduling systems, but it has two unique aspects. First and foremost, the tasks being scheduled

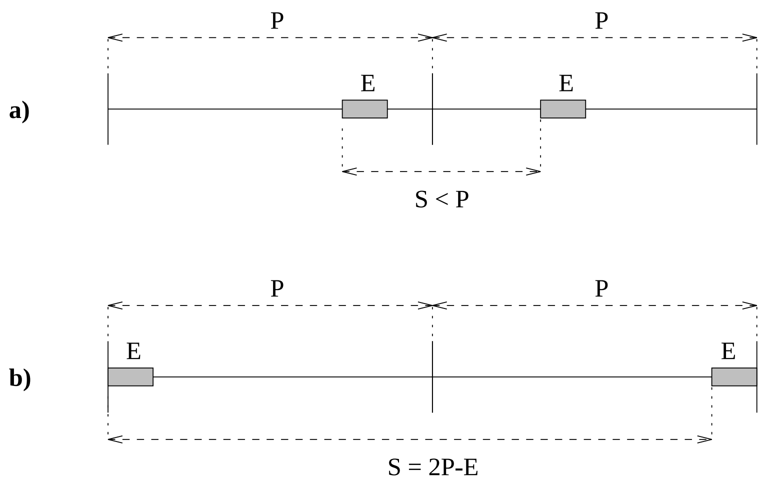

Figure 2: Possible relationships between task period (P) and invocation separation (S).

are automatically generated, so they are not as well-organized and optimized as human-generated tasks might be. One simple but confounding result is that TAP timing specifications do not fall on simple harmonic frequencies, so the least common multiple (LCM) of the TAP periods is generally extremely large. As a result, traditional schedulers that attempt to schedule calendars of task executions out to the LCM of the task periods (such as the Maruti scheduler [4]) will often be completely unable to deal with TAPs.

For example, suppose we have a set of five TAPs with specified periods of 1866, 617, 541, 411, and 250 time units respectively. The LCM of these periods is 10666566584250, meaning that at least 4×10^{10} invocations of the highest-frequency task would need to be enumerated to build a schedule out to the LCM. Clearly, this simple approach will not succeed.

To understand the second unique aspect of TAP scheduling, it will be helpful to make a distinction between a task's "period" and its "invocation separation"[1]. In the traditional interpretation of a task period P, future time is divided into a series of intervals, each P long. The task must be executed *exactly* once in each of these intervals. As a result, the separation between two task invocations may be much smaller or much larger than P, as illustrated in Figure 2. The maximum possible time between any two invocations of the task (S) is $2P - E$, where E is the execution time of the task.

The second special aspect of CIRCA's scheduling problem is that, instead of a period specification, each TAP is given to the Scheduler with a specification of the maximum acceptable invocation separation. Initially this may seem isomorphic with a period specification, differing only by simple linear factors of 2 and E. However, there are more subtle differences.

One of the primary differences relates to synchronization; when using a traditional period specification and scheduling one invocation per period, the invocations remain synchronized with

[1]A concept developed independently from, but essentially isomorphic with, the "distance constraint" of [2].

external clocks. That is, the tenth invocation of a task with a period of $P = K$ seconds will occur between $9K$ and $10K$ seconds after the schedule begins executing. This information can be useful if the task is communicating with or controlling an external process that has a similar periodic constraint: the respective invocations of each process will remain within some bounded distance of each other.

If the task is instead specified with a maximum invocation separation of $S = K$ seconds, then the only constraint on the tenth invocation of the task is that it occur before $10K$ seconds after the schedule begins executing. The invocations of the task may "skew" with respect to external clocks, so that synchronization is not possible. Lacking the lower bound, the invocation separation method provides less information but more flexibility in scheduling. As we shall see, this can lead to more effective scheduling algorithms, if the corresponding loss of synchronization is acceptable.

CIRCA specifies invocation separations because synchronous behavior is not necessary for the monitoring tasks it plans. Consider a monitoring system that must detect and react to some critical condition in the world. Obviously that condition must persist for some minimum amount of time in order to ensure that it can be detected. What constraints must be placed on a monitoring task to ensure that it detects a condition with minimum duration of C seconds? Simple: the monitoring task must be executed at least once every C seconds. Note, however, that there is no need to restrict the task to executing *exactly* once every C seconds; if the task executes more frequently, no penalty is incurred (except perhaps wasted computation). Thus a maximum invocation separation is the ideal specification for this type of monitoring task.

The reactive TAP plans built by CIRCA fit nicely into this model of monitoring. Because TAPs only invoke their actions in appropriate situations, there is no need to ensure that a TAP is not running faster than some rate; all TAPs may be executed repeatedly, as frequently as possible, and the system will continue to operate correctly. Thus there is no need for a lower bound on TAP invocation separation.

Because CIRCA's tasks have different timing specifications and, when converted to periods, these specifications have intractable LCMs, traditional scheduling mechanisms are not directly applicable. We have taken two approaches to addressing this inadequacy. In the first approach, we have developed a unique scheduling mechanism designed to take advantage of the increased scheduling flexibility associated with TAPs and their invocation separations.

4 Modifying Traditional Scheduling

The prototype Scheduler uses a modified deadline-driven scheduling algorithm [5, 11] to build a TAP schedule. Normally, this algorithm is used in the context of period-based scheduling, so that tasks are only considered "ready" for one invocation per period. However, the motivation for deadline-driven scheduling is not dependent on this aspect. As described in [11], Jackson's theorem justifies deadline-driven scheduling by noting that, if two tasks T_1, T_2 have respective execution times E_1, E_2 and deadlines D_1, D_2, where (without loss of generality) $D_1 \leq D_2$, then there are two cases of interest:

1. $E_1 + E_2 \leq D_1$: In this case, the tasks may be scheduled in either order.
2. $E_1 + E_2 > D_1$: In this case, T_1 must be executed first in order to meet its deadline.

Thus if we simply adopt the rule of executing the task with the earliest deadline (T_1 here), we will always find a feasible schedule if one is possible. Jackson's theorem is applicable to our scheduling problem because deadlines for each task are defined by the specified invocation separation and the previous task invocation.

To derive a cyclic schedule with this mechanism for choosing the next TAP to run, the Scheduler simulates the operation of a dynamic scheduler, incrementing a time counter and deciding which TAPs to run as simulated time passes. After the simulation has progressed far enough that all of the TAPs that must be scheduled have been invoked at least once, the Scheduler begins scanning the trace of the simulation, attempting to extract a loop of TAP invocations which meets all TAP timing requirements.

4.1 Modified Deadline-Driven Scheduling

The simple deadline-driven criterion for selecting the next TAP to run is optimal in the sense that, if any schedule is possible, this method will produce one. However, given the scheduling problem posed to CIRCA's Scheduler, the deadline-driven algorithm does not produce particularly efficient schedules. As a simple example, consider the problem of scheduling two TAPs, A and B, illustrated in Figure 3. If we use the trivial deadline-driven algorithm, the schedule of TAPs will have 11 invocations of TAP A, followed by one invocation of TAP B, and then repeat that pattern. This schedule is perfectly acceptable because it meets the frequency requirements for both of the TAPs. However, it is clearly not the shortest schedule that meets those requirements. In fact, the very simple schedule composed of alternating invocations of A and B also meets the requirements, and it is much shorter. This short schedule length is a major advantage from the CIRCA perspective, because the Scheduler is simulating this scheduling process forward to generate the appropriate loop of TAPs. The longer the loop, the longer it takes to generate, and the more resources that computation consumes. Therefore, we have modified the basic deadline-driven algorithm so that it will tend to produce shorter, more compact schedules.

The primary change to the scheduling algorithm is the addition of a second "level" of scheduling priorities, used to schedule TAPs that would not necessarily be chosen by the deadline-driven criterion, when slack time is available. Slack time is defined as the time between the current instant in the simulated schedule, and the latest possible start time of the TAP T_{DD} chosen by the deadline-driven criterion. If other TAPs can be found which fit within this slack time, they can be scheduled to run before T_{DD}. In terms of Jackson's theorem, this criterion detects when the

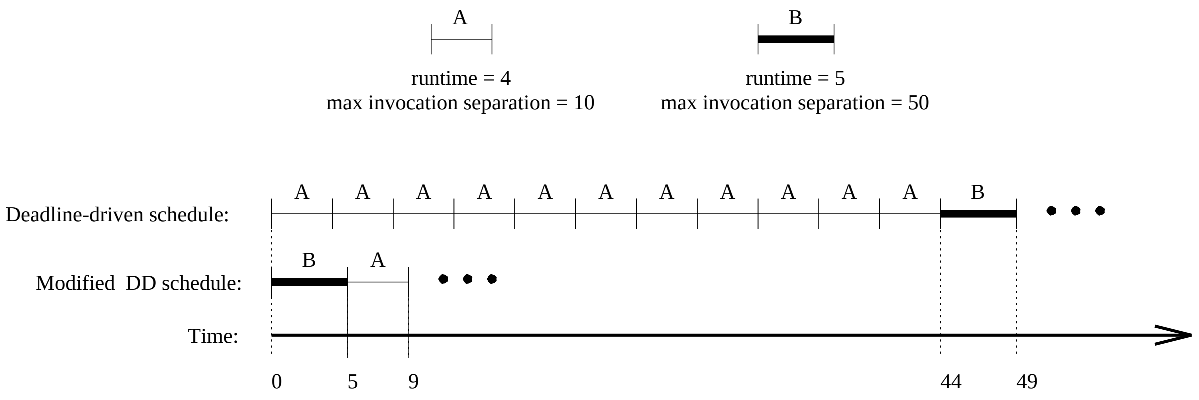
A
runtime = 4
max invocation separation = 10
B
runtime = 5
max invocation separation = 50
Deadline-driven schedule:
A A A A A A A A A A A B
Modified DD schedule:
B A
Time:
0
5
9
44
49

Figure 3: A simple example of how pure deadline-driven scheduling can produce undesirable, lengthy schedules.

first case ($E_1 + E_2 \leq D_1$) is occurring, and the tasks may be run in either order to still meet their deadlines. To take advantage of this situation, the Scheduler first finds T_{DD}, then finds the set of "feasible" TAPs whose runtimes will fit in the consequent slack time. If none are available, T_{DD} is chosen to run next. Otherwise, to maintain a "fair" distribution of invocations of the TAPs, the Scheduler chooses to run the feasible TAP which was least-recently invoked in the schedule so far. This has the effect of producing a modified round-robin effect, rotating the privilege of a slack-time invocation among the feasible TAPs.

With these modifications to the TAP selection criterion, the Scheduler is able to produce the second schedule shown in Figure 3. At time 0, the simple deadline-driven criterion indicates that T_{DD} is TAP A, because its deadline is 10, while TAP B has a deadline of 50. However, because A's runtime is 4, there are 6 units of slack time before it must be invoked. Since B's runtime of 5 fits in that slack time, the system selects B to run first. At the next scheduling point, time 5, T_{DD} is A again, but this time only 1 unit of slack time remains, B will not fit, and A is selected[2]. At this point, since both A and B have been scheduled, the system will begin scanning for acceptable loops in the schedule so far, and the simple loop BA meets all constraints.

4.2 The No-Holes Problem

Considered in isolation, this modified deadline-driven algorithm is quite successful at producing short schedules of guaranteed, critical tasks. However, it has one significant disadvantage when compared with traditional, period-based schedulers. Because a traditional scheduler only runs a task once per period, holes are left in the schedule, into which optional, if-time tasks (or even late-arriving critical tasks) may be fit on a dynamic basis. Our separation-based scheduler, on the other hand, leaves no unscheduled holes in the timeline; whenever the processor is available,

[2]Actually, even if B did fit it would not be selected here; the T_{DD} TAP is included in the least-recently-run round-robin, so A would be selected.

a critical TAP is selected to run. In systems where utilization levels are low and critical TAPs are being run more frequently than needed to meet their deadlines, it would be preferable to space their invocations out more and leave time for non-critical tasks. To address this concern, we further modified the Scheduler's operations to include non-critical TAPs.

4.3 The If-time Server TAP

If the Scheduler is able to produce a TAP schedule that includes all of the TAPs that must be guaranteed, it is possible that there are enough slack resources in the RTS to also guarantee some of the if-time TAPs, potentially speeding CIRCA's reactions. We have implemented an "if-time server" TAP, which tries to distribute the available slack time fairly amongst all of the if-time TAPs. When executed by the RTS, the if-time server TAP performs its own round-robin over the if-time TAPs. The server TAP is able to invoke any of the if-time TAPs because the server is specified with a WCET equal to the maximum of the WCETs of all the if-time TAPs. Deciding on a maximum invocation separation to assign to the server TAP is somewhat more difficult. Ideally, the if-time server TAP would be specified with an invocation separation that would cause it to be invoked frequently enough to use the slack resources, but not so frequently as to make a schedule impossible. Since it is not possible to directly determine this value, we have implemented a simple iterative heuristic to optimize the invocation separation assigned to the if-time server TAP.

As a starting point, the server TAP is assigned a separation equal to the length of the first schedule produced. If the Scheduler is able to produce a successful schedule with these constraints, the AIS then decreases the server TAP's invocation separation by some amount (currently by 25%), and repeats the scheduling process. This iteration terminates when the Scheduler fails, and the last successful schedule is restored and used. If the separation initially assigned to the server TAP does not result in a feasible schedule, the AIS can also increase the value iteratively for a few cycles, until a successful schedule is produced.

4.4 Discussion

We have shown how the prototype CIRCA Scheduler is able to produce cyclic schedules of TAPs that can be shorter than those built by simple deadline-driven scheduling, and how the if-time server TAP can allow the system to make guaranteed utilization of slack time. It is important to note that our modifications to the deadline-driven algorithm take effect only when the schedule utilization is fairly low; when the utilization is high, slack time is minimized and the second level of scheduling is never possible. As a result, our modifications alter the Scheduler performance for low-utilization domains, but in worst-case, high-utilization domains they have no effect, and the system defaults to pure deadline-driven scheduling (albeit not period-based).

Experiments using these mechanisms on hundreds of variations of the Puma domain have shown that the Scheduler produces efficient, short schedules very quickly, or else rapidly recognizes that a particular set of TAPs is not schedulable. Most Puma-domain schedules consist of between 10 and

35 TAP invocations, and are generated in a few seconds by an unoptimized Lisp implementation. Experimental results will be described in Section 6.

5 Adapting to Traditional Scheduling

A great deal of research effort has been invested in developing powerful scheduling methods based on the traditional interpretation of task periods. Therefore, although our invocation-separation scheduling methods are effective for the relatively simple tasks CIRCA derives, we are also interested in adapting these task specifications to be used with period-based schedulers. As described in the Introduction, the problem revolves around the task periods and their intractable LCMs.

To address this problem and allow CIRCA to use existing period-based schedulers, we have investigated several approaches to making small changes to the task periods in order to arrive at periods with a tractable LCM. There are two major constraints on these task period modifications. First, it is critical that the reactions planned by CIRCA are executed *at least* as frequently as the system originally specifies, since those specifications give maximum invocation separations that are needed to guarantee detection of (and reaction to) critical environmental events. Therefore, task periods (separations) may only be reduced, never increased, as an increase would make it possible for the system to miss critical events or violate deadlines. Second, when task periods are reduced, the overall load on the system is increased, because the given tasks must be executed more frequently. Thus it is important to minimize the changes to the task specifications, lest system utilization rise above the schedulable level.

5.1 Two-Factor Specialization for LCM Reduction

The problem of scheduling TAPs is closely related to recent research on scheduling for "pinwheel" [1] and "distance-constrained" tasks [2]. Two major differences are that TAPs have non-unit WCETs and are not preemptible. However, both Han & Lin [2] and Chan & Chin [1] have investigated "specialization" methods that can convert inconvenient task specifications into tractable ones. We can use these methods to modify TAP invocation separations to achieve more convenient LCMs[3].

Han & Lin describe specializations into a "two-factor" form where all distance constraints are reduced to values that can be factored into two numbers, one shared by all the specialized values (R) and one that is a power of two (i.e., 2^j). This yields specialized problem instances which fit nicely into the theory of pinwheel scheduling and the lower utilization ("density") bounds that have been derived for the schedulability of such problems.

However, this approach to specialization can lead to relatively large increases in the overall utilization of a task set. Although Han and Lin develop a method for finding optimal two-factor

[3]Although this is not the original purpose of their specialization methods.

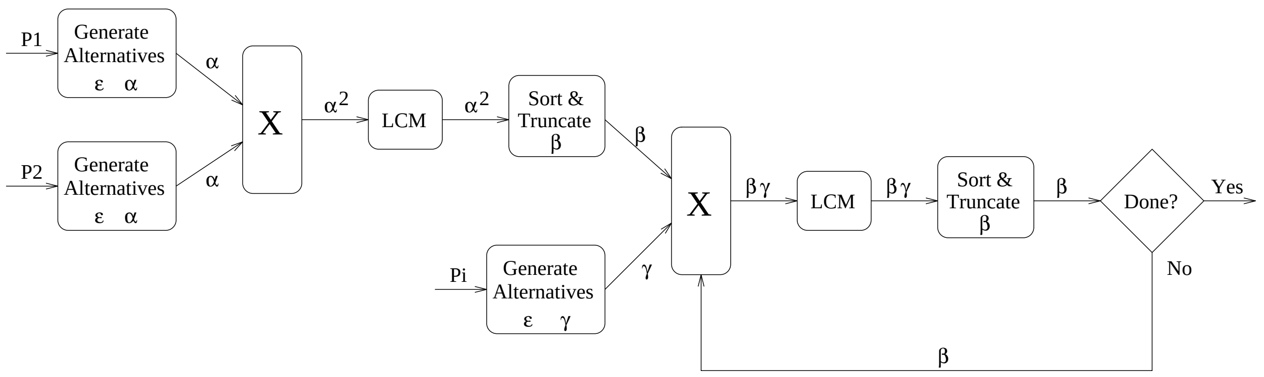

Figure 4: Flowchart of the LCM-reduction search. Line labels indicate the number of alternatives under consideration at each step.

specializations that minimize the resulting increase in utilization, the two-factor form itself is over-constraining. For example, our example tasks with separation specifications of {1866, 617, 541, 411, 250} are reduced to an optimal two-factor form of {1644, 411, 411, 411, 205.5}, requiring over a 33% decrease in the 617 specification.

5.2 Searching for the Least LCM

Following a more heuristic approach to addressing the LCM problem, we have developed an incremental band-limited search algorithm, illustrated by the flowchart in Figure 4, that can rapidly identify specializations that result in much smaller density increases than two-factor specializations.

The search algorithm operates by generating sets of alternative values for task specifications, using several input parameters to control this potentially explosive process. The ϵ value places a proportional limit on how much the system will modify the original specification, and α and γ limit the maximum number of alternatives generated. The system then computes the LCMs of all possible combinations of these specifications, discarding all but those with the β smallest LCMs. This process iteratively includes additional task specifications until all of the original ones have been considered.

While this brute-force approach may initially seem primitive and unlikely to succeed, in practice it is extremely effective at reducing the LCM for arbitrary sets of input periods. Given the periods of our running example and the parameters $\epsilon = .05, \alpha = 30, \beta = 10, \gamma = 100$, this search algorithm very quickly produces the set {1845, 615, 540, 410, 246} with maximum separation decrease of only 1.6%, and a tractable LCM of 22140. If α is increased to 70, the search will take slightly longer and will result in the even better set {1800, 600, 540, 400, 240}, which has an LCM of only 10800 with only 4% separation decrease.

Inserting this relatively simple search for a suitable LCM reduction allows CIRCA to invoke existing period-based schedulers on its automatically-generated sets of reactive tasks. In our experiments, we have used the scheduler built into the Maruti hard real-time operating system [4] to test these concepts and provide performance results for comparison with our separation-based scheduler.

6 Performance Comparison

To illustrate the comparative performance of these two approaches to scheduling, we will present results from tests run on large numbers of "artificial" task sets. In all of these experiments, the primary metric of concern is the percentage of task sets successfully scheduled by each method. We are also interested in the length of the schedules being produced, since this has an effect on the time needed to generate the schedule, the space needed to store the schedule, and the bandwidth needed to transmit the schedule to the RTS.

We are not particularly concerned with the relative speed of the different scheduling methods, for several reasons. Primarily, it is important to remember that CIRCA is designed to isolate the search-based reasoning processes of the AI planning system (and the Scheduler) from the hard real-time environmental interactions controlled by the RTS. As a result, even if the Scheduler and search-based LCM reduction algorithms require several seconds to find a schedule, this does not pose a threat to millisecond-scale reactions running on the RTS. Furthermore, compared to the search times required by the AIS to derive appropriate TAP plans, the scheduling time is negligible.

6.1 Random Task Set Experiments

To generate each artificial task set, a series of random values from 1 to 100 were generated to determine both task separation specifications and WCETs in accordance with a desired level of system utilization. These values were then scaled up by a factor of 10000, for compatibility with the minimum granularity of the Maruti scheduler.

Before the random task sets were passed to the Maruti scheduler, their periods were also reduced by as much as 5% (ϵ) using the search-based LCM-reduction method described above. Without that LCM reduction, virtually none of the random task sets could have been scheduled by Maruti, because their LCMs were all larger than the 32-bit integer range used by that system[4].

Figure 5 illustrates the performance results of the two schedulers on over 800 sets of five randomly-generated tasks with total system utilization levels ranging from 20% to 90%. CIRCA's separation-based scheduler is able to schedule a significantly larger percentage of these random task sets than the Maruti scheduler, even at higher utilization levels[5]. This result shows that the CIRCA scheduler is successfully taking advantage of the increased flexibility of separation-based task specifications.

[4]It is interesting to note that the problem of large LCMs has been so thoroughly ignored by real-time systems researchers that, in the original Maruti scheduler, there were no checks to make sure the LCM computation did not overflow its 32-bit integer representations.

[5]Note that there is no assurance that all of the task sets generated were theoretically schedulable— a 100% success rate is not necessarily possible on these task sets.

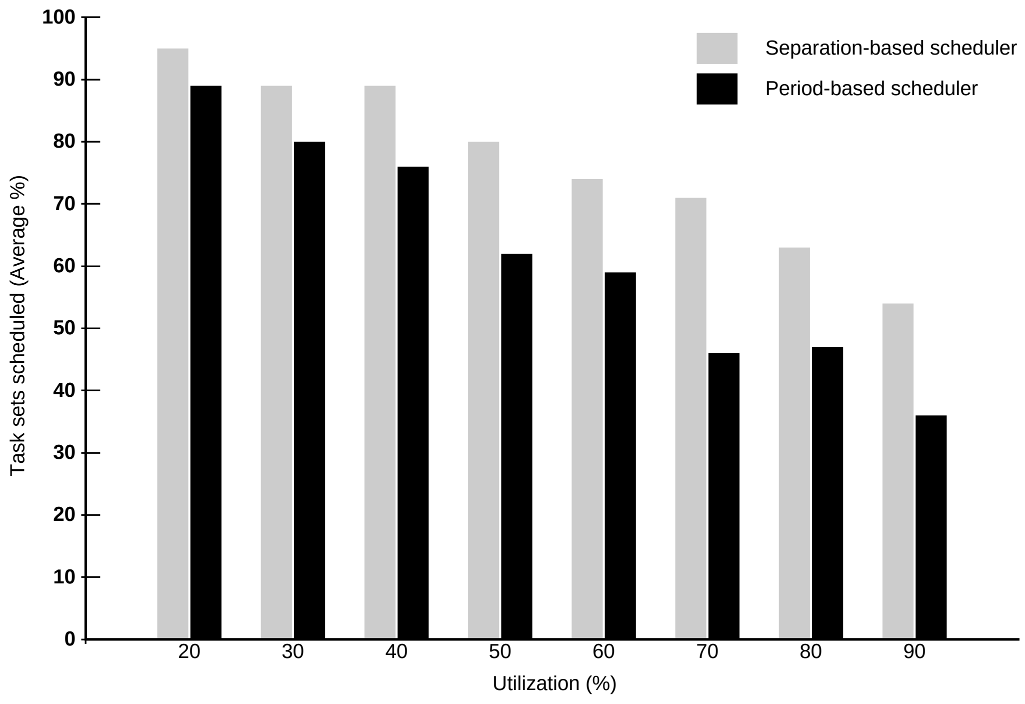

Figure 5: Scheduling results for randomly-generated sets of 5 tasks each.

Figure 6 shows an example five-task set with 80% utilization, for which both CIRCA and Maruti were able to produce schedules. The CIRCA schedule (without an if-time TAP server added) is only 7 task invocations long, while the Maruti schedule contains 275 invocations.

Figure 7 shows a similar five-task, 80% utilization set on which CIRCA's scheduler succeeded but Maruti's failed. In this case, even though the LCM was successfully reduced to a reasonable size with only a 2% increase in utilization, the Maruti scheduler was still unable to develop a successful interleaving. Close examination of a number of such cases seemed to indicate that the Maruti scheduler had the most trouble with task sets in which there was a wide variation in task periods. Presumably, the system would schedule many invocations of the higher-frequency tasks and then find that it could no longer fit in some lower-frequency task.

To test this theory that variance in task periods was causing trouble for the Maruti scheduler, we developed a second artificial task generation algorithm which is prone to choosing widely varying task periods. Rather than choosing utilization levels for each task in proportion to a set of previously-chosen random numbers, the algorithm instead assigns portions of the total remaining utilization sequentially, according to random percentages. As a result, if one of the first few tasks is generated with a high utilization percentage, the remaining tasks will all have small utilizations, usually growing smaller as each new task leaves less remaining utilization for consumption by the next.

Task	Worst-case Execution Time	Maximum Invocation Separation	Period	Reduced Period
1	150000	14710000	7430000	7140000
2	610000	4970000	2790000	2660000
3	210000	1870000	1040000	1020000
4	620000	4720000	2670000	2660000
5	840000	12300000	6570000	6460000

CIRCA schedule: (5, 1, 3, 2, 4, 1, 3)

Original LCM: 1400680953360000

Reduced LCM: 135660000

Maruti schedule length: 275 task invocations

Figure 6: An example task set with 80% utilization.

Task	Worst-case Execution Time	Maximum Invocation Separation	Period	Reduced Period
1	970000	9190000	5080000	5070000
2	590000	4050000	2320000	2210000
3	440000	117260000	58850000	57460000
4	440000	2360000	1400000	1360000
5	300000	17520000	8910000	8840000

CIRCA schedule: (2, 4, 3, 5, 1, 4)

Original LCM: 983153278800000

Reduced LCM: 689520000

Maruti scheduler failed.

Figure 7: Another example task set with 80% utilization.

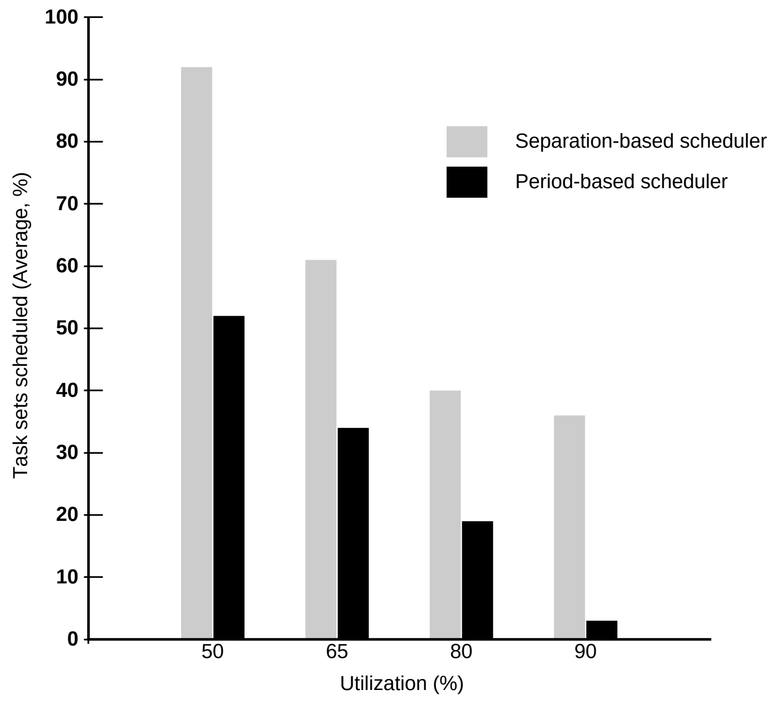

Figure 8: Scheduling results for randomly-generated sets of 5 tasks each, with widely varying task periods.

Figure 8 shows that the task sets produced by this different generator were much more difficult for the Maruti scheduler, resulting in as much an 80% decrease in performance, or a 30% drop in the absolute number of task sets scheduled. CIRCA's scheduler, on the other hand, was able to retain its excellent performance, suffering only a 30% decrease in performance at the 90% utilization level, and actually showing improved results on 50% utilization sets.

Experimental results on larger task sets also show that the CIRCA scheduler is more successful than the Maruti scheduler. Figure 9 shows the results for ten-task sets generated using the first random-number method. Comparing Figure 9 with Figure 5, we see that CIRCA's performance is similar, while the Maruti scheduler is uniformly less successful on the larger task sets. In fact, when the Maruti scheduler was run on ten-task sets from the second, high-variance random generator, it so rarely found a feasible schedule that the data is not worth plotting. The CIRCA scheduler, on the other hand, continued to display performance similar to the other graphs.

The MARUTI scheduler's performance degradation when faced with disparate periods and larger numbers of tasks is of particular interest because CIRCA tends to generate task sets with these characteristics. Scheduling experiments performed using task sets generated automatically by CIRCA showed even more clearly that traditional scheduling methods were inappropriate. CIRCA's scheduler was able to generate successful schedules in domains with 560% faster response-time

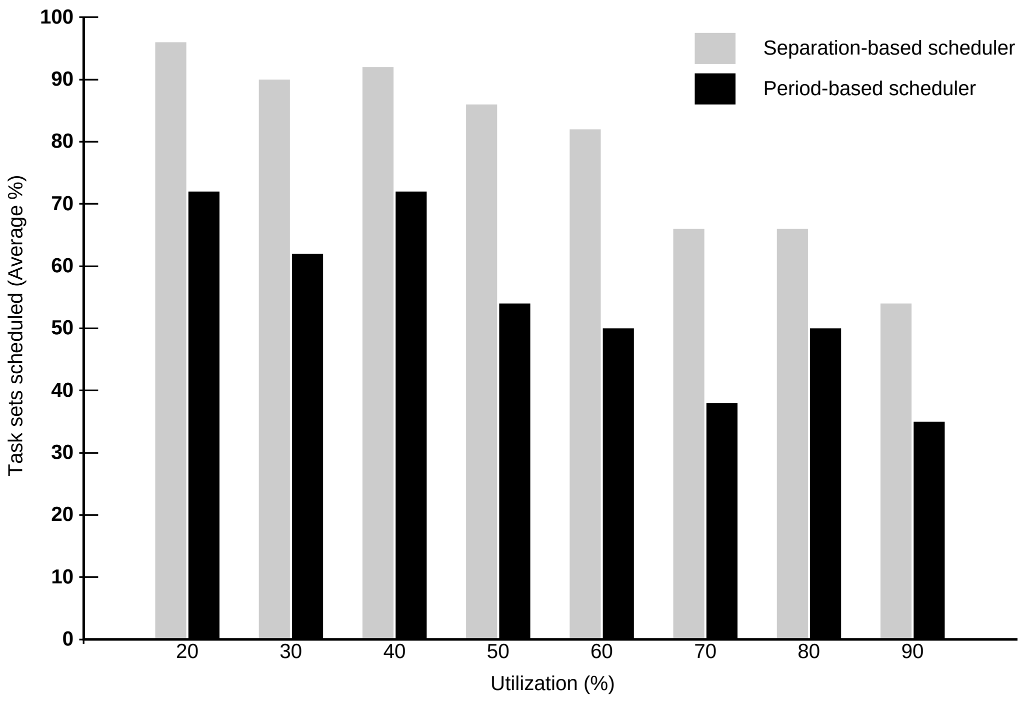

Figure 9: Scheduling results for randomly-generated sets of 10 tasks each.

requirements than Maruti (corresponding to higher system utilization).

7 Summary

Our experiences with CIRCA's automatically-generated real-time tasks bring into question some common assumptions of current RT research. In particular, we have seen that:

1. The specification of a task *period* is frequently overconstraining. Instead, *maximum invocation separation* specifications are well-suited to describing the timing constraints of reactive monitoring tasks.

2. Task timing specifications do not fall naturally on harmonic boundaries, and hence the LCM of a set of task periods is rarely tractable. As a result, scheduling methods that build "calendars" out to the major cycle (LCM) of the task set are impractical. We have developed an alternative calendar-building scheduler that avoids this problem.

3. Converting intractable task periods to values with a tractable LCM using existing "specialization" methods may yield very large increases in system utilization. We have developed a simple search-based algorithm that can rapidly generate high-quality specializations with much smaller utilization increases, hence improving schedulability.

The excellent experimental results to date are strong indications that our scheduling methods can provide significant performance improvements for real-time systems that do not require period synchronization conditions. Future work will involve developing the analytical basis for our modified deadline-driven scheduling algorithm based on maximum invocation separation specifications.

References

[1] M. Y. Chan and F. Y. L. Chin, "General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 755–768, June 1992.

[2] C.-C. Han and K.-J. Lin, "Scheduling Distance-Constrained Real-Time Tasks," in *Proc. Real-Time Systems Symposium*, pp. 300–308, 1992.

[3] T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao, and J. Y. Read, "Real-Time Knowledge-Based Systems," *AI Magazine*, vol. 9, no. 1, pp. 27–45, 1988.

[4] S. T. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala, "The MARUTI Hard Real-Time Operating System," *ACM Operating System Review*, vol. 23, no. 3, , June 1989.

[5] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[6] J. W.-S. Liu, K.-J. Lin, and S. Natarajan, "Scheduling Real-Time, Periodic Jobs Using Imprecise Results," in *Proc. Real-Time Systems Symposium*, pp. 252–260, December 1987.

[7] D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A Cooperative Intelligent Real-Time Control Architecture," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1561–1574, 1993.

[8] D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans," to appear in *Artificial Intelligence*, 1995.

[9] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, and J. K. Strosnider, "The Challenges of Real-Time AI," to appear in *IEEE Computer*, 1995.

[10] K. G. Shin and P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, January 1994.

[11] E. Walden and C. V. Ravishankar, "Algorithms for Real-Time Scheduling Problems," Technical Report CSE–TR–92–91, University of Michigan, Computer Science and Engineering, April 1991.