Response-time Calculation and Priority Assignment with Integer Programming Methods

Björn Lisper and Peter Mellgren
Dept. of Computer Engineering, Mälardalen University, Västerås, SWEDEN
bjorn.lisper@mdh.se, pmn99002@student.mdh.se

Abstract

Exact response-time calculation for cyclical tasks with fixed priorities requires that equations involving ceilings are solved. These equations are usually solved iteratively. For simple response-time problems with fixed priorities the iterations are guaranteed to converge if there is a solution. For more complex cases, however, like scheduling of real-time tasks in distributed systems, the iterative method does not always provide the best solution, in particular if the combined response-time/optimal priority assignment problem is considered. We show how to reformulate equations with ceilings into integer linear programming problems, which can be solved with known optimization methods. This reformulation is directly applicable to classical exact response-time calculation. We also show how optimal assignment of priorities can be incorporated in the model, which turns the optimization problem into a bilinear integer programming problem. A possible application is holistic scheduling with optimal priority assignments in distributed real-time systems.

1. Introduction

Fixed priority scheduling was introduced by Liu and Layland [7]. Exact response-time calculation for cyclical, independent tasks with fixed priorities was first formulated by Joseph and Pandya [5]. Since then, the model has been extended with task dependencies like blocking from tasks with lower priorities and release jitter. For instance, Audsley et. al. [2] have shown that in a uniprocessor system with fixed-priority tasks affected by blocking and jitter, the maximal response-time R_i for task i is given by

$$\begin{array}{rcl} w_i & = & C_i + B_i + \sum_{j \in hp(i)} \lceil \frac{w_i + J_j}{T_j} \rceil C_j \\ R_i & = & w_i + J_i^{max} \end{array} \qquad (1)$$

Here, for task i, T_i is the period, C_i is the maximal execution time, B_i is a blocking factor accounting for possible interference from tasks with lower priority, $J_i = J_i^{max} - J_i^{min}$ is the release jitter, and J_i^{max} (J_i^{min}) is the largest (smallest) delay from period start + release time until the task becomes ready. $hp(i)$ is the set of tasks with higher priorities than task i. Traditionally, these equations are solved iteratively. The procedure is guaranteed to converge if a solutions exists, although there is no upper bound for the time to converge.

The iterative method works when priorities are fixed. For simple systems of tasks there are methods to decide an optimal priority assignment without performing an exact response-time calculation, like rate-monotonic [7], deadline-monotonic [6] and for tasks with offsets (but no blocking factors) [1]. But there are cases where no good method for priority assignment is known. One such case is scheduling of tasks in distributed systems with networks with fixed-priority messages, sometimes called *holistic scheduling* [9].

This suggests that it might be worth looking for other methods to solve the equations arising in exact response-time calculation, in particular if they can be combined with a method to select priorities. We are investigating the possibility to use integer programming methods for this purpose. Here, we show how to reformulate the classical response-time calculation problem with fixed priorities into an integer linear programming (ILP) problem. The extended case, where priorities are not fixed, can be dealt with by expressing priorities through $0/1$-variables. The priority setting can be optimized together with the response-time calculation, as an integer programming problem where the objective function is quadratic.

2. Reformulating Almost-Linear Equations with Ceilings

(1) is an instance of the following equation:

$$x = \sum_{i=1}^{n} \lceil \frac{x + d_i}{a_i} \rceil c_i + b \qquad (2)$$

We now reformulate (2) into an ILP problem. The *least* solution of (1) gives the exact response-time. Thus, the task is to minimize x when (2) holds. Now, set

$$e_i = \lceil \frac{x+d_i}{a_i} \rceil, \quad i = 1, \ldots, n$$

Each e_i must be an integer. Therefore,

$$e_i = \frac{x+d_i}{a_i} + r_i, \quad i = 1, \ldots, n \tag{3}$$

where

$$0 \leq r_i < 1, \quad i = 1, \ldots, n \tag{4}$$

From (3) we obtain

$$x = a_i(e_i - r_i) - d_i, \quad i = 1, \ldots, n \tag{5}$$

and from (2)

$$x = \sum_{i=1}^{n} e_i c_i + b \tag{6}$$

We now use (5) and (6) to eliminate x, which yields

$$\sum_{j=1}^{n} e_j c_j + b = a_i(e_i - r_i) - d_i, \quad i = 1, \ldots, n$$

Solving this for r_i, and substituting into (4), gives

$$0 \leq a_i e_i - \sum_{j=1}^{n} e_j c_j - b - d_i < a_i, \quad i = 1, \ldots, n \tag{7}$$

Since $x = \sum_{j=1}^{n} e_j c_j + b$ we can now reformulate the task to find the least solution to (2) as "minimize $\sum_{j=1}^{n} e_j c_j$ under the constraints in (7)". This is an ILP problem in the n integer variables $e_1, \ldots, e_n$.

Exact response-time calculation for n tasks with priorities can now be reformulated into n independent ILP problems of the form (7), with the proviso that only the task itself and those with higher priorities will contribute with integer variables e_j. Thus, the exact response-time for the task with priority i can be calculated as the optimum of an ILP problem with i variables and $2i$ linear constraints.

3. Initial Experiments

We performed some initial experiments to see whether the ILP approach could compete with the traditional, iterative solution method for solving straightforward exact response-time calculation problems of the form (1). We used the ILP solver in the TOMLAB optimization environment [4]. This is a general purpose branch-and-bound ILP solver coded in Fortran. TOMLAB is a Matlab environment, with an interface to the solver. The iterative solver was also coded in Matlab. The experiments were run on a 350 MHz PII with 128 Mbyte memory under Windows NT 4.0, in Matlab 5.3. The results were however discouraging. For systems ranging from 6 to 25 tasks, the ILP solver was slower than the iterative solver with a factor roughly between 8 and 130, growing with the number of tasks. Therefore, although the ILP solution method clearly can be improved by using a specialized solver, we deemed it uninteresting to continue to work in this direction. Rather, we decided to look into how to extend response-time calculation by ILP into a method for optimal priority assignment.

4. Optimal Priority Assignment

We now describe how the ILP problem for finding response times for fixed-priority systems of tasks can be extended to an optimization problem where also the priorities are selected as part of the process.

Reconsider (2). For a system of n tasks with priorities, the exact response-time of task k will be decided by an equation of form

$$x_k = \sum_{i \in hp(k)} \lceil \frac{x_k + d_i}{a_i} \rceil c_i + b_k, \quad k = 1, \ldots, n \tag{8}$$

Rather than a set $hp(k)$ of tasks with higher priority than task k, we can express the priority relation with n^2 0/1-variables p_{ki}, where p_{ki} equals 1 exactly when $i \in hp(k)$. (8) can then be rewritten as

$$x_k = \sum_{i=1}^{n} p_{ki} \lceil \frac{x_k + d_i}{a_i} \rceil c_i + b_k, \quad k = 1, \ldots, n \tag{9}$$

Furthermore, b_k may in general vary with the setting of priorities, since it, according to (1), typically includes blocking factors from less prioritized tasks. Here, we assume[1] it is a sum of possible blocking factors b_{ik} from tasks i with lower priorities than task k. Thus, $b_k = \sum_{i=1}^{n} p_{ik} b_{ik}$.

We can now repeat the reformulation of (2) in Section 2. Introduce

$$e_{ki} = \lceil \frac{x_k + d_i}{a_i} \rceil, \quad i, k = 1, \ldots, n$$

These are n^2 nonnegative integer variables. As in Section 2, we obtain $2n^2$ inequalities, for $1 \leq i, k \leq n$:

$$0 \leq a_i e_{ki} - \sum_{j=1}^{n} p_{kj} e_{kj} c_j - \sum_{i=1}^{n} p_{ik} b_{ik} - d_i < a_i \tag{10}$$

[1]This assumption is somewhat unrealistic. For the priority ceiling and immediate inheritance semaphore protocols, the blocking factor is the maximum rather than the sum.

These inequalities are bilinear in the variables p_{kj} and e_{kj}, however restricted in that one factor always is a $0/1$-variable.

The priority variables p_{ki} are not independent: either task k has higher priority than task i, or the other way around. This is easiest expressed using boolean variables P_{ki}, where P_{ki} is true iff $p_{ki} = 1$. The following should hold, for all i, j, k:

$$\begin{array}{rl} P_{ii} = 0 & \text{(irreflexivity)} \\ P_{ij} \text{ XOR } P_{ji}, \quad i \neq j & \text{(total ordering)} \\ P_{ij} \wedge P_{jk} \implies P_{ik} & \text{(transitivity)} \end{array}$$

We reformulate the transitivity constraint as follows, in order to obtain linear constraints on the corresponding $0/1$-variables (see below):

$$\neg P_{ik} \implies \neg P_{ij} \vee \neg P_{jk}$$

These boolean conditions can be translated into linear constraints on the $0/1$-variables. There is a general translation from certain propositional formulae to constraints on the corresponding $0/1$-variables (X, Y are boolean variables, and x, y are the corresponding $0/1$-variables):

$$\begin{array}{rcl} X \vee Y & \rightarrow & x + y \\ X \text{ XOR } Y & \rightarrow & x + y = 1 \\ \neg X & \rightarrow & 1 - x \\ X \implies Y & \rightarrow & x \leq y \end{array}$$

We can thus translate the conditions on P_{ij} into constraints on p_{ij} (for $1 \leq i, j, k \leq n$, $i \neq j \neq k$):

$$\begin{array}{rcll} p_{ij} + p_{ji} & = & 1 & (11) \\ 1 - p_{ik} & \leq & (1 - p_{ij}) + (1 - p_{jk}) & (12) \end{array}$$

Summarizing, we have an optimization problem in n^2 non-negative integer variables and $(n-1)^2$ $0/1$-variables (eliminating the p_{ii}), with $2n^2$ bilinear inequalities (10) and $n(n-1) + n(n-1)(n-2)$ linear inequalities from (11) and (12). Half the $0/1$-variables can be eliminated using (11), and thus also these $n(n-1)$ linear constraints. The $n(n-1)(n-2)$ inequalities (12) are highly regular and can be generated on-the-fly as needed. Note, finally, that we can prescribe that task j must have higher priority than task i by setting $p_{ij} = 1$.

What are we optimizing? The response time calculations are now coupled through the linear constraints on the dependent priority variables. Thus, response times for individual tasks are not calculated in isolation anymore. One possibility is to minimize a weighted sum of the response times. This yields an optimization problem with a bilinear objective function. More interesting, perhaps, is to define a feasability test relative a deadline for each task. This leads to n more bilinear constraints of the form

$$\sum_{j=1}^{n} p_{kj} e_{kj} c_j + \sum_{i=1}^{n} p_{ik} b_{ik} \leq D_k, \quad 1 \leq k \leq n \qquad (13)$$

The problem is then to decide whether there are any feasible solutions. This can be done by an optimization algorithm for integer programming with bilinear constraints. Such algorithms exist and commercial solvers are available [3].

5. Optimal Priority Assignment in Distributed Systems

We now show how our optimal priority assignment method can be extended to distributed real-time systems with cyclic tasks with priorities. We will restrict our attention to distributed systems with networks that have fixed priority messages, like CAN. For such networks, exact response-time calculations for messages become very similar to response-time calculations for cyclic tasks with priorities [8, 10, 11]. We use the following equations for the maximal response-time R_i of CAN message i:

$$\begin{array}{rcl} w_i & = & B_i + \sum_{i \in hp(i)} \lceil \frac{w_i + J_j + \tau_{bit}}{T_j} \rceil C_j \\ R_i & = & w_i + J_i^{max} + C_i \end{array} \qquad (14)$$

Here, τ_{bit} is the time to transfer one bit on the CAN bus. Otherwise the notation is as in (1).

Now consider the following scenario. We have a system with processors connected through CAN buses. Tasks can communicate with tasks on other nodes, having the same time period, through CAN messages. During a single period, we allow finite acyclic chains of task-message communications. However, we allow a task to only receive and send a single message, respectively, per invocation.

We use the following notation. Messages are considered a special kind of tasks. All tasks (including messages) are globally numbered from 1 to n. Every processor and CAN bus is considered a *unit*. A priority variable p_{jk} is defined only when the tasks j and k share the same unit. If task i precedes task j, either by sending j as a message or by being a message received by j, then we define $prec(j) = i$. We also define powers of $prec$ through function composition in the standard way.

The major complication in the analysis, compared with the uniprocessor case, is that the jitter terms J_i become dependent on the the maximal response-time $R_{prec(i)}$ of the preceding task, if any. We assume the following:

$$\begin{array}{rcll} J_i^{max} & = & R_{prec(i)} + J_i^{max\_local} & (15) \\ J_i^{min} & = & R_{prec(i)}^{min} + J_i^{min\_local} & (16) \\ J_i & = & J_i^{max} - J_i^{min} & (17) \end{array}$$

Here, $R_{prec(i)} = R^{min}_{prec(i)} = 0$ if task i has no preceding task. We also assume that $J_i^{max\_local}$, $J_i^{min\_local}$, and $R^{min}_{prec(i)}$ are independent of the maximal response times.

We now derive a system of bilinear constraints, as in in Section 2. We have the following version of (5):

$$w_i + J_j + \tau_i = T_j(e_{ij} - r_{ij}), \quad 1 \le i \le n, j \in hp(i) \quad (18)$$

where $0 \le r_{ij} < 1$, and $\tau_i = \tau_{bit}$ if i is a message and $\tau_i = 0$ otherwise. We also have this version of (6):

$$w_i = \sum_{j \in hp(i)} e_{ij} C_j + b_i \quad (19)$$

Here, $b_i = C_i + B_i$ if i is an ordinary task and $b_i = B_i$ if it is a message. From (15) - (17) we obtain

$$J_i = R_{prec(i)} + J_i^{max\_local} - R^{min}_{prec(i)} - J_i^{min\_local} \quad (20)$$

From (1) and (14) we get

$$R_i = w_i + J_i^{max} + C'_i \quad (21)$$

where $C'_i = C_i$ if task i is a message and $C'_i = 0$ otherwise. Now assume that task i has N preceding tasks $prec(i), \dots, prec^N(i)$ in its communication chain. Expanding (21) in (20), and then (15) and (21) N times, we obtain

$$J_i = \sum_{j=1}^{N} w_{prec^j(i)} + K_i \quad (22)$$

where K_i is a constant. Substituting for J_i in (18) yields

$$\sum_{j=0}^{N} w_{prec^j(i)} + K'_i = T_j(e_{ij} - r_{ij}), \quad 1 \le i \le n, j \in hp(i) \quad (23)$$

We now eliminate each $w_{prec^j(i)}$ in (23) by substituting the right-hand side of its instance of (19). This yields

$$\sum_{j=0}^{N} \Big(\sum_{k \in hp(j)} e_{prec^j(i)k} C_k + b_{prec^j(i)} \Big) + K'_i = T_j(e_{ij} - r_{ij}) \quad (24)$$

for $1 \le i \le n$ and $j \in hp(i)$. Finally, we add priority variables p_{jk}, solve for r_{ij}, and substitute into the linear inequalities for r_{ij}. This yields a system of bilinear constraints in the same way as before.

6. Conclusions and Further Research

We have presented how to formulate exact response-time calculation and optimal priority assignment as a bilinear optimization problem. The approach works in theory also for optimal priority assignment in distributed real-time systems with fixed priority networks like CAN. What remains to do is experiments to verify that the approach is feasible also in practice.

A weakness in the current theory is the treatment of blocking factors. For common semaphore protocols the blocking factor is the maximum of the possible blockings from the tasks with lower priority. It is possible to reformulate maximum on a case-by-case basis, which leads to a number of bilinear systems to solve. Alas, the number of systems is $O(un^2)$ where u is the number of units in the system and n is the maximal number of tasks on a unit. Some other way must thus be found to deal with blocking factors of this kind.

7. Acknowledgments

We want to thank Kenneth Holmström for discussions on optimization methods and access to software for optimization, and Hans Hansson and Gerhard Fohler for discussions regarding task scheduling and priority assignment.

References

[1] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS-164, Dept. of Computer Science, University of York, England, Nov. 1991.

[2] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):226–292, 1993.

[3] K. Holmström. Personal communication.

[4] K. Holmström. The TOMLAB Optimization Environment in Matlab. *Advanced Modeling and Optimization*, 1(1):47–69, 1999.

[5] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[6] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, Dec. 1982.

[7] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. Assoc. Comput. Mach.*, 20(1):46–61, 1973.

[8] S. Punnekkat, H. Hansson, and C. Norström. Response Time Analysis under Errors for CAN. *Proc. IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 258–265, June 2000.

[9] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. *Microprocessing and Microprogramming*, 40(2/3):117–134, Apr. 1994.

[10] K. W. Tindell, A. Burns, and A. J. Wellings. Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[11] K. W. Tindell, H. Hansson, and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). *Proc. 15th IEEE Real-Time Systems Symposium*, pages 259–265, December 1994.