A priori task scheduling maximizing expected utility

Marius Minea*,**

* Department of Computing, "Politehnica" University of Timişoara
** Institute e-Austria Timişoara
marius@cs.utt.ro

Abstract

The paper addresses the problem of computing optimal schedules for a set of real-time tasks with hard and soft deadlines. The goal is to maximize a utility function based on task completion time, for expected task durations within given intervals. The approach allows an on-line choice between precomputed scheduling options at given time points, based on the actual elapsed execution time. The paper formalizes the problem, gives an expression for the optimal schedule and presents an algorithm to compute it.

Introduction Scheduling of applications which have both hard and soft real-time tasks appears in practice in a variety of applications, e.g., those involving multimedia, where besides guaranteed schedulability of critical tasks, some measure of quality of service is desired.

Schedulability of soft tasks has been studied in various contexts, often focusing on computations with non-essential subtasks, of which the goal is to execute as much as possible. The suggestion to use utility functions to represent the importance of tasks was first made in the thesis of Locke [5]. Burns et al. [2] present a framework for value-based scheduling and discusses different methods and types of value functions. Scheduling for hard and soft tasks in the context of multimedia systems is studied in [1]; here, the use of worst-case execution time for hard tasks vs. mean time for soft tasks is distinguished.

Our approach is inspired by and parallels that of [3]. Here, all tasks have execution times variable within an interval. Hard tasks have deadlines, whereas soft tasks have utility functions whose value is decreasing with completion time. While schedulability of hard tasks should be guaranteed even in the worst case, it is desirable to have an *expected* optimal utility. For the latter, we use a more accurate notion compared to [3, 4]: the expected value over the entire distribution of task durations, rather than merely the utility for the expected (average) duration of a task. As in the prior work, we consider that the entire task set executes once; this directly extends to a periodic task set.

Problem formulation We formalize the problem by considering a set T of tasks, with
– minimum and maximum durations $l(t), m(t) \in \mathbb{N}$ for each task in $t \in T$
– $d(t) = m(t) - l(t)$ the length of the possible duration interval for task $t \in T$
– $h(t) \in \mathbb{N} \cup \{\infty\}$ a hard deadline for each task; $h(t) = \infty$ if the task has no hard deadline
– a non-increasing utility function $u(t, \tau)$ for each task $t \in T$ and moment of completion $\tau \in \mathbb{R}$
– $c < 0$ and $d > 0$: the cost (negative utility) and delay of a one-time online scheduling decision

The approach to obtaining a maximal expected utility attempts to combine the benefits of static and dynamic scheduling. Whereas a static schedule minimizes runtime overhead, it needs to be conservative, scheduling hard tasks early enough to guarantee their worst-case on-time completion. However, if several tasks execute in less than maximum duration, enough slack may accumulate to allow inserting a soft task earlier on, increasing the total utility. Which subsequent schedule is optimal depends thus on the actual completion time of the already executed task subset.

**This research was partly sponsored by the Ministries of Education, Science, and Culture (BMBWK), and of Economy and Work (BMWA) of Austria under grants GZ 45.527/1-VI/B/7a/2002 and GZ 98.244/1-I/18/02.

The approach suggested by [3, 4] is then to allow a schedule with decision points: after having executed part of the task set according to a static schedule, the subsequent schedule is chosen from several precomputed options based on the current elapsed time. We associate a cost with this decision: a time delay ($d > 0$) as well as a negative utility ($c < 0$, e.g., by consuming some resources).

The goal is then to find both a set of optimal decision points as well as a set of schedules to select at each decision point based on the current elapsed time.

A formula for the maximal utility schedule With the above observations, we consider as preliminary step the utility value $U(t, T_r, \tau_s, \tau_w)$ obtained when scheduling task t before the remaining task set T_r, starting at time τ_s, but with schedules that are worst-case feasible even when starting at time $\tau_w \geq \tau_s$ (a partial schedule of tasks may start execution at an earlier time if the preceding tasks complete sooner than their worst case). We define $U = -\infty$ if the task set is not worst-case schedulable starting at τ_w. Our desired total maximum utility is then $\max_{t \in T} U(t, T \setminus \{t\}, 0, 0)$.

The above defined utility value can be computed recursively considering the following two cases:
i) The next task t' after t is chosen statically. Then, t' must maximize the expected utility value U, for a remaining taskset T_r starting at any time in $[\tau_s + l(t), \tau_s + m(t)]$, but worst-case schedulable if started at time $\tau_w + m(t)$ (for a maximal duration of t).
ii) There is a choice point after task t; the next task t' is chosen to maximize the expected utility of the remaining taskset T_r, with a known starting time ranging over the interval $[\tau_s + l(t) + d, \tau_s + m(t) + d]$. The value and time penalties c and d for inserting a choice point are taken into account.

The optimal expected utility is the highest of the two cases (with/without choice point after t):

$$U(t, T_r, \tau_s, \tau_w) = \frac{1}{d(t)} \int_{l(t)}^{m(t)} u(t, \tau_s + \tau)\, d\tau + \max(\max_{t' \in T_r} \frac{1}{d(t)} \int_{l(t)}^{m(t)} U(t', T_r \setminus \{t'\}, \tau_s + \tau, \tau_w + m(t))\, d\tau,$$
$$c + \frac{1}{d(t)} \int_{l(t)}^{m(t)} \max_{t' \in T_r} U(t', T_r \setminus \{t'\}, \tau_s + \tau + d, \tau_s + \tau + d)\, d\tau)$$

if $\tau_w + m(t) \leq h(t)$ and $-\infty$ otherwise. For simplicity, the above formula assumes evenly distributed task execution times within the given limits, but any distribution can be straightforwardly included.

For the base case of two tasks $T = \{t_1, t_2\}$ we obtain:

$$U(t_1, \{t_2\}, \tau_s, \tau_w) = \frac{1}{d(t_1)} \int_{l(t_1)}^{m(t_1)} [u(t_1, \tau_s + \tau_1) + \frac{1}{d(t_2)} \int_{l(t_2)}^{m(t_2)} u(t_2, \tau_s + \tau_1 + \tau_2)\, d\tau_2]\, d\tau_1$$

if (t_1, t_2) is schedulable at τ_w ($\tau_w + m(t_1) \leq h(t_1)$ and $\tau_w + m(t_1) + m(t_2) \leq h(t_2)$), else $-\infty$.

For example, if $u(t_i, \tau) = v_i - c_i\tau$, we obtain $U(t_1, \{t_2\}, t_s, \tau_w) = v_1 - c_1(t_s + m_1) + v_2 - c_2(t_s + e_1 + e_2)$, assuming schedulability at τ_w, and denoting $e_i = (l(t_i) + m(t_i))/2$.

The general formula above can be viewed as yielding both an expected utility function and the schedule(s) for which the optimum is obtained. If the maximum is the first term, the successor task t' of t is statically determined, independently of t's completion time. If the maximum stems from the second term, the choice point after t defines a *schedule tree*, with branches (schedule continuations) corresponding to different completion intervals for task t. Since U is a function of τ_s, it will itself have piecewise different expressions on subintervals for the starting time τ_s of the task subset.

It is thus natural to consider task utility functions $u(t, \tau)$ that are themselves piecewise defined on subintervals. It is easily seen that for polynomial utilities u, the degree of U in τ_s increases by one for each recursive application. An acceptable solution is to consider quadratic utilities u, and approximate U also by quadratic polynomials, preserving interval endpoint and average values.

Algorithm for optimal utility schedule The recursive utility formula suggests a dynamic programming approach, starting backwards with the scheduled last task. Each step increases by one the cardinality of the remaining task set for which optimal schedules for all possible starting times are computed. The final step produces the optimal schedule(s) for the entire task set, started at time 0.

We allow dependency relations between tasks, that condition the execution of a task by the completion of a subset of predecessors. Thus, in the backward computation we consider adding a task t to the remaining set T_r only if t has no successors outside T_r.

The optimal schedule and utility U is computed for the entire interval of possible starting times τ_s of the remaining task subset T_r, i.e., between the sum of the minimal and maximal execution

times of the already completed tasks, respectively. These schedules in turn may be parameterized (and thus split over intervals) by the worst-case starting time τ_w, for all values $\tau_w \geq \tau_s$.

The values carried around in the computation are representations of *schedule trees*. These contain a linear prefix of tasks, until the first choice point, and then a list of branches, one for each interval of completion times at the choice point. Each schedule is paired with its corresponding utility value (again, a function expressed piecewise over intervals, by means of its three polynomial coefficients).

When adding a new task to the schedule, the maximum of several piecewise utilities needs to be computed. All utilities which can be maximal for some value τ_s in the considered starting interval have to be maintained, potentially resulting in further interval splitting.

As shown in [3], the problem is worst-case exponential in the number of soft tasks. A related complexity problem is the potentially large number of branches in schedule trees, and the resulting split of the utility values into many intervals. Heuristics and approximations may be used for pruning the computation to its most promising branches.

Conclusions and future work We have presented a solution for scheduling systems composed of hard and soft real-time tasks to maximize a utility function based on task completion times, in a combined approach that allows on-line selection from a set of precomputed schedules. We have given a formula for the optimal utility, that compared to previous work accurately takes into account the distribution of task execution times.

Current and future work includes the evaluation of the presented algorithm; in particular, the limit to which exhaustive computation of the optimal schedule is possible, and the comparison of the obtained optimum with the results in [3, 4], where the computation is made directly with expected task durations, without considering their distribution. The next step consists of heuristics for making the approach scalable. One issue is limiting the number of candidates while building up the solution to a subset with highest expected value; another direction is limiting the search space by eliminating orderings of soft tasks that are provably suboptimal. Ultimately, the goal is to find a suitable application that demonstrates the usability of the approach.

Acknowledgements The author gratefully acknowledges being pointed to this problem by Petru Eles and Luis Alejandro Cortés during a visit at Linköping University in 2003.

References

[1] L. Abeni, G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. *Proc. Real-Time Systems Symposium*, pp.4-13, 1998.

[2] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. A. Stankovic, L. Strigini. The Meaning and Role of Value in Scheduling Flexible Real-Time Systems. *Journal of Systems Architecture*, 46(4):305–325, 2000.

[3] L. A. Cortés, P. Eles, Z. Peng. Static Scheduling of Monoprocessor Real-Time Systems composed of Hard and Soft Tasks. Tehnical Report, Linköping University, 2003.

[4] L. A. Cortés, P. Eles, Z. Peng. Quasi-Static Scheduling for Real-Time Systems with Hard and Soft Tasks. *Proc. DATE'04: Design, Automation and Test in Europe*, 2004.

[5] C. D. Locke. Best-Effort Decision Making for Real-Time Scheduling. Ph.D. thesis, Carnegie Mellon University, 1986.