

Statistical Quality Analysis of Schedulers under Soft-Real-Time Constraints

Hilbrandt Baarsma, Johann Hurink, Pierre Jansen

Universiteit Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands, h.e.baarsma@utwente.nl

This paper describes an algorithm to determine the performance of real-time systems with tasks using stochastic processing times. Such an algorithm can be used for guaranteeing Quality of Service of periodic tasks with soft real-time constraints. We use a discrete distribution model of processing times instead of worst case times like in hard real-time systems. Such a model gives a realistic view on the actual requirements of the system. The presented algorithm works for all deterministic scheduling systems, which makes it more general than most existing algorithms and allows us to compare performance between these systems. We show that the complexity of our algorithm is competitive with other algorithms that work for a wide range of schedulers.

1 Introduction

Many modern devices have to be able to process streams of data. These streams often consist of tasks arriving at regular intervals, where each task has to be processed within a fixed real time interval (RT). With the increasing use of RT systems, the techniques for building these systems have been described in many papers. In this paper we focus on single processor systems, but it is likely that the underlying ideas of these techniques can also be applied in a distributed RT environment. If several streams have to be processed on a single processor, there is a risk that some tasks may not complete before their deadlines. In hard real-time systems, this is not acceptable. However, with the increase of all kinds of multimedia devices, where occasionally missing a deadline would be tolerated, soft deadlines become more acceptable. In this context it is of importance to know how many deadlines are missed. Finding the number of missed deadlines is easy when dealing with deterministic processing times, but we run into problems when processing times are stochastic. Stochastic processing times are more realistic than using only worst case estimates. Using worst cases, may lead to an oversized system that, in reality, is idle most of the time. Therefore, if missing a deadline occasionally is acceptable, it becomes an important question how we can scale our system to achieve a performance level that we tolerate, in terms of missed deadlines.

Although fast algorithms exist to check if a given scheduling method will result in a deadline being missed, there are few efficient algorithms [1] [3] [4], to calculate the expected number of missed deadlines. This paper describes such an algorithm with three important features: it can take into account the effect of variation in processing times, it can compare hard vs. soft-real-time and it can compare a wide range of different scheduling techniques. The combination of these properties make it a valuable tool.

2 Analyzing performance using dynamic programming

We will use a dynamic programming algorithm to efficiently calculate the performance of a certain scheduling algorithm on a given set of tasks. We use the concept of states. Let $S_t = \{s_1^t, s_2^t, \dots\}$ be the set of all possible states at time t . Each state $s \in S_t$ contains a list Q_s and the probability p_s of being in this state. The list Q_s consists of tasks that still need to be processed, paired with their remaining processing time. If we do not allow preemption, then the state needs two extra

variables, containing the task currently being scheduled and its remaining processing time. To be able to evaluate how a scheduling method SM processes a task set Ω in a certain time period, we have to adjust every state, to reflect how the system evolves in time when scheduled by SM . For adjusting the states, the relevant events are when a new task enters the system or when a task reaches its deadline. In between two consecutive event times, t_i and t_{i+1} , no new states emerge. Only the values of the remaining processing times within the states change. Thus, the relevant times are given by the multiset $T = \{t_1, t_2, t_3, \dots\}$ of all deadline and release times. This multiset has the property that $t_1 \leq t_2 \leq t_3 \leq \dots$ and if times are equal, the deadline events are given before the release events. We define a function $F_{SM}(s_j^{t_i}, t_i, t_{i+1}) \rightarrow s_j^{t_{i+1}}$, that describes the change of the state $s_j^{t_i} \in S_{t_i}$ within the time interval $[t_i, t_{i+1}]$. The resulting state $F_{SM}(s_j^{t_i}, t_i, t_{i+1})$ belongs to $S_{t_{i+1}}$. Note, that the used function F depends on the scheduler used. If the set $S_{t_i} = \{s_1^t, s_2^t, \dots, s_k^t\}$ consists of all states at time t , we obtain $S_{t_{i+1}}$ by applying F_{SM} to all s_j 's from S_{t_i} . The first thing F_{SM} does is to calculate $t_{i+1} - t_i$ and, if this is not zero, it updates the state according to SM . This means that some tasks in the new state have less load than they had in $s_j^{t_i}$. If the event at t_i is a deadline of a task τ_j , F_{SM} checks if τ_j has any load left. If this is the case, task τ_j has missed its deadline, thus, the task is removed from Q_s and p_s is added to the expected number of errors for τ_j . If F_{SM} has been applied to all states in S_{t_i} , we check if there are any identical states in the resulting set $S_{t_{i+1}}$. If this is the case, we merge them, i.e. if the states s_1 and s_2 have $Q_{s_1} = Q_{s_2}$, we set p_{s_1} to $p_{s_1} + p_{s_2}$ and then delete s_2 from S . Two different states can end up being identical for several reasons. For example, during $[t_i, t_{i+1}]$ both states end up being idle, which happens when those states have all their tasks ready before their deadlines. How they reached this point is no longer important now.

If the event at t_i is a release event of τ_j , then the output of F_{SM} is not one state, but several states. Each output state corresponds to a different realization of the processing time C_j of τ_j and the sum of the probabilities corresponding to these new states is equal to the probability of the original state.

By using F_{SM} on all states at every event, we eventually reach the starting state again. At this point, the hyperperiod, we know for each task how many deadlines we expect to miss over time.

3 Complexity and Testing

In this section we derive the worst case time complexity of the method we introduced and compare it with the complexity of the method in [1], since this method is the closest to our own algorithm. We also describe some of the results of tests with our algorithm.

Since feasibility analysis of an arbitrary periodic task system is shown to be co-NP-hard in the strong sense [2], it is not likely that there is a way to get an exact performance measure in polynomial time, even more so when we are dealing with stochastic processing times. Our algorithm has a total complexity of $O(qnm^n)$, where n is the number of tasks, m is the maximum processing time over all tasks, while q represents the total number of deadline and release events.

If we compare this to the $O(q^3m^2)$ complexity of [1], we see that if the number of events, q , is high, our performance can be much better. This is because although [1] claims a polynomial time performance, it fails to take into account q is exponential in n . In situations where m or n is high, our algorithm might start to perform worse. This difference in complexity of our algorithm and the algorithm of [1] is explained by the fact that the latter has a smaller state space. In their algorithm, the processing times of separate tasks can be added because the algorithm has to do a separate calculation for each deadline, while in our algorithm, the results are calculated in one run.

We implemented our algorithm in C++ to see how it deals in practice with varying values of n , m ,

and other variables. We generated randomized task sets and tested how long our algorithm needs to complete a run. Our algorithm responds well to increases in m , showing an increase in runtime that is less than would be expected from the worst case complexity. Increasing n has a bigger impact on our algorithm's performance and task sets consisting of more than ten tasks may cause our algorithm to run several days on a normal desktop computer. These long run times depend heavily on the kind of scheduling used though. We used EDF and RM scheduling, both with and without preemption. Not using preemption has a significant positive effect on the complexity of our algorithm, since it reduces the size of the state space.

We were able to test our algorithm's versatility as well. For example we developed a simple modification to RM and EDF to better deal with overload situations. We showed it is very effective in reducing the expected number of deadline misses in overload situations. We also showed that using stochastic processing times to model a task's behavior leads to a much better estimate of the processor demand when compared to hard real-time and worst case estimates. With twenty percent less processing power we still make more than 99 percent of the deadlines.

4 Conclusion

Using methods derived from dynamic programming, allows to check effectively how many missed deadlines we may expect in a periodic system with soft real-time constraints. Although run times are still exponential in the worst case, this algorithm generally offers a significant boost in speed. Our algorithm can be modified to work with both hard and soft deadlines or deal with almost any kind of scheduling method. We have assumed discrete distributions of the processing times with a finite number of realizations. We believe these assumptions allows us to closely approach reality. Future research needs to be done, to see what kind of distributions are good approximations for the run times of different tasks.

References

- [1] Kanghee Kim and Jose Luis Diaz and Jose Maria Lopez (2005), An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations, *IEEE Trans. Comput.* **1**, 1460 – 1466.
- [2] Joseph Y.-T. Leung and M. L. Merrill (1980), A note on preemptive scheduling of periodic, real-time tasks, *IEEE Trans. Comput.* **1**, 115 – 118.
- [3] Mark K. Gardner (1990), *Probabilistic analysis and scheduling of critical soft real-time systems*, University of Illinois, Urbana Illinois.
- [4] T.-S. Tia and Z. Deng and M. Shankar and M. Storch and J. Sun and L.-C. Wu and J. W.-S. Liu (1995), Probabilistic performance guarantee for real-time tasks with varying computation times, *RTAS '95: Proceedings of the Real-Time Technology and Applications Symposium*