Improving Performance by Monitoring While Maintaining Worst-Case Guarantees

Syed Md Jakaria Abdullah Kai Lampka Wang Yi
Department of Information Technology, Uppsala University
Email: {jakaria.abdullah, kai.lampka, yi}@it.uu.se

***Abstract*—With real-time systems, feasibility analysis is based on worst-case scenarios. At run-time, worst-case situations are often very unlikely to occur. With the system being dimensioned for the worst-case, one faces low resource utilization and implicit loss in performance at run-time. We propose to use run-time monitoring for evaluating the deviation of job releases from their worst-case release bound. This allows us to compute a conservative bound on the future workload. Based on this, we design a scheme for reclaiming computation time, which has been originally allocated for jobs which are now known to be absent. By organizing the consumption of extra computing time in a dynamic and time-safe manner, we improve the run-time performance of applications and provably maintain the worst-case guarantees for their response times. We evaluate the usefulness of the presented approach by using randomly generated traces of job releases.**

I. Introduction

A. Motivation

Due to the significantly increased computing power of modern hardware, it can be believed that future (control) systems will have larger capabilities and be much more dynamic. This will lead to real-time systems which contain applications which have less regular arrival times of workloads, with the workload defined as number of job releases per fixed window of time. As an example for this one may consider a visual surveillance system. Modern video compression algorithms like the one of the HEVC video coding layer are based on inter-picture prediction of sequences of images. This significantly influences the amount of data as the complexity of movements of sensed objects varies or the movement of the sensing camera itself. Burstiness of workloads may have other reasons: (a) In cyber physical systems a computation is typically triggered by events of the (physical) environment, which can often not be predicted accurately, e. g., an autonomous car needs to react to objects the occurrence of which is unknown. (b) In distributed real-time systems, computations might be triggered by output events which are produced on other processing components. Variable processing times, communication delays, and interferences on shared resources make the prediction of precise triggering times extremely complicated if not impossible.

Complex task activation patterns commonly feature non-determinism to cope with unknown job release times. This comes at the price of severely overprovisioning computational resources, as real-time feasibility test [13] are based on worst-case assumptions like synchronous release of a maximum number of jobs. However, in most cases a real-time system will actually encounter much lower numbers of job releases and thereby be badly utilized. Exploiting the unused computation time in a timing-preserving manner is the contribution of this paper.

B. Own Contribution

This paper introduces a scheme for organizing slack reclamation for uniprocessor systems. Slack in the computation time stems from the absence of previously accounted releases of real-time jobs, where we make the extra computing time available in a timing safe manner. This preserves the feasability of a real-time system, i. e., the scheme does not introduce additional deadline misses.

Contrasting standard scheduling schemes and the common notion of free computing time, the scheme allows to consume the extra computing time even in the presence of not completed, i. e., pending real-time jobs.

The proposed scheme does not improve the schedulability of the deployed real-time system. However, its *early as possible consumption strategy of slack time* lowers the average response time of co-scheduled best-effort tasks.

C. Related Work

This work takes inspiration from real-time scheduling, specifically from request and demand bound functions [2], real-time servers [1], [5] and online slack reclamation schemes [7], [11], [12], [18], etc. However, unlike previous work in this area which is limited to either periodic or sporadic task activation patterns, we tend to extend slack reclamation for complex task activation patterns.

For modelling complex task activation patterns, we exploit arrival curves of the Real-time Calculus [17] or their pseudo-inverses [6]. Arrival curve based activation pattern modelling gives room for underspecification, as one abstracts over exact job release times. Moreover, arrival curves subsume standard activation patterns such as the model of strictly periodic job releases, the model of sporadic job releases and the one of periodic job releases with jitter and minimal distances (PJD-model) [16]. An example for the conversion of the later model into RTC arrival curves is discussed in [15].

For measuring deviations of actual job releases and their pre-defined bounding functions online, we exploit run-time monitoring of real-time task activations [4], [9], [14]. Specifically, we apply the scheme of dynamic counters of Lampka et al. [9].

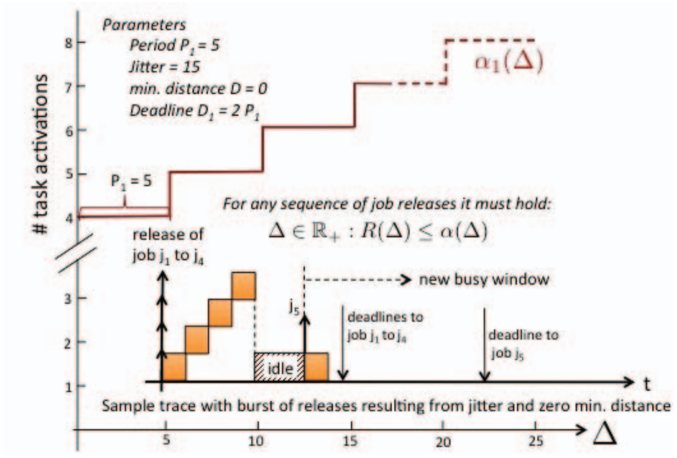


Fig. 1. Upper bounding of task activations and sample traces when following the upper bound

II. System model

We assume that a set of hard real-time tasks τ is mapped to uniprocessor. Each of the task $\tau_i \in \tau$ is equipped with unique worst-case execution time C_i and a relative deadline D_i. When a task τ_i is activated, we say that a job $j_{i,k}$ has been released, i. e., this job is the k'th invocation of the task τ_i. In case the processing of job $j_{i,k}$ has not been started, we say that the job (or task) is pending. In case the processing of job $j_{i,k}$ has started but has not terminated yet, we say that the job is not completed and completed otherwise. It is required that upon any activation of a task τ_i at time t the execution of the respective instance is completed at time $t + D_i$ latest.

For covering a wide range of job release patterns, this work assumes that *the job releases of a task τ_i are specified by a task-specific upper bounding function α_i*. The activation pattern of a task itself can be arbitrarily complex and does not need to be precisely known. The only thing required by the presented approach is that the the number of jobs released by task τ_i is bounded from above by a subadditive curve $\alpha_i : \mathbb{R}^+ \rightarrow \mathbb{N}_0$. In the following we denote this curve as release bound[1]. and require the following property to hold:

$$s, t \in \mathbb{R}_+ : 0 \leq s \leq t : \ \alpha_i(t - s) \geq R_i(t - s) \qquad (1)$$

with $R_i(t - s)$ as the cumulative counting function which reports the number of job releases of task τ_i in the interval $[s, t]$. For $s > t$, we define $\alpha_i(t - s) = R_i(t - s) = 0$. Note, there is an infinite set of traces of job releases described by release bound α_i, namely all traces for which Eq. (1) holds.

A. Example

An examples to the modelling of a job release pattern by means of an activation bound is provided by Fig. 1, where we illustrated the modelling of a PJD activation model. We have a period $P_1 = 5$, a relative deadline of each job which is twice the period, here $D_1 = 2P_1$, and for the WCET, we have $C_1 = 0.7$ units of time. As the jitter of the task activation model is 15 and the period is 5, the model includes bursts of job releases of size 5. With P_1 being an integer divider of the jitter, all of the staircases of α_1 have the same size, here 5. Otherwise, e. g., for a jitter of $J_1 = 16$, the first staircase would be of length 4 as $P - J \mod P$.

Besides the bounding curve, we also illustrates a sample trace of job releases. The trace is obtained by following the bounding curve, i. e., we see a bursty task activation pattern followed by a periodic job release sequence, where the first job outside the burst arrives after 5 time units and every other job after additional 5 time units. Note, there is an infinite set of traces described by an activation bound. This set includes all traces those number of job releases are bounded from above in the sense of Eq. (1) and for all points in time s and t with $t - s \geq 0$.

[1]In the context of RTC [17] the above α_i is called arrival curve. When weighted with the WCET of a task, the release bound resembles what is known in real-time scheduling as request bound function [2]

III. Time-safe Reclamation of Slack Time

A. History-aware bounding of job releases

The proposed scheme is based on monitoring of job releases as presented in [9].

With this scheme, job releases of a task τ_i are tracked with respect to a set of dynamic counters combined in a minimum computation. The current value of the j'th dynamic counter associated with task τ_i is stored by variable $DNC[i, j]$. This variable gives the number of jobs to be released instantaneously and is incremented each period $p_{i,j}$.

On the basis of dynamic counters we compute a time-variant bound on future job releases of task τ_i and for a time span $[now, now + \Delta]$:

$$\mathcal{F}_i(t, \Delta) \leq \min_{j \in J_i} \left(DNC[i, j] + \lceil \frac{\Delta}{p_{i,j}} \rceil + 1 \right). \qquad (2)$$

with J_i as index set of task τ_i for addressing its different dynamic counters. The above equation is a simplifying over-approximation of the bound presented in [9].

B. Main idea of the slack reclamation scheme

The busy period or busy window is defined as the maximum time span between two successive idling-phases of a processor which we denote T_{BW} in the following. Task invocations occurring prior to the idling time are not relevant for the work-conserving scheduling decisions and thus for satisfaction of deadlines. This restricts any feasibility analysis of real-time task set to consider at most sequences of job releases up to the maximum length of any busy window [10].

At analysis time, the worst case response time is constructed by assuming that job releases of task τ_i are bounded by the release bound α_i. This subsumes all placements of jobs within the busy window and such that $R_i(t - s) \leq \alpha_i(t - s)$ holds for all $t - s \leq T_{BW}$. With the starting of a new busy period, a history-aware refinement of α_i provided by $\mathcal{F}_i$ might indicate that there might be actually fewer task activations due to the specific release history of task τ_i.

The available slack time which we obtain from $(\alpha_i(\Delta = 0) - \mathcal{F}_i(t, \Delta = 0)) \cdot C_i$ cannot be arbitrarily consumed. To preserve feasibility of the system the slack time needs to be used in an EDF-conformant way. In addition, the consumed slack time and the time reserved for processing the potential jobs released by task τ_i needs to be strictly bounded by release bound α_i. This means that with only few job releases occurring while the busy period progresses slack time needs to be re-assigned to the task it originates from.

C. Example

For exemplifying the scheme, we consider the arrival curve illustrated in Fig. 1, i. e., an activation process with bursts of 4 and a period of 5 time units. Let the sample trace of Fig. 2 be decisive. The trace starts a time t_0 and such that t_0 is a renewal point[2]. Let the release of job j_1 be at time t_0 and the release of job j_2 be at time $t_0 + d$ as depicted also in Fig. 2. In addition, due to inactivity of other tasks, the processor is idling once it completed the processing of job j_1, here after time $t_0 + C_1$ as $t_0 + C_1 < t_0 + d$. Thus, job j_2 which is released at time $t_0 + d$ ends the idling of the processor and starts a new busy window.

When carrying out the feasibility analysis, it is assumed that the release history of a task is irrelevant when starting a busy period, i. e., all possible release patterns bounded by α_1 are possible. However, from the release history of the sample trace and the history-aware bound $\mathcal{F}$ it can be deduced that up to time $t_0 + P_1$, only another 2 jobs can arrive. This is because, with 2 jobs arriving in $[t_0, d]$ there is only room for another 2 jobs to be released in $[t_0 + d, t_0 + P_1)$ as $\mathcal{F}(t_0 + d, P_1 - d) = 2$. This contrast the assumption underlying the feasibility analysis. With the feasibility analysis, busy window and the task activation processes start synchronously such that feasibility is ensured for $\alpha_1(0) = 4$ being released at any time after time $t_0 + d$. With $\alpha_1(d) = 4$ and $\mathcal{F}(t_0 + d, P_1 - d) = 2$ this yields $(4 - 2)C_1$ units of computation time as extra computation time, as 2 jobs are guaranteed to not arrive in the time window $[t_0 + P_1 - d, t_0 + P_1)$ due to the release history of the task.

After time $t_0 + P_1$ and with no job actually being released, the slack time must be reduced by C_1 time units. This is because, in the time window $[t_0 + P_1, t_0 + 2P_1)$ an additional job can arrive at any time, i. e., $\mathcal{F}(t_0 + P_1, P_1) = 3$ and with $\alpha_1(P_1) = 4$, this yields $(4 - 3)C_1$ units of extra computation time, opposed to the $(\alpha_1(P_1) - 2)C_1$ units of extra computation time which have been available in the preceding interval $[t_0 + P_1 - d, t_0 + P_1)$. In other words, as one job has been released at time $t_0 + d$ and 3 can potentially arrive, it is guaranteed that only 1 of the assumed $\alpha_1(P_1) = 4$ jobs cannot arrive.

For the same reason, the slack time needs to be re-adjusted for the next period starting at $t_0 + 2P_1$. This is because, with no additional job release up to time $t_0 + 2P_1$, the history-aware bound $\mathcal{F}$ coincides with the static bound α_1.

[2]A renewal point is a point in time, where the past behaviour turns irrelevant for the future task activations, i. e., the full range of task activations bounded by α_1 is possible [9].

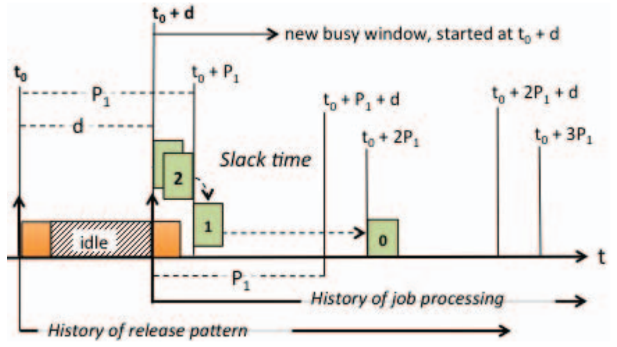


Fig. 2. Sample trace and slack time stemming from variations of job releases

Task	Period	Jitter	Delay	WCET	Relative Deadline
1	20	2	2	2	20
2	40	5	2	2	40
3	25	60	1	1	25
4	50	20	1	2	50
5	35	90	2	8	35

TABLE I
TASKSET WHERE TASK PARAMETERS ARE SPECIFIED IN MILISECONDS

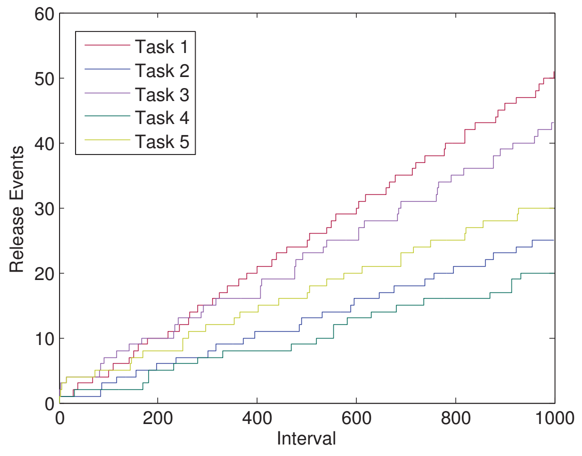
IV. EMPIRICAL EVALUATION

In this section we present an empirical evaluation of our slack reclamation scheme. As part of the evaluation, we implemented a trace generator and an online EDF scheduler with our slack reclamation scheme using Matlab scripting language.

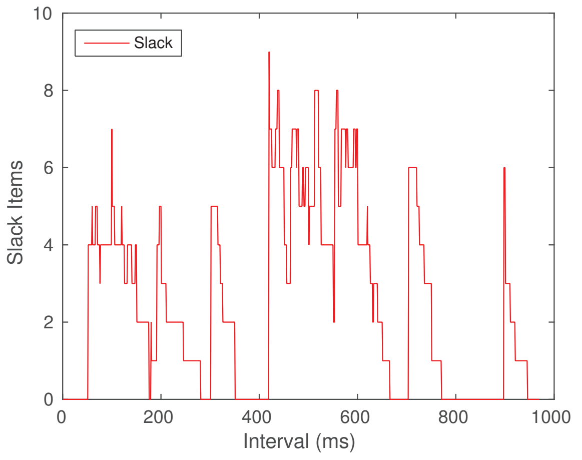
Our trace generator accepts task specified in PJD model together with relative deadline and WCET of it. Generation of the trace is inspired by [8], which uses a state machine to switch between two modes, namely eager and lazy mode. In eager mode, the generator tries to generate events following upper bound of the job releases. In lazy mode, the trace generator refrains from generating events unless it is going to violate the lower bound[3] of the job releases in next time instant. Unlike [8], we used a single user provided parameter M to randomly switch between eager and lazy mode. The switching time is uniformly sampled from the interval $[0, M \cdot P]$ where P is the task period. As a result, to get large variation in switching times we can use large values of M.

For the example case study, we generate trace from the taskset given in Table 1. Task 1 and 2 in this taskset have low jitter while the other three tasks have relatively large jitter in their activation pattern. For generating trace, we set M to be small for task 1 and large for other 4 tasks. The generated trace shown in Fig. 3(a) has both periodic activation pattern (task 1) and bursty activation pattern (rest of the tasks) in the same trace. We run this trace with an EDF scheduler modified with our slack reclamation mechanism (dynamic counters and refill) and record slack elements generated by the trace. In Fig. 3(b) we can see the slack generated by the trace Fig. 3(a). Interesting observations are: (1) maximum amount of slack is reclaimed when the schduler faces frequent idle periods. For our trace it is in the interval between 350 to 450 milliseconds when there are very few job activations. (2) Inventory of

[3]Number of RTC job releases are bounded form above and below [17].



(a) Job release events in a random trace of the example taskset



(b) Slack Time obtained when job releases follow the trace of Fig. 3(a)

Fig. 3. Example run of slack reclamation

reclaimed slacks are decreased by dynamic counter updates and if not used slack queue will return to the empty state after certain period of time. All the observations conform that our mechanism can correctly reclaim activation slacks during run-time if it is available in the corresponding run.

Our scheme utilizes start of a busy period as the instant of reclaiming activation slacks. It is possible that a workload has very long busy periods during run-time and its activation is strictly periodic. In such cases, our scheme is pessimistic as it depends on variations of the job releases. However, when the run-time activations do not conform worst-case activations, our scheme can reclaim considerable amount of slacks as shown in the example.

V. Conclusions and Future Work

We presented a dynamic slack reclamation scheme using run-time monitoring of job releases. Our scheme predicts availability of activation slacks at the beginning of a busy period thus allows best-effort jobs to consume it as early as possible. This can improve quality of service for best-effort jobs by improving their responsiveness. Another possible application of activation slacks can be graceful degradation of low-criticality jobs in mixed-criticality systems [19] or jobs with weakly-hard real-time constraints [3]. In future we would like to investigate usefulness of our scheme in the above mentioned areas and compare it's performance with the state of the art graceful degradation mechanisms.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *RTSS*, pages 4–13, 1998.
[2] S. K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Syst.*, 24(1):93–128, Jan. 2003.
[3] G. Bernat, A. Burns, and A. Llamosi. Weakly hard real-time systems. *IEEE Trans. Comput.*, 50(4):308–321, Apr. 2001.
[4] F. Bodmann, N. Muehleis, and F. Slomka. Situation aware scheduling for energy-efficient real-time systems. In *Proceedings of the 8th Workshop Cyber-Physical Systems - Enabling Multi-Nature Systems*, 2011.
[5] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Real-Time Systems. Springer-Verlag, Santa Clara, CA, USA, 2011.
[6] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *IEE Proceedings Computers and Digital Techniques*, 2005.
[7] R. Jejurikar and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the 42nd annual Design Automation Conference*, pages 111–116. ACM, 2005.
[8] S. Kuenzli and L. Thiele. Generating event traces based on arrival curves. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2006 13th ITG/GI Conference*, pages 1–18, March 2006.
[9] K. Lampka, K. Huang, and J.-J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *Proceedings of the 8th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011*, pages 267–276, Taipei, Taiwan, Oct 2011. ACM.
[10] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209. IEEE, 1990.
[11] J. Lehoczky. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *RTSS*, pages 110–123, 1992.
[12] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *ECRTS*, pages 193–200, 2000.
[13] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.
[14] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring arbitrary activation patterns in real-time systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, Dec 2012.
[15] S. Perathoner, K. Lampka, and L. Thiele. Composing heterogeneous components for system-wide performance analysis. In *Proceedings of Design, Automation and Test in Europe, 2011 (DATE 11)*, pages 1–6, Grenoble, France, Mar 2011. IEEE.
[16] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 287–292, New York, NY, USA, 2002. ACM.
[17] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. Intl. Symposium on Circuits and Systems*, volume 4, pages 101–104, 2000.
[18] T.-S. Tia, J.-S. Liu, and M. Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Real-Time Systems*, 10(1):23–43, 1996.
[19] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, pages 239–243, 2007.