IEICE TRANS. INF. & SYST., VOL.E101–D, NO.8 AUGUST 2018

LETTER

Proof and Evaluation of Improved Slack Reclamation for Response Time Analysis of Real-Time Multiprocessor Systems

Hyeongboo BAEK[†], ***Member***, **Donghyouk LIM**[††], ***and*** **Jinkyu LEE**[†a)], ***Nonmembers***

SUMMARY RTA (Response time analysis) is a popular technique to guarantee timing requirements for a real-time system, and therefore the RTA framework has been widely studied for popular scheduling algorithms such as EDF (Earliest Deadline First) and FP (Fixed Priority). While a number of extended techniques of RTA have been introduced, some of them cannot be used since they have not been proved and evaluated in terms of their correctness and empirical performance. In this letter, we address the state of the art technique of *slack reclamation* of the existing generic RTA framework for multiprocessors. We present its mathematical proof of correctness and empirical performance evaluation, which have not been revealed to this day.

key words: *response time analysis, real-time multiprocessor systems, slack reclamation*

1. Introduction

Real-time systems have been widely used in avionics, automobile, and medical devices, in which violation of timing requirements may result in a catastrophe [1]. To this end, the real-time systems community has developed various techniques for guaranteeing timing requirements, and RTA (Response Time Analysis) is one of the most popular techniques. Starting with FP (Fixed priority) [2] on uniprocessors [3], the generic RTA framework has been extended to multiprocessor systems for popular scheduling algorithms [4]–[6].

Slack reclamation is a novel technique, which is a part of the generic RTA framework to improve analytic capability by exploiting a slack value of each task (defined as the difference between the response time and the relative deadline). There have been two different slack reclamation strategies (forward and backward ones) for multiprocessor systems, which exploit a slack value of each task in opposite principles. While the forward strategy proposed a decade ago [4] scans the slack values from zero to the largest feasible value, the backward strategy introduced recently [7] investigates the values on the other way round. Although the forward strategy was exploited and studied in several studies [4]–[6], the backward strategy (which dominates the forward one) has not been identified in terms of its correctness and empirical performance to this day, and thus it has not been used.

In this letter, we focus on the backward strategy, and aim at identifying its correctness and empirical performance. We first prove the correctness of the backward strategy (Sect. 3). Then, we perform a case study, which applies the RTA framework with the backward strategy to EDF (Earliest Deadline First) [2], and demonstrate the effectiveness in terms of schedulability (Sect. 4). According to our case study, the backward strategy improves performance of the forward strategy up to 37%. We emphasize that the applicability of the backward strategy is not limited to EDF; it can be used for any existing scheduling algorithm whose RTA has been developed. Therefore, proof and evaluation of the backward strategy to be done in the letter makes it possible to further improve a number of existing scheduling algorithms, which indicates significance of our study. The contribution of this letter is summarized as follows: Formal proof of the correctness of the state of the art slack reclamation technique of RTA, and demonstration of its empirical effectiveness with a case study.

2. System Model

We consider a set of sporadic real-time tasks $\tau_i \in \tau$, which is expressed by $\tau_i(T_i, C_i, D_i)$, denoting the minimum separation (or period), the worst-case execution time, and the relative deadline, respectively [8]. We consider constrained-deadline tasks, in which $D_i \leq T_i$ holds. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units, executed at most C_i time units, and finished within D_i time units after its release. Without loss of generality, we let a quantum length be one time unit, and therefore all task parameters are integer values.

We consider a multiprocessor system consisting of m identical processors. We consider global, preemptive, and work-conserving scheduling algorithms, which imply that a job can be executed in any core allowing migration (*global*), a higher-priority job can preempt a lower-priority job at any time (*preemptive*), and any processor cannot be left idle as long as there exists at least one active job which is not currently-executing (*work-conserving*).

3. Slack Reclamation for the Generic RTA Framework

In this section, we first introduce how the generic RTA framework works with the forward and backward strategies of slack reclamation, and then we prove the correctness of

Manuscript received November 28, 2017.
Manuscript revised March 1, 2018.
Manuscript publicized May 2, 2018.

[†]The authors are with Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea.
[††]The author is with RTST, Republic of Korea.
a) E-mail: jinkyu.lee@skku.edu (Corresponding author)

DOI: 10.1587/transinf.2017EDL8260

the backward strategy.

The generic RTA framework calculates the response time of τ_k as follows; the response time (denoted by R_k) represents the largest finishing time of the job of τ_k (we let J_k^* denote such a job) among finishing times of all jobs released by τ_k. Let S_i denote a slack value of τ_i, meaning that every job of τ_i finishes its execution at least S_i time units earlier than its deadline. The framework computes $I_{k\leftarrow i}(\ell, S_i)$ for every $\tau_i \in \tau \setminus \{\tau_k\}$, representing an upper-bound of a cumulative length of time intervals such that jobs of τ_i execute but J_k^* cannot in an interval of length ℓ starting from the release time of J_k^*, under a target scheduling algorithm. Then, the generic RTA framework can compute a task's response time R_k with given slack values of other tasks, as follows [4].

Lemma 1 (Sect. 4.3 in [4]). *For given $\{S_i\}_{\tau_i\in\tau\setminus\{\tau_k\}}$, τ_k's response time is calculated as follows.*

Step 1. Set R_k to C_k.
Step 2. Check $R_k > D_k$; if it is true, we deem τ_k unschedulable, finally.
Step 3. Check Eq. (1); if it is true, we set the response time of τ_k to R_k, finally. Otherwise, we set R_k to the RHS (Right-Hand Side) of Eq. (1), and go to Step 2.

$$R_k \leq C_k + \left\lfloor \frac{1}{m} \cdot \sum_{\tau_i\in\tau\setminus\{\tau_k\}} \min\left(I_{k\leftarrow i}(R_k, S_i), R_k - C_k + 1\right)\right\rfloor. \tag{1}$$

Proof. The proof is given in [4]; here we present a high-level explanation. For J_k^* not to execute in a single time slot, there should exist m other jobs executed in the time slot. Also, the number of such time slots is at least $R_k - C_k + 1$, for J_k^* not to execute C_k time units within R_k time units. Therefore, if the RHS of Eq. (1) holds, C_k amount of executions are finished within R_k time units, implying that the response time is at most R_k. □

Using Lemma 1, the following RTA framework with the forward strategy of slack reclamation can compute response time of every task $\tau_k \in \tau$ along with slack value update.

Lemma 2 (Sect. 4.3 in [4]). *Setting S_k to 0 for every task $\tau_k \in \tau$, we calculate the response time of every task $\tau_k \in \tau$ by Lemma 1. Thereafter, for every task $\tau_k \in \tau$, we update $S_k = D_k - R_k$ in case of $D_k - R_k > S_k$. Then, we repeat this process until every task $\tau_k \in \tau$ satisfies $R_k \leq D_k$ (τ is deemed schedulable) or there is no slack value update (τ is deemed unschedulable).*

Proof. The proof is given in [4]; here we present a high-level explanation. Suppose that Lemma 2 deems τ schedulable but there exists a task τ_j satisfying $R_j > D_k$. This implies that Lemma 1 is wrong, which contracts. □

Here, we focus on the way how to update slack values by the forward strategy. As shown in Lemma 2, slack values are initially set to zero (which is the smallest possible one), and a task's slack value is updated on top of current slack values of other tasks. However, the forward strategy can be pessimistic as shown in the following example.

Algorithm 1 Backward strategy (τ)

```
1:  for k = 0 to n do
2:      R_k = C_k
3:      S_k = D_k − C_k
4:  end for
5:  Update ← TRUE
6:  while Update == TRUE do
7:      Update ← FALSE
8:      for k = 0 to n do
9:          r ← compute right-hand side of Eq. (1) in Lemma 1
10:         if r > D_k then
11:             return unschedulable
12:         end if
13:         if r > R_k then
14:             R_k ← r
15:             S_k ← D_k − R_k
16:             Update ← TRUE
17:         end if
18:     end for
19: end while
20: return schedulable
```

Example 1. *Suppose that $\tau = \{\tau_1(T_1 = 6, C_1 = 2, D_1 = 6), \tau_2(3, 2, 3), \tau_3(2, 1, 2)\}$ is scheduled on EDF on a two-processor platform. Then, if we apply RTA for EDF using Lemma 2 (detailed interference calculation will be described in Sect. 4), the first iteration with zero slack values yields $R_1 = 5$, $R_2 > 3$ (unschedulable), and $R_3 > 2$ (unschedulable), implying that $S_1 = 1$, $S_2 = 0$, and $S_3 = 0$. Even with further iteration, we cannot increase any slack value; therefore, RTA for EDF using the Lemma 2 deems τ unschedulable. However, τ is actually schedulable with $R_1 = 4$, $R_2 = 3$, and $R_3 = 1$, implying that $S_1 = 2$, $S_2 = 0$, and $S_3 = 1$.*

Note that the generic RTA framework with the forward strategy of slack reclamation does not scan $S_1 = 2$, $S_2 = 0$, and $S_3 = 1$. This is because, from $S_1 = 1$, $S_2 = 0$, and $S_3 = 0$, no slack value can be increased without increasing other slack values. Therefore, we need to find a better way to find feasible slack value combination.

We now illustrate how the backward strategy of slack reclamation works with Algo. 1 (briefly introduced in [7]). It first sets R_k to C_k and S_k to $D_k - C_k$ for every task $\tau_k \in \tau$ (Lines 1–4), and then it calculates the response time of every task $\tau_k \in \tau$ by Lemma 1 (Line 9). Thereafter, for every task $\tau_k \in \tau$, it updates $S_k = D_k - R_k$ in case of $D_k - S_k < R_k$ (Lines 13–17). Then, it repeats this process until there is no slack value update (τ is deemed schedulable) (Lines 6–20) or there exists at least one task τ_j satisfying $R_j > D_j$ (τ is deemed unschedulable) (Lines 10–11).

Then, we prove the correctness of the backward strategy in Algo. 1 as follows.

Theorem 1. *The resulting values $\{R_j\}_{\tau_j\in\tau}$ in Algo. 1 upper-*

bound the response time of $\{\tau_j\}_{\tau_j \in \tau}$.

Proof. (Our own new proof) Suppose that Algo. 1 calculates that the response time of every task $\tau_j \in \tau$ (denoted by R_j) is no larger than D_j, but there exists a task τ_k whose actual response time is strictly larger than R_k. We prove by showing contradiction.

Let t_0 denote the earliest time instant when there exists a job (invoked by τ_k) whose actual response time is about to be larger than the calculated response time R_k; we denote the job J. From the definition, every job of every task $\tau_k \in \tau$ finishes its execution at least $D_k - R_k$ time units earlier than its deadline, until t_0. Then, from Eq. (1), the response time of J cannot be larger than R_k until t_0, since the response time of every job of every task $\tau_k \in \tau$ is not larger than R_k with slack values $D_k - R_k$. This yields contradiction. □

Now, we explain the difference between the RTA frameworks with the forward and backward strategies of slack reclamation (i.e., Lemma 2 and Theorem 1). In Lemma 2, once a slack value of a task τ_k is set to S_k, the RTA framework guarantees that every job of τ_k finishes its execution at least S_k time units earlier than its deadline. Therefore, the lemma always seeks to find larger a task's slack value on top of valid slack values of other tasks. On the other hand, in Theorem 1, even if a slack value of a task τ_k is set to S_k, the RTA framework cannot guarantee its validity. Instead, the validity holds only when there is no slack update. Since the theorem starts the largest possible slack values, it is possible for Theorem 1 to find larger slack values than Lemma 2.

4. Case Study: RTA for EDF

In this section, we apply the generic RTA framework with the backward strategy, to EDF scheduling. To this end, we first explain existing inference calculation for EDF, and compare EDF schedulability by the framework with the backward strategy (i.e., Theorem 1), with that by the framework with the forward strategy (i.e., Lemma 2).

Under any work-conserving scheduling, $I_{k\leftarrow i}(\ell, S_i)$ is upper-bounded by $W_i(\ell, S_i)$ representing the maximum amount of execution of jobs of τ_i in an interval of length ℓ calculated as follows [4]:

$$W_i(\ell, S_i) = N_i(\ell, S_i) \cdot C_i + \min\left(C_i, \ell + D_i - S_i - C_i - N_i(\ell, S_i) \cdot T_i\right), \quad (2)$$

where $N_i(\ell, S_i) = \left\lfloor \frac{\ell + D_i - S_i - C_i}{T_i} \right\rfloor$. As shown in Fig. 1 (a), the amount of execution of jobs of τ_i in an interval of length ℓ is maximized when the first job of τ_i executes as late as possible, the other jobs of τ_i execute as early as possible, and the interval starts with the beginning of the first job's execution [4].

Under EDF, $I_{k\leftarrow i}(\ell, S_i)$ is upper-bounded by $E_{k\leftarrow i}(S_i)$ denoting the amount of execution of jobs of τ_i in an interval of length D_k when the deadline of the last job of τ_i is aligned with the job of τ_k of interest, calculated as follows [4]:

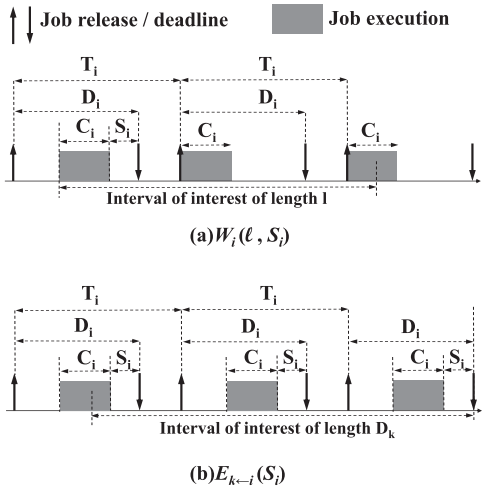

Fig. 1 Two worst-case scenarios under any work-conserving scheduling algorithm and EDF, for the maximum amount of execution of τ_i in an interval of length ℓ.

Table 1 Detailed calculation of response times of tasks in τ shown in Example 1 by Lemma 2 (the forward strategy).

Iteration #	S_1	S_2	S_3	R_1	R_2	R_3
1	0	0	0	$2 \to 4 \to 5$	$2 \to 4(> 3)$	$1 \to 2 \to 3(> 2)$
2	1	0	0	$2 \to 4 \to 5$	$2 \to 4(> 3)$	$1 \to 2 \to 3(> 2)$
3	1	0	0			

Table 2 Detailed calculation of response times of tasks in τ shown in Example 1 by Theorem 1 (the backward strategy).

Iteration #	S_1	S_2	S_3	R_1	R_2	R_3
1	4	1	1	$2 \to 3 \to 4$	2	1
2	2	1	1	$2 \to 3 \to 4$	$2 \to 3$	1
3	2	0	1	$2 \to 3 \to 4$	$2 \to 3$	1
4	2	0	1			

$$E_{k\leftarrow i}(S_i) = \left\lfloor \frac{D_k}{T_i} \right\rfloor \cdot C_i + \min\left(C_i, \max\left(0, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor \cdot T_i - S_i\right)\right). \quad (3)$$

That is, as depicted in Fig. 1 (b), to maximize the amount of higher-priority execution of jobs of τ_i than the job of τ_k of interest in an interval from its release time to its deadline (of length D_k), the interval should end with the deadline of the last job of τ_i and all jobs of τ_i execute as late as possible [4].

Finally, we can use the following upper-bound for EDF: $I_{k\leftarrow i}(\ell, S_i) \le \min(W_i(\ell, S_i), E_{k\leftarrow i}(S_i))$.

Example 2. *Using the upper-bound, Tables 1 and 2 present detailed calculation of response times of tasks in* τ *shown in Example 1 (*$\tau = \{\tau_1(T_1 = 6, C_1 = 2, D_1 = 6), \tau_2(3, 2, 3), \tau_3(2, 1, 2)\}$*). As shown in Table 1, Lemma 2 calculates that* $R_1 = 5$*,* $R_2 > 3$*, and* $R_3 > 2$*, which means* τ *is deemed unschedulable. On the other hand, Theorem 1 calculates that* $R_1 = 4$*,* $R_2 = 3$*, and* $R_3 = 1$*, which means* τ *is deemed schedulable, as shown in Table 2.*

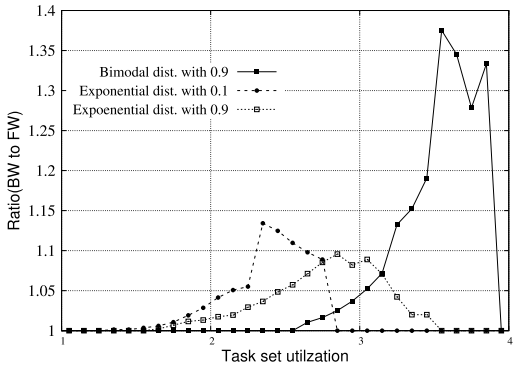

Fig. 2 Evaluation result for $m = 4$.

To demonstrate the existence of many task sets like τ in Example 1, we generate 100,000 task sets for $m = 2$, 4, and 8 using the technique widely used in a number of studies [9], [10]. Following the same method in [9], [10], we consider bimodal or exponential distribution for the task utilization (C_i/T_i); each task utilization distribution shows distinct average number of tasks in a task set (denoted by $\overline{n}$) and average task utilization (denoted by $\overline{C_i/T_i}$), and 10,000 tasks are generated based on each distribution, thereby generating 100,000 tasks in total.

We investigate how much performance improvement is achieved by the schedulaiblity analysis of the backward strategy (denoted by BW) compared to that of the forward strategy (denoted by FW) for different task utilization distributions in order to show various average numbers of tasks in a task set and various average task utilizations. Among the ten task utilization distributions (bimodal and exponential utilization distributions with five input parameters), we present bimodal distribution with 0.9 and exponential distributions with 0.1 and 0.9 since they produce well-separated $\overline{n}$ and $\overline{C_i/T_i}$ as the same was also exploited for a fair evaluation in [10]. In our simulation setting for $m = 4$ (among $m = 2$, 4 and 8), each utilization distribution shows $\overline{n}$ of 9.5, 20.5 and 12.3, and $\overline{C_i/T_i}$ of 0.77, 0.25 and 0.5, respectively. The similar trends are shown in [10]: $\overline{n}$ of 10.1, 43.1 and 14.5, and $\overline{C_i/T_i}$ of 0.68, 0.1 and 0.39, respectively for $m = 8$. The detail description of our task set generation method and the fairness of considered task utilization distributions are presented in the supplement file [11].

Figure 2 presents ratio of the number of task sets deemed schedulable by BW to that by FW (for constrained-deadline task sets for $m = 4$) for bimodal distribution with 0.9 and exponential distributions with 0.1 and 0.9; note that BW always better performs than FW as we discussed in the previous section. Each line in Fig. 2 plots the ratio under corresponding task utilization distribution over varying task set utilization ($U_{sys} \triangleq \sum_{\tau_i \in \mathcal{T}} C_i/T_i$).

We obtain a clear observation from Fig. 2 that BW much outperforms FW for a smaller value of $\overline{n}$ (naturally, a larger value of $\overline{C_i/T_i}$); for a given utilization distribution, values of $\overline{n}$ and $\overline{C_i/T_i}$ are inversely proportional as every utilization distribution shares the similar average U_{sys} owing to our task set generation method. For example, BW improves FW under bimodal distribution with 0.9 ($\overline{n} = 9.5$ and $\overline{C_i/T_i} = 0.77$) up to about 37% while it does under exponential distribution with 0.1 ($\overline{n} = 20.5$ and $\overline{C_i/T_i} = 0.25$) up to about 14% only. This is because both schedulability analysis of FW and BW are based on the upper-bounded execution (i.e., interference) calculated by $W_i(\ell, S_i)$ and $E_{k \leftarrow i}(S_i)$. Such upper-bounded executions are not exact ones, and thus the pessimism stemming from the upper-bounded interference of higher-priority tasks τ_i on a task τ_k of interest is proportional to the number of tasks in a task set. Therefore, there is a large room for BW's schedulability to be improved if each task set has fewer tasks (i.e., smaller $\overline{n}$).

5. Conclusion

In this letter, we addressed the backward strategy of slack reclamation of the existing generic RTA framework for multiprocessors, by proving the correctness and demonstrating its effectiveness. We performed a case study, in which the proposed framework is applied to EDF; the study demonstrates that the backward strategy improves performance of the forward strategy up to about 37%.

Acknowledgements

This research was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.R0190-15-2071, Open PNP platform for diversity of autonomous vehicle based on cloud map). This research was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2017R1A2B2002458).

References

[1] A. Eklund, T.E. Nicholsd, and H. Knutssona, "Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates," Proc. National Academy of Sciences of the United States of America, vol.113, no.28, pp.7900–7905, 2016.
[2] 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, 1973.
[3] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," Proc. IEEE Workshop on Real-Time Operating Systems and Software, pp.133–137, May 1991.
[4] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," Proc. IEEE Real-Time Systems Symposium (RTSS), pp.149–160, 2007.
[5] N. Guan, M. Stigge, W. Yi, and G. Yu, "New reposne time bounds for fixed priority multiprocessor scheduling," Proc. IEEE Real-Time Systems Symposium (RTSS), pp.387–397, 2009.
[6] R.I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," Real-Time Systems, vol.47, no.1, pp.1–40, 2011.
[7] S. Baruah, M. Bertogna, and G. Buttazzo, Multiprocessor Scheduling for Real-Time Systems, Springer, 2015.
[8] A. Mok, "Fundamental design problems of distributed systems for

the hard-real-time environment," Ph.D. thesis, Massachusetts Institute of Technology, 1983.

[9] T.P. Baker, "Comparison of empirical success rates of global vs. paritioned fixed-priority and EDF scheduling for hard real time," Tech. Rep. TR-050601, Dept. of Computer Science, Florida State University, Tallahasee, 2005.

[10] J. Lee, A. Easwaran, and I. Shin, "Contention-free executions for real-time multiprocessor scheduling," ACM Trans. Embedded Computing Systems, vol.13, no.69, pp.1–69, 2014.

[11] H. Baek, D. Lim, and J. Lee, "Supplement of "Proof and Evaluation of Improved Slack Reclamation for Response Time Analysis of Real-Time Multiprocessor Systems"," [Online]: http://cps.kaist.ac.kr/~khlee/files/ieice18_supplement.pdf.