Response Time Analysis of Lazy Round Robin

Yue Tang[1], Nan Guan[2], Zhiwei Feng[1], Xu Jiang[1], Wang Yi[1]

[1]Northeastern University, China
[2]The Hong Kong Polytechnic University, Hong Kong SAR

***Abstract*—The Round Robin scheduling policy is used in many real-time embedded systems because of its simplicity and low overhead. In this paper, we study a variation of Round Robin used in practical systems, named Lazy Round Robin, which is simpler to implement and has lower runtime overhead than ordinary Round Robin. The key difference between Round Robin and Lazy Round Robin lies in when the scheduler reacts to newly released task instances. The Round Robin scheduler checks whether a newly released task instance is eligible for execution in the remaining part of the current round, while the Lazy Round Robin scheduler does not react to any task release until the end of the current round. This paper develops techniques to calculate upper bounds of response time of tasks scheduled by Lazy Round Robin. Experiments are conducted to evaluate our analysis techniques and compare the real-time performance of Round Robin and Lazy Round Robin.**

I. Introduction

Round Robin (RR) is a simple scheduling policy in which computational or communication tasks are executed cyclically with a predefined order. Compared with other real-time scheduling policies, e.g., priority-based preemptive scheduling, RR is much simpler to implement and has much lower runtime overhead. Therefore, although RR is not as competitive as priority-based preemptive scheduling, it is still one of the mostly used scheduling policy for real-time embedded systems, especially those sensitive to system complexity and runtime overhead.

This paper considers a variation of Round Robin, named Lazy Round Robin (LRR), which is even simpler to implement and has lower runtime overhead than ordinary RR. The key difference between RR and LRR lies in when the scheduler reacts to newly released task instances. Under RR, the scheduler checks whether a newly released task instance is eligible for execution in the remaining part of the current round. This requires the scheduler to monitor the released instances and update related information in the scheduler in an almost real-time fashion. By contrast, LRR scheduler does not react to newly released task instances until the end of the current round. All the released task instances in the current round will be effectively received by the system altogether at the end of the current round, and executed in the following rounds. In Section IV we will introduce the LRR scheduling policy in details.

An example of realistic systems using LRR is the scheduling policy in the executors of ROS2 [1], the most popular and de-facto standard framework for robotic software development. In ROS2, processing tasks (*callbacks*) communicate with each other with the publish-subscribe paradigm using the DDS middleware [2]. The processing tasks (except those explicitly triggered by timers) in an *executor* (a core component in ROS2 multiplexing computing tasks) are executed by RR. When a task publishes a message to a topic, the subscriber task of that topic is not notified immediately. Instead, ROS2 uses a *ready set* to mark such information, and the executor will look up the ready set at the end of the current round to prepare the eligible tasks for the next round according to all the relevant published messages during the past period.

Compared with RR, LRR has been paid less attention in literature. To the best of our knowledge, the only existing work is [3], which studied the analysis of LRR in the context of ROS2. In this paper, we study the scheduling behavior of LRR and compare it with RR to explore the influence of its lazy reaction. Then we propose methods to calculate the response time bound of tasks scheduled under LRR, which is more precise than the analysis in [3]. Experiments are conducted to evaluate our proposed techniques and compare the real-time performance of RR and LRR.

II. Related Work

To satisfy more special requirements based on fair allocation, several variations of RR have been proposed. The first and most studied variation is Weighted Round Robin (WRR), which generalizes the ordinary RR by allowing to allocate resource to tasks proportional to their weights. There are a host of work analyzing the response time bound for tasks scheduled under WRR. [4] proposed a framework to derive utilization bounds as schedulability test with Network Calculus. [5] calculated response time bound with Compositional Performance Analysis framework, which improves the results in [4] by taking actual load bounds into account and avoiding overestimating unnecessary workload. [6] analyzed response time by calculating the interference in each scheduling round an instance experiences. [7] derived per-flow end-to-end delay bound in on-chip wormhole networks and proposed a weight adjustment algorithm. [8] computed weakly-hard real-time guarantees in the form of a deadline miss model for real-time messages with WRR scheduling.

Another variation of RR is Deficit Round Robin (DRR) [9]. Its difference from ordinary RR is: the unused (e.g., a packet is too large to be transmitted) resource in the previous round will be accumulated, while in ordinary RR the resource is cleared. The response time of tasks scheduled under DRR is analyzed with Network Calculus in [10]–[12].

However, none of the above variations of RR models lazy reaction of schedulers. As introduced above, [3] is the only work studying LRR to the best of our knowledge. However, the

978-3-9819263-5-4/DATE21/©2021 EDAA

analysis in [3] is pessimistic since it does not fully explore the scheduling features of LRR. Our work calculates more precise response time bound than [3].

III. System Model

A. Workload Model

We consider system Γ, which consists of a set of independent tasks $\{\tau_1, \tau_2, ..., \tau_n\}$. The worst-case execution time (WCET) of task τ_i is denoted by C_i.

Each task releases an infinite sequence of instances. The k^{th} instance released by a task τ_i is denoted by τ_i^k. The release pattern of τ_i is characterized by *arrival curve* $\alpha_i(\Delta)$, which upper-bounds the number of instances of τ_i released in any time interval of length Δ. We use $\overline{\alpha_i}(x) = inf\{\Delta : \alpha(\Delta) \geq x\}$ to denote the pseudo-inverse function [13] of $\alpha_i(\Delta)$, i.e., the length of any time interval in which x consecutive instances of τ_i is released, is lower-bounded by $\overline{\alpha_i}(x)$.

The *response time* of an instance is the time distance between its release time and finish time. The *worst-case response time* of a task is the maximal response time among all its instances. One of the targets of this paper is to calculate a safe upper bound of the worst-case response time for each task in Γ.

B. Resource Model

We assume that Γ executes on a processor with resource supply characterized by supply bound function $sbf(\Delta)$, which lower-bounds the amount of processing time available for the processor in any time interval of length Δ. We use $\overline{sbf}(x)$ to denote the pseudo-inverse function of $sbf(\Delta)$, i.e., $\overline{sbf}(x) = sup\{\Delta : sbf(\Delta) < x)\}$ upper-bounds the length of any time interval in which x units of processing time is provided.

IV. Lazy Round Robin

In this section, we first introduce the scheduling policy of LRR and then compare it with RR.

A. Lazy Round Robin

The scheduler maintains a *scheduling set* Ω, which records ready instances to be executed. Being in the scheduling set is prerequisite for an instance to execute. A task instance becomes ready and is put into Ω at time points when Ω is empty, which are called the *polling points*. When a task instance is finished, it is removed from Ω. We assume that the update of scheduling set is instantaneous, and an instance released exactly at a polling point becomes ready and is put into Ω immediately.

The time interval between two polling points is called a *processing window*. The scheduler selects ready instances in Ω to execute one-by-one *non-preemptively* in the current processing window. In each processing window, if more than one instance of τ_i exists in Ω, only the earliest one to be put into Ω is executed. The order to execute the ready instances of different tasks in a processing window depends on their priorities. Each task τ_i is characterized with a fixed and unique priority, and all its instances inherit this priority. We use $hp(\tau_i)$ to denote the set of tasks with higher priority than τ_i, and $lp(\tau_i)$ to denote the set of tasks with lower priority than τ_i.

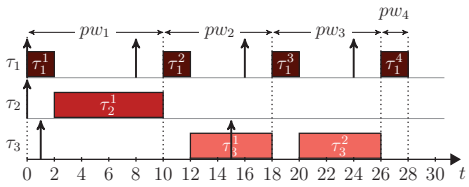


Fig. 1. An example illustrating the scheduling of LRR

TABLE I
Comparison between scheduling set and released instances

Time Points	Released Instances	Scheduling Set
$t = 0$	$\{\tau_1^1, \tau_2^1\}$	$\{\tau_1^1, \tau_2^1\}$
$t = 1$	$\{\tau_3^1\}$	$\{\tau_1^1, \tau_2^1\}$
$t = 8$	$\{\tau_1^2\}$	$\{\tau_2^1\}$
$t = 10$	$\{\}$	$\{\tau_3^1, \tau_1^2\}$
$t = 12$	$\{\}$	$\{\tau_3^1\}$
$t = 15$	$\{\tau_3^2\}$	$\{\tau_3^1\}$
$t = 16$	$\{\tau_1^3\}$	$\{\tau_3^1\}$
$t = 18$	$\{\}$	$\{\tau_3^2, \tau_1^3\}$
$t = 20$	$\{\}$	$\{\tau_3^2\}$
$t = 24$	$\{\tau_1^4\}$	$\{\tau_3^2\}$
$t = 26$	$\{\}$	$\{\tau_1^4\}$

Example 1. Fig. 1 illustrates the scheduling under LRR. Three tasks τ_1, τ_2, τ_3 are scheduled on a processor with $sbf(\Delta) = \Delta$. The WCET of τ_1, τ_2, τ_3 are $C_1 = 2, C_2 = 8, C_3 = 6$. τ_1, τ_2, τ_3 are released periodically with period 8, 36, 14, respectively. τ_1 has the highest priority, and τ_3 has the lowest priority. τ_1^1 and τ_2^1 are released at time point 0, and τ_3^1 is released at time point 1.

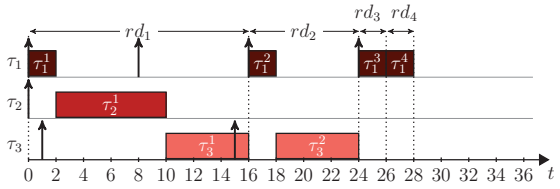
At time point 0, Ω is empty, and τ_1^1, τ_2^1 are put into Ω. τ_1^1 starts execution first due to its higher priority. At time point 1, τ_3^1 is released. It is not put into Ω until time point 10, when τ_1^1, τ_2^1 finish and are removed from Ω. Similarly, τ_1^2 is put into Ω at time point 10. Then τ_1^2 and τ_3^1 execute in priority order. Table I compares the content of scheduling set and the released instances at key time points (when some instances are released, put into Ω or finish). The difference between the scheduling set and the set of released instances indicates the lazy reaction: an instance is not put into scheduling set as soon as it is released, but waits until the end of the current round.

B. Comparison with Round Robin

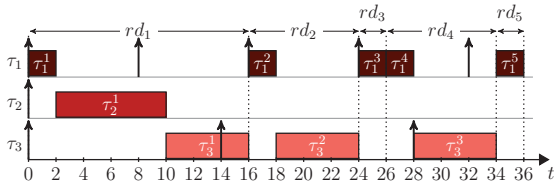
Fig. 2(a) shows the execution sequence of the system in Example 1 scheduled under RR. Here we consider RR with more generalized behavior: in each round, an instance executes to its WCET. We assume that the scheduler checks released instances in the order of τ_1, τ_2, τ_3, and τ_1^1 is selected to execute at 0.

The main difference between execution sequences in Fig. 1 and Fig. 2(a) is when τ_3^1 starts, which is caused by lazy updating scheduling set of LRR scheduler. Under RR, when τ_3^1 is released at 1, the scheduler immediately reacts to it and processes it as soon as its turn comes. However, under LRR, the scheduler does not react to τ_3^1 until the end of pw_1, i.e., time 10. At this point, both τ_1^2 and τ_3^1 are put into the scheduling set, and τ_3^1 is further delayed by τ_1^2 since τ_1^2 has higher priority.

This indicates that lazy reaction may increase an instance's response time, i.e., τ_3^1, by postponing the processing window



(a) Execution sequence with the same release pattern in Example 1



(b) Execution sequence with synchronous release

Fig. 2. Execution sequence under RR

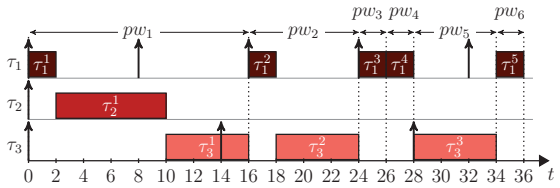


Fig. 3. Execution sequence with synchronous release under LRR

(round) in which it executes. On the other hand, some instance may benefit from this, i.e., the start time of τ_1^2 is advanced.

Next we consider the influence of synchronous release (all tasks are released simultaneously) under these two scheduling algorithms and focus on the worst-case response time of τ_3. Recall that synchronous release is essential for generating a task's worst-case response time under RR [5], [6]. Fig. 2(a) and Fig. 2(b) show the execution sequence with synchronous release and the same release pattern as Example 1 respectively. We can observe that the worst-case response time of τ_3 is larger when it is released simultaneously with τ_1 and τ_2. However, when advancing the release time of τ_3^1 by 1 under LRR, the worst-case response time is 16 (shown in Fig. 3), which is smaller than 17 in Fig. 1. This illustrates that under LRR synchronous release does not necessarily contribute to worse response time, which is different from RR. As a result, most of the existing analysis under RR is not directly applicable to LRR due to their assumption that synchronous release contributes to the worst-case response time.

V. Response Time Analysis

In this section, we propose two methods to calculate response time bound, which are mainly based on tasks' release patterns and scheduling features of LRR, respectively. Combining the results by these two methods generates the final response time bound.

A. Release Pattern-based Method

We say the processor is busy with task τ_i if one released instance of τ_i has not been finished (either being executed, or waiting in the scheduling set, or released but not put in scheduling set yet). We consider a *problem window* $[a, b]$ satisfying

- the processor is not busy with any task right before a and right after b
- the processor is busy with some task at any time point in $[a, b]$

We focus on the analysis of τ_i^k, the k^{th} instance of τ_i in an arbitrary problem window (k is arbitrary). All tasks other than τ_i, i.e., $\tau_j (j \neq i)$, are called interfering tasks. Without loss of generality, we let 0 be the start time of the problem window, t_1 be the release time of τ_i^k, t_2 be the time point when τ_i^k starts execution. Our target is to bound $t_2 - t_1$, with which we can calculate the response time of τ_i^k (the execution is non-preemptive). t_1 can be simply bounded by $\overline{\alpha_i}(k)$, since $\overline{\alpha_i}(k)$ lower-bounds the length of time interval k instances are released. To upper-bound t_2, we need to calculate the upper-bound of workload executed in $[0, t_2]$.

Lemma 1: The workload of an interfering task $\tau_j (j \neq i)$ executed in $[0, t_2]$ is upper bounded by $\alpha_j(t_2) \cdot C_j$.

Proof: By the definition of arrival curve, at most $\alpha_j(t_2)$ instances of τ_j are released in $[0, t_2]$. □

Then t_2 can be upper-bounded by the following lemma:

Lemma 2: t_2 is upper-bounded by the minimal value satisfying

$$sbf(t_2) = \sum_{j \neq i} \alpha_j(t_2) \cdot C_j + (k-1) \cdot C_i \tag{1}$$

Proof: We prove the lemma by contradiction, assuming $t*$ is the minimal solution of (1) and $t* < t_2$. By the scheduling model, each task τ_j with $j \neq i$ released in $[0, t*]$ is possible to interfere with the execution of τ_i^k. So the total workload executed in $[0, t*]$ is upper-bounded by $\sum_{j \neq i} \alpha_j(t*) \cdot C_j + (k-1) \cdot C_i$. On the other hand, the resource available in $[0, t*]$ is at least $sbf(t*)$. Since $\sum_{j \neq i} \alpha_j(t*) \cdot C_j + (k-1) \cdot C_i = sbf(t*)$, the workload that can execute in $[0, t*]$ is at most $sbf(t*)$, so all workload released by $t*$ has been finished by $t*$, which contradicts to the fact that $t*$ is a time point in the middle of a problem window (recall that we are analyzing the k^{th} instance of τ_i in a problem window and the processor must be busy with some task during $[0, t_2]$). □

Lemma 3: The response time of τ_i^k is upper-bounded by

$$R_\alpha(\tau_i^k) \leq \overline{sbf}(sbf(\overline{t_2}) + C_i) - \overline{\alpha_i}(k)$$

where $\overline{t_2}$ is the minimal solution of (1).

Proof: Let t_f denote the finish time of τ_i^k, so the response time of τ_i^k is $t_f - t_1$ (recall that t_1 is the release time of τ_i^k).The total workload in $[0, t_2]$ is upper-bounded by $\sum_{j \neq i} \alpha_j(t_2) \cdot C_j + (k-1) \cdot C_i$. Since $t_2 \leq \overline{t_2}$, the workload is further upper-bounded by $\sum_{j \neq i} \alpha_j(\overline{t_2}) \cdot C_j + (k-1) \cdot C_i$, and is upper-bounded by $sbf(\overline{t_2})$ as $\overline{t_2}$ is a minimal solution of (1). And the total workload before t_f is upper-bounded by $sbf(\overline{t_2}) + C_i$. Then we have $t_f \leq \overline{sbf}(sbf(\overline{t_2}) + C_i)$. Since $t_1 \geq \overline{\alpha_i}(k)$, then the lemma is proved. □

Lemma 4: Let $\overline{\Delta}$ be the minimal positive value of Δ satisfying the following equation:

$$\sum_{\tau_i \in \Gamma} \alpha_i(\Delta) \cdot C_i = sbf(\Delta) \tag{2}$$

then the length of an arbitrary problem window is upper-bounded by $\overline{\Delta}$.

The proof of the lemma is straightforward and thus omitted. Intuitively, the LHS of (2) upper-bounds the total workload of the entire system released in a time interval, and the RHS lower-bounds the total available resource in this time interval. The problem window must end if the total workload does not exceed the total available resource of this time interval, so the minimal solution of (2) upper-bounds the problem window length.

Theorem 1: The worst-case response time of an arbitrary task $\tau_i \in \Gamma$ is upper-bounded by

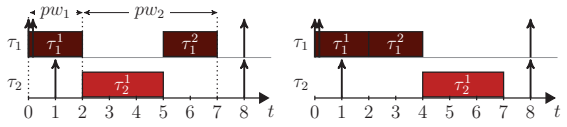
$$R_\alpha(\tau_i) = \max\{R_\alpha(\tau_i^k) \mid 1 \le k \le min(K_1, K_2)\}$$

where K_1 is the minimal value satisfying $R_\alpha(\tau_i^{K_1}) \le \overline{\alpha_i}(K_1+1)$, and $K_2 = \alpha_i(\overline{\Delta})$ with $\overline{\Delta}$ the minimal solution of (2) in Lemma 4.

A formal proof of the theorem is omitted due to space limit. Intuitively, there must exist K_1 satisfying $R_\alpha(\tau_i^{K_1}) \le \overline{\alpha_i}(K_1+1)$ as long as the total utilization is no larger than long term ratio of supply bound function [14]. Besides, $\alpha_i(\overline{\Delta})$ is the maximum number of instances of τ_i among all problem windows.

Discussion. The results calculated by Theorem 1 are pessimistic in some cases. It assumes that all instances released prior to t_2 start execution before t_2, which does not necessarily hold under LRR. Fig. 4 shows an example. Two tasks τ_1 and τ_2 execute on a processor with $sbf(\Delta) = \Delta$. $C_1 = 2$, and $C_2 = 3$. τ_2 has higher priority than τ_1. τ_1^1 and τ_1^2 are released simultaneously at 0, and τ_1^3 is released at 8. τ_2^1 and τ_2^2 are released at 1 and 8, respectively. Fig. 4(a) shows the execution sequence under LRR, and Fig. 4(b) shows the execution sequence with the assumption that all instances released before t_2 execute before τ_i^k. It is observed that the response time of τ_2^1 in Fig. 4(b) is larger than that in actual execution since it counts in the workload of τ_1^2, which should not execute before τ_2^1.

Above observation motivates our method in next subsection.



(a) Execution sequence under LRR (b) Execution sequence consistent with Theorem 1

Fig. 4. Example for pessimism in Theorem 1

B. Processing Window-based Method

In this subsection, we propose another method to calculate response time bound of τ_i, which combined with the results in Theorem 1 generates a tighter bound.

We consider a ***different*** problem window $[a', b']$ from Section V-A, which satisfies

- the processor is not busy with τ_i right before a' and right after b'
- the processor is busy with τ_i at any time point in $[a', b']$

We focus on the k'^{th} instance of τ_i in a problem window $[a', b']$. Without loss of generality, we let 0 be the start time of the problem window, t_1' be the release time of $\tau_i^{k'}$, t_2' be the time point when $\tau_i^{k'}$ starts execution. We say a problem window covers a processing window if they overlap for some time interval. We use pw_n to denote the n^{th} processing window covered by the analyzed problem window.

We first prove some properties about the execution of tasks in a problem window.

Lemma 5: For an arbitrary k', the release time of $\tau_i^{k'+1}$ is no later than the finish time of $\tau_i^{k'}$.

Proof: If $\tau_i^{k'+1}$ has not been released by the finish time of $\tau_i^{k'}$, then when $\tau_i^{k'}$ finishes, there are no unfinished instances of τ_i, and the problem window ends. □

Lemma 6: Instances released by τ_i execute in consecutive processing windows.

Proof: Assume $\tau_i^{k'}$ executes in pw_n. By Lemma 5, $\tau_i^{k'+1}$ is released by the finish time of $\tau_i^{k'}$ and put into Ω at the end of pw_n. Since $\tau_i^{k'+1}$ is the first to execute among all instances released after $\tau_i^{k'}$, it executes in pw_{n+1}. □

Lemma 7: The latest processing window for $\tau_i^{k'}$ to execute in is $pw_{k'+1}$.

Proof: We distinguish two cases based on whether there are unfinished instances at time point 0:

- All instances released prior to 0 have finished before 0. In this case, Ω is empty at 0 and there are no unfinished instances of τ_i right before 0. So when τ_i^1 is released, it becomes ready and put into Ω immediately. Then it executes in pw_1. By Lemma 6, $\tau_i^{k'}$ executes in $pw_{k'}$.
- There exists at least one unfinished instance of tasks other than τ_i at 0[1]. In this case, τ_i^1 is not put into Ω until the end of pw_1. Then it executes in pw_2. By Lemma 6, $\tau_i^{k'}$ executes in $pw_{k'+1}$.

Combining the two cases proves the lemma. □

Next we calculate upper bound of executed workload of interfering tasks in $[0, t_2']$.

Lemma 8: The workload of an interfering task $\tau_j (j \ne i)$ executed in $[0, t_2']$ is upper-bounded by

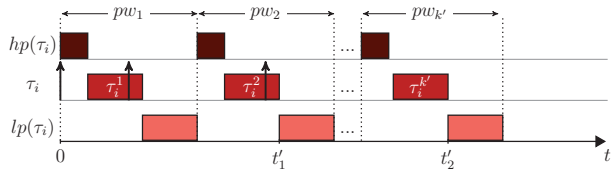
$$I_j = \begin{cases} (k'+1) \cdot C_j, & \tau_j \in hp(\tau_i) \\ k' \cdot C_j, & \tau_j \in lp(\tau_i) \end{cases}$$

Proof: By Lemma 7, the latest processing window for $\tau_i^{k'}$ to execute is $pw_{k'+1}$. By the scheduling model in Section IV-A, at most one instance of τ_j executes in a processing window, so at most $k'+1$ instances of τ_j execute from pw_1 to $pw_{k'+1}$.

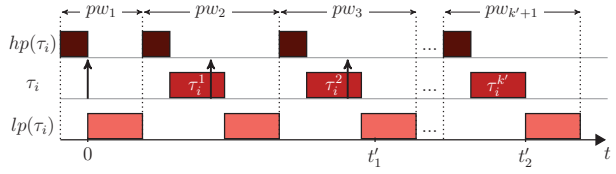
For tasks with higher priorites than τ_i, they execute before instances of τ_i in each processing window, so their workload executed in $[0, t_2']$ is upper bounded by $(k'+1) \cdot C_j$. For tasks with lower priorites than τ_i, they execute after instances of τ_i in each processing window, thus at most k' instances of lower-priority tasks execute before t_2'. □

An illustration for Lemma 7 and 8 is shown in Fig. 5. With Lemma 8, the pessimism of the example in Fig. 4 can be reduced: since τ_1 has lower priority, at most one instance of τ_1 can execute before τ_2^1.

[1]If there exists one unfinished instance of τ_i at 0, then 0 is already in a problem window, contradicting to our assumption that 0 is the start time of the considered problem window.



(a) $\tau_i^{k'}$ executes in $pw_{k'}$



(b) $\tau_i^{k'}$ executes in $pw_{k'+1}$

Fig. 5. Illustration for Lemma 7 and Lemma 8

Next we calculate the response time bound of $\tau_i^{k'}$.

Lemma 9: t'_2 is upper-bounded by the minimal value satisfying

$$sbf(t'_2) = \sum_{j \neq i} I_j + (k'-1) \cdot C_i \tag{3}$$

where I_j is calculated in Lemma 8.

Lemma 10: The response time of $\tau_i^{k'}$ is upper bounded by

$$R_{pw}(\tau_i^{k'}) \leq \overline{sbf}(sbf(\overline{t'_2}) + C_i) - \overline{\alpha_i}(k')$$

where $\overline{t'_2}$ is the minimal solution of (3).
The proof is similar to Lemma 3 and thus omitted.

Theorem 2: The worst-case response time of an arbitrary task τ_i is upper-bounded by

$$R_{pw}(\tau_i) = \max\{R_{pw}(\tau_i^{k'}) \mid 1 \leq k' \leq min(K'_1, K'_2)\} \tag{4}$$

where K'_1 is the minimal value satisfying $R_{pw}(\tau_i^{K'_1}) \leq \overline{\alpha_i}(K'_1+1)$, and $K'_2 = \alpha_i(\overline{\Delta})$ with $\overline{\Delta}$ the minimal solution of (2) in Lemma 4.

Proof: For any time interval $[a', b']$ satisfying the conditions for problem window in this subsection, there must exist a time interval $[a, b]$ satisfying the conditions in Section V-A with $a \leq a', b \geq b'$.

Since $\overline{\Delta}$ upper-bounds the length of a problem window $[a, b]$ satisfying the conditions in Section V-A, it must upper-bound the length of a problem window $[a', b']$ considered in this subsection. Then $\alpha_i(\overline{\Delta})$ upper bounds the number of instances released in a problem window $[a', b']$. On the other hand, if the earliest release time of $\tau_i^{K'+1}$ is no earlier than the latest finish time of $\tau_i^{K'}$, then the problem window $[a', b']$ must end. □

Combining Theorem 1 and Theorem 2, we have

Theorem 3: The worst-case response time of an arbitrary task τ_i is upper-bounded by

$$R(\tau_i) = min(R_\alpha(\tau_i), R_{pw}(\tau_i))$$

where $R_\alpha(\tau_i)$ and $R_{pw}(\tau_i)$ are calculated in Theorem 1 and 2 respectively.

VI. Evaluation

In this section, we conduct experiments with randomly generated workload to empirically evaluate our proposed response time analysis techniques, and compare the real-time performance of RR and LRR.

A. System Generation

We use the TDMA model [15] for $sbf(\Delta)$ of the processor: $sbf(\Delta) = (\lfloor \Delta'/c \rfloor \cdot s + \min(\Delta' \bmod c, s)) \cdot b$, where $\Delta' = \max(\Delta - c + s, 0)$. We set $s = 8, c = 10, b = 1$.

The arrival curve α_i of each task τ_i is generated following the PJD model [16]: $\alpha_i(\Delta) = min(\lceil (\Delta+J)/P \rceil, \lceil \Delta/D \rceil)$. We randomly choose P, J and D in $[20, 100]$, $[0, 5P]$ and $[0, P-1]$, respectively. The WCET of each task is randomly chosen in $[2, 7]$. Each task set contains 5 tasks[2].

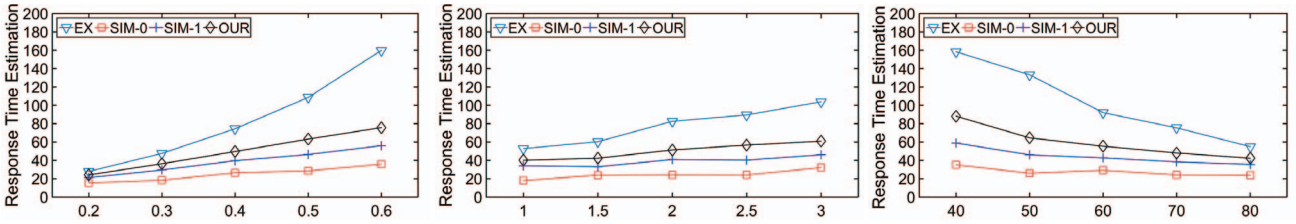
B. Response Time Analysis of Lazy Round Robin

We generated 1000 task sets. For each task set we compare the response time estimations obtained by following methods:

- OUR: the response time bound obtained by Theorem 3.
- EX: the response time bound obtained by analysis in [3].
- SIM-0: the maximal observed response time in simulation of the system, assuming all tasks release the first instances simultaneously, each task releases instances as soon as possible and each task instance executes to WCET. The simulation lasts until one instance of the analyzed task finishes before its subsequent instances are released. The result of SIM-0 is a lower bound on the actual worst-case response time, but still provides useful information to evaluate the precision of our method.
- SIM-1: similar to SIM-0, and the difference is: the first instance of the analyzed task is released one time unit later than first instances of interfering tasks, which are released simultaneously. We have found that in most cases postponing the release time of first instance of analyzed task by one time unit generates the largest response time, thus we only show results with this setting due to page limit. Besides, the difference between SIM-0 and SIM-1 has already proved that synchronous release does not necessarily generate worst-case response time under LRR.

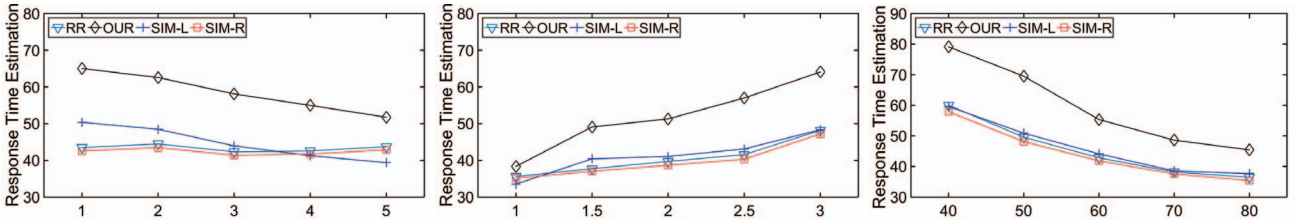
Fig. 6 shows the experiment results grouped by different parameters (the x-axis). In each figure, the four curves are the average response time estimation obtained by the above four methods. Each result on a curve is the average response time estimation of the same arbitrary task of all generated systems with that specific parameter falling in range corresponding to each x-axis. For example, the value 0.2 on the x-axis in Fig.6(a) means the total utilization range $[0.15, 0.25)$. Our analysis outperforms EX since exploring the scheduling features (as in Section V-B) of LRR helps exclude part of workload that causes pessimism in EX.

[2]The results with other parameters are omitted due to space limit since they have similar trends.



(a) x-axis: total utilization (b) x-axis: average J/P (PJD arrival curve model) (c) x-axis: average P (PJD arrival curve model)

Fig. 6. Experiment results under LRR.



(a) x-axis: task priority (b) x-axis: average J/P (PJD arrival curve model) (c) x-axis: average P (PJD arrival curve model)

Fig. 7. Comparison with RR.

C. Comparison with Round Robin

We generated 1000 task sets. For each task set we compare the response time estimations obtained by following methods:

- OUR: the response time bound obtained by Theorem 3.
- RR: the response time bound obtained by [5].
- SIM-L: the observed response time in simulation of the system under LRR, by getting the maximal response time of the analyzed task generated by SIM-0 and SIM-1 described in Section VI-B.
- SIM-R: the observed response time in simulation of the system under RR, following the critical instant proved in [5], [6], i.e., all tasks are released simultaneously at the time point when the analyzed task just misses its turn to execute. The predefined order for tasks to execute in a round is consistent with its priority under LRR: higher priority tasks execute earlier.

Fig. 7 shows the experiment results grouped by different parameters (the x-axis). We first focus on the comparison of SIM-L and SIM-R. There does not exist dominance relation between LRR and RR, i.e., one does not always generate smaller response time than the other. But it can be observed that RR is more beneficial for lower-priority tasks while LRR performs better for higher-priority tasks. Then we consider the comparison of OUR and RR. The response time bound obtained in Theorem 3 is always larger than that in [5], since in the worst case an instance executes in a later processing window (round) under LRR.

VII. Conclusion

We study the scheduling of LRR and compare it with RR. The key difference between these two scheduling algorithms lies in when they react to newly released instances. Then we propose response time analysis techniques for LRR and experiments are conducted to evaluate our proposed response time analysis techniques.

Acknowledgement

This work is supported by the Research Grants Council of Hong Kong (GRF 15204917 and 15213818) and National Natural Science Foundation of China (Grant No. 61672140).

References

[1] ROS2 Overview, "https://index.ros.org/doc/ros2/."
[2] A. Corsaro, G. Pardo-Castellote, and C. Tucker, "Dds interoperability demo." Object Management Group, 2009.
[3] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ros 2 processing chains under reservation-based scheduling," in *ECRTS*, 2019.
[4] J. Wu, J.-C. Liu, and W. Zhao, "Utilization-bound based schedulability analysis of weighted round robin schedulers," in *RTSS*, 2007, pp. 435–446.
[5] D. Thiele, J. Diemer, P. Axer, R. Ernst, and J. Seyler, "Improved formal worst-case timing analysis of weighted round robin scheduling for ethernet," in *CODES+ISSS*, 2013.
[6] R. Racu, L. Li, R. Henia, A. Hamann, and R. Ernst, "Improved response time analysis of tasks scheduled under preemptive round-robin," in *CODES+ISSS*, 2007, pp. 179–184.
[7] Y. Qian, Z. Lu, and Q. Dou, "Qos scheduling for nocs: Strict priority queueing versus weighted round robin," in *ICCD*, 2010, pp. 52–59.
[8] Z. Hammadeh and R. Ernst, "Weakly-hard real-time guarantees for weighted round-robin scheduling of real-time messages," in *ETFA*, 2018, pp. 384–391.
[9] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, p. 375–385, 1996.
[10] M. Boyer, G. Stea, and W. Sofack, "Deficit round robin with network calculus," in *VALUETOOLS*, 2012, pp. 138–147.
[11] A. Soni, X. Li, J.-L. Scharbarg, and C. Fraboul, "Integrating offset in worst case delay analysis of switched ethernet network with deficit round robin," in *ETFA*, 2018.
[12] ——, "Optimizing network calculus for switched ethernet network with deficit round robin," in *RTSS*, 2018, pp. 300–311.
[13] J. L. Boudec and P. Thiran, "Network calculus - a theory of deterministic queuing systems for the internet." Springer Verlag, 2012.
[14] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, p. 390–395, 1986.
[15] E. Wandeler and L. Thiele, "Real-Time Calculus (RTC) Toolbox," 2006. [Online]. Available: http://www.mpa.ethz.ch/Rtctoolbox
[16] K. Richter, "Compositional scheduling analysis using standard event models: The SymTA/S approach," *PhD thesis*, 2005.