Battery-Aging-Aware Run-Time Slack Management for Power-Consuming Real-Time Systems

Jaeheon Kwak[a], Kyunghoon Kim[b], Youngmoon Lee[c], Insik Shin[a], Jinkyu Lee[b,*]

[a] *School of Computing, KAIST, Republic of Korea*
[b] *Dept. of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea*
[c] *Dept. of Robotics, Hanyang University, Republic of Korea*

Abstract

As many safety- or mission-critical electric systems become equipped with batteries, we need to achieve seemingly conflicting goals: G1) timely execution of power-consuming tasks (for safety or mission) while G2) minimizing battery aging (for system sustainability). To this end, this paper proposes a novel scheduling framework for a set of power-consuming real-time tasks, which efficiently utilizes run-time slack (i.e., system idle time identified at run-time) to find the tasks' schedule that achieves both G1 and G2. The proposed framework is not only applicable to any existing prioritization policy (e.g., EDF and FP) but has also been proven to reduce battery aging by up to 32.6% without compromising G1.

Keywords: Real-time system, battery aging, timely execution, battery management, slack management

1. Introduction

As mobility becomes important, many real-time systems like satellites, electric vehicles (EVs), uncrewed aerial vehicles (UAVs), and drones are now equipped with battery packs as their main power source, becoming safety- or mission-critical electric systems [1]. Since the electric system's battery pack is typically shared by its sub-systems (e.g., sensors, motors, and computing units), each accommodating periodic power-consuming tasks, we need to achieve the following goals for scheduling a set of power-consuming real-time tasks:

G1. timely execution of the tasks to assure the safety or accomplish the mission,

G2. while minimizing battery aging that prolongs the system's lifetime.

Existing techniques (called *schedulability tests*), which assure G1 offline under a target prioritization policy, utilize static task parameters available offline, e.g., WCET (the Worst-Case Execution Time). On the other hand, achieving G2 depends on the actual task execution behavior available only at run-time, e.g., AET (the Actual Execution Time). Therefore, achieving both G1 and G2 necessitates the development of run-time mechanisms subject to the following requirements.

R1. We need to establish boundaries of run-time execution behavior for each task, which does not compromise G1 assured by a schedulability test under a target prioritization policy.

R2. Once R1 is addressed, we need to systematically find a battery-aging-favorable run-time execution behavior within the boundaries.

R3. Once R2 controls the run-time execution behavior for a task, the boundaries for other tasks to be executed after the task are also changed, which necessitates efficient update and reclamation of the boundaries for each task at run-time.

The R1–R3 are essential requirements for achieving both G1 and G2. In order to reduce battery aging (G2), it is essential to change the power load of the system by altering the execution behavior of tasks. Since these changes can violate the system's timing guarantees (G1), R1 is needed to find boundaries where execution behavior can be changed without violating G1. Nonetheless, G2 cannot be not easily achieved despite the capability to alter the execution behaviors of tasks. For instance, one may think that simply replacing a prioritization policy, which affects tasks'

*Corresponding author

Email addresses: `0jaehunny0@kaist.ac.kr` (Jaeheon Kwak), `kip0022@skku.edu` (Kyunghoon Kim), `youngmoonlee@hanyang.ac.kr` (Youngmoon Lee), `insik.shin@kaist.ac.kr` (Insik Shin), `jinkyu.lee@skku.edu` (Jinkyu Lee)

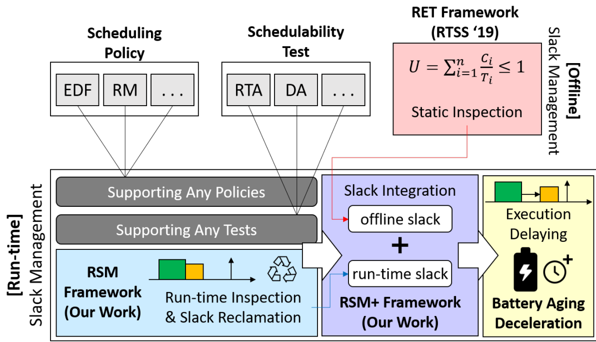


Figure 1: An overview of the proposed framework.

behaviors, would attain G2, but we have confirmed that prioritization policies show little difference in battery aging, to be described in Section 6. Thus, R2 is needed that utilizes the boundaries of R1 based on an understanding of battery-aging-favorable power load. When both R1 and R2 are achieved, the execution behavior changes caused by R2 can affect R1. This can either narrow the existing boundaries of R1, potentially violating G1, or widen them, degrading the achievement of G2. Therefore, R3 is also essential to update the boundaries for meeting G1 and G2.

To the best of our knowledge, there is no study that addresses R1–R3 together. Although there exist a group of studies that develop power- or energy-aware real-time scheduling (e.g., improving energy efficiency [2, 3, 4], managing power requirements [5, 6, 7, 8], simply reducing energy consumption [9, 10, 11], or conserving energy consumption in a temperature-aware manner [12, 13, 14, 15]) without compromising R1, they cannot address R2 and R3 because they do not contend with battery aging.

On the other hand, there exist a couple of relevant previous approaches that try to address both timing guarantees and batter aging [16, 17]; however, (i) they fail to optimize G2 as they focus on R1 and R2 only (but not R3) and (ii) their applicability and performance are limited to a certain schedulability test associated with EDF (Earliest Deadline First) as a prioritization policy.

To address R1, the existing approaches [16, 17] inflate WCET until the schedulability test barely compromises G1 and then utilize the difference between the inflated and original WCET as boundaries. As to R2, the approaches simplify the effect of a run-time execution behavior change on the battery aging, which eases exploiting the boundaries. However, since the boundaries are derived from static task parameters only (by the nature of schedulability tests), the approaches not only result in narrow boundaries for R1 (due to being oblivious of run-time information) but also fail to address R3 (due to absence of efficient run-time update/reclamation of boundaries), both of which make it impossible to fully achieve G2.

In this paper, we develop a novel run-time scheduling framework for a set of power-consuming real-time tasks, called *Run-time Slack Management* (`RSM`). By properly addressing the run-time requirements R1–R3, `RSM` not only can be applied to any pair of a schedulability test and prioritization policy but also fully addresses G2 without compromising G1. To this end, we first define notions of WCET-based and AET-based slacks; the former represents sub-system idle time identified at run-time assuming every task executes for its WCET, while the latter represents the time remainder unused by each task due to the difference between its WCET and AET. We utilize the notions such that they play a key role in addressing R1 as they help to derive a maximum tolerable duration for which each task can delay its execution without compromising G1.

Considering the maximum tolerable duration depends on how efficiently reclaim the two types of slacks at run-time, we propose a run-time mechanism `RSM` that tightly calculates and utilizes both slacks, addressing R3. Finally, we complete `RSM` by incorporating the technique for R2 in [16] into the run-time slack reclamation mechanism, which enables us to fully achieve G2. Following `RSM`, we develop `RSM`+, a framework that integrates `RSM` with the existing offline slack management scheduling method (i.e., `RET`). Fig. 1 depicts the relationship between `RET`, `RSM`, and `RSM`+. As shown in Fig. 1, the existing offline slack management approach `RET` calculates static offline slack based on a given schedulability test for a given scheduling policy. On the other hand, our proposed `RSM` obtains run-time slack based on the inspection of the run-time behavior of tasks, independent of scheduling policy or schedulability test. Since `RSM` and `RET` find slack at orthogonal timing, there is a chance to utilize both slacks together, which is achieved by `RSM`+ that integrates their respective slacks by leveraging the two methods. Finally, these slacks safely delay the execution of tasks to decelerate battery aging without compromising timing guarantees.

We demonstrate the battery aging reduction performance of our proposed frameworks on a precise battery emulator with various parameters. The evaluation considers various sets of scheduling policies, the number of tasks, utilization ratio, and task set parameters; it also estimates battery aging through the electrochemical model battery emulator, which is one of the most precise battery models [18, 19]. The evaluation results demonstrate that `RSM`+ not only reduces battery aging up to 32.6% and 28.2%, respectively, compared to the vanilla prioritization policy and existing approach while conserving timely execution of every job, but also showed the best average performance for all parameters: the number of tasks, scheduling policy, and utilization ratio.

We highlight the contributions of this paper as follows.

- We develop how to efficiently reclaim the notions of WCET-based and AET-based slacks at run-time (in Sections 3 and 4).
- We propose a novel run-time scheduling framework RSM that achieves both G1 and G2 by utilizing the slacks (in Section 4).
- We develop how to incorporate the existing offline slack management method into RSM, yielding RSM+ that exploits the advantages of both (in Section 5).
- We demonstrate the wide applicability of RSM and its accomplishment of both G1 and G2 (in Section 6).

2. Background

System model. Following the system model in [16], we consider a battery-powered electric system $\mathcal{S}$ with N sub-systems $\{\mathcal{S}^j\}_{j=1}^N$. Each sub-system $\mathcal{S}^j$ (e.g., sensors, motors, and computing units) operates a set of N^j real-time power consuming tasks $\tau^j = \{\tau_i^j\}_{i=1}^{N^j}$, and all sub-systems share a battery pack as a main power source. We assume that each real-time power-consuming task is periodic and has an implicit deadline. Each real-time power consuming task τ_i^j is modeled by (T_i^j, C_i^j, P_i^j), where T_i^j is both the period and the relative deadline, C_i^j is WCET, and P_i^j is the power consumption during execution. Each task τ_i^j invokes a series of jobs at every T_i^j time unit, and each job of τ_i^j released at t should complete its execution no later than $t + T_i^j$, where the AET of each job of τ_i^j is no larger than its WCET (i.e., C_i^j). Due to the periodicity, each sub-system at any time knows the next job release time of each task. A job is said to be *active* at t, if it is released no later than t and has remaining execution at t. Since there exists at most one active job of a task at any time, we use a task and its job interchangeably if no ambiguity arises. Each sub-system is allowed to execute only one job at any time, and a job cannot be preempted once it starts its execution. Therefore, apart from power consumption, scheduling a set of real-time power-consuming tasks within each sub-system is equivalent to real-time *uniprocessor* scheduling for *non-preemptive* tasks. On the other hand, the aging of the battery pack depends on the sum of power consumption of all sub-systems, implying that determining the job schedule of a sub-system entails consideration of that of other sub-systems. A more detailed explanation of the system model is given in [16].

Prioritization policy and schedulability test. In real-time scheduling, a *prioritization policy* $\mathcal{P}$ determines which job to be executed in which time interval. For the timing guarantee of real-time scheduling, there is an important notion of a *schedulability test* $\mathcal{T}_\mathcal{P}$ for a prioritization policy $\mathcal{P}$, as follows. If a set of real-time tasks passes the condition of $\mathcal{T}_\mathcal{P}$, timely execution (called *schedulablilty*) of every job of the task set is guaranteed by $\mathcal{T}_\mathcal{P}$ as long as the task set is scheduled by $\mathcal{P}$ in a work-conserving manner (which disallows to idle the sub-system if there exist at least one active job).

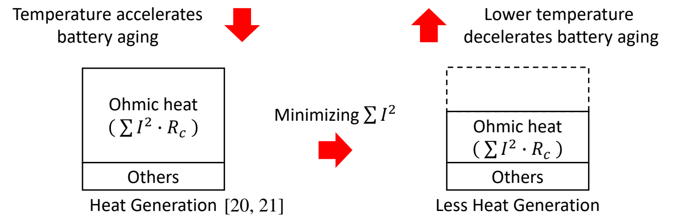


Figure 2: Our abstraction of reducing battery aging by minimizing Ohmic heat based on the battery aging model [20, 21, 22].

Battery aging model. Battery's capacity and performance degrade as aging processes due to various mechanisms such as SEI layer growth, active material loss, etc. This degradation is called battery aging, and researchers have revealed many factors related to it [20, 21, 23, 24]. Fig. 2 shows our battery aging model proposed by Jin et al. [22] and our abstracted approach to decelerate battery aging. Jin et al. modeled battery aging (Q) as Equation 1, where $k_{AM}, E_{AM}, k_{SEI}, E_{SEI}$ and R_{gas} denote some constants, respectively, and T, L and SoC denote temperature, length of a time interval and state of charge, respectively.

$$Q = \int_0^L \left[k_{AM} \cdot exp(\tfrac{-E_{AM}}{R_{gas} \cdot T}) SoC \cdot |I| + \tfrac{-k_{SEI}}{2\sqrt{t}} \cdot exp(\tfrac{-E_{SEI}}{R_{gas} \cdot T})\right] dt \quad (1)$$

Among lots of variables, high temperature is the common accelerator and is known to have the greatest effect on battery aging [23]. The battery temperature is determined by the internal heat generation by the battery itself and external heat transfer, and the former is more significant and dominated by Ohmic heat [20]. Since Ohmic heat is calculated by $I^2 \cdot R_c$, where R_c is the internal resistance of the battery and I is the current load, minimizing $\sum I^2$ is a simple yet effective abstraction to minimize battery aging for power-consuming real-time tasks [16].[1]

In this paper, we employ this abstraction as a means of achieving G2 (minimizing battery aging). Unlike our abstraction focusing on Ohmic heat, our battery aging evaluation considers various battery aging factors that our abstraction does not account for, using a sophisticated battery emulator, as shown in Fig. 2. The relationship be-

[1] Because of negligible change in voltage V_C of the battery internal R_C, the abstraction in [16] approximates Ohmic heat $I^2 R_c = I \cdot V_C$ with the current I, to be used in this paper.

tween achieving the abstraction and G2 will be addressed by evaluation results in Section 6.

Notations. For the ease of understanding, important terminologies and their notations used in this paper are presented in Table 1.

Symbol	Description
WCET	Worst-case execution time
AET	Actual execution time
$\mathcal{S}$	Battery-powered electric system
N	The number of sub-systems of $\mathcal{S}$
$\mathcal{S}^j$	Sub-system j of $\mathcal{S}$
N^j	The number of tasks of $\mathcal{S}^j$
τ^j	Task set of $\mathcal{S}^j$
τ_i^j	Task i of τ^j
T_i^j	Period of τ_i^j
C_i^j	WCET of τ_i^j
P_i^j	Power consumption of τ_i^j during its execution
$\mathcal{P}$	Prioritization policy
$\mathcal{T}_{\mathcal{P}}$	Schedulability test for $\mathcal{P}$
I	Current load
R_c	Internal resistance of a battery
V_c	Voltage of a battery
t_0	Current time
$rdyQ^j$	Ready queue of $\mathcal{S}^j$
τ_{resv}^j	Highest-priority job in $rdyQ^j$ (to be reserved)
d_{slack}^j	Maximum tolerable delay time of τ_{resv}^j
t_{resv}^j	Time that $\mathcal{S}^j$ is reserved for τ_{resv}^j
$t_{njr}^j(t_0)$	Next job release time after t_0 in $\mathcal{S}^j$
τ_{prev}^j	The previous τ_{resv}^j that assigned at t_{-1}.
τ_{other}^j	A job released at $t_{njr}^j(t_0) \in [t_0, t_{resv}^j]$
$WS^j(t_0)$	WCET-based slack of $\mathcal{S}^j$ at t_0
$AS^j(t_0)$	AET-based slack of $\mathcal{S}^j$ at t_0
$J_{HI/LO}$	Higher/Lower-priority job
IC_i^j	Inflated WCET of τ_i^j
I_i^j	Amount of inflated time of IC_i^j ($IC_i^j = I_i^j + C_i^j$)
U^j	Utilization ratio of $\mathcal{S}^j$ ($\sum_{\tau_i^j} C_i^j/T_i^j$)

Table 1: Important terminologies and their notations.

3. Run-Time Execution Behavior Analysis

We formally state the problem of this paper as follows.

Given an electric system $\mathcal{S}$ such that every sub-system's task set is deemed schedulable by a schedulability test $\mathcal{T}_{\mathcal{P}}$ associated with a prioritization policy $\mathcal{P}$,
Determine the schedule (i.e., which job starts its execution at which time instant) of every sub-system's task set, by changing the original schedule by $\mathcal{P}$ at run-time,
Subject to (i) achieving both G1 and G2 and (ii) being applicable to any pair of $\mathcal{T}_{\mathcal{P}}$ and $\mathcal{P}$.

To solve the problem, this section analyzes the effect of the run-time execution behavior on G1 and G2, which will be used for the development of RSM in Section 4. To this end, this section considers an electric system consisting of two sub-systems $\mathcal{S}^x$ and $\mathcal{S}^y$. In $\mathcal{S}^x$, there are four tasks: τ_a^x ($T_a^x = 20$, $C_a^x = 5$, $P_a^x = 5$), τ_b^x $(20, 3, 4)$ τ_c^x $(20, 2, 3)$, and τ_d^x $(9, 1, 3)$. The first jobs of τ_a^x, τ_b^x, τ_c^x and τ_d^x are released at $t = 0$, 0, 5 and 3, respectively. In $\mathcal{S}^y$, there is only one task τ_e^y $(4, 2, 4)$ whose first job is released at $t = 0$. Suppose that we apply FP (Fixed Priority) as a prioritization policy, in which jobs' priorities are inherited by their invoking tasks with the priority order of $\tau_a^x \succ \tau_b^x \succ \tau_c^x \succ \tau_d^x$. As we explained in the problem statement, the schedulability of each sub-system's task set is already guaranteed by a schedulability test $\mathcal{T}_{\text{FP}}$ associated with FP; this means, there is no job deadline miss, as long as the task set in each sub-system is scheduled by FP in a work-conserving manner.

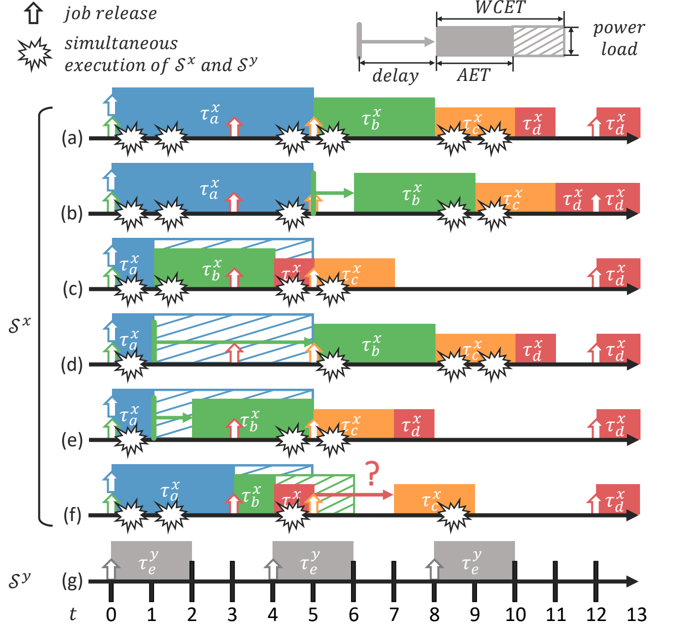


Figure 3: The effect of the run-time execution behavior on G1 and G2.

Fig. 3 depicts different schedules of $\mathcal{S}^x$ in $[0, 13)$, when the schedule of $\mathcal{S}^y$ is given. In particular, Fig. 3(a) illustrates the schedule of $\mathcal{S}^x$ when every job of tasks in $\mathcal{S}^x$ performs its execution during its WCET, which is the longest execution scenario considered by $\mathcal{T}_{\text{FP}}$. At the time instant of $t = 5$, at which the first job of τ_a^x finishes its execution, the first job of τ_b^x is the highest-priority job among active jobs at $t = 5$. Since the sum of execution of active jobs at $t = 5$ is $3 + 2 + 1 = 6$, and there is no job release until $t = 12$, there is an idle period of length 1 in $[5, 12)$, which is $[11, 12)$ as shown in Fig. 3(a). We define the duration of the idle period as follows.

Definition 1. *Let t' denote the earliest job release time*

in a sub-system $\mathcal{S}^x$ after t. Suppose that there is no job in $\mathcal{S}^x$, which starts its execution before t but does not finish its execution until t. The WCET-based slack of $\mathcal{S}^x$ at t is defined as the duration of idle time of $\mathcal{S}^x$ in $[t, t')$ when every job active at t in $\mathcal{S}^x$ is executed for its WCET.

In the situation shown in Fig. 3(a), the WCET-based slack of $\mathcal{S}^x$ at $t = 5$ is one time unit. Using the WCET-based slack at $t = 5$, we can delay the time to start τ_b^x's execution for one time unit without compromising the timely execution of any job active at $t = 5$, as shown in Fig. 3(b). This delay helps avoid simultaneous execution between any job on $\mathcal{S}^x$ and τ_e^y on $\mathcal{S}^y$ in $[5, 6)$, resulting in decelerated battery aging by decreasing Ohmic heat through the reduced $\sum I^2$ from 505 to 481. However, the delay cannot avoid simultaneous execution in other time slots, e.g., $[8, 10)$.

Different from the schedule in Fig. 3(a), each job's AET at run-time is usually less than its WCET. For example, Fig. 3(c) shows the situation where the AET of the first job of τ_a^x is one time unit, yielding earlier completion of the first jobs of τ_b^x, τ_c^x, and τ_d^x executed for their WCET. Since a schedulability test $\mathcal{T}_{\text{FP}}$ already guarantees timely execution of every job, even for its WCET, we focus on the remainder period unoccupied by the first job of τ_a^x (i.e., $[1, 5)$) due to the difference between its AET and WCET, which is defined as follows.

Definition 2. *Let J denote the most recently executed job on $\mathcal{S}^x$ before t, and t' denote the time instant at which J finishes its execution, assuming its AET is the same as its WCET. The AET-based slack of $\mathcal{S}^x$ at t is defined as $(t' - t)$; note that it is defined as 0, if $(t' - t) < 0$.*

In the situation shown in Fig. 3(c), the AET-based slack of $\mathcal{S}^x$ at $t = 1$ is $t' - t = 5 - 1 = 4$. If we delay the execution of the first job of τ_b^x for 4 time units (by fully utilizing the amount of the AET-based slack of $\mathcal{S}^x$ at $t = 1$), we can avoid simultaneous execution between any job on $\mathcal{S}^x$ and τ_e^y on $\mathcal{S}^y$ in $[1, 2)$ and $[4, 5)$ while we cannot avoid it in $[0, 1)$, $[5, 6)$ and $[8, 10)$, as shown in Fig. 3(d). On the other hand, utilizing a portion of the AET-based slack may help to further reduce the duration of simultaneous execution. If τ_b^x delays its execution by only one time unit as shown in Fig. 3(e), simultaneous execution happens only in $[0, 1)$ and $[4, 6)$, whose length is smaller than that in Fig. 3(d), decreasing $\sum I^2$ from 325 to 301.

Then, what if we have multiple AET-based slacks at different time instants? For example, consider a situation where the AET of the first jobs of τ_a^x and τ_b^x are 3 and 1, respectively, as shown in Fig. 3(f). In the situation, the AET-based slacks of $\mathcal{S}^x$ at $t = 3$ and $t = 4$ are $5 - 3 = 2$ (i.e., WCET - AET for τ_a^x) and $6 - 4 = 2$ (i.e., (3+WCET) - (3+AET) for τ_b^x), respectively. Since the calculation of those slacks is independent, the AET-based slack at $t = 4$, which is 2, does not address the situation where the AET-based slack at $t = 3$ is not used for delaying the execution of the first job of τ_b^x. In addition, it is unclear whether the first job of τ_d^x at $t = 4$ can utilize the unused AET-based slack at $t = 3$ and how much unused AET-based slack can be re-used, without incurring any job deadline miss. Including this issue, we need to address the following issues regarding WCET- and AET-based slacks to achieve both G1 and G2.

- Q1. How to exploit the WCET-based slack without compromising G1?
- Q2. How to efficiently calculate so-called cumulative AET-based slack (that includes unused AET-based slacks at previous time instants), and how to exploit the cumulative AET-based slack without compromising G1?
- Q3. How to utilize the answers of Q1 and Q2 in fully achieving G2?

Recent studies [16, 17] have also proposed approaches to reduce battery aging by delaying the execution of tasks while achieving both G1 and G2. However, since the previous approaches distribute slacks to each task offline (and then each task delays its execution with its own slack), tasks cannot fully utilize both WCET-based slack nor AET-based slack, failing to address Q1–Q3. For example, in the case of Fig. 3(a)–(e), τ_b^x's execution should be delayed for achieving G2; however, under the previous approaches, delaying τ_b^x is possible only if slacks happen to be distributed to τ_b^x offline, and if τ_b^x does not receive any slack, delaying the τ_b^x is impossible. Therefore, we need a novel approach that fully utilizes the WCET-based slack and AET-based slacks during run-time to achieve both G1 and G2, which remedies previous approaches.

4. `RSM`: Run-time Slack Management

In this section, we present a novel scheduling framework called `RSM` (Rum-time Slack Management), which fully achieves G2 without compromising G1.

4.1. Operation of `RSM` at Each Time Instant

Algorithm 1 presents the operation of `RSM` at t_0, which manages the following states of each sub-system $\mathcal{S}^j$: `running` (execution of a job), `idle` (no job execution due to non-existence of any active job), and `reserved` (no job execution despite the existence of active job(s) due to the sub-system reservation). Lines 1–3 put a task τ_i^j in the ready queue of $\mathcal{S}^j$ (denoted by $rdyQ^j$) if τ_i^j releases a new job, and Lines 4–6 set the state of $\mathcal{S}^j$ to `idle` if a job of τ_i^j in $\mathcal{S}^j$ completes its execution.

Algorithm 1 `RSM`: Run-time Slack Management at t_0

1: **if** τ_i^j releases its job **then**
2: $\quad rdyQ^j \leftarrow rdyQ^j \cup \{\tau_i^j\}$
3: **end if**
4: **if** a job of a task in $\mathcal{S}^j \in \mathcal{S}$ completes its execution **then**
5: $\quad$ Set the state of $\mathcal{S}^j$ to `idle`
6: **end if**
7: **if** $rdyQ^j \neq \emptyset$ and the state of $\mathcal{S}^j$ is `idle` **then**
8: $\quad \tau_{resv}^j \leftarrow$ the highest-priority job in $rdyQ^j$
9: $\quad d_{slack}^j \leftarrow \textsc{maxresv}(\mathcal{S}^j, t_0, t_{resv}^j, \tau_{resv}^j)$
10: $\quad t_{resv}^j \leftarrow t_0 + d_{slack}^j + C_{resv}^j$
11: $\quad rdyQ^j \leftarrow rdyQ^j \setminus \{\tau_{resv}^j\}$
12: $\quad$ Set the state of $\mathcal{S}^j$ to `reserved`
13: **end if**
14: **if** the state of any sub-system is changed to `reserved` **then**
15: $\quad$ **for** every `reserved` $\mathcal{S}^j \in \mathcal{S}$, sorted by $\frac{(t_{resv}^j - t_0)}{P_{resv}^j}$ **do**
16: $\quad\quad t_{start}^j \leftarrow \textsc{minIsquare}(\mathcal{S}^j, t_0, t_{resv}^j - C_{resv}^j)$
17: $\quad$ **end for**
18: **end if**
19: **if** $t_{start}^j = t_0$ **then**
20: $\quad$ Start to execute τ_{resv}^j on $\mathcal{S}^j$
21: $\quad$ Set the state of $\mathcal{S}^j$ to `running`
22: **end if**

If there is at least one job in the ready queue of $\mathcal{S}^j$ whose state is `idle` (Line 7), Lines 7–13 and Lines 14–18, respectively, perform two distinct stages: *sub-system reservation* and *execution timing decision*. Instead of executing τ_{resv}^j (i.e., the highest-priority job in the ready queue of $\mathcal{S}^j$) immediately at t_0, the *sub-system reservation* stage in Lines 7–13 delays the execution of τ_{resv}^j. To this end, we calculate d_{slack}^j, the maximum tolerable delay of the execution of τ_{resv}^j without incurring any job deadline miss in $\mathcal{S}^j$, by calling $\textsc{maxresv}(\mathcal{S}^j, t_0, t_{resv}^j, \tau_{resv}^j)$. We then reserve $\mathcal{S}^j$ until t_{resv}^j that consists of the current time (i.e., t_0), the maximum tolerable delay of the job execution (i.e., d_{slack}^j) and the maximum execution time of the job (i.e., C_{resv}^j). Section 4.2 will detail how the function of MAXRESV tightly utilizes the WCET-based and cumulative AET-based slacks, which is a key to achieving G2 without compromising G1.

Next, the *execution timing decision* stage in Lines 14–18 targets every sub-system $\mathcal{S}^j$ whose state is `reserved` and determines the time instant at which a delayed job of τ_{resv}^j actually starts its execution on $\mathcal{S}^j$ (denoted by t_{start}^j) within the boundary of the reservation (i.e., $t_0 \leq t_{start}^j \leq t_{resv}^j - C_{resv}^j$) by calling $\textsc{minIsquare}(\mathcal{S}^j, t_0, t_{resv}^j - C_{resv}^j)$. We employ Algorithm 3 in [16] for the function of MINISQUARE, which finds t_{start}^j that minimizes $\sum I^2$ based on a local minimum of given time series of expected I of other sub-systems. Regarding the order of sub-systems to determine t_{start}^j, we sort sub-systems by $(t_{resv}^j - t_0)/P_{resv}^j$ to prioritize a sub-system with a shorter reservation period of $[t_0, t_{resv}^j)$ (implying less candidate time instants for τ_{resv}^j to start its execution) and larger power consumption of τ_{resv}^j (implying a higher impact on $\sum I^2$).

Finally, if there exists a sub-system $\mathcal{S}^j$ whose t_{start}^j is the same as the current time t_0, we start the execution of τ_{resv}^j on $\mathcal{S}^j$ and set its state to `running` (Lines 19–22). Note that when the system $\mathcal{S}$ starts, the state, t_{resv}^j and t_{start}^j for every sub-system $\mathcal{S}^j \in \mathcal{S}$ are set to `idle`, $-\infty$ and $-\infty$, respectively.

Separating the *sub-system reservation* stage for G1 from the *execution timing decision* stage for G2, the structure of `RSM` offers an interface to address Q3 (i.e., efficiently achieving G2 without compromising G1). However, the full achievement of G1 and G2 through the interface depends on how to derive the latest t_{resv}^j without compromising G1 by addressing Q1 and Q2. This will be presented in the next subsection.

4.2. MAXRESV*: Calculation of Maximum Tolerable Execution Delay*

As we explained in the example associated with Fig. 3(a), we can calculate the WCET-based slack of $\mathcal{S}^j$ at t_0 (denoted by $WS^j(t_0)$) defined in Definition 1, as long as we know the information of the next job release time after t_0 in $\mathcal{S}^j$ (denoted by $t_{njr}^j(t_0)$) and a set of jobs in $\mathcal{S}^j$ active at t_0 (denoted by $rdyQ^j$ at t_0). Line 1 of Algorithm 2 records the calculation of $WS^j(t_0)$, by subtracting the sum of WCET of every active job at t_0 on $\mathcal{S}^j$, from the length of the interval between t_0 and $t_{njr}^j(t_0)$. Then, utilizing the WCET-based slack does not compromise G1 as follows.

Lemma 1. *Suppose that there is no job in $\mathcal{S}^j$, which starts its execution before t_0 but does not finish its execution until t_0, and let τ_{resv}^j denote the highest-priority active job at t_0 on $\mathcal{S}^j$. Then, delaying the start of the job execution of τ_{resv}^j until $t_0 + WS^j(t_0)$ does not compromise the timely execution of any job in $\mathcal{S}^j$.*

PROOF. By the definition of $t_{njr}^j(t_0)$, there is no job release on $\mathcal{S}^j$ in the interval of $\left(t_0, t_{njr}^j(t_0)\right)$, implying that there is no job deadline on $\mathcal{S}^j$ in the interval due to the periodicity of each task. Therefore, it suffices to prove that every job active at t_0 finishes its execution until $t_{njr}^j(t_0)$, which trivially holds by $WS^j(t_0)$ calculated in Line 1 of Algorithm 2.

As we explained in the schedule in Fig. 3(c), it is simple to calculate the AET-based slack at t_0 defined in Definition 2. However, different from the WCET-based slack, the calculation of the AET-based slack at t_0 does not consider the situation where the AET-based slack at t_{-1} ($< t_0$) is not used or partially used for job execution delay. This necessitates the management of a series of AET-

Algorithm 2 MAXRESV($\mathcal{S}^j, t_0, t^j_{resv}, \tau^j_{resv}$)

1: $WS^j(t_0) = t^j_{njr}(t_0) - t_0 - \sum_{\tau^j_x \in rdyQ^j \text{ at } t_0} C^j_x$
2: **if** $t^j_{resv} < t^j_{njr}(t_0)$ **then**
3: $\quad AS^j(t_0) = t^j_{resv} - t_0$
4: **else if** $t^j_{njr}(t_0) \le t^j_{resv}$ **then**
5: $\quad AS^j(t_0) = \max\left(t^j_{njr}(t_0) - 1 - t_0, t^j_{resv} - C^j_{resv} - t_0\right)$
6: **end if**
7: **return** $\max\left(WS^j(t_0), AS^j(t_0), 0\right)$

based slacks at different time instants in a *cumulative* manner. Therefore, we calculate $AS^j(t_0)$, the *cumulative* AET-based slack of $\mathcal{S}^j$ at t_0, such that delaying the job execution until $t_0 + AS^j(t_0)$ does not compromise the timely execution of any job in $\mathcal{S}^j$.

To this end, we review our design of RSM in Algorithm 1, in which each sub-system $\mathcal{S}^j$ updates t^j_{resv}, until which the job of τ^j_{resv} can reserve (i.e., idle or execute in) $\mathcal{S}^j$ without missing any job deadline in $\mathcal{S}^j$. At t_0, after the highest-priority task τ^j_{resv} is selected by Line 8 of Algorithm 1, Algorithm 2 (called by Line 9 of Algorithm 1) calculates d^j_{slack} by utilizing t^j_{resv} assigned at a previous time instant t_{-1} ($< t_0$) from Line 10 of Algorithm 1; therefore, utilizing t^j_{resv} is a key to reclaim unused AET-based slacks at previous time instants. Let τ^j_{prev} denote τ^j_{resv} assigned by Line 8 of Algorithm 1 at t_{-1}. Along with the time instant t^j_{resv}, we also focus on $t^j_{njr}(t_0)$, the next job release time instant after t_0 due to the following property for non-preemptive work-conserving scheduling: a higher-priority job J_{HI} released at $t^j_{njr}(t_0)$ can be blocked by at most one lower-priority job J_{LO}, only if J_{LO} starts its execution before $t^j_{njr}(t_0)$. Therefore, if we want to rely on the target schedulability test to guarantee the timely execution of every job in a situation, it is not allowed to start the execution of J_{LO} at or after $t^j_{njr}(t_0)$ in the situation before J_{HI} finishes its execution.

Focusing on the time instants t^j_{resv} and $t^j_{njr}(t_0)$, we now derive $AS^j(t_0)$ with two cases (the latter of which consists of two sub-cases) as shown in Fig. 4: (Case 1) no job is released in $[t_0, t^j_{resv}]$ implying $t^j_{resv} < t^j_{njr}(t_0)$ in Line 2 of Algorithm 2, and (Case 2) a job (denoted by τ^j_{other}) is released at $t^j_{njr}(t_0) \in [t_0, t^j_{resv}]$ implying $t^j_{njr}(t_0) \le t^j_{resv}$ in Line 4 of Algorithm 2. All cases in Fig. 4 illustrate the following situation at t_0: τ^j_{prev} finishes its execution at t_0 although it reserves $\mathcal{S}^j$ until t^j_{resv}, and at t_0, τ^j_{resv} determines how long it can delay its execution. Note that each case in the figure describes a different time instant $t^j_{njr}(t_0)$ at which τ^j_{other} is released.

Since τ^j_{resv} is the highest-priority job at t_0, Case 1 implies that τ^j_{resv} is still the highest-priority job until t^j_{resv}. Since it was already guaranteed at t_{-1} ($< t_0$) that τ^j_{prev} can reserve $\mathcal{S}^j$ until t^j_{resv} without compromising any job deadline miss, executing the highest-priority job (i.e., τ^j_{resv})

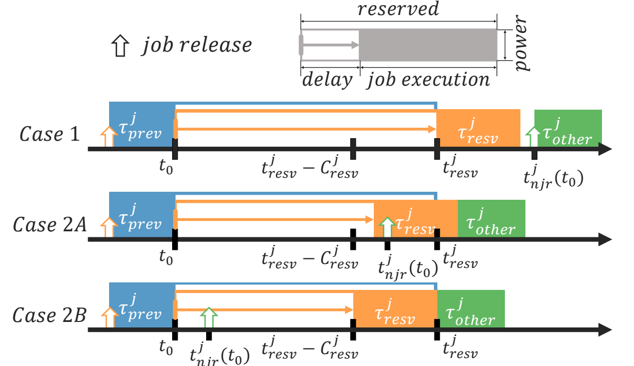


Figure 4: Different cases for calculation of $AS^j(t_0)$ in Algorithm 2.

no later than t^j_{resv} does not compromise any job deadline miss. Therefore, we can delay starting the execution of τ^j_{resv} until t^j_{resv}, as shown in Case 1 of Fig. 4, yielding Line 3 of Algorithm 2.

Under Case 2, τ^j_{other} could be the highest-priority job in $[t^j_{njr}(t_0), t^j_{resv}]$, and therefore executing τ^j_{resv} at or after $t^j_{njr}(t_0)$ may compromise the timely execution of τ^j_{other}. However, if we start the execution of τ^j_{resv} no later than $t^j_{njr}(t_0) - 1$ (i.e., before $t^j_{njr}(t_0)$), we can guarantee that τ^j_{other} suffers at most one lower-priority job blocking, which yields a legal schedule and therefore makes it possible for the target schedulability test to guarantee timely execution of τ^j_{other} and following jobs (as long as all jobs after τ^j_{resv} are scheduled in a work-conserving manner with the target prioritization policy), yielding the first max argument of Line 5 of Algorithm 2.

Since it is already guaranteed that τ^j_{prev} can reserve $\mathcal{S}^j$ until t^j_{resv} without compromising any job deadline miss, it also does not yield any job deadline miss to finish the execution of τ^j_{resv} until t^j_{resv} for both cases, implying that τ^j_{resv} can delay starting its execution until $t^j_{resv} - C^j_{resv}$, which is addressed in the second max argument of Line 5 of Algorithm 2, as shown in Case 2B of Fig. 4. Note that $t^j_{resv} - C^j_{resv} - t_0$ (in the second max argument of Line 5) is always smaller than $t^j_{resv} - t_0$ (in Line 3), so we do not apply the second max argument of Line 5 to Line 3.

The cumulative AET-based slack also does not compromise G1.

Lemma 2. *Suppose that there is no job in $\mathcal{S}^j$, which starts its execution before t_0 but does not finish its execution until t_0, and let τ^j_{resv} denote the highest-priority active job at t_0 on $\mathcal{S}^j$. Then, delaying the start of the job execution of τ^j_{resv} until $t_0 + AS^j(t_0)$ does not compromise timely execution of any job in $\mathcal{S}^j$.*

PROOF. Suppose that there is a job deadline miss. We show the contradiction when $AS_j(t_0)$ was calculated by (Case 1) Line 3 in Algorithm 2, (Case 2A) the first max

argument of Line 5, and (Case 2B) the second max argument of Line 5, which accords with Fig. 4.

(Case 1) Since τ^j_{resv} is the highest-priority task until t^j_{resv}, the supposition contradicts that the reservation by τ^j_{prev} until t^j_{resv} does not incur a job deadline miss.[2]

(Case 2A) Since the schedule in the interval starting at $t^j_{njr}(t_0) - 1$ is a legal schedule by the target prioritization policy $\mathcal{P}$ in a work-conserving manner, the supposition contradicts a schedulability test $\mathcal{T}_{\mathcal{P}}$ guarantees timely execution of every job subject to a legal schedule.

(Case 2B) Since the execution of τ^j_{resv} finishes no later than t^j_{resv}, the supposition contradicts that the reservation by τ^j_{prev} until t^j_{resv} does not incur a job deadline miss.

By Lemmas 1 and 2, we can delay the start of τ^j_{resv} for as much as d^j_{slack} in Line 9 of Algorithm 1, without incurring any job deadline miss.

4.3. Properties of RSM

Lemma 3. *The proposed run-time slack management RSM in Algorithm 1 can be applied to any pair of a prioritization policy $\mathcal{P}$ and its schedulability test $\mathcal{T}_{\mathcal{P}}$.*

Proof. First, since RSM does not use any specific job priority, any prioritization policy $\mathcal{P}$ can be applied to RSM. Second, since RSM does not use any property specialized for a certain schedulability test, any schedulability test $\mathcal{T}_{\mathcal{P}}$ can be applied to RSM.

Theorem 1. *The proposed run-time slack management RSM in Algorithm 1 satisfies the two requirements of the problem statement in Section 3: (i) achieving both G1 and G2 and (ii) being applicable to any pair of a prioritization policy and its schedulability test.*

Proof. Since (ii) is addressed by Lemma 3, it suffices to address (i). Regarding G1 in (i), Lemmas 1 and 2 guarantee no job deadline miss even in the presence of the execution delay for $\max\big(WS^j(t_0), AS^j(t_0), 0\big)$, which is the return value of $\textsc{maxresv}(\mathcal{S}^j, t_0, t^j_{resv}, \tau^j_{resv})$ in Algorithm 2. Therefore, no job deadline miss is guaranteed as long as the execution of τ^j_{resv} assigned in Line 8 of Algorithm 1 is finished no later than t^j_{resv} assigned in Line 10. This is achieved by making τ^j_{resv} start its execution no later than $t^j_{resv} - C^j_{resv}$ in Lines 16 and 20–21, which proves the achievement of G1 in (i).

As to G2 in (i), unless the sub-system utilization $U^j = \sum_{\tau^j_i} C^j_i / T^j_i$ is 1.0 *and* every job's AET is equal to WCET[3], RSM derives a positive value of d^j_{slack} (i.e., the maximum tolerable delay of the job execution), which is used to determine t^j_{start} (i.e., the actual time instant to start the job execution) that reduces battery aging. Therefore, RSM can reduce battery aging, while Section 6 shows how much battery aging is reduced.

Run-time complexity of RSM. The only additional steps for RSM are Lines 7–13 of Algorithm 1 that calls MAXRESV (i.e., the *sub-system reservation* stage) and Lines 14–18 of Algorithm 1 that calls MINISQUARE (i.e., the *execution timing decision* stage). The former operates in constant time while the latter exhibits $O\big(N^j \cdot \log(N^j) + d^j_{slack}\big)$ time-complexity due to sorting at Line 15 and checking the interval of length d^j_{slack} in MINISQUARE [16]. Therefore, RSM requires a little additional computational cost at run-time.

5. Integrating Run-Time and Offline Slack Management Frameworks

Our proposed RSM functions at run-time, so it can be utilized orthogonally with existing slack management frameworks that compute slack offline. In this section, we demonstrate how RSM can be integrated with offline slack management frameworks. As an example, we propose RSM+, a synergistic run-time and offline slack management framework that combines RSM with an offline slack management framework proposed by Kwak et al. [16].

5.1. Existing Offline Slack Management Framework

Kwak et al. [16] proposed a Reserved Execution Time (RET) framework, which is an offline slack management framework that statically utilizes unchangeable slack. RET intentionally inflates the WCET of tasks up to a limit that passes a target schedulability test at the offline time. Let IC^j_i and I^j_i denote the task i of sub-system j's inflated WCET (C^j_i) and the amount of inflated time, respectively, such $IC^j_i = I^j_i + C^j_i$. Tasks are then executed as if their WCET is IC^j_i, while the time of I^j_i is secured as offline slack. In this way, RET can use slack, but it cannot update nor reclaim its offline slack, compromising R3 in Section 1.

Fig. 5 depicts cases where RET, RSM, and RSM+ use slack to delay and run task τ^j_a and then reserve a sub-system j for the subsequent task τ^j_b. When to start running τ^j_a, RET delays the execution of τ^j_a with I^j_a and reserves its corresponding sub-system. However, after the reservation, it cannot reclaim the unused I^j_a for τ^j_b.

5.2. RSM+: Run-Time and Offline Integrated Slack Management

Our proposed RSM+ integrates RSM and RET. RSM and RET secure slack at orthogonal stages, but both of them utilize

[2] This is a mathematical induction, where the basis case with $t^j_{resv} = -\infty$ trivially holds.

[3] The former disallows a schedulability test to guarantee the schedulability of a set of non-preemptive tasks, and the latter is infeasible in reality.

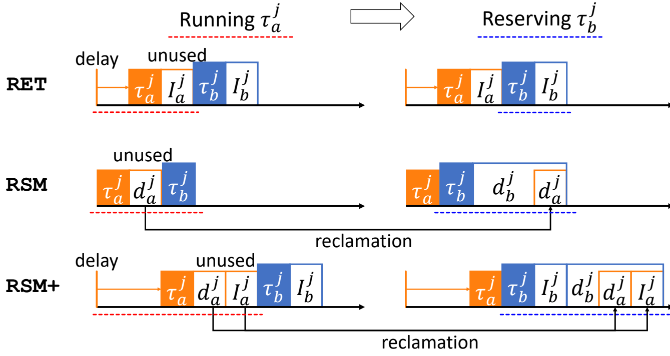


Figure 5: Comparing RSM+ with RSM and RET; RSM+ is superior to others, in terms of the efficiency of slack management.

slack for execution delay at run-time. Based on this principle, RSM+ integrates RSM and RET by securing slacks using the methods of RET and RSM independently and combining the slacks obtained from each method. The behaviors of RSM+ during offline and run-time stages are as follows:

Offline Stage. Before the system is operated, RSM+ inflates the WCET of the tasks as long as all task-sets pass their schedulability tests, according to the methodology of RET. During the WCET inflation, RSM+ prioritizes tasks in decreasing order of $\frac{P_i^j * C_i^j}{I_i^j}$ to give priority to tasks that consume more energy and make tasks be inflated evenly. Through this WCET inflation, the WCET of task i in a sub-system j becomes IC_i^j, and it obtains offline slack I_i^j.

Run-time Stage. The run-time behavior of RSM+ is not much different from RSM. RSM+ schedules and manages tasks in the same way as RSM in Algorithm 1. Since the actual WCETs of tasks are not changed, the calculation of MAXRESV (Algorithm 2) also does not change. Therefore, RSM+ calculates the same as RSM to get its run-time slack d_i^j.

Finally, RSM+ simultaneously utilizes the offline slack and run-time slack to reserve sub-systems. Line 10 of Algorithm 1 should be replaced by Equation (2) in RSM+, where $t_{resv}^j, t_0, d_{slack}^j, I_{resv}^j$, and C_{resv}^j denote time to reserve, current time, run-time slack, offline slack, and WCET. Since run-time slack and offline slack yield separate real-time guarantees in an independent manner, the two slacks can be integrated by addition.

$$t_{resv}^j \leftarrow t_0 + d_{slack}^j + I_{resv}^j + C_{resv}^j \quad (2)$$

In Fig. 5, both RSM and RSM+ utilize d_a^j to delay running τ_a^j and reclaim the unused slack for τ_b^j. The difference is that RSM+ additionally secures slacks I_a^j and I_b^j, providing chances of longer delay to tasks.

5.3. Advances of RSM+

Utilizing run-time and offline slack management together enables managing slack more efficiently than using only one. First, RSM+ reclaims offline slack, which is not possible in RET. RET cannot reclaim remaining unused slacks, as its slacks are calculated offline and thus fixed. On the other hand, RSM+ secures unused slacks by run-time slacks as RSM's AET-based slack calculation covers them. In Fig. 5, τ_b^j of RET is unavailable to reclaim unused slack I_a^j and waits for the duration of I_a^j before its execution, while τ_b^j of RSM+ reclaims it by d_b^j.

Second, RSM+ more evenly distributes slack opportunities to tasks compared to RSM. In Fig. 3., we discussed the chances for τ_b^x, τ_c^x, and τ_d^x to have slacks through RSM, but we did not discuss that there is nearly no such chance for τ_a^x. In RSM, there will be tasks with relatively few slack claim opportunities depending on their execution order, which causes a slack imbalance among tasks. RSM+ resolves the slack imbalance by distributing offline slacks to tasks in advance. τ_a^j of RSM in Fig. 5 has much less slack than τ_b^j, showing an imbalance, while RSM+ shows a more balanced slack distribution among tasks through offline slack management. As such, using run-time slack and offline slack together compensates for the disadvantages of each, enabling more efficient slack management.

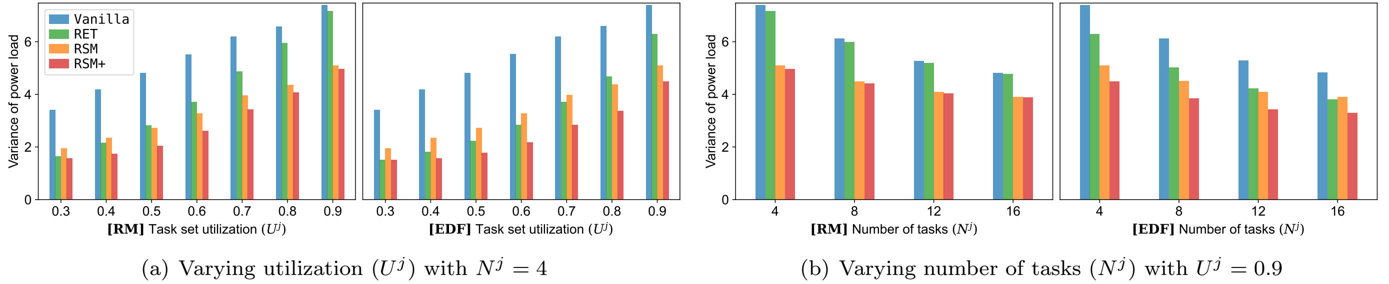
6. Evaluation

6.1. Experiment Setup

Target system. This paper focuses on safety- or mission-critical electric systems such as drones, UAVs, and EVs. These systems consist of several sub-systems in charge of actuation, sensing, computation, etc., and require high power loads (particularly for drones and UAVs). Considering these characteristics, we target a system that demands an average 5C load and consists of four sub-systems.[4] To model the workload of the system, we generate task sets for each sub-system $\mathcal{S}^j$, by varying utilizing ($U^j = \sum_{\tau_i^j} C_i^j / T_i^j$) and the number of tasks (N^j). For each task τ_i^j, T_i^j is uniformly chosen in $[10ms, 1000ms)$, C_i^j is generated based on UUniFast-Discard algorithm [25], and P_i^j is randomly generated such that the average system-wide discharge C-rate becomes 0.5C per 0.1 utilization. The AET of each task τ_i^j is employed to distribute normally between 10ms and its C_i^j (i.e. WCET) as the AET is known to follow a normal distribution [26, 27].

Battery emulator. We develop our own accurate battery emulator to evaluate battery aging based on

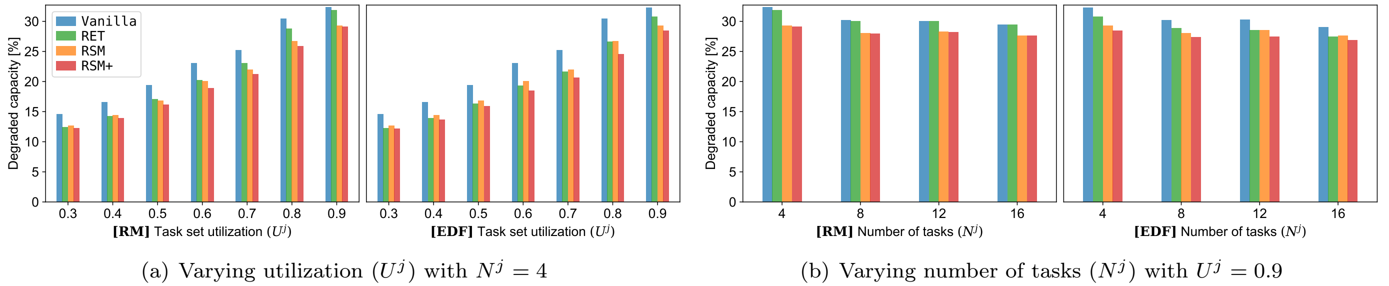
[4] The discharge C-rate of 1C means a constant current rate that completely discharges a fully-charged battery in one hour. xC is a constant current rate x times of 1C.



(a) Varying utilization (U^j) with $N^j = 4$

(b) Varying number of tasks (N^j) with $U^j = 0.9$

Figure 6: The power load variance, under different parameters of the utilization U^j, the number of tasks N^j and the prioritization policy.



(a) Varying utilization (U^j) with $N^j = 4$

(b) Varying number of tasks (N^j) with $U^j = 0.9$

Figure 7: The percentage of degraded capacity, under different parameters of the utilization U^j, the number of tasks N^j and the prioritization policy.

the thermal-electrochemical pseudo-2D (P2D) model [20, 28], the capacity degradation model for graphite anode LIBs [22], and cell parameters [29]. The P2D model mathematically calculates the ion transport, kinetics, and thermodynamics of a battery and is regarded as one of the most precise battery models [30]. The battery emulator models a 26650-size 2.3Ah lithium iron phosphate (LFP) battery cell, widely used in drones and EVs, at room temperature ($25°C$), and emulates the battery's internal state and aging with an error of less than 1% [20] and a 1.51 RMS error [22], respectively.

Experiment process. As an underlying prioritization policy, we apply EDF and RM (Rate Monotonic, FP that prioritizes a task with a smaller period). We generate task sets for 4,000 sub-systems (i.e., 1,000 systems), each of which passes the schedulability tests of non-preemptive uniprocessor scheduling under RM and EDF [31, 32, 33]. We compared the following run-time scheduling mechanisms.

- `Vanilla`: The vanilla non-preemptive and work-conserving scheduling (without being idle deliberately), which is traditional RM and EDF [31, 32, 33],
- `RET`: An existing approach based on the WCET inflation [16],
- `RSM`: Our approach in Algorithm 1 as it is, and
- `RSM+`: Our approach of Algorithm 1 by setting each task's WCET to the WCET inflated by `RET`.

We conduct our experiments in two stages: power load generation and battery emulation. The first stage simulates the power load of 1,000 different electric systems under every pair of a run-time scheduling mechanism and underlying prioritization policy. For the second step, we emulate the battery aging for 5,000 hours with the battery emulator, also under every pair of a run-time scheduling mechanism and underlying prioritization policy. The emulation repeats fully discharging the battery according to the generated power load and fully charging it at 1C. Note that our power load variance evaluation targets the whole 1,000 generated electric systems, while our battery aging evaluation targets 100 out of them due to overlong emulation time (taking 30 to 60 hours per system). From the two-stage experiment, we measure power load variance (for $\sum I^2$) and the percentage of degraded capacity (for battery aging).

6.2. Evaluation Results: Minimizing $\sum I^2$

Although the ultimate goal of `RSM` is to minimize battery aging, we first inspect system-wide power load variance to evaluate how `RSM` effectively reduces $\sum I^2$ as it is designed to minimize $\sum I^2$. We evaluate the power load variance with varying parameters of the system utilization U^j, the number of tasks N^j, and the prioritization policy $\mathcal{P}$, which are illustrated in Figs. 6(a) and 6(b). From the figures, we make the following observations.

O1. RSM results in a smaller power load variance than RET under many settings. In particular, as U^j gets larger or N^j gets smaller, RSM becomes more effective in reducing the power load variance.

O2. RSM+ always outperforms all other approaches in terms of minimizing the power load variance due to its effectiveness in various settings.

O3. In general, the trend of evaluation results for the power load variance under RM is similar to that under EDF, but there exist non-negligible differences.

Regarding O1, while we can easily find many settings in which RSM is more advantageous than RET in terms of reducing the power load variance, we observe that RSM becomes more effective in reducing the power load variance when U^j is higher. For example, in the settings under RM with $N^j = 4$, the ratio between the power load variance of RSM and RET is 118.8%, 96.3%, 84.1%, and 71.7%, respectively, for $U^j = 0.3$, 0.5, 0.7, and 0.9. This is because the amount of inflating WCET under RET becomes smaller as U^j becomes higher, while RSM efficiently reclaims and utilizes the run-time slack from the difference between AET and WCET for every U^j. When it comes to N^j, we observe that a smaller N^j yields more reduced power load variance of RSM. For example, in the settings under RM with $U^j = 0.9$, the ratio between the power load variance of RSM and RET is 71.1%, 75.2%, 78.6%, and 81.7%, respectively, for $N^j = 4$, 8, 12 and 16, due to the similar reason.

As to O2, we observe RSM+ outperforms other approaches for all settings. When $N^j = 4$ under RM, as shown in Fig. 6(a), RSM+ reduces Vanilla's power load variance by an average of 54.3%, 57.6%, 44.8%, and 33.0%, respectively, for $U^j = 0.3$ to 0.9 with a step of 0.2. Considering it is generally difficult to reduce power load variance with higher U^j due to limited flexibility to change each job execution time, the results demonstrate the effectiveness of RSM+ for various settings, which is different from RET, that reduces Vanilla's power load variance by an average of 51.8%, 41.3%, 21.4%, and 3.2% for the same settings. The reason why RSM+ dominates others for various settings is that it integrates the merit of RSM (effective for high U^j and small N^j) and RET (effective for low U^j).

For O3, we observe that the trend of power load variance under EDF is mostly similar to that under RM in Figs. 6(a) and (b). In particular, Vanilla makes almost no difference in the power load variance between RM and EDF, which indicates it is not easy to reduce the power load variance using any vanilla prioritization policy oblivious to power load. Since RSM does not have any mechanism that employs any job priority, RSM also yields a similar power load variance between RM and EDF. On the other hand, RET and RSM+ under RM yield less power load variance reduction than those under EDF. This comes from RET's dependency on the schedulability tests; since the schedulability test for RM allows less inflation of WCET than that for EDF, RET itself and RSM+ that utilizes the inflated WCET by RET exhibit such a different power load variance reduction between RM and EDF. However, RSM+ shows a smaller gap between the power load variance under RM and EDF than RET, as it employs features of not only RET but also RSM.

6.3. Evaluation Results: Minimizing Battery Aging

Now, we validate the effectiveness of RSM and RSM+ in minimizing battery aging. First, we validate our abstraction to minimize battery aging by minimizing $\sum I^2$ to lower battery temperature. We calculate the correlation coefficients between the power load and temperature, as well as between the power load variance and capacity degradation, which are 0.99 and 0.95, respectively, indicating robust positive correlations. As intended by our abstraction, minimizing $\sum I^2$ effectively reduces battery temperature and battery aging.

In accordance with the positive correlation between the power load variance and capacity degradation, the percentage of degraded capacity in Figs. 7(a) and (b) exhibits a similar trend to the power load variance in Figs. 6(a) and (b). Similar to the evaluation results of the power load variance, we observe that RSM yields a smaller capacity degradation than RET in many settings, and RSM+ *always* yields the smallest capacity degradation. In particular, RSM+ reduces capacity degradation by up to 32.6% and 28.2%, compared to Vanilla and RET, respectively.

Note that since the correlation coefficient is not 1.0, there exists a minor discrepancy between minimizing the power load variance and capacity degradation. For example, in the case of EDF with $U^j = 0.8$ and $N^j = 4$, the power load variance of RET (4.67) is larger than that of RSM (4.37) in Fig. 6(a), but the percentage of degraded capacity of RET (26.6%) is slightly smaller than RSM (26.7%) in Fig. 7(a). Although the minor discrepancy is a limitation that comes from the abstraction of complex battery behaviors, it does not damage our general claim, the effectiveness of RSM in achieving battery aging.

7. Related Work

Battery aging models and control. Battery aging is one of the most important factors in many industries from battery manufacturers to device or car makers. Many studies have investigated various aging mechanisms [19, 20, 21, 22, 23, 24, 34, 35, 36, 37]. Many researchers in the electrochemistry community analyzed and modeled batteries and their aging [19, 20, 21, 22, 23, 24, 34]. In addition, plenty of research aims to use such models to

control, predict, and inspect battery aging on various systems [35, 36, 37]. These studies provide principles to combat battery aging, however, they do not deal with timing guarantees and thus cannot be applied to real-time systems.

Battery aging-aware real-time systems. Recently, only a few studies have aimed to slow down battery aging in real-time systems requiring timing guarantees. Kwak et al. [16] proposed a scheduling framework, `RET`, to decelerate battery aging while meeting timing constraints by timely delaying real-time power-consuming tasks. `RET` framework searches for the opportunity to delay execution time (i.e., slack) in offline by analyzing the schedulability test of the designated scheduling policy. This way, `RET` delays execution to diminish heat generation from batteries to decrease battery aging. Similarly, Jang et al. [17] exploited the principles of `RET` framework to reduce battery aging in satellite systems. They rather increase heat generation from batteries considering the low-temperature environments of satellites. Unlike existing offline frameworks, the proposed online `RSM` framework analyzes execution delay opportunities at run-time, effectively reducing aging in the case of the static scheduler and high utilization task sets, as demonstrated in Section 6. We also proposed `RSM`+ framework by combining the benefits of both `RET` and `RSM`, thus showing the most effective solution space in decelerating battery aging in real-time systems.

Energy-, or power-aware schedulers for real-time systems. There have been a group of studies that address scheduling algorithms for improving energy efficiency [2, 3, 4], managing power requirements [5, 6, 7, 8], simply reducing energy consumption [9, 10, 11] or conserving energy consumption in a temperature-aware manner [12, 13, 14, 15] while guaranteeing the constraints of real-time systems. Such scheduling algorithms may help with battery usage, but expecting them to reduce battery aging is difficult as they are not directly related to battery aging. Specifically, these studies primarily focus on reducing either instantaneous power or overall energy consumption both of which affect battery aging. In contrast, our solution aims to directly reduce battery aging by efficiently scheduling both power and energy consumption.

8. Conclusion

In this paper, we proposed a run-time slack management framework that mitigates the battery aging in power-consuming real-time systems. The framework not only guarantees the timely execution of power-consuming real-time tasks but also is applicable to any scheduling policies and schedulability test without design changes. We also extended the proposed framework by integrating it with an existing offline slack management framework. We evaluated the proposed framework on a precise battery emulator varying multiple parameters and found that it significantly reduced battery aging (up to 32.6%) compared to other approaches. In future work, we will apply the proposed slack management framework to other domains besides battery aging and study a new framework for multi-core and mixed-criticality systems.

Acknowledement

This work was supported by the National Research Foundation of Korea (NRF) grant (2022R1A4A3018824, 2022R1G1A1003531, 2022K2A9A1A01097764, RS-2023-00248143) funded by the Korea government (MSIT). This work was also supported in part by ERC (NRF-2018R1A5A1059921) funded by the Korea government (MSIT).

References

[1] M. S. Ziegler, J. E. Trancik, Re-examining rates of lithium-ion battery technology improvement and cost decline, Energy & Environmental Science 14 (4) (2021) 1635–1651.

[2] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, P. Li, Energy-efficient scheduling for real-time systems based on deep q-learning model, IEEE transactions on sustainable computing 4 (1) (2017) 132–141.

[3] M. Chetto, H. El Ghor, Scheduling and power management in energy harvesting computing systems with real-time constraints, Journal of Systems Architecture 98 (2019) 243–248.

[4] D. Ramegowda, M. Lin, Energy efficient mixed task handling on real-time embedded systems using freertos, Journal of Systems Architecture 131 (2022) 102708.

[5] Y.-w. Zhang, R.-f. Guo, Power-aware scheduling algorithms for sporadic tasks in real-time systems, Journal of Systems and Software 86 (10) (2013) 2611–2619.

[6] E. Kim, J. Lee, L. He, Y. Lee, K. G. Shin, Offline guarantee and online management of power demand and supply in cyber-physical systems, in: RTSS, 2016, pp. 89–98.

[7] B. Ranjbar, T. D. Nguyen, A. Ejlali, A. Kumar, Power-aware runtime scheduler for mixed-criticality systems on multicore platform, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 40 (10) (2020) 2009–2023.

[8] M. Ansari, S. Safari, A. Yeganeh-Khaksar, M. Salehi, A. Ejlali, Peak power management to meet thermal design power in fault-tolerant embedded systems, IEEE Transactions on Parallel and Distributed Systems 30 (1) (2018) 161–173.

[9] J. Luo, N. K. Jha, Battery-aware static scheduling for distributed real-time embedded systems, in: Proceedings of the 38th annual Design Automation Conference, 2001, pp. 444–449.

[10] J. L. C. Hoffmann, A. A. Fröhlich, Online machine learning for energy-aware multicore real-time embedded systems, IEEE Transactions on Computers 71 (2) (2021) 493–505.

[11] S. Moulik, Z. Das, R. Devaraj, S. Chakraborty, Seamers: A semi-partitioned energy-aware scheduler for heterogeneous multicore real-time systems, Journal of Systems Architecture 114 (2021) 101953.

[12] S. Moulik, Reset: A real-time scheduler for energy and temperature aware heterogeneous multi-core systems, Integration 77 (2021) 59–69.

[13] Y. Sharma, S. Chakraborty, S. Moulik, Eta-hp: an energy and temperature-aware real-time scheduler for heterogeneous platforms, The Journal of Supercomputing 78 (8) (2022) 1–25.

[14] Y. Sharma, S. Moulik, Cetas: a cluster based energy and temperature efficient real-time scheduler for heterogeneous platforms, in: SAC, 2022, pp. 501–509.

[15] Y. Sharma, S. Moulik, Fats-2tc: A fault tolerant real-time scheduler for energy and temperature aware heterogeneous platforms with two types of cores, Microprocessors and Microsystems 96 (2023) 104744.

[16] J. Kwak, K. Lee, T. Kim, J. Lee, I. Shin, Battery aging deceleration for power-consuming real-time systems, in: RTSS, 2019, pp. 353–365.

[17] S. Jang, H. Yang, A real-time scheduling approach to mitigation of li-ion battery aging in low earth orbit satellite systems, Electronics 10 (1) (2021) 86.

[18] K. Smith, C.-Y. Wang, Power and thermal characterization of a lithium-ion battery pack for hybrid-electric vehicles, Journal of power sources 160 (1) (2006) 662–673.

[19] A. Barré, B. Deguilhem, S. Grolleau, M. Gérard, F. Suard, D. Riu, A review on lithium-ion battery ageing mechanisms and estimations for automotive applications, Journal of Power Sources 241 (2013) 680–689.

[20] A. Bizeray, S. Zhao, D. Howey, Lithium-ion battery thermal-electrochemical model-based state estimation using orthogonal collocation and a modified extended kalman filter, Journal of Power Sources 296 (20) (2015) 400–412.

[21] X. Lin, K. Khosravinia, X. Hu, J. Li, W. Lu, Lithium plating mechanism, detection, and mitigation in lithium-ion batteries, Progress in Energy and Combustion Science 87 (2021) 100953.

[22] X. Jin, A. Vora, V. Hoshing, T. Saha, G. Shaver, R. E. García, O. Wasynczuk, S. Varigonda, Physically-based reduced-order capacity loss model for graphite anodes in li-ion battery cells, Journal of Power Sources 342 (2017) 750–761.

[23] M. Alipour, C. Ziebert, F. V. Conte, R. Kizilel, A review on temperature-dependent electrochemical properties, aging, and performance of lithium-ion cells, Batteries 6 (3) (2020) 35.

[24] H. J. Ploehn, P. Ramadass, R. E. White, Solvent diffusion model for aging of lithium-ion battery cells, Journal of The Electrochemical Society 151 (3) (2004) A456.

[25] R. I. Davis, A. Burns, Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, in: RTSS, 2009, pp. 398–409.

[26] M. Berkelaar, Statistical delay calculation, a linear time method, in: proc. TAU, Vol. 97, Citeseer, 1997, pp. 15–24.

[27] M. H. J. Saldanha, A. K. Suzuki, Determining the probability distribution of execution times, in: ISCC, 2021, pp. 1–6.

[28] A. M. Bizeray, J. Reniers, D. A. Howey, Spectral_li-ion_spm: Initial release, doi:10.5281/zenodo.212178 (2016).

[29] L. Zhang, C. Lyu, G. Hinds, L. Wang, W. Luo, J. Zheng, K. Ma, Parameter sensitivity analysis of cylindrical lifepo4 battery performance using multi-physics modeling, Journal of The Electrochemical Society 161 (5) (2014) A762–A776.

[30] L. Xu, X. Lin, Y. Xie, X. Hu, Enabling high-fidelity electrochemical p2d modeling of lithium-ion batteries via fast and non-destructive parameter identification, Energy Storage Materials 45 (2022) 952–968.

[31] L. George, N. Rivierre, M. Spuri, Preemptive and non-preemptive real-time uniprocessor scheduling, Ph.D. thesis, Inria (1996).

[32] K. Jeffay, D. Stanat, C. Martel, On non-preemptive scheduling of periodic and sporadic tasks, RTSS (1991).

[33] A. Burns, K. Tindell, A. Wellings, Effective analysis for engineering real-time fixed priority schedulers, IEEE Transactions on Software Engineering 21 (5) (1995) 475–480.

[34] X. Zhang, Thermal analysis of a cylindrical lithium-ion battery, Journal of Power Sources 56 (1) (2011) 1246–1255.

[35] M. Jafari-Nodoushan, B. Safaei, A. Ejlali, J.-J. Chen, Leakage-aware battery lifetime analysis using the calculus of variations, IEEE Transactions on Circuits and Systems I: Regular Papers 67 (12) (2020) 4829–4841.

[36] L. Liu, H. Sun, C. Li, T. Li, J. Xin, N. Zheng, Managing battery aging for high energy availability in green datacenters, IEEE Transactions on Parallel and Distributed Systems 28 (12) (2017) 3521–3536.

[37] G. Karimi, X. Li, Thermal management of lithium-ion batteries for electric vehicles, International Journal of Energy Research 37 (1) (2013) 13–24.