Thermal-Aware Data Freshness: Formulation and Evaluation

***Abstract*—In safety-critical systems, one can find several applications such as Adaptive Cruise Control (ACC) with task chains, whereby a task produces output data that is consumed by another task in the chain. Typically, there is a freshness requirement for the data consumed by a task for the data to be valid. Working with invalid data can put the system at serious risk. Similarly, the reliable operation of the applications on the processor requires the processor temperature to remain within the allowed peak temperature. There isn't any other work in literature that presents an analysis framework to derive periods of a task set considering both end-to-end data freshness and thermal constraints. In this paper, we attempt to fill this gap systematically by presenting a thermal-aware data freshness analysis framework considering a 2-task scenario and then extend it to an n-task scenario. We perform evaluations for schedulability based on the task periods derived using the thermal-aware data freshness formulation proposed in this work.**

***Index Terms*—Data Freshness, Peak Temperature, Thermal Management, Real-Time Scheduling, Schedulability**

I. INTRODUCTION

Modern day automotive systems integrate a large number of tasks/applications on the available set of resources provided by the automotive platform. This integration and sharing of resources reduces the overall cost of the automotive as opposed to exclusive resource allocation for each task/application. However, the resource efficiency comes at the cost of complexity in resource allocation as many of the applications have strict timing constraints, which if violated can lead to serious consequences. In addition, the tasks may exchange some data between each other where some tasks are producers of data, some are consumers and some are both a producer and a consumer. They may also have different rates of execution, which in turn determines how old/fresh the sampled data by consumer tasks/applications is.

In a safety critical scenario, as in automotive applications such as Adaptive Cruise Control (ACC) [1], the freshness of data is of utmost importance. For example, in the ACC application, a task that periodically senses the angular velocity of the wheel provides input to another periodic task that calculates the linear velocity of the vehicle [1]. The freshness of the angular velocity data is important for ACC to derive the correct linear velocity, which will enable the vehicle to keep a safe distance from the leading vehicle. A safety critical application may consist of a chain of tasks and each pair of producer-consumer tasks may have a local data freshness requirement. However, it may also be very critical to adhere to the end-to-end freshness requirement of the entire application, which may have critical impact on the performance of the system. Therefore, it is imperative for all safety critical systems running applications such as ACC to undergo rigorous theoretical analysis to determine if the end-to-end freshness requirements are satisfied along with the individual deadline requirements of the tasks in the chain.

A. Motivation

Given a task chain consisting of periodic tasks, it is a non trivial problem to determine the period of each task such that the end-to-end freshness requirement is met while optimizing some objective such as total system utilization. In [2], the authors present a novel technique to determine the periods for a task chain considering end-to-end freshness requirement under very little assumptions regarding the underlying scheduling algorithm. The analysis for end-to-end data freshness becomes even more complex when other practical aspects are taken into consideration such as thermal limits of a processor and mode changes in an application functionality.

When we consider thermal capacity of a processor in terms of the peak processor temperature that is safe for its reliable operation, the end-to-end data freshness analysis presented in [2] cannot be applied. We highlight the above problem with the help of an example here. Assume there are 3 tasks in a chain, which are denoted by a_0, a_1 and a_2. Let the end-to-end data freshness requirement be defined as the maximum time taken between the output of task a_0 and the consumption of input by task a_2. In the normal scenario without thermal constraints, the tasks can be assigned periods according to the maximum staleness based analysis presented in [2]. However, the periods so assigned may result in execution patterns of tasks a_0, a_1 and a_2 that may lead to an instantaneous temperature higher than the peak temperature requirement of the processor. In order to avoid such increase in the instantaneous temperature, one common thermal management technique is to put the processor to intermittent sleep between processing or insert idle times. However, insertion of idle times has a non trivial influence on the data freshness value of the task chain. Therefore, in this work, we propose a method to solve the problem of analyzing end-to-end data freshness for a task chain consisting of n tasks under peak temperature constraints.

To the best of our knowledge, the thermal-aware data freshness formulation proposed in this paper is the first one that considers end-to-end data freshness and thermal constraint jointly in order to derive the periods of the tasks in a task chain. More specifically, our contributions in the paper are:

1) We first formulate the 2-task scenario and the optimization problem to derive the task periods under end-to-end and thermal constraints.
2) Then we extend our formulation to an n-task scenario.
3) We evaluate our theoretical framework with several task chains derived from the MiBench [3] automotive benchmark and demonstrate the variations in schedulability percentages with varying thermal and freshness constraints.

The paper is further organized in the following manner. In Section II, we present an overview of the related work in data freshness and temperature based analysis. Then we present the problem formulation for the thermal-aware data freshness formulation in Section III. Section IV discusses the thermal-aware data freshness formulation for a 2-task scenario which is extended to an n-task scenario in Section V. We present the experiments conducted for the thermal-aware data freshness formulation in VI. Finally, we present our conclusions in Section VII.

II. Related Work

There are two distinct lines of work related to our work in this paper namely data freshness based analysis and thermal management for real-time systems.

Data freshness is considered as a data quality metric in systems such as Web Servers [4] and Real-Time Databases [5], [6]. A comprehensive overview of freshness as a data quality metric and associated analysis framework is presented in [7], [8]. In [5], the authors studied the effects of update policies on freshness of derived data for a distributed real-time database and proposed a novel update policy. However, none of the above works modeled real-time tasks and derived task period parameters to ensure target freshness requirements. In [9], the authors propose a preemptive Last-Come First-Served policy and optimize freshness, throughput, and delay performance in multi-server information-update system. A deferrable scheduling algorithm is proposed in [10] for satisfying data freshness constraints while minimizing the update workload. This is done by deferring the sampling time of a transaction job as late as possible while ensuring temporal validity of the data. In contrast, we provide an analytical framework to derive the period of tasks under end-to-end data freshness constraint and thermal constraint.

Some prior works have looked at deriving task periods to satisfy individual latency requirements under dynamic priority scheduling [11] and static priority scheduling [12] methods. A heuristic was proposed in [13] to derive a feasible period-deadline combination in order to ensure schedulability of the task set under the assumption that task deadlines are piecewise first-order differentiable functions of the respective periods. Under EDF scheduling, task periods and deadlines were selected in [14] in order to enhance the control performance of a system. However, the above works do not consider data freshness constraint.

The authors in [15] have looked at the problem of period selection combined with data freshness requirement. They considered three classes of timing constraints namely freshness, correlation, and separation. An iterative pruning-based heuristic is used to derive the period, offset, and deadline of tasks such that the end-to-end constraints are met. In contrast, our work proposes an analytical framework to derive the task periods under data freshness and thermal constraints. Moreover, in [15], the periods considered for producer tasks were harmonic with respect to the periods of the consumer tasks. We do not impose any such a restriction. Similarly, there are several works on thermal management in embedded systems [16], [17] , which consider schedulability of tasks but do not consider data freshness as we do in this paper.

III. Problem Formulation

A. Task Model

A task a_j is modeled as a triple (P_j, e_j, D_i), where P_j is the period, e_j is the execution time and D_j is the relative deadline of task a_j. The execution time of i^{th} job of task a_j is denoted as (e_j^i). In this work, we assume that the relative deadline is equal to the period of the task for simplicity of explanation, i.e., $D_j = P_j$. The proposed method can be easily extended to a case where $D_j \neq P_j$. The worst case execution time of the task a_j is denoted by e_j^{max} and the best case execution time by e_j^{min}. We call a task as producer when it generates data for another task to process and consume. The task that receives the data is called the consumer.

Considering context switch time: The additional delay due to context switch within the range denoted as $[ct^{min}, ct^{max}]$ can be considered as part of the execution time, making the total execution time range for task a_j as $[e_j^{min} + ct^{min}, e_j^{max} + ct^{max}]$. Therefore, the entire proof explained in further sections will be valid if context switch or any such delay is considered, which can be accommodated in the execution time.

B. System Model

In this work, we consider a uni-processor platform for the execution of tasks. Every task interacts with its producer and/or consumer task via a shared memory, i.e., the consumer task has to consume the value from the same memory location that the producer task stored the result in.

C. Thermal Model

If the processor's average power is $P_{proc}(t)$ and ambient temperature is $T_{amb}(t)$, then the processor's instantaneous temperature is modeled as follows
$T(t) = T(0).e^{-t/RC} + (T_{amb}(t) + P_{proc}(t).R).(1 - e^{-t/RC})$
where t is the time instant, R and C are the thermal resistance and capacitance, respectively, and T(0) is the initial temperature of the processor [18].
Considering the types of nature of a task in thermal environment, we have two different types,

- Hot Task, whose execution leads to the increase in temperature of the system. and,

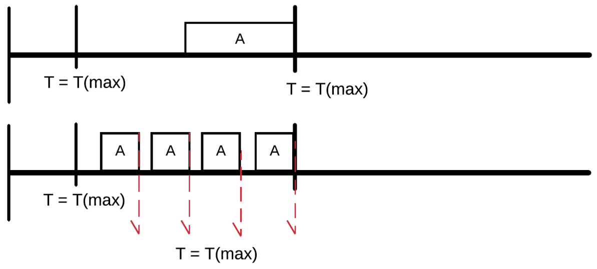


Figure 1: Non Splitting vs Splitting Mechanism

First case denotes the non splitting and the second case denotes the splitting mechanism. Red lines denotes all the cases of temperature, T = T_{max} while considering splitting

- Cold Task, whose execution leads to the decrease in temperature of the system.

All the tasks in the task set are considered as hot tasks because if any one of them is cold, the thermal constraints are relaxed for the overall system making the execution thermally safe. The peak temperature that a processor should not exceed is denoted by T_{max}.

The thermal management mechanism used in this work is based on insertion of idle times. The idle times can be inserted as one chunk before the task (as shown in the top half of Fig. 1) or the task can be split into multiple chunks and an idle time can be inserted before each chunk (as shown in the bottom half of Fig. 1). The idle time required to cool the processor is derived based on the following two cases -

- The temperature at the start time of another task is T_{max}
- The temperature at the start time of the task is less than T_{max} but the execution of the fraction of the task leads to the increase in the temperature to T_{max}, which requires an idle time for the processor to cool off.

In this work, we follow the splitting mechanism [18], which is described next.

Splitting Mechanism

Consider a situation where temperature at the start of a task is T_{max}, thereby having the need to insert idle time to reduce the processor's temperature. As illustrated in the Fig. 1, there are two ways of insertion of idle times -

- Insert the idle time to lower the temperature such that the whole task A can run without reaching T_{max} in between.
- Split task A into m parts such that after the execution of every part, the temperature rises to T_{max}. m is chosen optimally such that the preemption overhead cost never outweighs the benefit of splitting.

The total idle time required when we follow the splitting technique is always lesser than the original non-splitting mechanism [18].

The idle time is a function of (a) temperature at the start time of the task i.e. $T(t)$, and (b) execution time of the remaining fraction of the job i, of the task a_j that needs to split i.e. $e_j^{i,f}$. So we denote idle time as $T_{idle}(T(t), e_j^{i,f})$ which is a unique pair for a given task and given initial temperature. The fraction of the task a_j executed is denoted as $a_j^{i,f}$ where i is the job and f is the fraction of the task a_j executed.

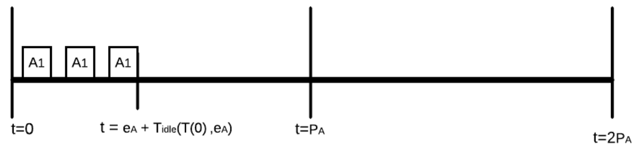


Figure 2: Generic Scenario for Two Task Result

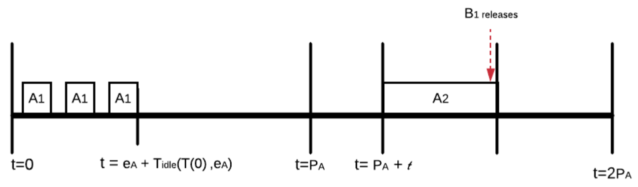


Figure 3: Scenario for B_1 executes after start time of A_2 when the second instance of A never splits

IV. Two Task Result

For the two task scenario, we denote the producer task by A and consumer task by B, where A_i and B_i denotes the i^{th} job of task A and B respectively. The two task results depends on the initial configuration of the system consisting of the initial temperature as its primary constituent. We obtain the maximum staleness bound, by considering two different cases -

- $T_{initial}$, where first instance of A reaches T_{max} either from the start or in between the execution and follows the splitting mechanism.
- $T_{initial}$, where first instance of A never reaches T_{max} in between its execution and therefore, never splits.

A. *Formulation - Initial Temperature such that first instance of A splits*

The idle time insertion for the first instance of task A are illustrated in Fig.2. In this case, $e_A^1 + T_{idle}(T_0, e_A) \leq P_A$ must follow, as the first instance of A has to meet its deadline for maintaining the schedulability. So, now we have to decide which of the following execution - the second instance of A denoted as A_2 or first instance of B denoted as B_1 has to be executed, which brings us to the two sub-cases.

1) B_1 executes after start time of A_2: Let A_2 execute after some time τ from P_A as illustrated in Fig. 3. Referring to the timeline diagram, it is evident that the processor gets some time precisely, t = P_A + τ - (e_A^1 + $T_{idle}(T_0, e_A)$) to cool implying that the temperature at the start time of A_2 is not

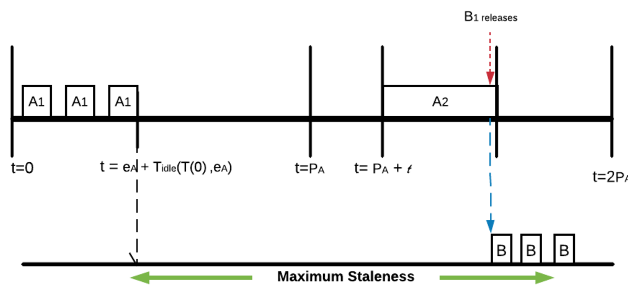


Figure 4: Maximum Staleness Scenario when the second instance of A never splits

T_{max}. Now we consider two cases here, where, either the execution of A_2 never leads to increase in the temperature to T_{max} or, the temperature rises to T_{max} in between the execution.

a) Temperature at t = P_A + τ + execution of A_2 is less than T_{max}: In this case, assume that T(t = P_A + τ + execution of A_2) = T_{max} - α such that $\alpha > 0$. So B_1 execution happens at time instant defined as ϵ before the finish time of A_2 where $\epsilon \rightarrow 0$.

Theorem IV.1. *In the Fig. 4, where temperature at t = P_A + τ + execution of A_2 $< T_{max}$, the maximum staleness factor is,*

$$S_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B^{1,f}/m) \quad (1)$$

Proof. The execution of B_1 may rise to T_{max} in between. As illustrated in the Fig.4, if B_1 splits, then the data generated by A_1 should be consumable till the last part of B_1 as no task can generate its own local copy, thereby concluding that every part of the task B_1 should be able to consume the data generated by A_1. There is no need of including the last part's execution in the calculation of data freshness as no further data is being consumed by B_1.

Assuming m is the optimal number of parts of B_1 derived from the splitting policy, the staleness factor denoted by S is computed,

$$\begin{aligned} S = P_A + \tau - (e_A^1 + T_{idle}(T_0, e_A)) + (e_A^2 - \epsilon) \\ +(e_B^1 + T_{idle}(T_{max} - \alpha, e_B^{1,f})) - (e_B^{1,f}/m) \end{aligned} \quad (2)$$

where $e_B^{1,f}$ is the execution time left for B_1 to execute after reaching T_{max} in between.
Maintaining the strict thermal constraint and obtaining the maximum staleness factor, also requires the schedulability to be met by the task set. So, imposing the condition, we get

$$P_A + \tau + e_A^2 + (e_B^1 + T_{idle}(T_{max} - \alpha, e_B^{1,f})) \leq 2P_A \quad (3)$$

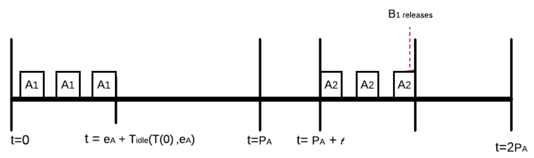


Figure 5: Scenario for B_1 executes after start time of A_2 when the second instance of A splits

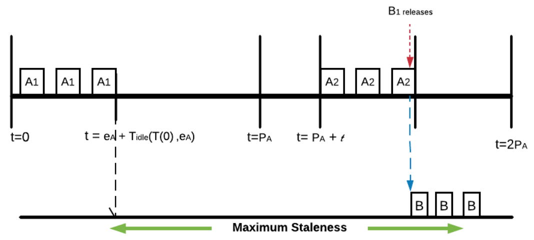


Figure 6: Maximum Staleness Scenario when the second instance of A splits

To find S_{max} we can state that the Eq.3 can be used with the equality operation as otherwise, we will obtain a lesser staleness bound than the desired. Using the Eq.2 and Eq.3 i.e. substituting, the variable τ, we obtain

$$\Rightarrow S_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B^{1,f}/m) \quad (4)$$

, assuming $\epsilon \rightarrow 0$ and m chosen optimally for obtaining the minimum total idle time.

□

Lemma IV.2. *The maximum staleness S_{max} remains the same even when B_1 doesn't split, when B_1 executes after the start time of A_2.*

Proof. Unlike depicted in Fig. 4, the maximum staleness factor in the case where the B_1 doesn't split, will be considered only till the start of B_1. This would result in a difference of $e_B - e_B/m$ (Assuming m is the number of parts of B_1 derived from the splitting policy) in the staleness factor, thereby lessening the staleness, making no difference in the **maximum** staleness.

□

b) Temperature at t = P_A + τ + execution of $A_{f,2}$ is T_{max}: In this case, the following Fig. 5 and Fig.6, illustrate the staleness equation.

Theorem IV.3. *In the Fig. 6, where temperature at t = P_A + τ + execution of A_2 $= T_{max}$, the maximum staleness factor is,*

$$S_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B/m) \quad (5)$$

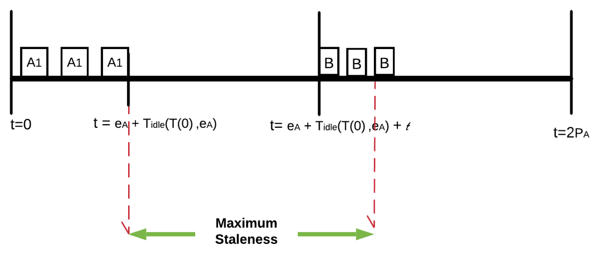


Figure 7: Staleness Scenario when first instance of B splits

Proof. Consider, the temperature at t = $P_A + \tau$ as T_{start}. Also, the temperature at the beginning of B's execution is considered $\approx T_{max}$ because $\epsilon \to 0$.

$$\begin{aligned} S = (P_A - e_A^1 - T_{idle}(T_0, e_A)) + \tau \\ +(e_A^2 + T_{idle}(T_{start}, e_A^{2,f}) - \epsilon) \\ +(e_B^1 + T_{idle}(T_{max}, e_B)) - (e_B^1/m) \end{aligned} \tag{6}$$

, where $e_A^{2,f}$ is the execution time left for A_2 after reaching T_{max} in between.
In this case, the schedulability is defined as

$$\begin{aligned} P_A + \tau + e_A^2 + T_{idle}(T_{start}, e_A^{2,f}) + e_B^1 + \\ T_{idle}(T_{max}, e_B) \leq 2P_A \end{aligned} \tag{7}$$

To find S_{max} we can state that the Eq.7 can be used with the equality operation as otherwise, we will obtain a lesser staleness bound than the desired. Using the Eq.6 and Eq.7, i.e. substituting τ, we get the final equation as,

$$\Rightarrow S_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B/m) \tag{8}$$

, assuming $\epsilon \to 0$ and m is chosen optimally for obtaining the minimum total idle time. □

Notice that in Eq.4 and Eq.8, m are two different values. By theory of approximation, if we assume α to be a very small quantity then we can say that $e_B/m_1 \approx e_B^{1,f}/m_2$. This approximation helps us maintain a stricter thermal constraint. One might wonder that why should we consider the start of B_1 just before ϵ time of finish time of A_2. This is because if we consider any other case, the staleness factor is no more the maximum.

2) Assuming B_1 executes before the start time of A_2: Consider Fig. 2 as the starting point for this sub-case too. Assume that after time τ after $e_A^1 + T_{idle}(T_0, e_A)$, B_1 starts its execution, such that $\tau \geq 0$. Here also, we can witness two sub-cases - B_1 executes and never reaches T_{max} and other one where it reaches T_{max} in between.

a) Temperature at t = $e_A^1 + T_{idle}(T_0, e_A) + \tau + e_B^{1,f}$ is T_{max}: Assume that the temperature at the start time of B_1 is T_{start}.

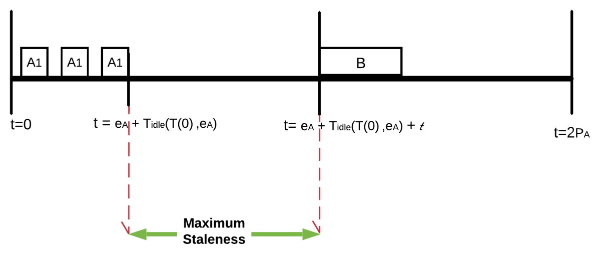


Figure 8: Maximum Staleness Scenario when the first instance of B never splits

Theorem IV.4. *In this sub-case, when B_1 reaches T_{max} in between its execution, the maximum staleness factor is calculated as,*

$$\begin{aligned} S_{max} = 2P_A - 2e_A - T_{idle}(T_0, e_A) - \delta \\ -T_{idle}(T_\delta, e_A) - (e_B^{1,f}/m) \end{aligned} \tag{9}$$

Proof. As shown in Fig. 7,

$$S = \tau + e_B^1 + T_{idle}(T_{start}, e_B^{1,f}) - (e_B^{1,f}/m) \tag{10}$$

, where $e_B^{1,f}$ is the execution time of B_1 left after it reaches T_{max} in between. Let δ be the time difference between the finish of B_1 and the start of A_2, and the temperature at the start of A_2 is T_δ The schedulability condition is stated as

$$\begin{aligned} e_A^1 + T_{idle}(T_0, e_A) + \tau + e_B^1 + T_{idle}(T_{start}, e_B^{1,f}) \\ +\delta + e_A^2 + T_{idle}(T_\delta, e_A) \leq 2P_A \end{aligned} \tag{11}$$

Using Eq.10 and Eq.11, i.e. substituting τ, we get,
For staleness to be maximum, τ needs to be maximised and it is possible when the Eq.11 follows the equality as when it becomes equal, τ reaches it peak.

$$\begin{aligned} \Rightarrow S_{max} = 2P_A - 2e_A - T_{idle}(T_0, e_A) - \delta \\ -T_{idle}(T_\delta, e_A) - (e_B^{1,f}/m) \end{aligned} \tag{12}$$

□

b) Temperature at t = $e_A^1 + T_{idle}(T_{max}, e_A) + \tau + e_B < T_{max}$: As shown in the Fig. 8 and Fig. 7, we can say that the staleness factor in this case will be definitely be less, implying that the maximum staleness bound is unaffected by considering this case.

3) Discovering the period P_A: Let $d_{A\to B}$ denote the upper bound on data freshness for data produced by task A and consumed by task B.

Theorem IV.5. *If the first instance of A splits, the period of task A is,*

$$P_A \leq \frac{1}{2}[d_{A\to B} + e_A + T_{idle}(T_0, e_A)) + (e_B/m)] \tag{13}$$

Proof. From Eq.4, Eq.8 and Eq.12, we can deduce that the maximum staleness is,

$$S_{max} = 2P_A - (e_A^1 + T_{idle}(T_0, e_A)) - (e_B/m) \tag{14}$$

Therefore,

$$d_{A\to B} \geq 2P_A - (e_A + T_{idle}(T_0, e_A)) - (e_B/m) \tag{15}$$

Therefore, the period of task A,

$$P_A \leq \frac{1}{2}[d_{A\to B} + e_A + T_{idle}(T_0, e_A)) + (e_B/m)] \tag{16}$$

□

The aim of our result is to obtain the maximum data freshness under strict thermal bounds, satisfying the objective of minimizing the maximum CPU utilization.

$$U(t) = \frac{e_A^{max}}{P_A} \tag{17}$$

Notice that we assumed the minimum execution times of the task set for maximum staleness in the formulation.
To minimize utilization, we have to ensure the period to be at its maximum. Using period obtained from Eq.16, we get,

Utilization in Two-Task Scenario,

$$U(t) = \frac{e_A^{max}}{\frac{1}{2}[d_{A\to B} + e_A^{min} + T_{idle}(T_0, e_A^{min})) + (e_B^{min}/m)]} \tag{18}$$

where we have considered e_A^{max} because our solution will ensure freshness even with best-case execution of the first task, while minimizing the maximum utilization the system could experience.

B. Formulation - Initial Temperature such that first instance of A never splits

We will follow the same exact procedure as done in the case when A_1 splits, but before that lets look at the result we have obtained in Eq. 14.

Lemma IV.6. *Staleness is maximum when B_1 executes after the start time of A_2.*

Lemma IV.7. *Under thermal constraints, Maximum staleness solely depends on the initial temperature T(0) and the B_1's start temperature.*

From the above two lemmas we can say that there are only two cases we need to consider,

1) B_1 splits: From Eq.14, we can say that if A_1 does not split, then $T_{idle}(T_0, e_A)) = 0$. The staleness factor becomes,

$$S_{max} = 2P_A - e_A^1 - (e_B/m) \tag{19}$$

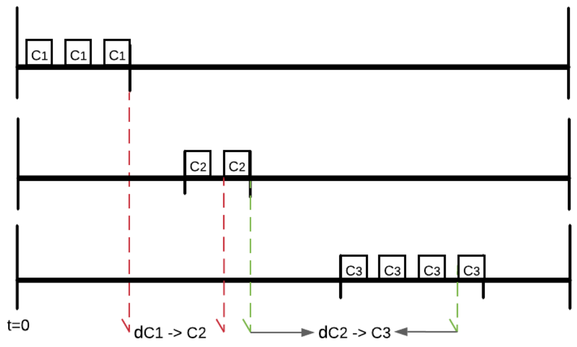


Figure 9: Staleness Factor Scenario For Three-Task set

2) B_1 does not split: From Eq.19, we can say that if B_1 does not split, then the staleness is measured only till the start of B_1, implying that e_B/m need not be considered as B executes without any split. The staleness factor becomes,

$$S_{max} = 2P_A - e_A^1 \tag{20}$$

Amongst these two cases, the **maximum staleness factor** is

$$S_{max} = 2P_A - e_A^1 \tag{21}$$

, therefore, **Utilization :**

$$U(t) = \frac{e_A^{max}}{\frac{1}{2}[d_{A\to B} + e_A^{min}]} \tag{22}$$

V. N Task Result

For the N-task scenario, C_1 to C_n denotes the task chain of length n, where C_i denotes a task. In this case, C_i acts as a producer task for C_{i+1} and, acts as a consumer task for C_{i-1}. In this section, we extend our two task formulation to N-task result.
Key points which can be derived from the Eq. 14 are :

- Staleness is maximum when A_2 starts before the execution of B_1 begins.
- In two task result, freshness bound didn't consider any task's execution in between the execution of the producer and consumer task, but in N-task scenario, its quite possible. But if we look closely to the result obtained, freshness bound is independent of any activity happening in between the two tasks. Freshness bound is independent of the idle time of A_2 implying that initial temperature of the start of A_2 is not the part of the equation and as we know that any task's execution in between solely affects the temperature, we can conclude that irrespective of any fraction of task running in between pair of tasks, the freshness bound remains unchanged.

Theorem V.1. *In N-task, where C_i denotes the task in a task set where $1 \leq i \leq n$, the end to end data freshness bound is,*

$$d_{C_1\to C_{n+1}} = \sum_{i=1}^{i=n} d_{C_i\to C_{i+1}} + \sum_{i=2}^{i=n} \gamma_i * (e_{C_i}/m_i) \tag{23}$$

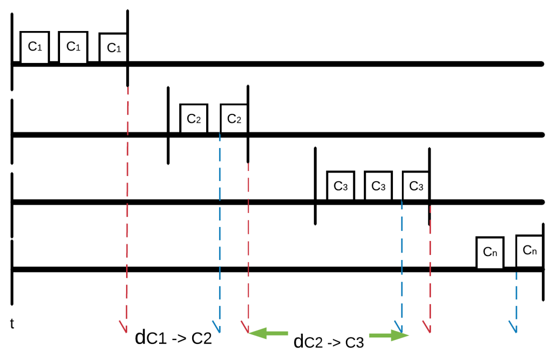


Figure 10: Staleness Factor Scenario For N-Task set

where $\gamma_i \epsilon \{0,1\}$

Proof. Proving using Principle of Mathematical Induction,

Base Case :
As shown in the Fig. 9, we can say that if C_1, C_2 and C_3 all split, then

$$d_{C_1 \to C_3} = d_{C_1 \to C_2} + e_{C_2}/m + d_{C_2 \to C_3} \tag{24}$$

In case C_1 or C_3 does not split, it does not make any difference to the Eq.24 but if C_2 does not split then the equation transforms as,

$$d_{C_1 \to C_3} = d_{C_1 \to C_2} + d_{C_2 \to C_3} \tag{25}$$

C_1 and C_3 split influences the data freshness value.
So we can conclude that if $\gamma \epsilon \{0,1\}$,

$$d_{C_1 \to C_3} = d_{C_1 \to C_2} + \gamma * (e_{C_2}/m) + d_{C_2 \to C_3} \tag{26}$$

Induction Step :
Assuming that the result is true for n task result, we will try to prove that it exists for n+1 task result.
As shown in the Fig. 10, for n task set, we get

$$d_{C_1 \to C_n} = \sum_{i=1}^{i=n-1} d_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n-1} \gamma_i * (e_{C_i}/m_i) \tag{27}$$

. where $\gamma_i \epsilon \{0,1\}$
So, we have to obtain the result for n+1 task which should be

$$d_{C_1 \to C_{n+1}} = \sum_{i=1}^{i=n} d_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n} \gamma_i * (e_{C_i}/m_i) \tag{28}$$

where $\gamma_i \epsilon \{0,1\}$

One more addition of a task leads to the equation,

$$\begin{aligned} d_{C_1 \to C_n} = & \sum_{i=1}^{i=n-1} d_{C_i \to C_{i+1}} + d_{C_n \to C_{n+1}} + \\ & \sum_{i=2}^{i=n-1} \gamma_i * (e_{C_i}/m_i) + \gamma_n * (e_{C_n}/m_n) \end{aligned} \tag{29}$$

where γ_n represents the possibility of the split of n^{th} task. If we look carefully at Eq.29, then we can see that it is

$$d_{C_1 \to C_{n+1}} = \sum_{i=1}^{i=n} d_{C_i \to C_{i+1}} + \sum_{i=2}^{i=n} \gamma_i * (e_{C_i}/m_i) \tag{30}$$

. where $\gamma_i \epsilon \{0,1\}$

Hence proved

□

In the above derivation, the consecutive two task stitching mechanism is considered. The above case, can be extended to m task stitching depending on which tuple represents the minimum data freshness amongst all.
So, we get our optimization formulation as,

Given : Let $E = \{1, 2, 3...n\}$, e_i^{max}, e_i^{min} for each $i \,\epsilon\, E$

Find : $d_{C_i \to C_j}$ for each $(i,j) \,\epsilon\, E$

Objective : Minimize $U = \sum_{i=1}^{i=n-1} \frac{e_{C_i}^{max}}{P_{C_i}}$ which is,

$$\sum_{(i,*)\epsilon E} \frac{2 * e_{C_i}^{max}}{min(d_{C_i \to C_*}) + e_{C_i}^{min} + \gamma_i * (T_{idle}(T_{0,i}, e_{C_i}^{min})) + \beta_i * (e_{C_*}^{min}/m_{C_*})}$$

where γ_i and β_i $\epsilon\{0,1\}$ and $T_{0,i}$ denotes the temperature at the start of the execution of task C_i.
γ_i and β_i are 0 or 1 depending on whether in between tasks split or not and the initial temperature of every part splits their first task or not respectively.

Subject To :
$$\forall\, k, \sum_{i:\{i,j\}\epsilon E_k} d_{C_i \to C_j} + \sum_{i:\{i,*\}\epsilon E_k} (e_{C_i}/m_i) \leq d_{C_1 \to C_n}$$
and, $i : \{i,j\}\epsilon E, d_{C_i \to C_j} \geq 0$

In this formulation, stitching of the tasks has been done such that it yields the minimum data freshness.

VI. Evaluation

A. N-Task Theory Evaluation

We have implemented and evaluated our theory with values provided by MiBench [3]. Our evaluation platform for taskset values is i.MX6 with ARM A9. We used tasks from the automotive section of the benchmark. We use utilization based schedulability test for EDF.
Our evaluation focuses on verifying whether the proposed theory guarantees the thermal and real-time scheduling constraints while maintaining data freshness. Further, we evaluate how schedulability and utilization changes while varying the desired freshness and temperature ranges. And lastly, we examine and see how schedulability changes with varying chain lengths as we modify the temperature constraints.

1) Experimental Setup: The task graphs we obtained have between 2 and 12 tasks. In total, we collected 16 two-task scenarios and 20 three-task scenarios from the benchmark suite. For the experiments, we simulated EDF scheduler on Linux. To gather as many relevant examples as possible for the special cases, we sometimes combined several serial tasks into one large task. However, we always considered the temperature ranges from the real use case scenarios, so that the results could be used meaningfully. We used WCETs of the first hardware configuration given in the benchmark for subset of given range of period values, and chose BCET to be the half of WCET. We considered context switch time as 1% of WCET, for it to be a part of execution time for evaluation.

2) Experiments: Optimization code used for derivation of periods of the tasks in a chain and the EDF Simulation along with the thermal model, used to schedule the task chain with given freshness bound and temperature constraints are shown in Fig 11.

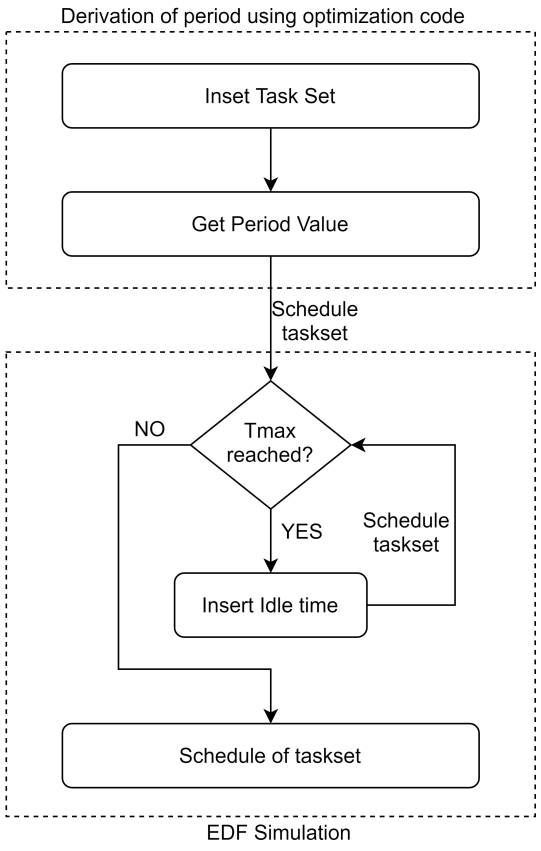
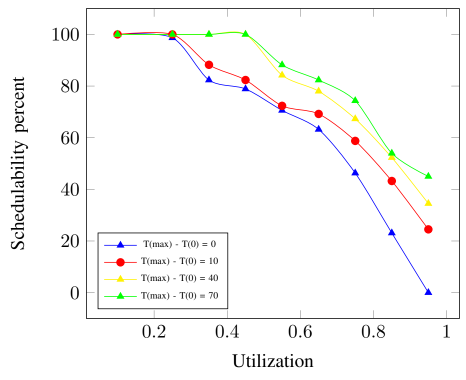


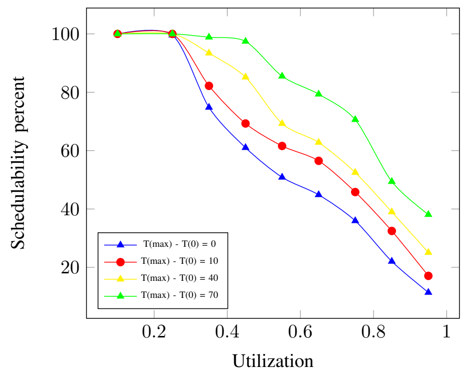
Figure 11: Evaluation Flow Diagram

a) ***Schedulability variation with Utilization factor :*** For the first experiment, we ran task chains of different length for various percentage of freshness bound. We set up the tasks to run for 99% of the specified WCET to account for scheduler overhead and context switching.The final values we got are the schedulability percentage for uniformly spread utilization values over a set of temperature ranges defined for the processor.

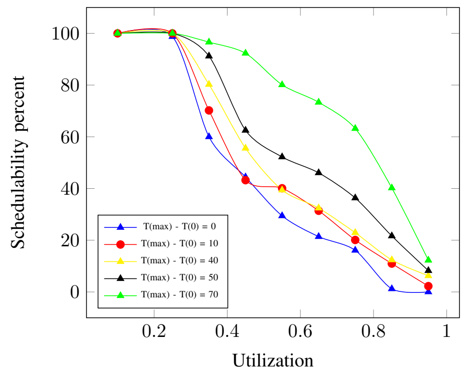
Parameters of the thermal model i.e. T_{idle} and m, of all the task sets for all the considered temperature ranges, are calculated by our implementation, following the detailed scheduling and data freshness experiment for all task sets. Tasks were all



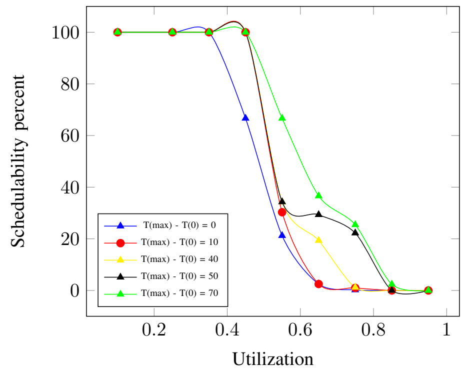
(a) Freshness bound = 80%



(b) Freshness bound = 55%



(c) Freshness bound = 30%



(d) Freshness bound = 12%

Figure 12: Schedulability Percentage and Utilization at various temperature ranges for different freshness bounds. (a)80%, (b)50%, (c)30%, and (d)12%

run on the same core with no external loads added.
We choose to vary data freshness bound with respect to the end-to-end data freshness bound of the task set. Fig. 12 shows variation of schedulability for different temperature ranges and various freshness bounds (12%, 30%, 50%, and 80%).
In Fig. 12(a), we fix the freshness bound at 80%. Here, for a constant utilization (assume a line parallel to y-axis), we observe that as the thermal bound gets more constrained, the average schedulability decreases. From the perspective of a particular schedulability percentage, (assuming the line parallel to x-axis), we observe that as the thermal bound gets more constrained, processor utilization decreases drastically implying that the higher the thermal constraint, the less is the utilization, inferring that the longer task chains are less likely to get scheduled under strict thermal bounds.

Now, on varying data freshness bound from 80%, Fig. 12(a), to 12%, Fig. 12(d), we observe that for each temperature difference, graph shifts from concave downward to convex upward. This happens because, as the data freshness bound decreases, the period of tasks would decrease, which in turn makes the schedulability difficult due to close deadlines.

It is also noticeable that, across different data freshness bounds, there is a steep decrease in schedulability if we consider both, strict thermal constraint ($T_{max} - T(0) = 0$) and strict data freshness bound (12%). If either of the constraint is relaxed, we can witness decrease in schedulability, but lesser than the previous case(where both the constraints were strict), and finally, if we relax both the constraints, schedulability is at its best.

*b) **Schedulability variation under different freshness bound:*** For the second experiment, for a particular task set we have found out idle time and 'm' value for different temperature ranges. Using these values we calculate the maximum staleness factor, S_{max}. Then, we find the minimum possible staleness value for which the task set is schedulable, stating in other words that if we reduce the staleness factor any further, the periods obtained would not allow the task set to be schedulable. This freshness value is assigned to be S_{min}. Now, we schedule all the possible periods, knowing this range of freshness bound (S_{min} to S_{max}), for different thermal constraints, i.e. temperature difference from 0 to 80. With thermal constraints implemented, we find the percent of these task chains that were schedulable, and then plot these values in the graph 13.

In Fig. 13, Firstly we can see that for a constant staleness time as the temperature difference increases, the schedulabity of task increases. For staleness time ranging from 21.79s to 23.2s, when temperature difference is changed from 0 to 80, average schedulability is 46.75% with a minimum of 12.32% and maximum of 76.8%. Secondly, for a constant temperature difference as the freshness bound increases, the schedulablity increases in a concave upward fashion. Lastly, for a constant schedulability, as the thermal bound increases, maximum staleness factor increases, which concurs with the theoretical formulation as stated in Eq. 14 .

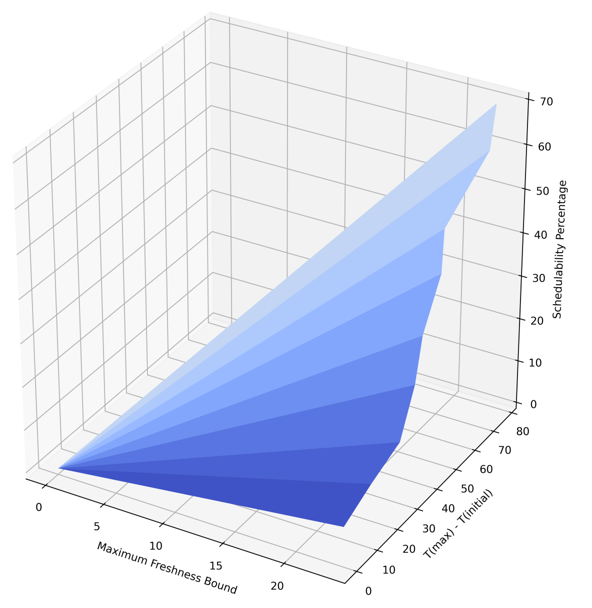


Figure 13: Schedulablilty with different freshness bounds under various temperature constraints

B. Optimization Formulation Evaluation

To evaluate the optimization formulation we implemented the described optimization problems in MATLAB. As described in the previous evaluation, we obtain the values of idle time and m for all the tasks for various temperature ranges considered. The data freshness for every task chain has been fixed at 30% of the maximum freshness bound. These parameters acts as the input to the experiment where we have considered BCET of every task to be exactly half of their WCET. With this, we have procured all the inputs to the optimization problem. The output for the respective task chains, are the periods of the tasks, which we ran for the chain length of 2 to 12 and various temperature constraints, to obtain their schedulability percentage. The average schedulability of the tasks with and without thermal constraints are plotted in Fig. 14 with detailed values mentioned in Table I.

We can observe that for a particular chain length, schedulability decreases with decrease in the temperature difference as the idle time required increases rapidly. From the Fig. 13, it is noticeable that task sets with chain length more than 9 are not schedulable. And to obtain a particular schedulability percent, if the chain length is really large, then a high temperature difference is required whereas, if the chain length is considerably small, then even a strict thermal bound i.e. low temperature difference can help us achieve the same schedulability percent. We can see the average schedulability percent drops very steeply as compared to the one without the thermal constraint because as the chain length increases, scheduler find it difficult to schedule the task chains under strict freshness bounds and thermal constraints because of insertion of huge idle times,

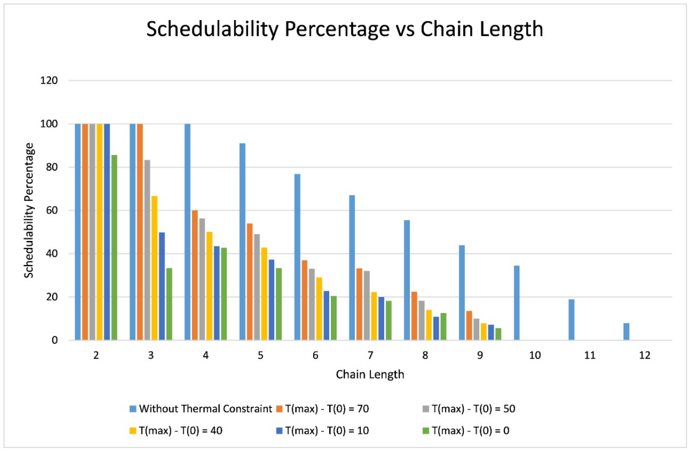


Figure 14: Schedulablilty with Different Chain Lengths

thereby frequently missing the deadlines.
From Table I, we can infer that for all the chain lengths under consideration, the average percent drop in the schedulability (average of all lengths), increases with the decrease in the temperature difference.

Temperature bound	70	50	40	10	0	NIL
Average schedulability	52.52	47.73	41.56	36.44	31.49	79.27
Percent drop in schedulability	33.75	39.79	47.58	54.04	60.18	-

Table I: Average schedulability at different temperature bounds

Percent drop in schedulability is a metric which defines the drop in the schedulability under temperature constraints w.r.t to without any temperature constraint for all the chain lengths.
Average in this case, has been considered for all chain lengths unlike in Fig 13, where we consider average for all the cases of a particular length.

It should be noted that, the schedulability values obtained without thermal constrained is very optimistic, unlike real world scenario, where we have to consider the thermal bound for the safety and durability of processors. Fig. 14 shows the advantages of having thermal constraints for a task set with a particular chain length. The schedulability percent for a particular chain length (length > 3) does not vary much for different temperature differences implying that even if the processor would have a unexpectedly high temperature rise, schedulability would not be affected by such an unanticipated event. An exception to this would be task set with chain length 3, where we can observe a huge difference for different temperature differences but in real-life scenarios, the length of the task sets are really huge.

VII. Conclusion

In this work, we proposed a thermal-aware data freshness formulation to analyze the dependency of temperature on end-to-end data freshness. We formulate the problem for a 2-task scenario and then extended the formulation of an n-task scenario. The proposed framework was used to derive the periods of tasks in the task chains while optimizing the system utilization. We evaluated the proposed framework by performing experiments using the MiBench automotive benchmark. In future, we plan to extend this work considering multiple modes of operation for the system.

References

[1] G. R. Goud, N. Sharma, K. Ramamritham, and S. Malewar, "Efficient real-time support for automotive applications: A case study," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2006, pp. 335–341.
[2] D. Golomb, D. Gangadharan, S. Chen, O. Sokolsky, and I. Lee, "Data freshness over-engineering: Formulation and results," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2018, pp. 174–183.
[3] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Fourth Annual IEEE International Workshop on Workload Characterization*, 2001, pp. 3–14.
[4] A. Labrinidis and N. Roussopoulos, "Exploring the tradeoff between performance and data freshness in database-driven web servers," *The VLDB Journal*, vol. 13, no. 3, pp. 240–255, 2004.
[5] Y. Wei, S. H. Son, and J. A. Stankovic, "Maintaining data freshness in distributed real-time databases," in *Proceedings of 16th Euromicro Conference on Real-Time Systems*. IEEE, 2004, pp. 251–260.
[6] K.-D. Kang, S. H. Son, and J. A. Stankovic, "Managing deadline miss ratio and sensor data freshness in real-time databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 10, pp. 1200–1216, 2004.
[7] M. Bouzeghoub and V. Peralta, "A framework for analysis of data freshness," in *Proceedings of the International Workshop on Information Quality in Information Systems*. ACM, 2004, pp. 59–67.
[8] V. Peralta, "Data freshness and data accuracy: a state of the art," *Reportes Técnicos 06-13*, 2006.
[9] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing data freshness, throughput, and delay in multi-server information-update systems," *arXiv preprint arXiv:1603.06185*, 2016.
[10] M. Xiong, S. Han, and K.-Y. Lam, "A deferrable scheduling algorithm for real-time transactions maintaining data freshness," in *Proceedings of 26th IEEE Real-Time Systems Symposium*. IEEE, 2005, pp. 11–pp.
[11] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proceedings of 17th IEEE Real-Time Systems Symposium*. IEEE, 1996, pp. 13–21.
[12] D. Seto, J. Lehoczky, and L. Sha, "Task period selection and schedulability in real-time systems," in *Proceedings of 19th IEEE Real-Time Systems Symposium*, Dec 1998, pp. 188–198.
[13] T. Chantem, X. Wang, M. Lemmon, and X. Hu, "Period and deadline selection for schedulability in real-time systems," in *Proceedings of Euromicro Conference on Real-Time Systems*, July 2008, pp. 168–177.
[14] Y. Wu, G. Buttazzo, E. Bini, and A. Cervin, "Parameter selection for real-time controllers in resource-constrained systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 610–620, 2010.
[15] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, pp. 579–592, 1995.
[16] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software*, 2008, pp. 191–201.
[17] X. Zhou, J. Yang, M. Chrobak, and Y. Zhang, "Performance-aware thermal management via task scheduling."
[18] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang, "Thermal-aware resource management for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2857–2868, 2018.