Boost-At-The-Tail: Work-Triggered Frequency Boosting for Fixed-Priority Scheduling

Behnam Khodabandeloo
Department of Information Technology, Uppsala University, Sweden

Chengzi Huang
Department of Information Technology, Uppsala University, Sweden

Pontus Ekberg
Department of Information Technology, Uppsala University, Sweden

Abstract

Fixed-priority (FP) scheduling is widely used in safety-critical real-time systems due to its simplicity and analyzability, yet it may fail to schedule task sets whose worst-case response-time bounds exceed deadlines by small margins. Meanwhile, modern processors increasingly support short-term frequency boosting, providing additional processing capacity subject to thermal and power constraints. This paper introduces *Boosted-FP*, a fixed-priority scheduling framework that augments a given FP policy with controlled, limited frequency boosting. The key idea is to activate boosting based on the worst-case remaining execution demand of the currently executing job, thereby confining high-frequency execution to the tail of jobs. Per-task boost parameters are computed offline using standard fixed-priority response-time analysis under the chosen priority assignment, while boost activation decisions are made online using worst-case remaining-work information.

We show that *Boosted-FP* preserves hard real-time guarantees by relating its execution behavior to that of a task set with reduced execution times. Leveraging the sustainability of fixed-priority schedulability analysis with respect to execution-time reductions, we establish that any task set schedulable under the modified execution bounds is also schedulable under the proposed framework. We evaluate the framework under a global boost-budget constraint and show experimentally that boost usage remains bounded and often well below the offline provisioned budget. Together, these results demonstrate that controlled, work-triggered boosting can safely extend the schedulable region of fixed-priority scheduling without abandoning static priorities.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases Fixed Priority Scheduling, Boost frequency, Execution slack

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2026.4

Supplementary Material *Software (ECRTS 2026 Artifact Evaluation approved artifact)*: `https://doi.org/10.4230/DARTS.12.2.10`

Acknowledgements The authors thank the anonymous reviewers for their careful reading and constructive feedback, which significantly improved the presentation of this paper.

1 Introduction

Fixed-priority (FP) scheduling remains a cornerstone of hard real-time systems, particularly in safety-critical domains where predictability, analyzability, and certification are paramount [7]. This dominance is reflected in practice: most commercial and certified real-time operating systems implement fixed-priority preemptive scheduling as their primary policy. Classic FP schedulers such as Rate Monotonic (RM) and Deadline Monotonic (DM) are attractive due to their simplicity and mature worst-case response-time analysis [19, 18, 1]. However, FP scheduling is inherently conservative: even when total utilization is below one, worst-case interference may cause response-time bounds to exceed deadlines [18].

In practice, many task sets deemed unschedulable under FP analysis are only marginally so. Deadline violations typically arise from small response-time overruns caused by accumulated higher-priority interference rather than sustained overload [18]. While such task sets may be feasible under dynamic-priority algorithms such as EDF [19], replacing FP scheduling is often undesirable due to certification costs, legacy constraints, and reduced predictability [7]. Accordingly, we seek mechanisms that recover marginal unschedulability while preserving the original fixed-priority policy and its established analysis.

Dual-priority scheduling [6, 11] partially relaxes FP by introducing controlled priority promotion. Although effective, it requires runtime priority transitions and additional scheduler states, complicating validation and certification. We instead explore whether additional processing capacity can be supplied without modifying priority semantics.

Modern processors provide short-term frequency boosting mechanisms that allow execution at elevated speeds for bounded durations subject to thermal and power constraints. Recent industrial experience demonstrates that controlled DVFS boosting can safely expand computing capacity at large scale. In particular, Piga *et al.* [21] report a large-scale production deployment at Meta in which controlled DVFS boosting created capacity equivalent to roughly half a datacenter while maintaining reliability and managing power-capping risks. While that work targets throughput-oriented datacenter services, it highlights the practical viability of bounded frequency boosting as a capacity-expansion mechanism. However, it does not address hard real-time schedulability or fixed-priority analysis.

This paper investigates whether bounded, work-triggered frequency boosting can compensate limited response-time overruns under FP scheduling while preserving hard real-time guarantees. We propose *Boosted-FP*, a framework that augments a given fixed-priority policy with short boost intervals activated near job completion based on worst-case remaining execution demand. Boost execution is confined to precisely those intervals where it most effectively reduces response-time bounds. Under bounded boost availability, this enables FP scheduling to accommodate task sets with utilization arbitrarily close to one while retaining classical analyzability.

The framework separates offline analysis from runtime control. Offline, per-task boost parameters are derived using standard FP response-time analysis. Online, the scheduler selects nominal or boost frequency without modifying task priorities. We establish correctness by relating the boosted system to an equivalent task set with reduced execution times and leveraging the sustainability of FP schedulability with respect to execution-time reductions [4].

The contributions of this paper are as follows:

- We introduce a work-triggered, boost-at-tail mechanism that integrates bounded frequency boosting into FP scheduling without altering priorities.
- We derive offline boost parameters via classical FP response-time analysis and define runtime activation rules based on remaining execution demand.
- We prove schedulability by reducing the boosted system to an equivalent reduced-execution-time task set using a sustainability argument.
- We propose a dynamic boost-cancellation optimization that reclaims early-completion slack while preserving guarantees.
- We experimentally demonstrate that bounded work-triggered boosting expands the FP schedulable region while keeping boost activity limited in practice.

2 Background and Preliminaries

2.1 Fixed-Priority Response-Time Analysis

We briefly recall the standard response-time analysis (RTA) for preemptive fixed-priority scheduling. Consider a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ scheduled on a uniprocessor under a fixed-priority policy. Each task τ_i is characterized by a worst-case execution time C_i, period T_i, and relative deadline D_i. We assume constrained deadlines, i.e., $D_i \leq T_i$ for all tasks. Let $hp(\tau_i)$ denote the set of tasks with higher priority than τ_i. Under classical RTA, the worst-case response time of task τ_i is computed as the smallest fixed point of the recurrence

$$R_i^{(k+1)} = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j, \tag{1}$$

with initialization $R_i^{(0)} = C_i$. The iteration continues until convergence or until $R_i^{(k+1)} > D_i$. The task τ_i is deemed schedulable if the iteration converges to a value $R_i \leq D_i$; otherwise, it is declared unschedulable. This fixed-point formulation of response-time analysis was introduced by Joseph and Pandya [16]. Rate Monotonic Scheduling (RMS), introduced by Liu and Layland [19], is a special case in which priorities are assigned in order of increasing task periods.

2.2 Sources of Unschedulability

Under fixed-priority scheduling, the response time of a task τ_i is determined by the fixed point of Eq. (1). A deadline violation occurs when the recurrence converges to a value $R_i > D_i$. A key property of Eq. (1) is that R_i appears on both sides of the recurrence through the interference term $\left\lceil \frac{R_i}{T_j} \right\rceil C_j$. Therefore, reducing the *effective execution time* of τ_i can have a *compound* effect on the computed response time: (1) It directly decreases the base term C_i. (2) It may also decrease the number of higher-priority jobs that interfere, because a smaller response-time estimate can reduce one or more ceiling terms $\left\lceil \frac{R_i}{T_j} \right\rceil$.

In particular, if a reduction in effective execution time causes the fixed point to cross a release boundary of some higher-priority task τ_j, i.e., the value of $\left\lceil \frac{R_i}{T_j} \right\rceil$ decreases by one, then the computed interference term is reduced by an entire C_j. Thus, a small reduction in effective execution time can remove one full higher-priority job from the worst-case interference bound, which may be sufficient to restore $R_i \leq D_i$. This non-linear effect underlies the possibility of restoring schedulability through bounded execution-time reductions. This observation motivates mechanisms that compensate deadline violations by reducing *effective execution time* through controlled acceleration, rather than by changing task priorities. This perspective is adopted in the subsequent execution-time reduction view and in the design of work-triggered boosting.

3 System Model

We consider a uniprocessor real-time system consisting of a finite set of independent periodic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by a tuple (C_i, D_i, T_i), where C_i, T_i, and D_i denote the worst-case execution time (WCET), period, and relative deadline, respectively. We focus on *implicit-deadline* task systems, for which $D_i = T_i$ for all tasks. All WCETs are specified with respect to execution at the processor's *nominal frequency*. Tasks release an infinite sequence of jobs strictly periodically. The j-th job of task τ_i, denoted by

$\tau_{i,j}$, is released at time $r_{i,j} = (j-1)T_i$ and has an absolute deadline $d_{i,j} = r_{i,j} + D_i$. Tasks are independent, fully preemptible, and do not share resources. We assume a discrete-time model in which execution times, periods, and deadlines are expressed in integer time units. This assumption is standard in response-time analysis and simplifies the presentation of ceiling operations and reduction increments. The system is scheduled under a preemptive *fixed-priority* policy with a total order over tasks; for a task τ_i, let $hp(\tau_i)$ denote the set of tasks with higher priority than τ_i. We assume that the total nominal utilization $U_{\text{all}} \triangleq \sum_{\tau_i \in \Gamma} \frac{C_i}{T_i}$ satisfies $U_{\text{all}} \leq 1$, which is a necessary condition for feasibility on a uniprocessor. Actual schedulability under the selected fixed-priority assignment is verified using standard response-time analysis.

3.1 Processor Model: Thermal-Aware Boosting

The processor supports two discrete operating frequencies: a nominal frequency f_{n} and a boost frequency $f_{\text{B}} = s_{\text{B}} \cdot f_{\text{n}}$, where $s_{\text{B}} > 1$ is a fixed boost speedup factor. When executing at the boost frequency, the processor completes s_{B} units of nominal work per unit time. We assume that execution demand scales proportionally with frequency; microarchitectural effects such as memory stalls are subsumed into WCET estimation. The processor can switch dynamically between f_{n} and f_{B} at runtime; any frequency-transition overhead is assumed negligible for analysis purposes and is conservatively included in the WCET estimates. Modern processors limit short-term boosting through hardware thermal management mechanisms [14]. Processor temperature is modeled as a time-varying state $T(t)$. Boost execution is governed by a hardware thermal hysteresis mechanism with two thresholds $T_{\text{a}} < T_{\text{b}}$. Boosting is permitted only while $T(t) \leq T_{\text{b}}$. If $T(t) > T_{\text{b}}$, boosting is disabled by hardware and is re-enabled only once the temperature falls below T_{a}. The system monitors temperature online and enforces this rule at runtime. In this paper, we focus on executions in which boosting remains available (i.e., thermal throttling is avoided), an assumption supported by the experimental observations reported in Sections 7.2 and 7.4. If boost is disabled by hardware thermal control, the system executes at the nominal frequency. Such executions fall outside the normal-mode setting considered in this paper; we discuss possible system-level responses informally in Appendix A.

We adopt a conservative bounded-rate abstraction of thermal dynamics consistent with standard first-order thermal models (e.g., [10, 9]). When executing at the boost frequency and $T(t) \leq T_{\text{b}}$, temperature increases at most at rate $r_{\text{h}} > 0$. When the processor is idle and $T(t) > T_{\text{a}}$, temperature decreases at least at rate $r_{\text{c}} > 0$. When executing at the nominal frequency, temperature is assumed not to increase in a manner that affects boost eligibility. This abstraction is used only to justify bounded boost availability and is not used directly in schedulability analysis.

The platform provides a boost availability contract over arbitrary intervals of length L: if $T(t_0) \leq T_{\text{b}}$ at the beginning of an interval $[t_0, t_0 + L)$, then all boost requests issued within the interval are guaranteed to be honored up to a budget of k time units. This abstraction is consistent with modern processors that regulate short-term boosting via hardware-managed power and thermal windows. For example, Intel Turbo Boost permits elevated performance operation (e.g., PL2) for a bounded duration controlled by a configurable time constant (Tau) [13]. The budget k is chosen conservatively such that even k consecutive time units of boost execution starting from temperature T_{b} cannot violate the hardware thermal limit $T_{\max}$. Under the bounded-rate abstraction, a sufficient condition is $T_{\text{b}} + kr_{\text{h}} \leq T_{\max}$. Because consecutive boost execution represents the thermal worst-case scenario, any execution containing at most k boosted time units within the interval, regardless of interleaving with

nominal or idle execution, is also thermally safe. Our schedulability analysis relies solely on the parameters k and L; the thermal abstraction serves only as a correctness argument for the existence of this contract.

Execution semantics

Let $C_{i,j}^{\mathrm{rem}}(t)$ denote the worst-case remaining execution demand of job $\tau_{i,j}$ at time t, measured in units of nominal work. At release, $C_{i,j}^{\mathrm{rem}}(r_{i,j}) = C_i$. While a job executes at the nominal frequency, $C_{i,j}^{\mathrm{rem}}(t)$ decreases at unit rate; while executing at the boost frequency, it decreases at rate s_{B}. If a job is preempted, its worst-case remaining execution demand is preserved.

4 Problem Definition

Given the execution platform defined in Section 3, we formalize the following problem. Consider a fixed-priority task set Γ scheduled preemptively on a uniprocessor at the nominal frequency, and a processor that additionally supports a boost frequency subject to a platform-provisioned availability bound (L, k). We focus on normal-mode executions in which the bounded boost availability assumed by the analysis is satisfied. The question is whether budget-limited boost execution can be exploited, using only the worst-case remaining execution demand of the currently executing job, to compensate bounded response-time overruns under fixed-priority scheduling while preserving hard real-time guarantees.

Formally, the objective is to design a scheduling mechanism that:

- preserves the fixed-priority dispatching semantics and priority assignment,
- selectively applies boost execution based on the worst-case remaining execution demand of the currently executing job,
- ensures that the platform-provisioned boost bound (L, k) is respected through offline admission,
- admits conservative schedulability analysis based on standard fixed-priority response-time techniques.

The goal is to guarantee that all jobs meet their deadlines under worst-case execution assumptions whenever the mechanism's sufficient schedulability conditions hold and the normal-mode boost-availability assumptions are satisfied. (For a brief outline of possible fallback policies in case the required boost frequency cannot be provided by the processor, see Appendix A.)

5 Boosted-FP

We propose *Boosted-FP*, a fixed-priority scheduling framework that augments static-priority scheduling with controlled frequency boosting. The key idea is to activate boost execution based on the worst-case remaining execution demand of the currently executing job, thereby applying boosting only near job completion and only when it is beneficial.

5.1 Offline Computation of Boost Parameters

We formulate offline parameter selection as the problem of finding a minimum execution-time reduction vector $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_n)$ such that the reduced task set with execution times $C_i' = C_i - \Delta_i$ is schedulable under fixed-priority scheduling, and such that each reduction can be realized by bounded boost execution. Each reduction Δ_i is mapped to a per-job boost allowance

$$w_i = \left\lceil \frac{\Delta_i}{s_{\mathrm{B}} - 1} \right\rceil, \tag{2}$$

which ensures that w_i units of boost execution compensate for an effective reduction of at least Δ_i units of nominal work. We define the worst-case cumulative boost demand over any interval of length L as

$$\mathrm{BoostUB}(L) \triangleq \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil w_i. \tag{3}$$

Intuitively, $\mathrm{BoostUB}(L)$ bounds the worst-case total number of boosted time slots requested if every job uses its entire allowance. This quantity upper-bounds the total number of boost time units that may be requested by all jobs within any interval of length L under the proposed runtime policy. This bound assumes that all jobs release as early as possible and that each job requests its full allowance w_i. In our evaluation, we set L to the hyperperiod H to obtain a conservative, cycle-aligned bound; the analysis applies to any platform-defined window length L. A task set is admitted in normal mode only if

$$\mathrm{BoostUB}(L) \leq k, \tag{4}$$

where k is the platform-provisioned boost budget per interval. We incorporate the admission constraint directly into the offline search: any intermediate reduction vector for which $\mathrm{BoostUB}(L) > k$ holds is treated as infeasible and is not committed.

▶ **Lemma 1** (Maximum realizable reduction). *The execution-time reduction Δ_i satisfies*

$$0 \leq \Delta_i \leq \left\lfloor \frac{s_{\mathrm{B}} - 1}{s_{\mathrm{B}}} C_i \right\rfloor.$$

Proof. Under continuous boosted execution, the worst-case execution time reduces from C_i to C_i/s_{B}. Hence, the maximum reduction is $C_i - C_i/s_{\mathrm{B}} = \frac{s_{\mathrm{B}}-1}{s_{\mathrm{B}}} C_i$. Since Δ_i is integer-valued, the bound follows. ◀

By Lemma 1, for each task τ_i the maximum admissible execution-time reduction is $\Delta_i^{\max} \triangleq \left\lfloor \frac{s_{\mathrm{B}}-1}{s_{\mathrm{B}}} C_i \right\rfloor$, and the search space is restricted to $\Delta_i \in [0, \Delta_i^{\max}]$.

5.1.1 Baseline Approach

A straightforward solution enumerates all reduction vectors satisfying $0 \leq \Delta_i \leq \Delta_i^{\max}$ and selects a reduction vector that minimizes $\mathrm{BoostUB}(L)$ (Eq. (3)) among all vectors that pass fixed-priority response-time analysis. The number of such vectors is $\prod_{i=1}^{n}(\Delta_i^{\max} + 1)$. Since $\Delta_i^{\max} = \Theta(C_i)$ in the worst case, this yields $O((C_{\max})^n)$ candidate vectors, where $C_{\max} \triangleq \max_{1 \leq i \leq n} C_i$ denotes the maximum WCET among all tasks. Each candidate requires schedulability testing of all tasks via classical fixed-priority response-time analysis, which incurs a pseudo-polynomial cost of $O(n^2 \cdot I)$, where I is the number of fixed-point iterations. Computing $\mathrm{BoostUB}(L)$ for a candidate vector requires $O(n)$ time and is dominated by the cost of schedulability testing. Hence, the total complexity is $O((C_{\max})^n \cdot n^2 \cdot I)$, which is exponential in the number of tasks and therefore infeasible in practice. The exponential factor arises from the combinatorial number of reduction vectors, independent of the cost of schedulability testing. We therefore propose an efficient constructive algorithm.

5.1.2 Efficient Approach

Tasks are processed in order of decreasing priority (i.e., from highest to lowest). Throughout the procedure, we maintain a reduced task set Γ' with execution times $C'_k = C_k - \Delta_k$. For each task τ_i, we first perform classical fixed-priority response-time analysis using the current

execution times $\{C'_k\}$. If the resulting worst-case response time R_i satisfies $R_i \leq D_i$, then τ_i is schedulable and we set $\Delta_i = 0$ and $w_i = 0$. Otherwise, if $R_i > D_i$, we determine the smallest reduction Δ_i such that τ_i becomes schedulable, subject to $\Delta_i \leq \Delta_i^{\max}$.

Monotonicity of fixed-priority response-time analysis with respect to execution times implies that reducing any C_k cannot increase any response-time bound. Hence, we perform a monotone integer search over Δ_i. Specifically, we apply binary search over the interval $[0, \Delta_i^{\max}]$ to find the smallest value of Δ_i such that τ_i becomes schedulable under fixed-priority analysis. If a feasible Δ_i is found, we compute w_i via (2). The update is committed only if the resulting reduction vector satisfies $\mathrm{BoostUB}(L) \leq k$; otherwise, the candidate is rejected and the search continues. If no feasible Δ_i satisfying both schedulability and the budget constraint exists within $[0, \Delta_i^{\max}]$, we invoke backtracking.

Backtracking incrementally shifts reductions to higher-priority tasks to reduce the interference experienced by an unschedulable task τ_i. Specifically, when τ_i is found unschedulable under the current reduction vector $\mathbf{\Delta}$, we repeatedly scan tasks in $hp(\tau_i)$ in increasing order of priority distance from τ_i. For each candidate task τ_j, if $\Delta_j < \Delta_j^{\max}$, we tentatively increase Δ_j by one unit and set $w_j = \lceil \Delta_j/(s_{\mathrm{B}} - 1) \rceil$. The increment is committed only if Lemma 1 holds and the resulting reduction vector satisfies $\mathrm{BoostUB}(L) \leq k$; otherwise, the increment is discarded. After each committed increment, response-time analysis for τ_i is repeated. The scan restarts after each committed increment and continues until either τ_i becomes schedulable or all higher-priority tasks reach their maximum admissible reductions, in which case infeasibility is declared. Each committed backtracking step strictly increases some Δ_j by one, and no Δ_j can exceed $\Delta_j^{\max}$. Therefore, after at most $\sum_i \Delta_i^{\max}$ committed increments, the algorithm must either find feasible parameters for all tasks or exhaust all feasible increments and declare infeasibility. Hence, the offline procedure always terminates.

The efficient approach is *sound*: whenever it terminates with a reduction vector $\mathbf{\Delta}$, the derived parameters $w_i = \lceil \Delta_i/(s_{\mathrm{B}} - 1) \rceil$ satisfy Lemma 1, the reduced task set passes fixed-priority response-time analysis, and the budget constraint $\mathrm{BoostUB}(L) \leq k$ holds. The latter follows since the algorithm initializes with $\mathrm{BoostUB}(L) = 0$ and commits a reduction update only if the resulting vector preserves $\mathrm{BoostUB}(L) \leq k$, thereby maintaining the budget constraint as an invariant throughout execution. The procedure is *complete with respect to fixed-priority schedulability* within the search space $\Delta_i \in [0, \Delta_i^{\max}]$: if a reduction vector within this space renders the task set schedulable under classical fixed-priority response-time analysis, the monotonicity of response-time bounds with respect to execution-time reductions guarantees that the backtracking procedure will eventually reach a schedulable vector unless all admissible reductions are exhausted. However, the algorithm is not complete with respect to the combined schedulability-and-budget constraint, since reductions are assigned monotonically and previously committed reductions are never decreased. Consequently, feasible reduction vectors that require trading reductions among tasks under a binding boost budget may not be explored. Finally, since reductions are assigned greedily, the returned solution is not guaranteed to minimize $\mathrm{BoostUB}(L)$, although it satisfies all correctness constraints.

Example

To illustrate the backtracking procedure, consider a task set $\tau_1(T_1{=}10, C_1{=}4)$, $\tau_2(T_2{=}12, C_2{=}4)$, $\tau_3(T_3{=}24, C_3{=}14)$, under RM priorities (highest priority first) with $s_{\mathrm{B}} = 2$, giving $\Delta_1^{\max} = 2$, $\Delta_2^{\max} = 2$, $\Delta_3^{\max} = 7$. We assume the boost budget k is non-binding so that the schedulability constraint is the only active constraint. Processing τ_1 and τ_2 in turn, classical RTA shows that both are schedulable under nominal execution times ($R_1 = 4$, $R_2 = 8$); we therefore set $\Delta_1 = \Delta_2 = 0$ and $w_1 = w_2 = 0$. We then process τ_3.

Under the current reduction vector $\boldsymbol{\Delta} = (0, 0, \Delta_3)$, RTA fails to converge below the deadline (i.e., the recurrence either reaches a fixed point with $R_3 > D_3$ or grows past D_3) for every $\Delta_3 \in [0, \Delta_3^{\max}]$ – even at $\Delta_3 = 7$, the interference from τ_1 and τ_2 alone is sufficient to render τ_3 unschedulable. Binary search therefore returns no feasible Δ_3, and the algorithm invokes backtracking. The scan visits $hp(\tau_3)$ in increasing order of priority distance from τ_3, i.e., τ_2 first. We tentatively increase Δ_2 from 0 to 1 (with $w_2 = \lceil 1/(s_{\mathrm{B}} - 1) \rceil = 1$), commit the increment, and re-run RTA on τ_3. τ_3 remains unschedulable for all $\Delta_3 \in [0, 7]$, so the scan restarts and increments Δ_2 again, from 1 to $2 = \Delta_2^{\max}$. With $\boldsymbol{\Delta} = (0, 2, \Delta_3)$, binary search now finds the smallest feasible value $\Delta_3 = 6$, which yields $R_3 = 20 \leq D_3 = 24$. The procedure terminates successfully with $\boldsymbol{\Delta} = (0, 2, 6)$ and $\boldsymbol{w} = (0, 2, 6)$. Had backtracking exhausted Δ_2 without success, the scan would have continued to τ_1; had it then exhausted Δ_1 as well, the algorithm would have declared infeasibility.

Complexity

For each task τ_i, determining the smallest feasible reduction Δ_i via binary search requires $O(\log \Delta_i^{\max})$ feasibility tests. Each feasibility test invokes fixed-priority response-time analysis of one task, which costs $O(n \cdot I)$ in the worst case, where I denotes the number of fixed-point iterations. During backtracking, each committed step increases some reduction parameter Δ_j by one and no Δ_j can exceed $\Delta_j^{\max}$. Hence, the total number of committed increments is bounded by $\sum_i \Delta_i^{\max}$. After each committed increment, response-time analysis for the currently failing task is repeated, and scanning the set $hp(\tau_i)$ incurs $O(n)$ additional overhead. Therefore, backtracking performs at most $O(\sum_i \Delta_i^{\max})$ response-time analyses. Consequently, the overall offline complexity is

$$O\left(n \cdot I \cdot \sum_{i=1}^{n} \left(\log \Delta_i^{\max} + \Delta_i^{\max} \cdot \max_j (\log \Delta_j^{\max}) \right) \right) \Longrightarrow O\left(n^2 \cdot I \cdot C_{\max} \cdot \log C_{\max} \right),$$

which is pseudo-polynomial in WCET and period parameters and polynomial in the number of tasks.

5.2 Thermally-Gated Runtime Boost Activation Policy

We define the runtime boost activation policy for the normal-mode setting considered in this paper, where boosting remains available throughout the analyzed interval. Scheduling priorities follow the underlying fixed-priority order, e.g., Rate Monotonic Scheduling. Whenever a job $\tau_{i,j}$ is selected for execution at time t, the processor frequency is determined based on the job's worst-case remaining execution demand $C_{i,j}^{\mathrm{rem}}(t)$.

The processor executes at the boost frequency if and only if the work-trigger condition holds:

$$C_{i,j}^{\mathrm{rem}}(t) \leq s_{\mathrm{B}} w_i \tag{5}$$

Otherwise, execution proceeds at the nominal frequency f_{n}. We do not maintain an online boost-budget counter. Instead, we rely on offline admission (Eq. (4)) and the platform-availability argument in Lemma 2 to ensure that the global boost budget within any window of length L is not exceeded under the normal-mode assumptions. Intuitively, the work-trigger condition confines boosting to the final portion of a job's execution. Moreover, since $C_{i,j}^{\mathrm{rem}}(t)$ is a conservative bound on worst-case remaining execution demand and most jobs complete before their worst-case bound, the trigger is typically not reached in common-case executions. Hence, boost slots are activated only occasionally, mitigating thermal stress in practice. Evaluating the boost trigger requires only a constant-time check of Eq. (5) and simple arithmetic updates of $C_{i,j}^{\mathrm{rem}}(t)$, and thus incurs negligible runtime overhead.

5.2.1 Effective Execution-Time Bound

▶ **Lemma 2** (Boost availability under platform contract). *Consider an analysis interval of length L and a platform-provisioned boost budget k. Assume the platform availability contract applies to the interval. If the task set satisfies* $\mathrm{BoostUB}(L) \le k$, *then all boost requests issued by the online policy within the interval are guaranteed to be honored.*

Proof. By definition, $\mathrm{BoostUB}(L)$ upper-bounds the total number of boost time units that may be requested by all jobs within any interval of length L in the worst case. Since the platform guarantees that up to k boost time units are available within such an interval and $\mathrm{BoostUB}(L) \le k$, the boost budget cannot be exhausted. Hence, no boost request is denied. ◀

▶ **Lemma 3** (Effective execution bound (normal mode)). *In normal mode, under the runtime policy defined in Eq.* (5)*, any job of task τ_i executes for at most $C_i^{\mathrm{eff}} = C_i - (s_{\mathrm{B}} - 1)w_i$ units of processor time, excluding preemption gaps.*

Proof. While executing at the nominal frequency, a job's worst-case remaining execution demand decreases at unit rate. If the work-trigger condition $C_{i,j}^{\mathrm{rem}}(t) \le s_{\mathrm{B}} w_i$ becomes true while $\mathrm{BoostEnabled}(t)$ holds, the job may execute at the boost frequency, in which case its worst-case remaining execution demand decreases at rate s_{B}. Note that since boosting is permitted only when both $C_{i,j}^{\mathrm{rem}}(t) \le s_{\mathrm{B}} w_i$ and $\mathrm{BoostEnabled}(t)$ hold, and since $C_{i,j}^{\mathrm{rem}}(t)$ decreases at rate s_{B} during boosted execution, the cumulative boosted execution time of job $\tau_{i,j}$ is implicitly bounded by w_i, even in the absence of an explicit per-job boost counter. Formally, once boosting starts at time t_0 with $C_{i,j}^{\mathrm{rem}}(t_0) \le s_{\mathrm{B}} w_i$, if the job executes at boost frequency for ℓ time units, then $C_{i,j}^{\mathrm{rem}}(t_0 + \ell) = C_{i,j}^{\mathrm{rem}}(t_0) - s_{\mathrm{B}}\ell \ge 0$, which implies $\ell \le C_{i,j}^{\mathrm{rem}}(t_0)/s_{\mathrm{B}} \le w_i$. If the job enters the boost region when $C_{i,j}^{\mathrm{rem}}(t) = s_{\mathrm{B}} w_i$ and then executes at boost speed for w_i units of processor time, it completes $s_{\mathrm{B}} w_i$ units of nominal work. The remaining execution demand $C_i - s_{\mathrm{B}} w_i$ is then executed at nominal speed. Summing both phases yields a total execution time of $C_i - (s_{\mathrm{B}} - 1)w_i$. The bound holds in normal mode provided that the platform availability contract holds and $\mathrm{BoostUB}(L) \le k$ (Lemma 2), which guarantee that boost requests issued under the trigger condition are honored. ◀

Together, the offline parameter selection and the thermally gated runtime policy ensure that, in normal mode, *Boosted-FP* behaves equivalently to a fixed-priority scheduler operating on a reduced task set.

5.3 Example

We illustrate *Boosted-FP* using the periodic task set $\tau_1(T_1{=}6, C_1{=}3)$, $\tau_2(T_2{=}8, C_2{=}2)$, and $\tau_3(T_3{=}12, C_3{=}3)$, scheduled under Rate Monotonic (RM) priorities. The processor supports a boost speed $s_{\mathrm{B}} = 1.5$. The hyperperiod of the task set is $H = \mathrm{lcm}(6, 8, 12) = 24$. We assume that boost eligibility is maintained throughout execution and that the platform availability contract holds over the hyperperiod. Standard response-time analysis under nominal execution speed shows that τ_3 violates its deadline, while τ_1 and τ_2 are schedulable. Reducing C_3 by one unit restores schedulability, yielding $\Delta_3 = 1$ and $\Delta_1 = \Delta_2 = 0$. The corresponding boost parameter is $w_3 = \left\lceil \frac{1}{s_{\mathrm{B}} - 1} \right\rceil = 2$, and the feasibility condition $s_{\mathrm{B}} w_3 \le C_3$ is satisfied. Thus, $w_1 = w_2 = 0$ and $w_3 = 2$. Over one hyperperiod, $\mathrm{BoostUB}(H) = \left\lceil \frac{24}{12} \right\rceil w_3 = 2 \cdot 2 = 4$. Hence, the task set is admissible in normal mode for any platform budget $k \ge 4$. Since $s_{\mathrm{B}} w_3 = 1.5 \times 2 = 3 \ge C_3$, the work-trigger condition $C_{3,j}^{\mathrm{rem}}(t) \le s_{\mathrm{B}} w_3$ is satisfied immediately

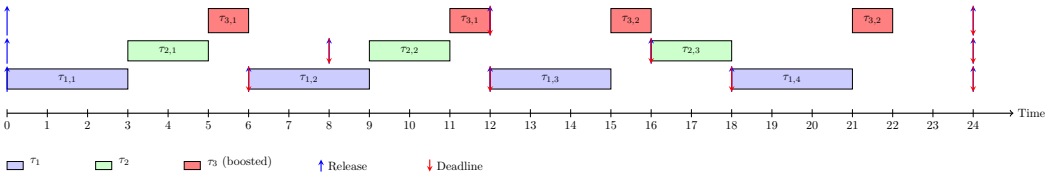


Figure 1 Critical-instant schedule over one hyperperiod ($H = 24$) for $\tau_1(6, 3)$, $\tau_2(8, 2)$, $\tau_3(12, 3)$ with $s_{\mathrm{B}} = 1.5$ under *Boosted-FP*. Upward blue arrows indicate job releases, and downward red arrows denote absolute deadlines.

upon release. Consequently, each job of τ_3 executes entirely at boost speed whenever it is scheduled. Each job therefore completes in $C_3/s_{\mathrm{B}} = 3/1.5 = 2$ units of processor time. Tasks τ_1 and τ_2 execute exclusively at nominal speed. Figure 1 shows the resulting schedule over one hyperperiod, where darker segments indicate execution at boost speed. This example illustrates how *Boosted-FP* restores schedulability while confining boost execution to the task that requires it.

5.4 Schedulability Analysis

Recall from Section 5.1 that the offline procedure computes, for each task τ_i, a reduction parameter Δ_i and a corresponding boost parameter w_i. Based on these values, we define a hypothetical task set Γ' that is identical to the original task set Γ, except that each task τ_i has execution time $C'_i = C_i - \Delta_i$. By Lemma 3, any job of task τ_i executes under *Boosted-FP* for at most $C_i^{\mathrm{eff}} = C_i - (s_{\mathrm{B}} - 1)w_i$ units of processor time, excluding preemption gaps. By construction of w_i (Eq. (2)), we have $(s_{\mathrm{B}} - 1)w_i \geq \Delta_i$, and therefore $C_i^{\mathrm{eff}} \leq C_i - \Delta_i = C'_i$. Hence, each job of τ_i under *Boosted-FP* executes for no more processor time than assumed for τ_i in the reduced task set Γ'.

▶ **Theorem 4** (Normal-Mode Schedulability of *Boosted-FP*). *If the reduced task set Γ' with execution times $C'_i = C_i - \Delta_i$ is schedulable under the* same fixed-priority order *as in the original system according to response-time analysis, and the platform availability contract holds and $\mathrm{BoostUB}(L) \leq k$, then the original task set Γ is schedulable under Boosted-FP.*

Proof. Since Γ' is schedulable under fixed-priority scheduling, for each task τ_i the worst-case response time computed using classical response-time analysis satisfies $R'_i \leq D_i$. Under *Boosted-FP*, by Lemma 3 and Lemma 2, every job of τ_i executes for at most $C_i^{\mathrm{eff}} \leq C'_i$ time units, and likewise every higher-priority task executes each job for at most its corresponding reduced execution time in Γ'. Moreover, *Boosted-FP* preserves the same fixed-priority order and release pattern as assumed in the analysis of Γ', and only reduces execution demand relative to Γ'. Fixed-priority schedulability analysis is sustainable with respect to execution-time reductions, meaning that decreasing task execution times cannot cause a previously schedulable task set to become unschedulable. Therefore, since the execution demand of each task under *Boosted-FP* is no greater than that assumed in Γ', the response time of each task under *Boosted-FP* is upper-bounded by its response time in Γ', which does not exceed its deadline. ◀

The schedulability guarantee established above is conservative, as it relies on worst-case execution times and ceiling effects in the computation of Δ_i and w_i. In practice, boost execution may be required less frequently than assumed by the analysis. Nonetheless, by interpreting work-triggered boosting as an effective execution-time reduction and leveraging

sustainability of fixed-priority schedulability analysis, *Boosted-FP* preserves hard real-time guarantees while remaining fully compatible with standard fixed-priority response-time analysis techniques. The schedulability analysis in this paper targets normal-mode operation. Integrating explicit thermal dynamics into the schedulability envelope is an important direction for future work.

6 Boost Cancellation (Optimization)

The schedulability analysis in Section 5.4 assumes a conservative worst-case scenario in which every job fully utilizes its assigned boost allowance. In practice, jobs may complete before exhausting their worst-case execution budget, yielding unused computation capacity. This section introduces *dynamic boost cancellation*, an optional runtime optimization that exploits such capacity to suppress unnecessary boost execution. Boost cancellation strictly reduces boost usage relative to the worst case assumed in the analysis. Consequently, all schedulability guarantees established in Section 5.4 remain valid.

Many slack-reclamation and slack-stealing techniques (e.g. [15]) in the real-time literature could, in principle, be applied to reuse unused execution capacity in our setting. In this section, rather than designing an optimal reclamation policy, we outline a simple slack-reuse procedure used in our experiments that is compatible with our execution semantics and opportunistically cancels unnecessary boosts without affecting the schedulability guarantees.

6.1 Execution Slack

For each job $\tau_{i,j}$, the scheduler maintains $C^{\text{rem}}_{i,j}(t)$, which denotes an *upper bound* on the job's remaining execution demand measured in units of nominal work. At release time $r_{i,j}$, $C^{\text{rem}}_{i,j}(r_{i,j}) = C_i$, and $C^{\text{rem}}_{i,j}(t)$ is monotonically non-increasing over time. A nominal slot, when $\tau_{i,j}$ executes, reduces $C^{\text{rem}}_{i,j}$ by one unit, whereas a boosted slot reduces it by s_{B} units.

▶ **Definition 5** (Execution Slack). *Let $\tau_{i,j}$ complete at time t_f. The* execution slack *generated by $\tau_{i,j}$ is*

$$\delta_{i,j} = \begin{cases} C^{\text{rem}}_{i,j}(t_f) - (s_{\text{B}} - 1)w_i, & \text{if } C^{\text{rem}}_{i,j}(t_f) > s_{\text{B}} w_i, \\ \dfrac{C^{\text{rem}}_{i,j}(t_f)}{s_{\text{B}}}, & \text{otherwise.} \end{cases}$$

By definition, $\delta_{i,j} \geq 0$. The first case corresponds to jobs that complete within their nominal execution region, in which case both unused nominal and boost reservations contribute to the execution slack. The second case corresponds to jobs that complete during boosted execution, where execution slack arises only from the unused portion of the reserved boosted time. Execution slack therefore represents the portion of processor time provisioned by the schedulability analysis that remains unused upon job completion.

6.2 Dynamic Boost Cancellation

Dynamic boost cancellation is applied only when a job becomes eligible for boost execution according to the boost-trigger condition defined in Section 5.2. Recall that a job $\tau_{i,j}$ is boosted only when its remaining worst-case execution bound satisfies

$$C^{\text{rem}}_{i,j}(t) \leq s_{\text{B}} \cdot w_i, \tag{6}$$

where w_i is the boost allowance of task τ_i and $s_{\text{B}} > 1$ is the boost factor.

Dynamic boost cancellation operates on a per-job basis. Each active job $\tau_{i,j}$ maintains a slack credit variable $S_{i,j}$, which is initialized to zero at job release. When $\tau_{i,j}$ completes, it is removed from the active set and its slack credit is discarded (equivalently, set to zero). Slack credits are maintained only for active jobs. Thus, a job $\tau_{i,j}$ can only receive slack generated while it is active, i.e., slack produced after its release and before its completion. Slack generated before release is not visible to $\tau_{i,j}$, and upon completion its credit is discarded. Although the platform is uniprocessor and executes at most one job at a time, multiple jobs may be *active* (released but not yet completed) and reside in the ready queue. The slack-credit updates described below are performed by the scheduler atomically at discrete events (job completion and scheduling decisions), and therefore do not require concurrent access or synchronization. Accordingly, the term "active jobs" refers to jobs that are pending at the same time, not to jobs executing in parallel.

When a job $\tau_{k,\ell}$ completes and produces execution slack $\delta_{k,\ell}$, this slack is made available to all active jobs of lower priority:

$$S_{i,j} \leftarrow S_{i,j} + \delta_{k,\ell} \qquad \forall\, \tau_{i,j} \text{ such that } \tau_i \prec \tau_k.$$

Whenever job $\tau_{i,j}$ satisfies the trigger condition (6), the scheduler first checks whether sufficient slack credit exists: $S_{i,j} \geq (s_{\mathrm{B}} - 1)$. If the condition holds, boost execution is canceled and $\tau_{i,j}$ executes at nominal speed for the current slot. Otherwise, boost execution proceeds normally. Because execution slack is replicated across multiple active lower-priority jobs, slack consumption must be performed consistently. Therefore, upon canceling a boost for job $\tau_{i,j}$, the scheduler subtracts $(s_{\mathrm{B}} - 1)$ units of slack from the slack credit of all active jobs of priority lower than or equal to τ_i:

$$S_{k,\ell} \leftarrow \max\{0,\, S_{k,\ell} - (s_{\mathrm{B}} - 1)\} \qquad \forall\, \tau_{k,\ell} \text{ such that } \tau_k \preceq \tau_i.$$

Dynamic boost cancellation adapts boost usage to actual execution behavior without weakening the schedulability guarantees established in Section 5.4. Since dynamic boost cancellation never increases execution demand or interference beyond the bounds assumed in the schedulability analysis, all hard real-time guarantees remain valid. We next formalize the slack-credit bookkeeping and show that replicated credits cannot cause over-commitment.

▶ **Lemma 6.** *Dynamic boost cancellation does not increase any job's execution demand or interference beyond the bounds assumed in the schedulability analysis.*

Proof. Each canceled boost slot replaces one unit of boosted execution by one unit of nominal execution together with $(s_{\mathrm{B}} - 1)$ units of previously generated execution slack. This slack corresponds to computation capacity already reserved by the analysis but not consumed by earlier jobs. At any time, the slack credits stored in the system correspond to execution capacity already reserved by the reduced task set Γ' but not consumed in practice. Slack is generated only when jobs complete early. Each canceled boost consumes exactly $(s_{\mathrm{B}} - 1)$ units of such slack and replaces one boosted slot with one nominal slot. Therefore, boost cancellation merely redistributes computation capacity that was already assumed by the worst-case model in Section 5.4. Since fixed-priority schedulability is sustainable with respect to execution-time reductions, all response-time bounds remain valid. ◀

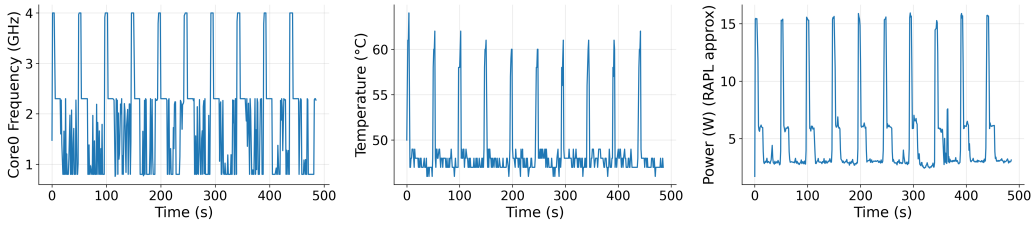
7 Experimental Evaluation

We evaluate *Boosted-FP* under a Rate Monotonic (RMS) priority assignment along three dimensions: (i) schedulability-region expansion under bounded boost budgets, (ii) the effectiveness of dynamic boost cancellation and its associated thermal and energy impact, and (iii) energy consumption relative to a global frequency-scaling baseline.

7.1 Experimental Energy Model

To quantify the relative power and thermal impact of boosted execution, we employ a CMOS-inspired energy model [8] used solely for experimental comparison. When executing at operating point (V_{dd}, f), dynamic power is modeled as $P_{\text{dyn}} = C_{\text{eff}} V_{dd}^2 f$, and leakage power as $P_{\text{leak}} = I_{\text{leak}} \cdot V_{dd}$, where C_{eff} is the effective switched capacitance per cycle (a technology- and workload-dependent constant aggregating gate switching activity) and I_{leak} is the leakage current. Both are treated as constants in our model since we report only relative energy quantities normalized to E_{n}, the dynamic energy consumed per slot at f_{n} (defined below). We assume that supply voltage scales approximately linearly with frequency ($V_{\text{dd}} \propto f$). Under this assumption, dynamic power scales cubically with frequency, whereas leakage power scales linearly. The processor supports two discrete frequencies: nominal f_{n} and boost $f_{\text{B}} = s_{\text{B}} f_{\text{n}}$. Let E_{n} denote the dynamic energy consumed per slot at f_{n}. Since dynamic power scales cubically with frequency and execution duration scales inversely with frequency, the dynamic energy consumed per slot scales quadratically with frequency. Hence, execution at f_{B} consumes $s_{\text{B}}^2 E_{\text{n}}$ dynamic energy per slot. When the processor is idle, no useful switching occurs; however, background activity incurs a fraction of nominal dynamic energy, modeled as αE_{n}, where $\alpha \in [0, 1]$. Since our comparisons normalize by the same schedule horizon H, and since the focus is relative dynamic switching stress, we report dynamic-energy–based metrics; incorporating leakage would not change qualitative trends. This model is not used in the schedulability analysis and serves only as a proxy for relative power and thermal stress induced by boosting.

7.2 Platform-Level Validation of Boosting Model



(a) Frequency. **(b)** Temperature. **(c)** Power.

Figure 2 Measured single-core behavior under alternating BOOST/NOMINAL/IDLE, showing bounded short-term boosting consistent with the proposed model.

We validate that the assumptions underlying our execution model, distinct nominal and boost operating points, and bounded short-term boosting, are observable on commodity hardware. The goal of this experiment is to confirm that the abstraction used in our analysis is consistent with practical processor behavior.

Measurements were conducted on a laptop running Ubuntu 22.04 equipped with an 11th Gen Intel® Core™ i7-11800H processor (8 cores, nominal frequency 2.30 GHz) with Intel Turbo Boost enabled. We instrumented the processor using the Linux `intel_pstate` interface and Intel RAPL energy counters. A custom script periodically alternates the processor between three phases: **BOOST** (Turbo enabled, maximum performance), **NOMINAL** (Turbo disabled, maximum performance), and **IDLE** (Turbo disabled, minimum performance). During BOOST and NOMINAL phases, a single core is stressed using `stress-ng`; during IDLE, the system remains largely inactive. We sample core frequency (kHz), package

temperature (°C), and RAPL energy (µJ) once per second and derive instantaneous power (W) from successive energy readings. The three-phase pattern is repeated multiple times to observe steady-state behavior and transitions. This setup emulates the average workload model used in our analysis: tasks execute predominantly at nominal speed, occasionally experience short-term boost execution, and are separated by idle or low-activity intervals.

The measurements reveal three clearly separated regimes corresponding to BOOST, NOMINAL, and IDLE phases. In BOOST mode, frequency approaches the turbo range, with increased power and temperature. In NOMINAL mode, frequency stabilizes near the nominal base operating range with reduced power and thermal growth. In IDLE mode, frequency and power drop to minimal levels and temperature decays. Figure 2 illustrates representative time series of frequency, power, and temperature across BOOST, NOMINAL, and IDLE phases. These observations support the model introduced in Section 3: (i) the processor provides two distinct operating points; (ii) boosted execution incurs higher but bounded power and thermal cost; and (iii) idle intervals enable recovery. Hence, the work-triggered boosting abstraction and the (L, k) boost availability contract are compatible with observed hardware behavior.

Mapping measurements to model parameters

The platform-level traces also offer concrete reference points for the model parameters s_{B}, L, and k used in our analysis. First, the measured BOOST and NOMINAL frequency plateaus correspond to the two-frequency abstraction of Section 3.1: with the i7-11800H, the steady BOOST frequency reaches roughly twice the steady NOMINAL frequency, which is consistent with realistic boost speedup factors in the range $s_{\mathrm{B}} \in [1.3, 2.0]$ used in our experiments (we use $s_{\mathrm{B}} = 1.5$ as a moderate, conservative choice). Second, the measured BOOST plateaus are themselves bounded in length: each BOOST burst sustains elevated frequency for a bounded duration before either returning to NOMINAL or being throttled, which directly mirrors the (L, k) contract. Specifically, the maximum continuous BOOST duration we observed serves as an empirical lower bound on the achievable k within a window L equal to the burst-plus-recovery cycle. Third, the IDLE phases show clear temperature decay, validating the bounded-rate cooling assumption used to re-establish boost eligibility. We do not commit to a specific (L, k) pair on this platform, since these parameters are platform-configurable (e.g., via Intel's PL2/Tau settings [13]); instead, we use this experiment to confirm that all three model components – bounded-rate heating during boost, bounded-rate cooling during idle, and a finite per-window boost capacity – are observable on commodity hardware. This micro-benchmark demonstrates that commodity processors already exhibit the execution modes required by *Boosted-FP*: bounded short-term boosting and natural thermal recovery during idle intervals. We emphasize that our schedulability results depend only on the abstract (L, k) contract and not on this specific platform.

Discussion

A practical concern is that some short-term boosting mechanisms, such as Intel Turbo Boost, are ultimately governed by hardware thermal and power controllers. Software can request a high-performance operating point, but the hardware decides whether the request can be served based on the current thermal and power state of the processor. Thus, per-job boost activation cannot be guaranteed by software alone.

Our use of boosting is therefore conditional on operating within a safe thermal region. The proposed policy requests boost only for a bounded amount of execution, as captured by the per-window bound (L, k), and the experimental observations in Section 7.2 indicate

that, under such controlled usage, short boost intervals can be sustained while temperature remains within the hardware-managed range. These observations support the practicality of the two-frequency abstraction used in the paper, but they do not by themselves constitute a formal platform guarantee.

Formally, the schedulability analysis assumes that the target platform provides a boost-availability guarantee: every boost request issued by an admitted task set is served, and the total boosted execution requested within any window of length L does not exceed the platform-provisioned budget k. Such a guarantee would have to be provided by the processor/platform specification or established through a conservative platform-specific characterization.

7.3 Acceptance Ratio Improvement

7.3.1 Setup

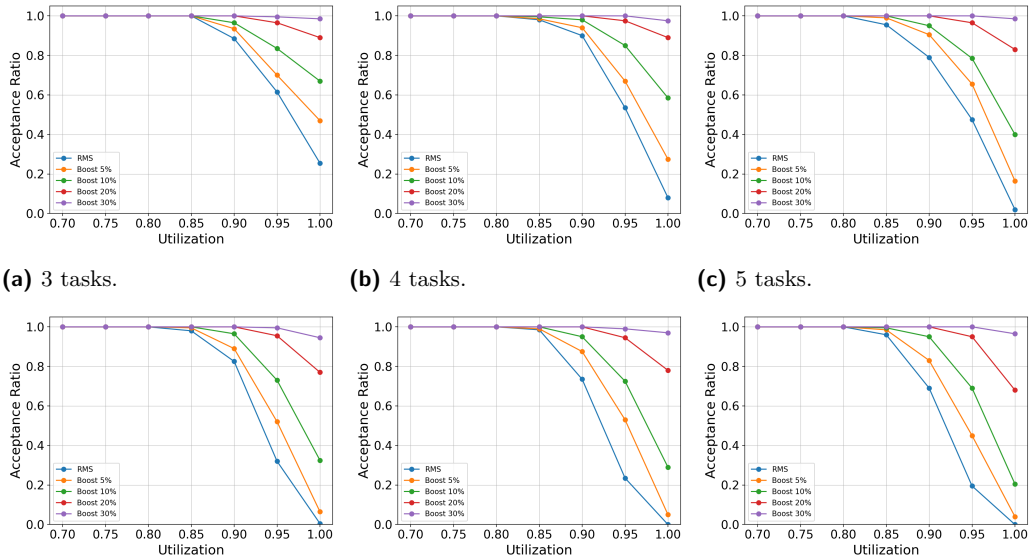
We generate implicit-deadline periodic task sets using the UUniFast algorithm. For each task set size $n \in \{3, 4, 5, 6, 7, 8\}$, periods are drawn from a log-uniform distribution [12] in $[10, 200]$. Target utilization values range from 70% to 100% in steps of 5%, and for each pair (n, U) we generate 200 task sets. The WCET of each task is computed from its target utilization u_i and period T_i as $C_i = \min(T_i, \max(1, \lfloor (U_i T_i) \rfloor))$. We assume a nominal frequency f_{n} and a boost frequency $f_{\mathrm{B}} = 1.5 f_{\mathrm{n}}$, corresponding to $s_{\mathrm{B}} = 1.5$. The offline boost upper bound over one hyperperiod H is computed according to Eq. (3) We set the platform window to $L = H$ and express the platform budget as $k = \gamma H$, where γ is the allowed boosted-time fraction. A task set is *accepted* by RMS if it is schedulable under standard response-time analysis. A task set is *accepted* by *Boosted-FP* if: (i) the offline reduction algorithm produces feasible parameters, (ii) the reduced task set is schedulable under RMS, and (iii) the boost bound satisfies BoostUB $\leq \gamma H$, where γ denotes the allowed boost percentage. We evaluate $\gamma \in \{5\%, 10\%, 20\%, 30\%\}$. For each utilization bin U, the acceptance ratio is $\mathrm{AcceptanceRatio}(U) = n_{\mathrm{acc}}(U)/200$, where $n_{\mathrm{acc}}(U)$ denotes the number of task sets accepted at utilization U.

7.3.2 Results

Figure 3 reports the acceptance ratio as a function of system utilization for the baseline RMS and *Boosted-FP* under different *boost budget bounds*, and across varying numbers of tasks. As expected, the acceptance ratio of RMS drops sharply as utilization approaches one, reflecting the increasing prevalence of response-time violations in dense task sets. This degradation becomes more severe as the number of tasks increases, indicating that RMS is particularly sensitive to both high utilization and growing system complexity.

In contrast, *Boosted-FP* yields consistently higher acceptance ratios than RMS across all utilizations and task-set sizes. At $U = 1.0$, the average acceptance-ratio improvement over RMS is 0.122 (0.040–0.215), 0.359 (0.215–0.505), 0.747 (0.635–0.815), and 0.911 (0.730–0.975) for boost-budget bounds $\gamma \in \{5\%, 10\%, 20\%, 30\%\}$, respectively. With $\gamma = 30\%$, acceptance approaches unity, reaching at least 0.945 for all evaluated task-set sizes even at full utilization. These results demonstrate that bounded, work-triggered boosting substantially enlarges the schedulable region under fixed-priority scheduling, especially in highly loaded systems.

The horizontal axis in Figure 3 reports *nominal* utilization $U_{\mathrm{all}} = \sum_i C_i/T_i$, in which all WCETs are expressed at the nominal frequency f_{n}. Because Boosted-FP runs at $f_{\mathrm{B}} = s_{\mathrm{B}} f_{\mathrm{n}}$ for at most a γ-fraction of any window of length L, the *average effective compute capacity* made available within that window is $(1-\gamma) + s_{\mathrm{B}}\,\gamma \;=\; 1 + (s_{\mathrm{B}} - 1)\,\gamma$ times nominal capacity. For our setting $s_{\mathrm{B}} = 1.5$, this corresponds to up to 1.025×, 1.05×, 1.10×, and 1.15× nominal



(a) 3 tasks. **(b)** 4 tasks. **(c)** 5 tasks.

(d) 6 tasks. **(e)** 7 tasks. **(f)** 8 tasks.

Figure 3 Acceptance ratio versus utilization for RMS and *Boosted-FP* under boost budgets.

capacity for $\gamma \in \{5\%, 10\%, 20\%, 30\%\}$, respectively. A point at nominal utilization $U = 1.0$ on the $\gamma = 30\%$ curve thus represents a workload whose demand fits entirely within a system that, on average, offers up to 115% of nominal capacity within any window L. This connects the nominal U-axis to the effective compute budget the platform contracts to deliver and explains why the schedulable region under Boosted-FP can extend close to (and at $\gamma = 30\%$ even fully cover) the $U = 1.0$ line that strict FP cannot cross.

To better understand the source of the schedulability gains, we examine the structure of the offline reductions Δ_i for task sets accepted by *Boosted-FP* at $U = 1.0$ under a boost budget of $\gamma = 0.30$. Across accepted task sets with $n \in \{3, \ldots, 8\}$ tasks, the average number of tasks with non-zero reduction ranges from 1.21 to 2.71, indicating that only a small subset of tasks requires adjustment. Moreover, reductions are strongly concentrated in the lowest-priority levels: on average, between 75.3% and 94.0% of the total reduction is assigned within the lowest-priority quartile, and between 97.9% and 100% within the lowest half of the priority spectrum. To normalize reductions across task sets with different periods, we compare the aggregate reduction utilization $\sum_i \Delta_i/T_i$ against the total nominal utilization $U_{\text{all}} = \sum_i C_i/T_i$. Under $\gamma = 0.30$, the resulting reduction rate ranges from 5.23% to 7.82% of U_{all}. These results indicate that *Boosted-FP* primarily repairs marginal response-time violations rather than globally reshaping the workload.

7.4 Dynamic Boost Cancellation

Although the boost budget bound guarantees schedulability in the worst case, actual job execution times are typically smaller than their WCET. Consequently, reserved boost capacity may remain unused in common-case executions. Dynamic boost cancellation reclaims this slack at runtime by replacing boost execution with accumulated early-completion slack when available. This preserves the worst-case boost bound and schedulability guarantees while reducing common-case energy and thermal overhead.

7.4.1 Setup

We evaluate task sets that are not schedulable under RMS but are accepted by *Boosted-FP* under a system-wide boost bound of 30% of the hyperperiod (BoostUB $\leq 0.3H$). For each task set, we simulate 20 consecutive hyperperiods (capped at $H \leq 10^6$ time slots). Each job $\tau_{i,j}$ has WCET C_i at nominal speed, but its actual demand is $C_{i,j}^{\text{act}} = \left\lceil \frac{C_i}{\phi_{i,j}} \right\rceil$, where $\phi_{i,j} \geq 1$ is drawn from a scaled Weibull distribution (clamped to avoid extreme outliers), ensuring $1 \leq C_{i,j}^{\text{act}} \leq C_i$. Early completions generate slack that is broadcast to pending lower-priority jobs. A job becomes boost-eligible only after executing its offline nominal budget C_i'. Within this tail region, boost slots are canceled whenever sufficient slack is available; otherwise, execution proceeds at boost speed. This preserves the worst-case boost bound while reducing boost usage in typical executions.

7.4.2 Metrics

We report three metrics:

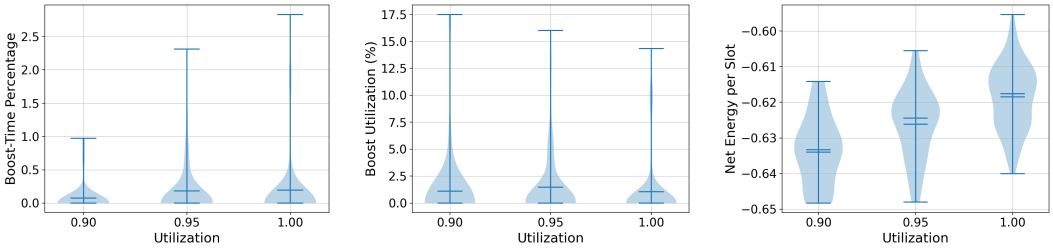
- **Boosted-Time Percentage:** $100N_{\text{B}}/H$, where N_{B} is the number of boosted slots over one hyperperiod of length H. This is a direct indicator of the power/thermal stress induced by boosting (lower is better)
- **Boost Utilization:** N_{B}/BoostUB, indicating how much of the offline provisioned boost budget is exercised at runtime and thus quantifies the effectiveness of dynamic boost cancellation (lower is better).
- **Net Normalized Energy per Slot:** $\big((s_{\text{B}}^2 - 1)N_{\text{B}} - (1-\alpha)N_{\text{I}}\big)/H$, where N_{I} is the total number of idle time slots over one hyperperiod of length H. This metric reports the average deviation from nominal per-slot energy and is used as a proxy for the net thermal/power stress induced by boosting. Negative values indicate that idle-time savings outweigh boost overhead and positive values indicate that boost overhead dominates idle-time savings. We set $\alpha = 0.2$ for idle energy computation.

7.4.3 Results

Figure 4 summarizes dynamic boost cancellation for representative 3-, 5-, and 8-task systems (other sizes exhibit similar trends). Each row fixes task-set size; columns report Boosted-Time Percentage, Boost Utilization, and Net Normalized Energy per Slot for $U \in \{0.9, 0.95, 1.0\}$. Boost usage increases with utilization as available slack decreases. However, boost activity remains limited in typical executions. For 3-task systems, the median Boosted-Time Percentage is zero at all utilizations (means: 0.07, 0.18, 0.19). For 5 tasks, the mean remains below 0.48 (median < 0.03). For 8 tasks, the mean increases from 0.34 to 1.19, with median 1.02 at $U = 1.0$. Thus, early completions cancel most boost slots even at high load.

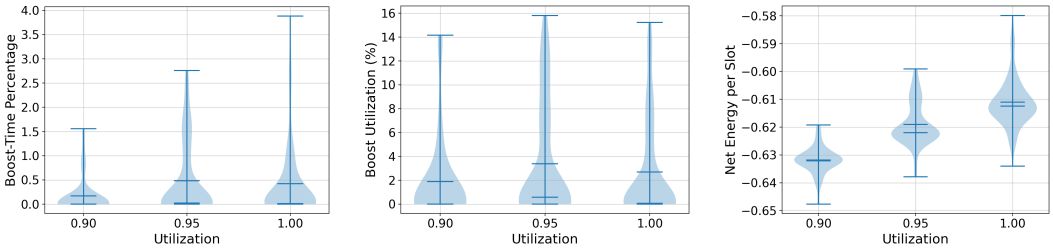
Increasing task count raises variance and upper tails due to tighter interference, but central tendencies remain small across all sizes. At $U = 1.0$, maximum Boosted-Time Percentage is 2.83, 3.88, and 4.04 for 3-, 5-, and 8-task systems, respectively. Maximum Boost Utilization remains below 16%, well under the 30% offline budget bound. Mean Boost Utilization at $U = 1.0$ is 1.06%, 2.64%, and 6.75% for 3-, 5-, and 8-task systems, confirming that only a small fraction of the provisioned boost budget is exercised at runtime. Net Normalized Energy per Slot is negative in all scenarios (mean between -0.64 and -0.60), indicating that idle-time savings offset occasional boosting. Overall, dynamic boost cancellation preserves schedulability while substantially reducing common-case boost usage and associated energy/thermal overhead.

(A) 3 Tasks



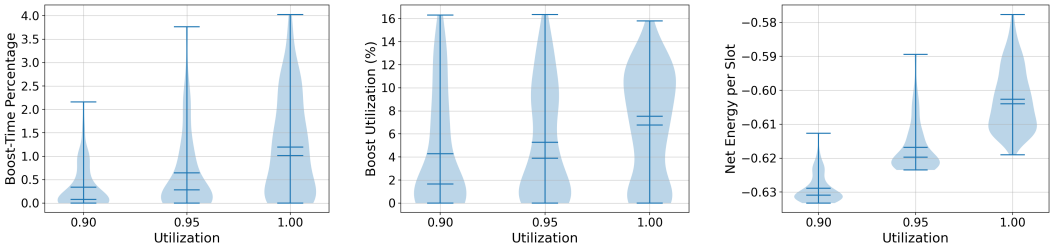
(a) Boosted-Time Percentage. **(b)** Boost Utilization. **(c)** Net Norm. Energy/Slot.

(B) 5 Tasks



(d) Boosted-Time Percentage. **(e)** Boost Utilization. **(f)** Net Norm. Energy/Slot.

(C) 8 Tasks



(g) Boosted-Time Percentage. **(h)** Boost Utilization. **(i)** Net Norm. Energy/Slot.

Figure 4 Dynamic boost cancellation results across three task-set sizes (rows). Each row reports (left) Boosted-Time Percentage, (middle) Boost Utilization, and (right) Net Normalized Energy per Slot.

7.5 Dynamic Energy Efficiency

We compare *Boosted-FP* against an offline analytical baseline that ensures schedulability via global frequency scaling. Unlike our approach, the baseline does not exploit work-triggered boosting or execution slack reclamation; instead, it uniformly accelerates all execution whenever the task set is not schedulable at nominal frequency under the given fixed-priority (FP) policy.

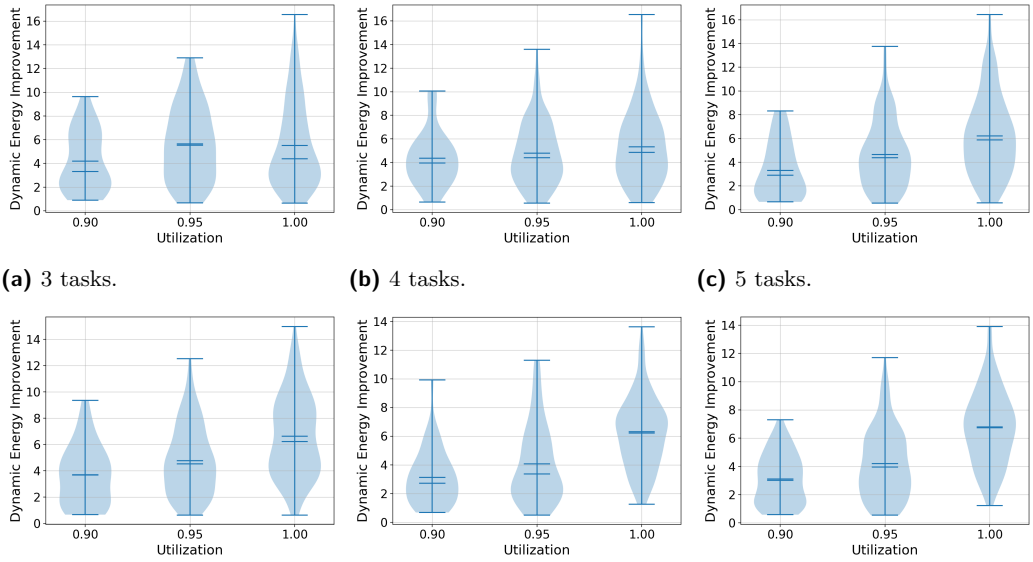
7.5.1 Offline Global Frequency-Scaling Baseline

The baseline executes all jobs at a single scaled frequency $f_\beta = \beta f_\mathrm{n}$, where $\beta \geq 1$. It is applied only to task sets that are not schedulable at nominal speed. Assuming implicit deadlines ($D_i = T_i$), the response time of task τ_i under scaling is upper-bounded by

$R_i(\beta) \leq (C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{R_i(\beta)}{T_j} \right\rceil C_j)/\beta$. A sufficient condition for $R_i(\beta) \leq D_i$ is $\beta \geq (C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j)/D_i$. Accordingly, we compute an upper bound

$$\beta_{\mathrm{GS}} = \max\left\{1,\ \max_i \frac{C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j}{D_i}\right\}.$$

Since this bound is sufficient but not necessarily tight, we refine it via binary search over $\beta \in [1, \beta_{\mathrm{GS}}]$, invoking exact response-time analysis at each step. This corresponds to classical processor-speedup (resource-augmentation) analysis for fixed-priority scheduling [20]. The baseline serves as an idealized analytical reference with continuously tunable frequency and is used solely for quantitative comparison.



(a) 3 tasks. **(b)** 4 tasks. **(c)** 5 tasks.

(d) 6 tasks. **(e)** 7 tasks. **(f)** 8 tasks.

Figure 5 Energy-improvement of *Boosted-FP* over the offline global frequency-scaling baseline across different task-set sizes and utilizations.

7.5.2 Results

The experimental setup is identical to that used in Subsection 7.4. For idle energy computation, we set $\alpha = 0.2$. We define energy improvement as $100 \times (E_{\mathrm{GS}} - E_{\mathrm{BF}})/E_{\mathrm{GS}}$, where E_{GS} and E_{BF} denote the dynamic energy consumption under global scaling and *Boosted-FP*, respectively. Positive values indicate lower energy consumption by *Boosted-FP*. Figure 5 reports energy improvement for task sets with 3–8 tasks and utilizations $U \in \{0.9, 0.95, 1.0\}$. Across all configurations, *Boosted-FP* consistently reduces energy relative to global scaling. At $U = 1.0$, mean improvement ranges from 5.5% (3 tasks) to 6.7% (7–8 tasks), with medians between 4.4% and 6.8%. Even at $U = 0.9$, mean improvements remain around 3–4%. Energy gains increase with utilization. For example, in 6-task systems, mean improvement rises from 3.7% at $U = 0.9$ to 6.6% at $U = 1.0$. This reflects that global scaling must uniformly raise processor speed under heavy load, whereas *Boosted-FP* concentrates additional capacity in short boosted intervals. Energy improvement also grows with task-set size. Larger systems

lead the baseline to select higher scaling factors, while *Boosted-FP* benefits from additional slack-reclamation opportunities. Overall, *Boosted-FP* preserves schedulability while achieving consistent and scalable energy-efficiency gains over static worst-case global frequency scaling.

8 Related Work

This work relates to extensions of fixed-priority scheduling, dual-priority techniques, and dynamic voltage and frequency scaling (DVFS) in real-time systems. We summarize the most relevant directions and highlight how *Boosted-FP* differs from existing approaches.

Dual-priority schemes initially assign jobs a low priority and promote them later to improve schedulability under fixed-priority scheduling and to approximate dynamic-priority behavior [5, 6, 11]. By delaying promotion until a job approaches its deadline, such techniques reduce interference on lower-priority tasks while preserving the fixed-priority framework. While dual-priority variants can schedule task sets that are unschedulable under classical fixed-priority analysis, they require runtime priority changes and additional scheduling states, increasing implementation and certification complexity in safety-critical systems. *Boosted-FP* addresses marginal unschedulability differently. It preserves a single static priority assignment and selectively accelerates execution near job completion via bounded frequency boosting. This achieves an interference-reduction effect similar to priority promotion, without introducing dynamic priority transitions.

DVFS has been widely studied for energy reduction in real-time systems [22, 2, 3]. Classical approaches reduce processor speed under EDF or fixed-priority scheduling while maintaining schedulability, often exploiting slack or utilization bounds to determine safe operating points. In contrast, *Boosted-FP* assumes nominal execution speed and applies bounded acceleration only when necessary. Boosting is conservatively modeled as an effective reduction in execution time, enabling schedulability proofs via domination arguments over a reduced task set under standard fixed-priority response-time analysis. Whereas techniques such as RT-DVS [22] aim to reduce energy consumption while preserving real-time schedulability guarantees, *Boosted-FP* targets schedulability expansion. The approaches are therefore complementary.

Slack reclamation methods exploit early job completions for energy savings or improved responsiveness under EDF and fixed-priority scheduling [15, 23, 17]. *Boosted-FP* is compatible with such techniques but differs in purpose: rather than reclaiming slack to reduce processor speed, it uses bounded acceleration to compensate for offline-identified execution-time reductions. The optional boost-cancellation mechanism leverages dynamic slack only to avoid unnecessary boost execution, without altering schedulability guarantees. Overall, *Boosted-FP* combines elements of dual-priority scheduling and DVFS-based execution control while remaining grounded in classical fixed-priority analysis. By decoupling execution acceleration from priority management and confining boosting to bounded, work-triggered regions, it retains analyzability and implementation simplicity.

9 Conclusion

We presented *Boosted-FP*, a fixed-priority scheduling framework that leverages bounded, thermally gated frequency boosting to improve schedulability without changing task priorities. Boost activation is driven by conservative worst-case remaining-work information and is confined to the tail of jobs. We showed that work-triggered boosting can be analyzed as an effective execution-time reduction, enabling schedulability guarantees via classical fixed-priority response-time analysis and a domination argument, and we derived sufficient conditions to bound total boost usage. Experiments demonstrate that even modest boost

budgets substantially expand the schedulable region near high utilization while requiring only limited boost time in the worst case. Future work includes extending the framework to multiprocessor platforms, integrating more detailed thermal and power models, and exploring adaptive selection of boost parameters based on observed execution behavior.

References

1 Neil C. Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993. doi:10.1049/sej.1993.0034.

2 H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pages 95–105, 2001. doi:10.1109/REAL.2001.990600.

3 Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1), January 2016. doi:10.1145/2808231.

4 Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 159–168, USA, 2006. IEEE Computer Society. doi:10.1109/RTSS.2006.47.

5 Alan Burns. Dual priority scheduling: Is the processor utilisation bound 100%? In *Proceedings of the 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS), in conjunction with ECRTS*, 2010. Available online.

6 Alan Burns and Andrew J. Wellings. Dual priority assignment: A practical method for increasing processor utilisation. In *Proceedings of the Fifth Euromicro Workshop on Real-Time Systems (RTS)*, pages 48–53, Oulu, Finland, 1993. IEEE. doi:10.1109/EMWRT.1993.639052.

7 Alan Burns and Andy J. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 4 edition, 2009.

8 A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992. doi:10.1109/4.126534.

9 Thidapat Chantem, X. Sharon Hu, and Robert P. Dick. Online work maximization under a peak temperature constraint. In *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 105–110, San Francisco, CA, USA, 2009. ACM. doi:10.1145/1594233.1594257.

10 Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 141–150. IEEE, 2009. doi:10.1109/RTAS.2009.30.

11 R. Davis and A. Wellings. Dual priority scheduling. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, RTSS '95, page 100, USA, 1995. IEEE Computer Society. doi:10.1109/REAL.1995.495200.

12 P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, July 2010.

13 Intel Corporation. 12th Generation Intel Core Processors Datasheet, Volume 1 of 2: Processor Line Thermal and Power. https://edc.intel.com/content/www/de/de/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/004/processor-line-thermal-and-power/. Accessed: 2026-02-23.

14 Intel Corporation. How intel technologies boost your cpu's performance. Technical report, Intel, 2021. Technology brief; describes hardware frequency boosting and thermal limits. URL: https://www.intel.com/content/www/us/en/gaming/resources/how-intel-technologies-boost-cpu-performance.html.

15 Ravindra Jejurikar and Rajesh Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the 42nd Annual Design Automation Conference*, DAC '05, pages 111–116, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1065579.1065612.

16 M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986. doi:10.1093/comjnl/29.5.390.

17 W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '02, page 788, USA, 2002. IEEE Computer Society. doi:10.1109/DATE.2002.998389.

18 John P. Lehoczky, Lui Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 10th IEEE Real-Time Systems Symposium (RTSS)*, pages 166–171, 1989. doi:10.1109/REAL.1989.63567.

19 C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

20 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 140–149, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258570.

21 Leonardo Piga, Iyswarya Narayanan, Aditya Sundarrajan, Matt Skach, Qingyuan Deng, Biswadip Maity, Manoj Chakkaravarthy, Alison Huang, Abhishek Dhanotia, and Parth Malani. Expanding datacenter capacity with dvfs boosting: A safe and scalable deployment experience. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '24), Volume 1*, pages 1–16, New York, NY, USA, 2024. ACM. doi:10.1145/3617232.3624853.

22 Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2001. doi:10.1145/502034.502044.

23 D. Zhu, R. Melhem, and B.R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(7):686–700, 2003. doi:10.1109/TPDS.2003.1214320.

A Informal Discussion of Out-of-Normal-Mode Thermal Events

This appendix does not propose or analyze a specific fallback mechanism. Instead, it outlines possible system-level responses when boost availability is lost due to hardware thermal control. Such executions fall outside the normal-mode setting used in the schedulability analysis of this paper.

One possible response is to continue executing all tasks at the nominal frequency, accepting that the guarantees derived for Boosted-FP no longer apply during the throttled interval. This may be appropriate for systems in which deadline misses can be tolerated temporarily or handled at the application level.

Another possible response is a criticality-aware policy. For example, tasks with hard deadlines could continue to receive service under the original fixed-priority order, while tasks with soft deadlines or lower criticality could be delayed, degraded, or skipped until boost availability is restored. This resembles the high-level idea of mixed-criticality scheduling, but we do not instantiate or analyze such a policy here.

A third possible response is workload adaptation, such as reducing input rates, dropping optional jobs, or switching to lower-quality service modes. These mechanisms are system-dependent and require separate analysis. Integrating such out-of-normal-mode behavior with Boosted-FP is left for future work.