Work-in-Progress: A Practical Linux Framework for Weakly-Hard Tasks with Constant Bandwidth Server

Marcus Chen*, Pascal Reich*, Yidi Wang†, and Hyunjong Choi*
*San Diego State University, USA
mchen6446, preich5404, hyunjong.choi @sdsu.edu
†Santa Clara University, USA
yidi.wang@scu.edu

***Abstract*—Weakly-hard real-time systems enhance resource efficiency by allowing bounded deadline misses, but practical support in existing platforms remains limited. In this work-in-progress, we present the *CBS-based Weakly-Hard Framework*, which maps tasks with (m, K) constraints into Constant Bandwidth Server (CBS) parameters under the `SCHED_DEADLINE` policy in Linux. The proposed approach requires no kernel modifications and guarantees schedulability through constrained-deadline EDF analysis. We implemented the framework as a user-space API and evaluated it on a Raspberry Pi platforms. Experimental results show improved schedulability compared to existing methods and demonstrate potential opportunities for Quality-of-Service (QoS) enhancement.**

I. Introduction

A weakly-hard real-time system is defined by a bounded distribution of deadline meets and misses within a specified window [4]. Unlike conventional real-time systems that classify tasks as strictly hard or soft, weakly-hard systems recognize that occasional deadline misses can be tolerated in practice. By constraining the maximum number of misses in any K-job window, they provide a guaranteed minimum level of service while improving resource efficiency. Despite these merits, there is still no practical framework that enables weakly-hard tasks to run directly through standard application programming interfaces (APIs). One of the latest attempts, the job-class-level scheduling (JCLS) framework [7], makes progress toward practicality but requires extensive kernel-level modifications, limiting its deployment in existing systems.

The Constant Bandwidth Server (CBS) [2] was originally proposed to handle aperiodic tasks by reserving CPU resources through a server-based approach. Since Linux kernel version 3.14, CBS has been implemented as the `SCHED_DEADLINE` real-time scheduling policy, which integrates EDF scheduling with resource reservation mechanisms [3, 9]. By providing predictable bandwidth allocation, CBS offers a natural foundation for enforcing the minimum execution requirements of a given taskset. Building on this idea, we investigate how a weakly-hard taskset can be mapped into CBS parameters, thereby enabling their execution within existing Linux systems without requiring kernel-level modifications.

In this paper, we introduce a work-in-progress effort toward a practical approach that enables weakly-hard tasks to run in the Linux system by leveraging the constant bandwidth server. We propose the *CBS-based Weakly-Hard Framework*, which converts weakly-hard tasks with (m, K) constraints into CBS parameters, allowing them to execute under the `SCHED_DEADLINE` scheduling policy. This paper makes the following contributions:

- We enable weakly-hard tasks to run in the Linux system without requiring significant kernel-level modifications by leveraging the CBS under the `SCHED_DEADLINE` policy.
- We utilize the schedulability test of EDF scheduling with constrained deadlines to ensure feasibility under weakly-hard guarantees.
- We implement the proposed framework as an API library on Linux and evaluate it on a real embedded platform (Raspberry Pi 5).

The rest of the paper is organized as follows: Sec. II introduces the task and server models. Then, we propose CBS-based weakly-hard execution framework in Sec. III. Sec. IV describes the API implementation and evaluates the proposed approach. Sec. V concludes the paper and discusses possible extensions of this work.

II. System model

This paper considers a uniprocessor system where the CPU runs at a fixed clock frequency. The system executes a taskset consisting of N periodic or sporadic real-time tasks with constrained deadlines.

Task model. Each task τ_i is characterized as follows:

$$\tau_i := (C_i, D_i, T_i, (m_i, K_i))$$

- C_i: The worst-case execution time of each job of a task τ_i.
- D_i: The relative deadline of each job of τ_i ($D_i \leq T_i$).
- T_i: The minimum inter-arrival time between consecutive jobs of τ_i. If τ_i is a periodic task, T_i is the period of τ_i.
- (m_i, K_i): The weakly hard constraints of τ_i ($m_i < K_i$), where m_i is the number of allowed deadline misses in any K_i consecutive windows. If τ_i is a hard real-time task, $m_i = 0$ and $K_i = 1$.

Each task can have a release jitter $\mathcal{J}_i$, where $\mathcal{J}_i \leq D_i - C_i$. The j-th job of a task τ_i is denoted as $J_{i,j}$.

Utilization. To represent the resource demand and effective performance of weakly-hard systems, we adopt the utilization metrics introduced in [7]:

- **Maximum utilization** U_i^M **of** τ_i (Def. 1 of [7]): The maximum amount of CPU resource that τ_i can demand, defined as $U_i^M = \frac{C_i}{T_i}$. This corresponds to the resource usage when every job of τ_i meets its deadline.

- **Minimum utilization U_i^m of** τ_i (Def. 2 of [7]): The effective CPU demand of task τ_i when it misses the maximum number of deadlines permitted by its (m_i, K_i) constraint, defined as i.e., $U_i^m = \frac{C_i}{T_i} \times \frac{K_i - m_i}{K_i}$.

Note that each task requires at least U_i^m of CPU resource to be schedulable with respect to its weakly-hard constraint.

Server model. We consider M subsystems $s_k \in \Psi$, where $k = 1, 2, ..., M$. Each subsystem is managed by a reservation server characterized by a budget Q_k and a period P_k. The server bandwidth is defined as $\alpha_k = Q_k/P_k$ and the worst-case service delay is bounded by $2(P_k - Q_k)$. We assume that each subsystem executes a taskset Γ_k that consists of n_k periodic tasks described by the task model introduced earlier. As an initial step in this work, we restrict our framework to the case where each server s_k handles only a single task τ_i (i.e., $\forall k = 1, 2, ..., M,\ n_k = 1$). While this assumption simplifies the design, it also ensures timing isolation among tasks, consistent with AUTOSAR specifications [1]. As future work, we plan to extend the design by introducing a *local scheduler* that supports allocating multiple tasks to a single server s_k, thereby reducing the overhead of creating a large number of servers.

III. CBS-BASED WEAKLY-HARD FRAMEWORK

The goal of our framework is to support weakly-hard tasksets on Linux platforms without requiring kernel-level modifications. Instead of introducing a new scheduling policy, we leverage the Constant Bandwidth Server, which is already implemented in the Linux kernel as the `SCHED_DEADLINE` based on EDF scheduling policy. The key idea of our framework to guarantee that each task continues to satisfy its weakly-hard constraint (m, K) when interpreted in its origianl time domain, even though it is executed under CBS. To achieve this, each task is mapped to a server whose budget (Q_k) and period (P_k) are configured according to it's parameters including weakly-hard constraint. This design not only preserves the weakly-hard guarantees but also enables us to directly apply the existing EDF schedulability analysis used for CBS.

A. Mapping a weakly-hard task to a CBS

In this subsection, we describe how to transform the parameters of each weakly-hard task τ_i into CBS parameters.

Budget of a server (Q_k). The minimum required budget of a server is set to the worst-case execution time of its associated task τ_i, i.e., $Q_k = C_i$, since the server must always be capable of executing a full job of τ_i within its relative deadline D_i.

Period of a server (P_k). To determine the server period, we consider the maximum number of consecutive deadline misses that τ_i can tolerate while still satisfying its constraint. This value, referred to as the *miss threshold*, was introduced in [7] and is defined as:

$$w_i = \max(\left\lfloor \frac{K_i}{K_i - m_i} \right\rfloor - 1, 1)$$

The w_i parameter provides an upper bound on the number of consecutive deadline misses that τ_i may incur while meeting its weakly-hard requirement. This threshold serves as a key element in deriving an appropriate CBS server period P_k, thereby ensuring that the resulting schedule preserves the timing guarantees of the weakly-hard model.

Based on w_i, a period of a server can be determined as the following definition.

Def. 1. *The maximum admissible server period P_k for a task τ_i is defined as:*

$$P_k = \begin{cases} T_i & \text{, if } \frac{m_i}{K_i} < 0.5, \\ (w_i + 1)\,T_i & \text{, otherwise.} \end{cases}$$

where w_i denotes the miss threshold of τ_i.

The server period can be effectively determined by the spacing between budget replenishments. Since the server budget is set to $Q_k = C_i$ and the server deadline equals the task deadline, i.e., $D_k = D_i$, any budget available after D_i cannot contribute to the current job's completion. Given that task τ_i can tolerate at most w_i consecutive deadline misses, the server can defer providing budget for up to $w_i \cdot T_i$. However, immediately after this interval at least one job must meet its deadline to prevent violating the weakly-hard constraint. Therefore, the replenishment must occur no later than $(w_i + 1)T_i$, which establishes the upper bound on the server period.

In the special case where $\frac{m_i}{K_i} < 0.5$, the miss threshold is always $w_i = 1$. If the server is configured with a period of $2T_i$, it completes one job execution and then provides no budget in the following T_i interval, and repeating this pattern. Such alternating behavior of a job completions and misses ultimately violates the weakly-hard constraint. For instance, a task with $(m_i, K_i) = (1, 3)$ that misses every other deadline fails to satisfy its weakly-hard guarantee, i.e., "miss-meet-miss" does not satisfy $(1, 3)$. This observation is consistent with the findings in [7], which shows that when $\frac{m_i}{K_i} < 0.5$, schedulability must be verified by exploring all possible execution patterns using a reachability tree.

B. Constrained-deadline EDF analysis

Since our framework leverages `SCHED_DEADLINE` in the Linux kernel, an implementation of EDF scheduling augmented with a CBS mechanism, we apply the schedulability test of constrained-deadline EDF based on the processor demand criterion, following the Theorem 4.6 in [6]. The feasibility condition is given by:

$$U < 1 \quad \text{and} \quad \forall t \in \mathcal{D},\ \ \text{dbf}(t) \le t,$$

where $U = \sum_{i=1}^{n} U_i$ is the total utilization, and

$$\mathcal{D} = \{d_\ell \mid d_\ell \le \min\left[H, \max(D_{\max}, L^*)\right]\},$$

with H denoting the hyperperiod of the taskset, $D_{\max} = \max_{\forall i}\{D_i\}$, and

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1 - U}.$$

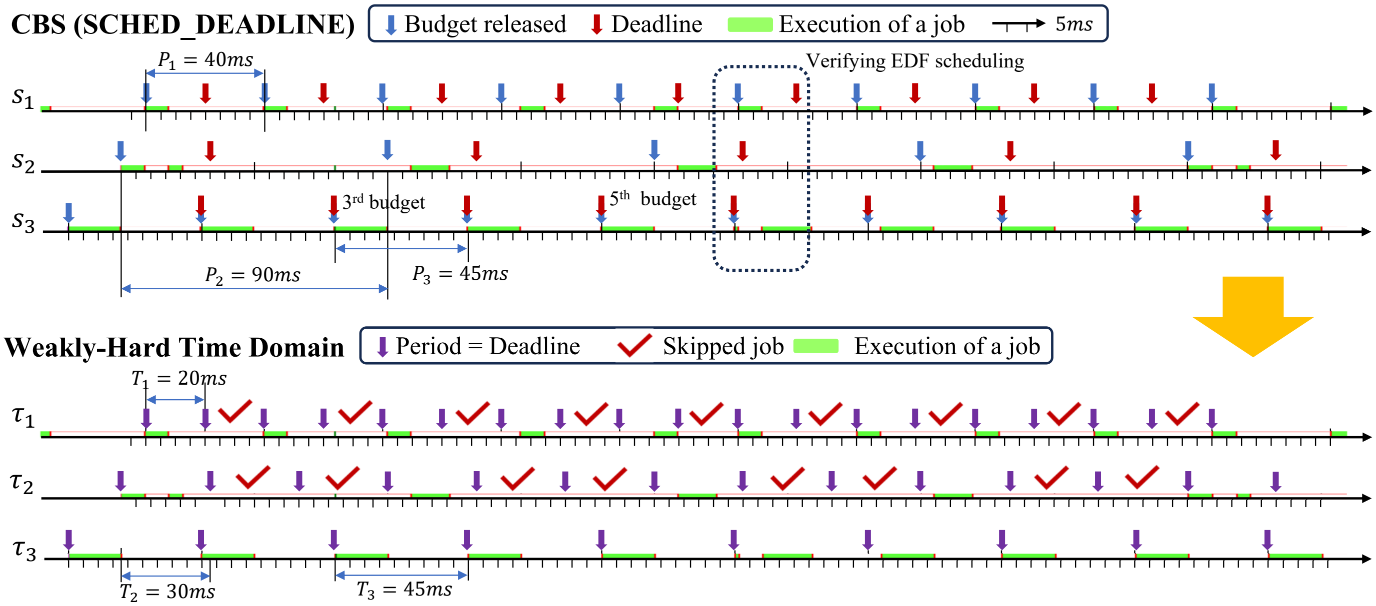

Fig. 1: CBS scheduling (top) and interpretation in weakly-hard time domain (bottom)

Here, $\mathcal{D}$ is the set of candidate time instants at which the demand bound function (dbf) needs to be checked. The term L^* provides an upper bound on the interval length beyond which the processor demand test does not need to be evaluated, thus limiting the number of points to check.

We evaluate the schedulability test by comparing it prior works in Sec. IV.

C. Example

In this subsection, we demonstrate the execution of an example task set under the proposed framework. Our framework is implemented as an open-source API library[1], which simplifies the deployment of weakly-hard tasksets. We tested the library using the example taskset shown in Table I on a Raspberry Pi 5 platform.

Table. I presents a taskset consisting of three weakly-hard tasks with a total maximum utilization (U^M) of 1.44 and a minimum utilization (U^m) of 0.7130. Based on the analysis in Sec. III-B, this taskset is schedulable after conversion to CBS parameters according to Def. 1. To validate its behavior on real hardware, we executed the taskset for 5 minutes and collected scheduling events using `trace-cmd`, which were then visualized with KernelShark.

TABLE I: Weakly-hard taskset and CBS parameters [ms]

Tasks	C_i	D_i	T_i	(m_i, K_i)	Q_k	P_k
τ_1	10	20	20	(1, 2)	10	40
τ_2	15	30	30	(2, 3)	15	90
τ_3	20	45	45	(1, 3)	20	45

Fig. 1 illustrates the observed scheduling behavior. The top panel shows how each server executes its reserved budget to run corresponding task. Blue arrows mark the start of a new budget, while red arrows indicate the deadlines of a budget. These timing events can be accurately reconstructed via KernelShark. For example, we know that the third and fifth budgets of s_3 begin executions immediately at the release time since no other budgets are pending. From these reference points, we can infer the release times and deadlines of all other budgets. The trace confirms that all budgets complete within their deadlines, ensuring correct execution of the associated jobs. Furthermore, EDF scheduling is observed in the dotted region, i.e., s_1 preempts s_3 because its deadline is earlier. When viewed in the weakly-hard time domain (bottom panel), the trace shows that tasks meet their weakly-hard constraints: τ_1 skips every other job and satisfies the weakly-hard constraint of $(1, 2)$, while τ_2 meets one job out of any three consecutive jobs.

[1]The API is available together with the analysis framework in the project repository as a open-source. https://github.com/rteclab/CBS-WH

IV. Evaluation

A. Implementation

We implemented an API library to simplify the use of proposed framework. The library employs the system call, `SYS_sched_setattr`, to register weakly-hard tasks as servers by leveraging `SCHED_DEADLINE` scheduling interface. The converted CBS parameters (described in Sec. III-A) are directly mapped to the `SCHED_DEADLINE` interface [9] as `dl_runtime` $= Q_k$, `dl_deadline` $= D_k$, and `dl_period` $= P_k$. This API supports two modes of use: 1) registering a weakly-hard task for standalone execution by calling the function, `weaklyhard_register()` within the task, and 2) creating multiple weakly-hard threads using `weaklyhard_create()`.

B. Experiments

In this subsection, we present experiments that demonstrate the behavior of our proposed framework.

Comparison of schedulability tests. We first compare the schedulability performance of our approach (**CBS-WH**) against three existing methods:

- **WSA** [10]: the offset-free weakly-hard schedulability analysis for fixed priority scheduling, one of the representative analytical studies on weakly-hard systems
- **RTO-RM** [8]: the Red-Task-Only version of the skip-over algorithm
- **JCLS** [7]: a new job-class-level scheduling (JCLS) with analysis framework for weakly-hard tasks, one of the latest studies on weakly-hard systems

For the experiments, We randomly generate $1,000$ tasksets for each utilization, each consisting of 20 tasks, using the UUniFast algorithm [5]. Task periods are chosen randomly from the interval $[10, 1000]$ ms with deadlines set equal to periods, i.e., $T_i = D_i$. The weakly-hard constraint K was selected from the $\{5, 10, 15\}$, following the motivation in [11].

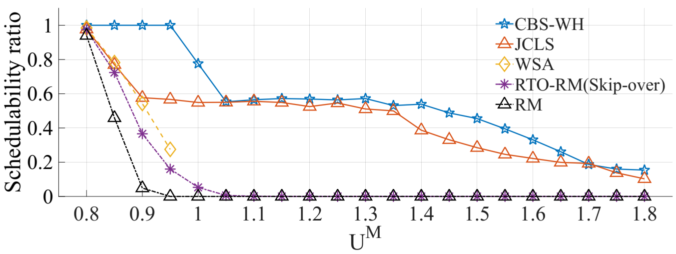

Fig. 2: Schedulability under different weakly-hard analysis

Fig. 2 shows the schedulability ratio of weakly-hard tasksets under different approaches. At $U^M = 0.95$, CBS-WH schedules 100% of the tasksets, while the other methods schedule only 56% for JCLS, 26% for WSA, and 16% for RTO-RM. CBS-WH consistently outperforms the other approaches when $U^M \leq 1$, primarily because it leverages EDF scheduling, which dominates fixed-priority strategies. For $U^M > 1$, CBS-WH still results slightly higher schedulability ratio than JCLS, as JCLS employs dynamic priorities only within a task, whereas CBS-WH benefits from fully dynamic priority scheduling.

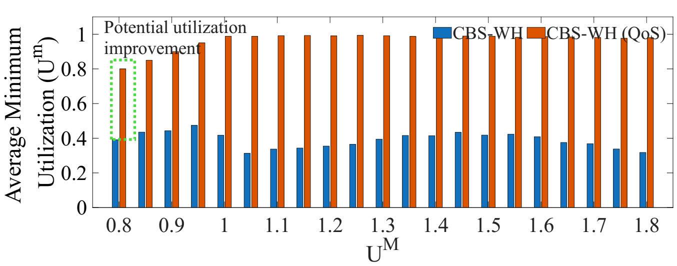

Fig. 3: Enhancement of total minimum utilization (U^m)

Quality of service enhancement. In this experiment, we aim to improve utilization by executing more jobs than those required by the specified weakly-hard constraint, thereby evaluating the potential utilization enhancement for weakly-hard tasksets under the proposed framework. As shown in Fig. 3, the CPU remains underutilized where the blue bars represent the average minimum total utilization of schedulable tasksets under CBS-WH. To achieve enhancement of utilization, we progressively decrement w_i of each task (from τ_1 to τ_{20}) while maintaining schedulability, which effectively increases the number of completed jobs within K consecutive job window. The red bars in 3 demonstrate that substantial utilization can be exploited across all utilization levels, thus, improving the quality of service for the given tasksets. We plan to formalize a QoS metric and develop an adaptive algorithm that dynamically adjusts w_i to enhance overall system performance as a future work.

V. Conclusion and Future Work

This paper presented a practical framework for executing weakly-hard tasksets on Linux using the CBS mechanism under `SCHED_DEADLINE`. While our current focus has been on mapping individual weakly-hard tasks to CBS servers, several important directions remain for future exploration:

Supports for multiple tasks allocation. We plan to design a local scheduler that allows multiple weakly-hard tasks to be dispatched within a single server. This will require incorporating all parameters of the individual tasks while preserving weakly-hard guarantees, and it will help reduce the overhead of creating one server per task.

System supports for processing chains. In modern autonomous and intelligent systems, tasks are often interdependent and form processing chains. Extending our framework to support such chains requires end-to-end latency analysis under weakly-hard constraints, which is a challenging open problem. We aim to develop a practically usable framework that can analyze and guarantee performance for such chain models, making our approach more applicable to real-world applications.

Integration into open-source frameworks. We intend to integrate our framework into widely used open-source platforms such as ROS 2, Cyber-RT, and AUTOSAR. This will facilitate adoption in industry and research communities, and demostrate the practicality of weakly-hard models in complex cyber-physical systems.

References

[1] AUTOSAR Release 4.1, Specification of Operating System. https://www.autosar.org, 2013. Accessed: 2025-09-15.

[2] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, page 4–13. IEEE, 1998.

[3] L. Abeni, G. Lipari, and J. Lelli. Constant bandwidth server revisited. *ACM SIGBED Review*, 11(4):19–24, 2015.

[4] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001.

[5] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] G. C. Buttazzo. *Periodic Task Scheduling*, chapter 4, pages 79–118. Springer, 3 edition, 2011.

[7] H. Choi, H. Kim, and Q. Zhu. Job-Class-Level fixed priority scheduling of weakly-hard real-time systems. In *IEEE RTAS*, 2019.

[8] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *RTSS*, 1995.

[9] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli. Deadline scheduling in the linux kernel. *Software: Practice and Experience*, 2016.

[10] Y. Sun and M. D. Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM TECS*, 2017.

[11] M. Yayla, K.-h. Chen, and J.-j. Chen. Fault Tolerance on Control Applications : Empirical Investigations of Impacts from Incorrect Calculations. In *EITEC*, 2018.