Work in Progress: Guaranteeing weakly-hard timing constraints in server-based real-time systems

Nasim Samimi, Mitra Nasri, Twan Basten, Marc Geilen
Eindhoven University of Technology
Eindhoven, The Netherlands

***Abstract*—Ensuring deadlines of hard real-time applications in server-based deployments is a challenging problem, particularly if the workload arrives following an arbitrary arrival curve. This work extends the "$(\mathcal{M},\mathcal{K})$-firm weakly hard" model to server-based systems, ensuring timely processing of real-time requests to the server. We introduce an admission policy to regulate the remote server workload and prevent deadline misses while attempting to admit more requests than the minimum required, when possible. We guarantee the weakly hard constraints through optimal resource allocation and server configuration.**

***Index Terms*—Server-based systems, admission test, response time analysis, arrival curve, real-time systems**

I. Introduction

Server-based systems are commonly utilised to provide centralised resources allowing computing nodes with limited resources to offload their tasks. However, these servers may not always be able to meet hard or firm timing constraints (where a failure to meet deadlines may result in intolerable/unwanted consequences) when the server is overloaded and there are not enough resources to process all the requests, transmitted from computing nodes, before their deadlines.

With the increasing use of application offloading to remote servers (e.g. cloud servers [1]), it is beneficial to define a model for a guaranteed quality of service in these systems. We exploit the $(\mathcal{M},\mathcal{K})$-firm weakly hard model commonly used to represent timing constraints in real-time systems, where at least $\mathcal{M}$ out of any $\mathcal{K}$ consecutive jobs must meet the deadline. We extend this model to server-based deployments, where the server guarantees to process $\mathcal{M}$ out of any $\mathcal{K}$ consecutive requests to a remote server and meet their deadline requirements. Our work proposes a job-level admission test and scheduling approach that ensures $(\mathcal{M},\mathcal{K})$-firmness constraints for server-based applications.

Many works focus on task-level admission control in distributed systems [2], [3], where the goal is to either accept all jobs of a task or none. We focus on a job-level admission test to be able to accept most jobs of a task on the server and only reject jobs that the server cannot process due to its transient overloads.

To ensure that requests are handled effectively, three conditions are necessary: (i) the weakly-hard constraints must be met, (ii) requests that are not ensured to meet the deadline should be recognised quickly so that a fallback scenario can be used, and (iii) the selection of $\mathcal{M}$ out of $\mathcal{K}$ requests should minimise the required resources.

The issue of selecting $\mathcal{M}$ out of $\mathcal{K}$ jobs in systems with weakly hard constraints has been addressed in various works based on the classification of jobs as mandatory and optional to meet the deadlines for a given set of resources which considers periodic and/or sporadic tasks [4]–[11]. However remote server requests do not necessarily follow periodic patterns. To support a wide class of arrival patterns for server requests and capture the potential communication delays that may result in a bursty arrival of requests to the server, we exploit *arrival curves* [12]. The literature has studied the problem of bounding deadline misses to ensure weakly-hard constraints of tasks with arrival curves [13]–[17]. These works, however, do not predict whether a job will miss its deadline before its admission to enable a fallback scenario in case the server is overloaded.

Once a request is admitted its deadline must be satisfied necessitating a minimum guaranteed resource. We ensure guaranteed resources using reservation-based scheduling. These allocated reservation servers must meet the deadlines of the requests that are admitted by providing enough resources to those requests. They ensure timing predictability for tasks mapped to them. We use the constant-bandwidth server (CBS) currently used in Linux for real-time applications.

There exists research [18], [19] on configuring reservation-server parameters to operate with the lowest possible utilisation that we can adapt to our work. Nevertheless, in many server-based applications (especially in the cloud), the incoming requests are independent and can arrive in bursts. In this situation, an application might receive multiple requests, which must be distributed over more than one core to be able to meet their deadlines. We name such applications *multi-CBS* applications. This poses a new challenge to distribute the workload among CBSs mapped onto different cores while minimising the allocation of resources to the CBSs dedicated to such applications.

We schedule CBSs according to the preemptive scheduling model as used in existing server-based systems. Furthermore, we propose a partitioned earliest deadline first (EDF) scheduling approach aiming for a higher resource utilisation and preventing migration overhead improving timing predictability, analysability, and deadline guarantees [20].

This paper. We propose a job-level admission test that ensures $(\mathcal{M},\mathcal{K})$ and deadline constraints, optimises resource usage for tasks defined by arrival curves, and enables fallback scenarios on computing nodes. Our specific contributions are:

2642-7346/24/$31.00 ©2024 IEEE
DOI 10.1109/RTAS61025.2024.00043

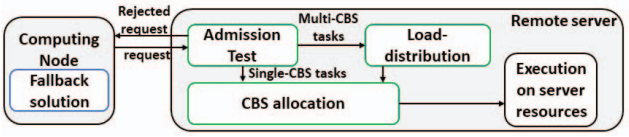

Figure 1: Runtime phase

- An approach to derive the minimum required service from the server to maintain $(\mathcal{M}, \mathcal{K})$ constraints of requests restricted by an arrival curve.
- An online admission policy with practical runtime (e.g., in microseconds) that maximises resource utilisation while guaranteeing that admitted requests meet their deadlines.
- An approach to map multi-CBS applications to CBSs located on different cores while minimising the total required amount of resources.
- A load-distribution method to distribute workloads of multi-CBS applications among allocated CBSs.

II. System Model

1) Architecture of the system: We assume a server-based architecture with nodes that have limited computational resources. These nodes are connected to a server with a multi-core platform that can run given applications when requested. When a node sends a request to the server, the request includes information about the type of application, input data, and deadline. The server then notifies the node whether the request has been accepted or not. If the request is rejected, the node will have to execute the application locally or not all all, which may result in degraded quality (see Fig. 1). Because the focus of our work is on the server-side of the problem and the admission test, we work with a simplifying assumption that the deadline of an incoming request (job) accommodates communication delays along with a contingency plan in case the server rejects the job.

A server application is called a *task* and incoming requests to that application are called *jobs*. Jobs are assumed to be independent and single-threaded. We assume cores to be homogeneous.

2) Workload model: We assume a set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n weakly-hard real-time tasks (offloaded to the remote server during system configuration). A task τ_i is represented by an upper arrival curve, the worst-case job execution time (C_i^{max}), and a relative deadline (D_i). $\mathcal{M}_i$ and $\mathcal{K}_i$ define its weakly-hard timing constraints, namely, at least $\mathcal{M}_i$ jobs out of any $\mathcal{K}_i$ consecutive jobs that arrive in the server must execute on the server and meet their deadlines.

3) Workload arrival model: We assume that the workload that can be sent to the server follows an arrival curve, a flexible way to represent a wide class of known arrival models such as periodic, sporadic, bursty, and irregular arrivals. An arrival curve represents the maximum amount of workload (maximum number of jobs times their WCET) that can arrive in the server within any given time interval.

4) Scheduling and execution models: A CBS is assigned specifically to one task. CBSs do not overrun their budget. We assume a constant context-switching overhead for preempting a CBS by higher-priority CBSs.

If a task is not schedulable on a single core, e.g., due to its bursty arrival curve, more than one CBS is allocated to the task, each running on a different core. The jobs of the corresponding task are distributed among the allocated CBSs by the load-distribution method (Sec. III-B).

III. Overview of the solution

The primary challenge in this work is to design an admission test. The admission test must strategically select and admit $\mathcal{M}$ out of any consecutive $\mathcal{K}$ jobs of each task. A CBS will be allocated to each task. The strategy of the admission test in selecting the jobs to admit must minimise the resource demand of the allocated CBS. We may have tasks that require more than one core ('multi-CBS tasks') to be schedulable. These tasks require a different approach to be analysed, which creates new challenges, as well. The identification of multi-CBS tasks, deriving the minimum resources that are sufficient for their maximum resource demand at design-time, and distributing their workload among the CBSs that are allocated to the corresponding task at run-time, are the challenges in managing multi-CBS tasks. Finally, we must compute the CBS parameters to ensure the deadlines of the tasks while minimising the utilisation of the CBS (budget per period). The resulting CBSs use partitioned scheduling with the least number of cores possible.

A. Admission policy

Selecting the $\mathcal{M}$ admitted jobs evenly across the $\mathcal{K}$ jobs reduces the amount of admitted workload in a burst (of the arrival curve of the task), regardless of when the burst occurs. This will then help us to allocate a CBS with low utilisation to the task. Thus, we derive an admission sequence that dictates a fixed $\mathcal{M}$ out of $\mathcal{K}$ admission pattern, categorising the jobs as mandatory or optional.

1) Admission sequence: We characterise the rejection and admission of $\mathcal{K}$ consecutive jobs by a static binary sequence of length $\mathcal{K}$, where '0' means rejection and '1' means admission. This sequence is denoted by $\zeta = (\zeta_1, \ldots, \zeta_{\mathcal{K}}) \in \{0,1\}^{\mathcal{K}}$. This sequence repeats for every next set of $\mathcal{K}$ jobs. We aim to construct an optimal admission sequence ζ_{opt} that minimises the number of admitted jobs in any given interval by minimising the maximum number of 1s in any word of length 2 to length $\mathcal{K}$. We compute the indices of the terms that can be set to 1 as $ind(\mathcal{M}, \mathcal{K}) = \left\{ \left\lceil \frac{i\mathcal{K}}{\mathcal{M}} \right\rceil \;\middle|\; 1 \leq i \leq \mathcal{M} \right\}$.

2) Deriving new arrival curve: Suppose that we admit the jobs following the sequence derived before. We determine a new arrival curve representing the maximum workload of task τ that must be *admitted* in an interval of time. To derive the new arrival curve denoted by α' for a task with arrival curve α, we find the minimum interval of time Δ_l in which a given amount of workload $l \times C^{max}$ for l jobs must be admitted: $\Delta_l = \min\{\Delta \in \mathbb{R}^{\geq 0} \mid \alpha'(\Delta) = l \times C^{max}\}$. For any l admitted

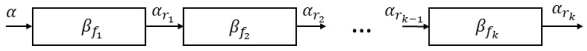

Figure 2: Assigning resources to tasks that need more than one core

jobs, we compute the shortest sequence of jobs, where the maximum of l jobs are admitted. The minimum interval for a sequence's arrival is derived from the arrival curve.

3) Opportunistic admission of optional jobs: The CBS design is based on the worst-case workload scenario and worst-case job execution times. But such worst-case workloads might not always occur, potentially leading to server under-utilisation. When jobs finish their execution earlier than their WCET, some of the reserved processor time between the actual completion time (CT) and the worst-case completion time (CT^{max}) of the job will become available to other jobs. To benefit from such occasions, we employ an opportunistic strategy at runtime. Mandatory jobs are admitted. In addition, when a (set of) job(s) complete earlier, we quantify the generated slack ($CT^{max} - CT$). This slack gradually decreases over time with a slope of -1 and finishes at CT^{max}.

We use this free processor time FPT to schedule optional jobs (with a 0 in the admission sequence). If the FPT exceeds the worst-case execution time of an optional job J, we allocate that time to job J and admit the job. Part of the FPT is then consumed and FPT will be updated as $FPT - C^{max}$.

B. Load distribution of multi-CBS tasks

Due to the context-switching overhead of the CBSs, increasing the number of CBSs wastes resources. Hence, the admitted jobs of the task should be distributed between the minimum number of CBSs with the minimum total utilisation. The output of the presented techniques is a set of CBSs (and the number of full cores in case of multi-CBS tasks).

1) Resource allocation: We propose a similar approach to interface-based design [21] to derive the required amount of resources for multi-CBS tasks. As illustrated in Fig. 2, the arrival curve α of such a task consumes the resources provided by a full core (represented by β_{f_1}). The remaining (unprocessed) workload is represented by α_{r_1}. In the next step, the remaining workload (α_{r_1}) consumes the resources provided by β_{f_2}. This generates a new remaining curve represented by α_{r_2}. This procedure continues until the remaining curve is schedulable on a single core. Let α_{r_k} be the last remaining curve that is schedulable on a single core. We allocate k full dedicated cores to this task and then derive the optimal server parameters for the curve α_{r_k}.

2) Load distribution: The goal of load-distribution is to distribute the jobs of multi-CBS tasks among the allocated CBSs of the corresponding task. We propose a customised load-distribution method for multi-CBS tasks that follows the same strategy at runtime as the resource allocation step uses to respect the $(\mathcal{M}, \mathcal{K})$ constraint and the deadline of any admitted job. We must look for a CBS that can satisfy the job's deadline. This is achieved by comparing the worst-case response time of the job with its deadline on each CBS or core and allocating the job to the first CBS or core that meets the deadline.

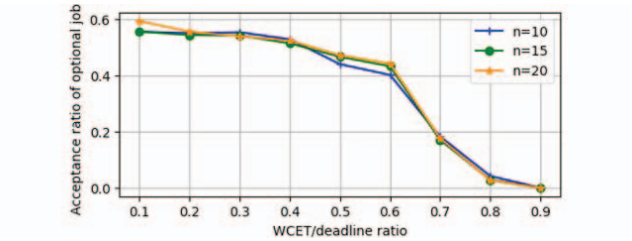

Figure 3: Acceptance ratio of optional jobs per WCET/deadline ratio for $n = 10$, 15 and 20 tasks.

C. Task-set scheduling

Task-set scheduling concentrates on deriving the minimum number of cores (sets of CBSs) required for the task set to be schedulable. The goal is to find the minimum number of cores such that all the CBSs are schedulable. We employ the Worst-Fit Decreasing algorithm [22] to schedule the rest of the CBSs according to the partitioned EDF scheduling policy.

IV. Evaluation

We evaluate (i) the admission policy's performance under non-worst-case job arrivals and execution times, and (ii) the runtime efficiency of our admission test and load-distribution method. We measure the *acceptance ratio* as the average proportion of optional jobs admitted out of the total number of optional jobs across all tasks within a task set. We also measure the runtime of the admission test and load-distribution method for each job, individually, and report the average runtime.

We generated task sets of 10, 15, and 20 tasks across nine configurations (0.1 to 0.9) by generating 50 task sets per point, with deadlines ensuring C^{max}/D matched the configuration. Minimum inter-arrival times ranged from 1 to 5 minutes, maximums were 5 to 10 minutes longer, and worst-case execution times varied between 1 and 10 minutes.

Results. Fig. 3 illustrates the average acceptance ratio for systems (with 10, 15, and 20 tasks) as the ratio of WCET/deadline increases from 0.1 to 0.9. The acceptance ratio decreases (0.6 to 0) as WCET/deadline ratio increases because then it is more likely that the jobs with close arrival times are assigned to different CBSs. In addition, this figure shows that increasing the number of tasks does not impact the acceptance ratio. This is because the admission test of each task is performed independently of the other tasks.

Fig. 4(a) and (b) illustrate the average runtime of the admission test to assess one job as a function of the WCET/deadline ratio and number of tasks, respectively. The average runtime of our admission test is in the order of $450\mu s$, though it slowly increases as the WCET/deadline ratio increases, i.e., from $300\mu s$ to $600\mu s$. Fig. 4(b) shows that increasing the number of tasks has a negligible impact on the runtime of the admission test (around $400\mu s$ on average). This is because the admission test of each task is independent from other tasks.

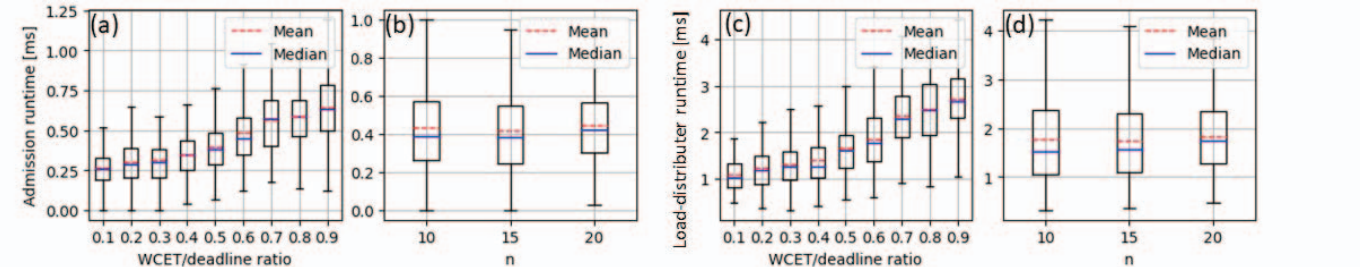

Figure 4: Runtime of the admission test (in milliseconds) as a function of (a) WCET/deadline ratio, (b) number of tasks, and the runtime of load-distribution method (in milliseconds) as a function of (c) WCET/deadline ratio, and (d) number of tasks

Fig. 4(c) and (d) show the average runtime of the load-distribution method. As shown in Fig. 4(c), by increasing the WCET/deadline ratio, the runtime of the load-distribution method increases linearly ($1ms$ to $3ms$). By increasing the WCET/deadline ratio, the number of CBSs increases as the number of tasks that require multi-CBS increases. Therefore, the load-distribution method has to look up the status of more CBSs. Fig. 4(d) shows that increasing the number of tasks has a negligible impact on the processing time of load-distribution (less than $2ms$ on average). This occurs because each task's load distribution is independent of others.

The load-distribution method's higher average runtime compared to the admission test is due to the former requiring computation of the job's WCRT on the CBS and its comparison with the deadline, unlike the latter's simpler sequence for mandatory jobs and comparison between job C^{max} and server FPT. Additionally, the number of cores increases with the WCET/deadline ratio (from 28 to 40 on average) and the number of tasks (from 25 to 50 on average). The number of CBSs grows with the WCET/deadline ratio (from 30 to 40 on average) and task numbers (25 to 50 on average).

V. Conclusion

We presented a resource-management framework that provides practical solutions for guaranteeing deadline requirements of real-time applications in server-based architectures. Our approach ensures the efficient utilisation of resources while preventing admitted requests from missing deadlines. Our framework accommodates multi-CBS applications, allowing for workload distribution across cores. These contributions can pave the way for reliable and efficient real-time systems in remote servers.

We are currently working on proving the optimality of the admission sequence proposal, deriving the admission curve and determining the optimal CBS parameters. Additionally, we will work on ensuring the optimality of resources allocated to multi-CBS tasks and guaranteeing deadlines with the proposed load-distribution method. We plan to expand our work to cloud platforms and employ dynamic resource scaling policies.

VI. Acknowledgement

This work was supported by the EU ECSEL project TRANSACT (grant no. 101007260).

References

[1] J. Lee, J. Wang, D. Crandall, S. Sabanovic, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," *IEEE IRC*, pp. 36–43, 2017.
[2] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE TC*, vol. 38, pp. 1110–1123, 1989.
[3] W. Zhao and K. Ramamritham, "Distributed scheduling using bidding and focused addressing," *RTSS*, pp. 103–111, 1985.
[4] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," *RTSS*, pp. 110–117, 1995.
[5] A. Marchand and M. Silly-Chetto, "RLP: Enhanced QoS support for real-time applications," *RTCSA*, pp. 241–246, 2005.
[6] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE TPDS*, vol. 10, pp. 549–559, 1999.
[7] T. Semprebom, C. Montez, and F. Vasques, "(m,k)-firm pattern spinning to improve the GTS allocation of periodic messages in IEEE 802.15.4 networks," *Eurasip JWCN*, 2013.
[8] L. Niu and G. Quan, "Energy minimization for real-time systems with (m, k)-guarantee," *IEEE TVLSI Systems*, vol. 14, pp. 717–729, 2006.
[9] H. Choi, H. Kim, and Q. Zhu, "Toward practical weakly hard real-time systems: A job-class-level scheduling approach," *IEEE IoT Journal*, vol. 8, pp. 6692–6708, 2021.
[10] G. Bernat and A. Burns, "Combining (m,n)-hard deadlines and dual priority scheduling," *RTSS*, pp. 46–57, 2002.
[11] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, pp. 1443–1451, 1995.
[12] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," *ISCAS*, vol. 4, pp. 101–104, 2000.
[13] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Extending typical worst-case analysis using response-time dependencies to bound deadline misses," *EMSOFT*, 2014.
[14] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks," *IEEE TECS*, vol. 18, 2019.
[15] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," *DATE*, pp. 515–520, 2012.
[16] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst, "Typical worst case response-time analysis and its use in automotive network design," *DAC*, 2014.
[17] L. Ahrendts, S. Quinton, and R. Ernst, "Finite ready queues as a mean for overload reduction in weakly-hard real-time systems," *RTNS*, 2017.
[18] E. Wandeler and L. Thiele, "Optimal TDMA time slot and cycle length allocation for hard real-time systems," *ASP-DAC*, 2006.
[19] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," *RTSS*, 2005.
[20] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," *RTSS*, vol. 0, pp. 99–110, 2016.
[21] L. de Alfaro and T. A. Henzinger, "Interface-based design," *Engineering Theories of Software Intensive Systems*, pp. 83–104, 2005.
[22] S. Baruah, "Partitioned EDF scheduling: A closer look," *Real-Time Systems*, vol. 49, pp. 715–729, 2013.