Work-in-Progress: Evaluating Task Dropping Strategies for Overloaded Real-Time Systems

Yiqin Gao
ENS Lyon, France
yiqin.gao@ens-lyon.fr

Guillaume Pallez
Inria Bordeaux, France
guillaume.pallez@inria.fr

Yves Robert
ENS Lyon, France
Univ. Tennessee Knoxville, USA
yves.robert@ens-lyon.fr

Frédéric Vivien
Inria Lyon, France
frederic.vivien@inria.fr

***Abstract*—This paper discusses evaluation criteria and scheduling strategies for the analysis of overloaded real-time systems. This work builds upon techniques from queueing theory and proposes a new approach for real-time systems.**

***Index Terms*—real-time system, overloaded system, task dropping**

I. Introduction

Problem statement: This work deals with overloaded real-time systems. Specifically, we consider a single periodic task to be processed by a server: task instance number i (or job i), $i \geq 1$, is released at time $r_i = \tau \times (i-1)$ and placed into the waiting queue; it must complete not later than time $d_i = r_i + D$, where τ is the period and D the relative arbitrary deadline (we have $D \geq \tau$). More precisely, if job i is not completed by its deadline d_i, it is considered as failed, and any time spent by the server to execute part of it has been wasted. We further assume that job execution times are stochastic and obey some probability distribution $\mathcal{D}$.

The goal is to optimize some metric. There are many important criteria to guide the scheduling strategies, some user-oriented, and some platform-oriented. We review several of them in Section III. Our major contribution is to use dynamic strategies that take decisions on the fly, and that decide to execute a job, or not, based upon the current system state. These strategies are described in Section IV.

Motivation: Overloaded systems have become the norm in scientific computing. With the advent of parallel computing for many societal advances, traditional computing systems are under a much higher load than was typically the case a few years ago. High-Performance Computing (HPC) systems run with a utilization of almost 100% and a job waiting time of days [18].

In order to cope with this increase in submissions, several approaches are considered. In Cloud computing, the typical approach is to have dynamic pricing, with prices increasing as the resources become more scarce [22]. In HPC, high waiting times are usually a deterrent with important negative consequences, such as limiting the number of very large jobs on HPC platforms [18].

In this work, we focus on real-time systems but borrow the idea from HPC or Cloud computing that when not all jobs can be executed, the server can refuse (or *drop*) some jobs. A key difference with many existing approaches is that we allow the server to take dynamic decisions, i.e., to use up-to-date information about the system state to make the decision of accepting or dropping a job. Such information includes much more than the number of jobs in the waiting queue; e.g., when the server becomes ready, it can accept a job based upon the time spent since its release or upon the time left before its deadline, or both.

Contributions: We review several optimization criteria and discuss new parametrized algorithmic strategies which select which jobs should be dropped and when. We provide ways to compute the optimal value of the parameters for these strategies.

II. Related work

Real-time applications: Real-time jobs are released periodically and must complete successfully before a fixed time-interval called the deadline. In the literature, real-time systems are classified as *hard* or *soft* [1]. For hard real-time jobs, no deadline should be missed: it is mandatory that each job completes before its deadline. On the contrary, for soft real-time jobs, some deadlines can be missed. There are two scenarios then: either a job that missed its deadline must still be completed, or it can be ignored. The latter case is referred to as *firm* real-time jobs: firm jobs are allowed to miss their deadline, and there is no value to complete them after their deadline. Applications of firm real-time jobs include multimedia applications, satellite-based tracking, financial forecast systems, and robotic assembly lines [13], [15].

In the classical setting with firm real-time applications, several periodic tasks are input to the system. Each periodic task is composed of instances (or jobs) that are released with a given period and deadline. The objective is to maximize the number of jobs that successfully complete before their deadline. There are many variations: for instance, some tasks may have two different types, skippable or not [16], and only skippable jobs are allowed to miss their deadline. Our problem is similar to scheduling overloaded systems that allow skips [14]. However, contrarily to some related work we do not impose that at least m of k consecutive jobs must meet their deadlines [11], [21] and we do not allow to modify a task period [2].

In this work, we revisit the problem with firm real-time applications in a new framework. We deal with a single

periodic task and assume that all jobs are skippable, which simplifies the scheduling, but we do not assume that the set of jobs admit a finite worst-case execution time (WCET), which dramatically complicates it. We assume that the job execution times obey some known probability distribution $\mathcal{D}$, and we provide experiments with a wide range of standard distributions. Dealing with such stochastic execution times introduces new challenges: If the support of the distribution $\mathcal{D}$ is unbounded, some jobs may execute for an arbitrarily long duration, thereby putting the following jobs at risk. However, if we decide to interrupt a job to launch a new one, the time already spent to execute it is lost, and there is no guarantee that the new job will complete faster than the interrupted one.

This job model assumes that some jobs may not be executed in the end. In fact, there are three cases: (i) some jobs are launched and reach completion before their deadline, meaning that they are successfully executed; (ii) some jobs are launched, but they are interrupted before completion, either because their deadline is missed or the scheduler has interrupted them, meaning that their execution has failed; and (iii) some jobs are not launched at all.

Queuing networks: In this work, we deal with the list of waiting jobs via a queue: while the machine is occupied, all submitted jobs are waiting for their turn (or waiting to be killed) in a queue. It strongly differs from what is done in queueing theory systems [12] where the typical constraint is to select in which order jobs should be executed to optimize an objective such as response time. Furthermore, job execution times obey the same probability distribution; to select which jobs we execute next we use a simple First-Come-First-Served strategy amongst the jobs that have not been *dropped*.

The closer to our work is the job dropping model [4], where upon arrival of a job, the system selects, at admission, whether the job should be *dropped* (i.e., should be rejected). The decision to drop a job often depends on a function of queue parameters (number of jobs in the queue, average load of the queue): linear function (average queue size) [9] or other more complicated functions [8], [19]. Then, all jobs from the queue are executed (for example following a FCFS strategy). Contrarily to the job dropping model, we propose to drop jobs *dynamically*.

Job pruning in heterogeneous computing systems: In a series of publications [5], [6], [10], [17], [20], job pruning techniques have been investigated. The authors consider an oversubscribed system to which jobs are submitted at random times. When a job is released, it is at first stored in a *batch queue*, and then can be allocated to the *machine queue* of a processor by the mapping process. At each mapping event (completion oor arrival of a job), the success probability of all jobs in the machine queue is recomputed, via a costly convolution over all possible durations of the jobs in the queue weighted by their respective probabilities. Jobs in the batch queue are also considered when there remains a processor with available positions after the previous computation. Jobs with low probability to meet their deadline are dropped from the machine queue and deferred in the batch queue. Deferring jobs means that their assignment to a processor is postponed. In contrast, a job dropped from the machine queue is definitely removed from the system. Contrarily to our problem, there are several job types, several processor types, and job arrival dates are random rather than periodic. This calls for a very costly solution where a whole convolution over a large time window must be recomputed at each mapping event. Some jobs can be dropped after some duration, a strategy also investigated in this work. The striking difference is that with a single job type and periodic releases, we are able to determine the optimal value of the key scheduling parameters once and for all and to apply them on the fly, thereby providing a schedule whose cost is constant and independent of the number of jobs.

III. Optimization criteria

There are two natural important optimization criteria:

Deadline miss ratio [3], [7]: This is a frequently used metric in soft real-time systems where one is only looking for probabilistic guarantee that the deadline miss ratio of a task is below a threshold. Note that this metric is directly related to the *task execution rate* metric, which corresponds to the expected number of jobs executed per time unit.

Utilization [18]: This corresponds to the proportion of time spent doing computation that ends up in a success (i.e., the completion of a job before its deadline).

In addition, we study the following two criteria:

Mean response time of successfully completed jobs [7], [18]: This corresponds to the delay between the release of the job and its completion. When deadlines are enforced, this criterion is less important but does provide qualitative information about the various solutions.

Mean rejection time: This new criterion is important for an overloaded system. Intuitively, it corresponds to the time it takes for the system to inform a user that its job will not be executed. Of course, this criterion must be coupled with another, otherwise the best strategy would be to reject all jobs at submission time.

IV. Scheduling strategies

In all strategies, we execute the jobs admitted in the waiting queue in an Earliest-Deadline-First (EDF) fashion. Because all jobs have the same relative deadlines, this is equivalent to a First-Come-First-Served (FCFS) strategy.

Admission policies: As discussed in Section II, existing strategies in queueing networks focus on an admission policy: at submission, the scheduler decides whether the job is admitted in the queue or whether it is rejected by the system. This has the advantage of providing almost minimal rejection time. Such dropping mechanisms usually consider the size of the queue as input [4]. As baseline strategies, we admit the next job only if the queue contains strictly less than k jobs, where $k = 1$ or $k = 2$. However, when jobs have stochastic nature and are presented with deadlines, we expect such solutions to have sub-optimal performance.

Dynamic dropping mechanisms: The main focus of this work is on strategies with no admission policy (all jobs go in the queue) but with dropping mechanisms. We dynamically drop jobs if conditions are not met. There are three natural approaches:

- S-MAX: Drop the job if it has not started $s_{\max}$ units of time after its release;
- L-MAX: Drop a running job after $l_{\max}$ units of time;
- D-MAX: Drop an unfinished job (running or not) $d_{\max}$ units of time after its release.

Due to lack of space, we only study the performance of the S-MAX strategy in this paper. Our main result relies on a Markov model to characterize the best parameter values:

Theorem 1. *Given a periodic inter-arrival time of jobs whose execution time follow a discrete probability distribution and with identical relative deadlines, we can compute in polynomial time the optimal value for the parameter $s_{\max}$ to maximize either the deadline miss ratio, or the utilization of the system.*

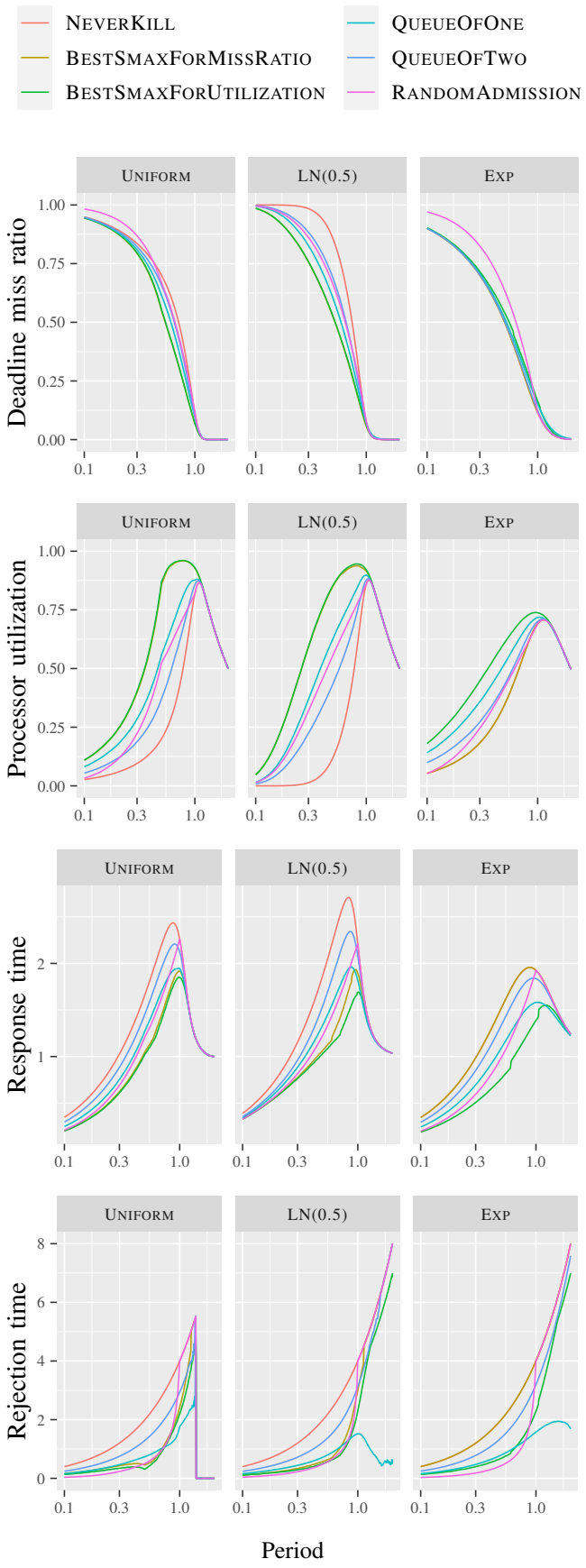
Proof Sketch. We define a quantum duration q, and discretize time into quanta: a period τ has $\frac{\tau}{q}$ quanta, and the server has $S = \frac{D-\tau}{q} + 1$ possible states when job i arrives: it can be available 0, 1, ..., $\frac{D-\tau}{q}$ quanta after the release of the job. Thus, given a parameter set (τ, D and $\mathcal{D}$), we construct a Markov chain with S states, and we compute its transition probability matrix. We show that this Markov chain is irreducible and aperiodic, and compute its limit stationary distribution. Using this limit distribution and the success probability of a job after any given quantum, we determine the asymptotic expected probability that a task is executed successfully. This calculation is done in $O(S^3)$ steps. The best value $s_{\max}$ is computed either by trying all values (adding a factor S to the complexity) or via a binary search (adding a factor $\log(S)$ to the complexity). □

V. EVALUATION

Framework: In order to evaluate the strategies, we propose a simple framework: we use standard probability distribution functions (PDF) to generate job execution times, namely Uniform, Exponential, and LogNormal. All distributions have a mean execution time normalized to 1. Jobs are released periodically with a period $\tau \in [0.1, 2]$, and we have a relative deadline $D = 4\tau$. We measure the average performance via simulation over the execution of 1,000,000 jobs.

We compare six strategies: admission policy with $k = 1$ or $k = 2$ (QUEUEOFONE and QUEUEOFTWO), random admission (RANDOMADMISSION) where the admission ratio is equal to the period (hence, a randomly admitted load of 1), the strategy that admits all jobs and never interrupts any (NEVERKILL), and the two S-MAX strategies that admit all jobs and use the optimal value of $s_{\max}$ either for the deadline miss ratio (BESTSMAXFORMISSRATIO) or for system utilization (BESTSMAXFORUTILIZATION).

Fig. 1. Performance of the different heuristics.



Analysis of the results: We sum up some high-level observations from the results displayed on Figure 1:

- The S-MAX dynamic dropping policy is extremely efficient for utilization under heavy load. This is not surprising as it was designed for this objective. However, S-MAX seems to be quite dominant for all metrics for the distributions under study. More analysis should be done to understand if this is a bias due to the selection of the distributions, and find the limits of its performance.
- The only metric for which S-MAX does not dominate is rejection time. It still has good performance which is correlated to the high-utilization. To have a high utilization, one needs to have few running jobs dropped before they reach their deadline. Hence, the only jobs dropped are those that have not started (this is less true for the Exponential distribution). This balances the fact that the S-MAX policy does not automatically drop jobs at submission. More analysis should be done here to study if one can improve this. In particular, we observe that the RANDOMADMISSION policy has a better rejection time under heavy workloads. This is coherent and hints that we can probably do better by coupling those two algorithms.
- With an overloaded system and deadlines, an admission policy with $k = 1$ is usually more efficient than with $k = 2$, which is consistent with the fact that there is no job shortage. In accordance, the NEVERKILL algorithm (corresponding to the behavior of an unbounded queue, or $k = \infty$) has poor performance.

About the overall performance:

- Deadlines have an important impact on utilization: when the system is overloaded, without deadlines one could expect an utilization of 1 since there is always work to be done. With the best heuristic, we get a utilization close to 1 only with a deadline greater than 2 (corresponding to $\tau \geq 0.5$).
- The high rejection time for underloaded systems correspond to jobs dropped exactly at their deadlines: by construction the deadline increases with the period. This is inevitable because we use unbounded distributions. However, looking at the deadline miss ratio, we see that there are very few jobs that are dropped.

VI. FUTURE WORK

This work has discussed admission and dynamic dropping policies to deal with overloaded service systems. As a future step, algorithmically we plan to study the importance of various parameters for dynamic dropping ($l_{\max}$, $d_{\max}$). Then we believe that combining a posteriori mechanisms with a priori mechanisms will lead to significant improvements.

Indeed, with deadlines, one of the main drawbacks of mechanisms that admit jobs depending on the size of the queue is the following: when a job arrives while another is running, the new job is queued or dropped if the queue is full. Because of deadlines, the longer the wait of a job in the queue, the less likely its success. An alternative approach would be: when a new job arrives and the queue is full, drop the oldest job from the queue and replace it by the newest job. Of course, this comes with a cost for user-oriented objectives.

Acknowledgments: The work of Yiqin Gao was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

We thank the reviewers for their valuable comments.

REFERENCES

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

[2] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings 19th IEEE RTSS*, pages 286–295, 1998.

[3] L.-C. Canon, A. K. W. Chang, Y. Robert, and F. Vivien. Scheduling independent stochastic tasks under deadline and budget constraints. *The Int. J. of High Perf. Computing Applications*, 34(2):246–264, 2020.

[4] A. Chydzinski. Queues with the dropping function and non-poisson arrivals. *IEEE Access*, 8:39819–39829, 2020.

[5] C. Denninnart, J. Gentry, A. Mokhtari, and M. A. Salehi. Efficient task pruning mechanism to improve robustness of heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 2020.

[6] C. Denninnart, J. Gentry, and M. A. Salehi. Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning. In *2019 IEEE IPDPS Workshops*, pages 6–15. IEEE, 2019.

[7] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *23rd IEEE RTSS*, pages 289–300. IEEE, 2002.

[8] C.-W. Feng, L.-F. Huang, C. Xu, and Y.-C. Chang. Congestion control scheme performance analysis based on nonlinear red. *IEEE Systems Journal*, 11(4):2247–2254, 2015.

[9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM T. on Network.*, 1(4):397–413, 1993.

[10] J. Gentry, C. Denninnart, and M. A. Salehi. Robust dynamic resource allocation via probabilistic task pruning in heterogeneous computing systems. In *IPDPS*, pages 375–384. IEEE, 2019.

[11] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, 1995.

[12] M. Harchol-Balter. Open problems in queueing theory inspired by datacenter computing. *Queueing Systems*, 97(1):3–37, 2021.

[13] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, USA, 2nd edition, 2011.

[14] G. Koren and D. Shasha. Skip-over: algorithms and complexity for overloaded systems that allow skips. In *RTSS*, pages 110–117, 1995.

[15] C. Lin, T. Kaldewey, A. Povzner, and S. Brandt. Diverse soft real-time processing in an integrated system. In *RTSS*, pages 369–378, 2006.

[16] A. Marchand and M. Chetto. Dynamic scheduling of periodic skippable tasks in an overloaded real-time system. In *AICCSA*, pages 456–464, 2008.

[17] A. Mokhtari, C. Denninnart, and M. A. Salehi. Autonomous task dropping mechanism to achieve robustness in heterogeneous computing systems. *arXiv preprint arXiv:2005.11050*, 2020.

[18] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari. Job characteristics on large-scale systems: long-term analysis, quantification, and implications. In *SC'20*, pages 1–17. IEEE, 2020.

[19] V. Rosolen, O. Bonaventure, and G. Leduc. A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffic. *ACM SIGCOMM Computer Communication Review*, 29(3):23–43, 1999.

[20] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. Chong, J. Apodaca, L. D. Briceno, T. Renner, V. Shestak, J. Ladd, et al. Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system. *Journal of Parallel and Distributed Computing*, 97:96–111, 2016.

[21] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings 21st IEEE RTSS*, pages 239–248, 2000.

[22] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, 36(13):1381–1419, 2006.