

Controlled Degradation, Not Dispatch Reordering: Semantics-Preserving Overload Control in Lingua Franca

Yutaka Matsubara, Wenhung Kevin Huang, and Akihito Iwai

Abstract—Logical-time reactive languages such as Lingua Franca (LF) make timing part of the programming model, but overload handling is usually left to implementation-level scheduling. This letter formulates *controlled degradation*, which sheds application-declared *degradable* reactions while preserving LF determinism, and asks a focused question: when high-criticality (HC) and low-criticality (LC) reactions are released at the same logical-time tag, can the runtime shed optional LC work, without violating LF semantics, to protect HC timing and bound overload propagation across tags? We give the mechanism as a pressure-driven, per-tag LC budget enforced at the ready-set boundary with semantic guards and safe fallback, argue that shedding a degradable reaction is observationally equivalent to its non-triggering, and show that, once retained per-tag work exceeds capacity, no work-conserving dispatch order can prevent HC misses; only shedding can. On a multi-worker LF/C runtime on a PREEMPT_RT Raspberry Pi at a natural 2 ms period, degradation keeps HC within its deadline (HC-miss falls from 100% to $\approx 0\%$ at the overload onset), bounds the longest run of overrun tags to at most tens (vs. the full trace), and exposes a tunable LC/timeliness trade-off, with LF determinism preserved by construction; the benefit depends on worker parallelism, which we characterize.

Index Terms—Lingua Franca, logical time, reactive systems, mixed criticality, overload, graceful degradation, imprecise computation, real-time scheduling.

I. INTRODUCTION

TIME-CENTRIC reactive software separates *logical time*, which defines program behavior, from *physical* execution timing, which depends on the runtime and platform. This separation supports deterministic behavior even under concurrent or variable execution, and is attractive for cyber-physical systems. Overload, however, remains a practical challenge: when many reactions become ready at the same logical-time tag, the runtime may have freedom to choose a dispatch order, but reordering alone does not reduce the amount of work that

must complete before *backlog* (physical lag relative to logical time) propagates to later tags.

We study this problem in Lingua Franca (LF) [1], [2]. Our starting point is a previously published Master Scheduler (MS) that exposes the LF ready set as a semantics-preserving scheduling interface [3], with guarded intervention, overhead measurements, and an initial degradation demonstration. Controlled degradation is the LF, logical-time instance of a classical idea—shedding *optional* work under overload, as in imprecise computation [4], elastic tasks [5], and weakly-hard models [6]—but with two distinguishing requirements: the shed must preserve LF determinism, and its benefit must be expressed in terms of *logical-time progress*, not just utilization.

The distinction matters for mixed-criticality systems [7], [8], which are increasingly orchestrated with LF in domains such as software-defined vehicles [9]. That work argues, at the architecture level, for orchestrating mixed-criticality *services* in LF and identifies overload as an open challenge; the present letter is complementary and narrower: it specifies and evaluates a concrete *runtime* mechanism—a per-tag, pressure-driven low-criticality (LC) budget at the ready-set boundary—with a semantics-preservation argument and a backlog-based evaluation. If high-criticality (HC) and LC reactions are released synchronously, an HC-first policy can only reorder reactions already in the ready set; it cannot reduce LC work volume and therefore has no leverage once per-tag work exceeds capacity.

This letter makes the following contributions over the prior MS paper [3] (which contributed the ready-set interface and a single overload demonstration): (1) a *semantics-preserving formulation* of degradation, for application-declared degradable reactions, as a pressure-driven, *per-tag* LC budget, with an argument that shedding such a reaction is observationally equivalent to its non-triggering (Sec. III); (2) a *structural impossibility argument* that, once retained per-tag work exceeds capacity (HC and LC sharing the runtime), no work-conserving dispatch order (including HC-first) can prevent HC misses, isolating shedding as the necessary lever (Sec. III); (3) *backlog-propagation metrics* that connect physical overload to logical-time progress across tags (Sec. IV); and (4) a multi-worker evaluation on a PREEMPT_RT Raspberry Pi at a natural 2 ms period: HC protection at the overload onset, bounded backlog, a tunable trade-off, and two pitfalls that silently disable shedding (Sec. III, V).

Y. Matsubara is with the Graduate School of Informatics, Nagoya University, Nagoya, Japan (e-mail: yutaka@ertl.jp).

W. K. Huang and A. Iwai are with DENSO CORPORATION, Kariya, Japan.

Manuscript submitted to TCRS '26 (IEEE Embedded Systems Letters special issue).

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This is the author's accepted manuscript of an article accepted for publication in *IEEE Embedded Systems Letters*. DOI: 10.1109/LES.2026.3731770. The final published version is available on IEEE Xplore at <https://doi.org/10.1109/LES.2026.3731770>.

II. BACKGROUND AND RELATED WORK

An LF program executes reactions at logical-time tags. At each tag, the runtime forms a *ready set* of reactions whose dependencies are satisfied; a scheduler may choose among ready reactions but may not advance or reorder tags in a way that violates LF semantics [1], [2]. The prior MS [3] is a user-space *semantic control plane* at this boundary. Runtime hooks expose each tag's ready set to the MS, which returns an *advisory* pick; a semantic guard admits a pick only if it is a current ready-set member, and on guard failure or a missing pick the runtime falls back to the default LF scheduler. The MS can thus influence intra-tag order and, here, shed degradable reactions, but cannot advance or reorder tags; at worst it degenerates to stock LF scheduling. It provides no OS-level enforcement, and we make no hard real-time claim.

Handling overload by discarding optional work is well studied: imprecise computation drops optional sub-tasks [4], elastic models compress task rates [5], weakly-hard models bound the pattern of misses [6], and skip-over models permit selected task instances to be skipped under overload [10], [11]; mixed-criticality scheduling instead protects HC by mode changes [7], [8]. Our per-tag LC budget is, in this lineage, a skip-over policy applied at logical-time tags. The contribution is not a new scheduling-theoretic result but optional-work shedding *inside a logical-time runtime*: performed at the ready-set boundary, shown to preserve LF determinism, and evaluated by logical-time backlog rather than utilization alone. We are not aware of prior work that frames overload shedding in an LF runtime in terms of bounded backlog propagation across logical-time tags.

III. CONTROLLED DEGRADATION AT THE READY-SET BOUNDARY

System model. At each logical tag g the runtime exposes a ready set R_g . Each reaction is statically classified as high-criticality (HC) or low-criticality (LC); LC reactions whose absence the application admits are marked *degradable*. The MS observes, per dispatch, the runtime lag ℓ (physical minus logical time) and the ready-queue length q , and asserts an overload signal $p = (\ell \geq \theta_\ell) \vee (q \geq \theta_q)$. The control knob is a *per-tag LC budget* b : at most b degradable reactions are admitted per tag while p holds. Reactions are released with a common period P (one tag per period) and executed by m workers; W denotes the total retained execution time of the reactions admitted at one tag (a scalar, not a set) and $C = mP$ the per-tag capacity (instantiated as W_b in Sec. V). The thresholds are sized to react before backlog spans a tag— $\theta_\ell = 300 \mu\text{s}$ ($\approx P/6$) and $\theta_q = 2$ reactions.

Mechanism. Algorithm 1 gives the decision applied to each ready reaction. HC and non-degradable reactions always run; when not overloaded, LC also runs; under overload, the first b LC reactions of the tag run and the rest are shed (removed from R_g and never invoked). A per-tag counter u , reset when the tag advances, enforces the budget independently of period and worker count. Every shed is checked by a semantic guard; if the guard fails the default LF scheduler runs, so no unsafe scheduler intervention is performed (safe fallback).

Algorithm 1: per-tag pressure-driven LC shedding

State: pressure p , admitted count u , current tag g_c .
 ON_METRICS(ℓ, q): $p \leftarrow (\ell \geq \theta_\ell) \vee (q \geq \theta_q)$
 SHOULD_SHED(r, g):
 1: **if** $g \neq g_c$ **then** $g_c \leftarrow g$; $u \leftarrow 0$ (new tag)
 2: **if** $\text{HC}(r) \vee \neg \text{degradable}(r)$ **then return** RUN
 3: **if** $\neg p$ **then return** RUN (no overload)
 4: **if** $u < b$ **then** $u \leftarrow u+1$; **return** RUN (within budget)
 5: **if** $\neg \text{guard}(r)$ **then return** RUN (safe fallback)
 6: remove r from R_g ; **return** SHED

Fig. 1. The decision is $O(1)$ per reaction; under pressure it sheds degradable LC beyond the per-tag budget.

Semantics preservation. *Proposition.* Let r be a reaction the application declares *degradable*, meaning r produces no mandatory output and no other reaction's required behavior depends on r 's outputs or persistent state at tag g (degradability is thus an application-level contract, not inferred by the runtime). Then shedding r at g yields an execution observationally equivalent—at every other reaction and at the program outputs—to one in which r is not triggered at g . *Argument.* The MS removes r before invocation, so r emits no outputs and performs no state update. A reaction influences the program solely through the (tag, value) events it produces, so the event stream seen by every other reaction, the dependency graph, and the advancement of logical time are identical to the case where r 's trigger yields no event at g ; only r 's own optional effects are omitted, and only at shed tags. Guards reject any other intervention. Thus, under this degradability contract, degradation trades *completeness* of optional work for *timeliness* without changing LF semantics. Note that degradable reactions are not statically removable optional code: absent pressure they run and deliver their optional value; shedding is load-adaptive.

Why reordering is insufficient. We assume HC and LC share the same workers and that LF advances tags in order. Dispatch is work-conserving: any ordering of the ready set runs the same total work W per tag. If W exceeds the per-tag capacity C (workers $\times$ period) and remains so (sustained overload), physical time falls behind logical time by at least $W - C$ each tag and backlog grows without bound, *independent of dispatch order*. Because tags are processed in order, an HC reaction, however prioritized within its tag, is eventually released into a saturated pipeline and misses. Therefore, in this regime, no work-conserving policy (including HC-first) can prevent HC misses; only reducing W (shedding) can. This is confirmed empirically: the dispatch-only MS mode tracks the baseline (Sec. V).

Two pitfalls (design insights). The budget must be *per logical tag*, not per fixed time window: a time-window budget shorter than the per-tag execution span resets between sequentially executed reactions and bounds nothing, so shedding silently fails at low worker counts. And shedding must be *pressure-driven*, not conditioned on an HC reaction being simultaneously ready: under sequential execution an HC-first order runs HC before LC, so an “HC-also-ready” guard is rarely satisfied and shedding under-fires precisely when

backlog is building. Per-tag, pressure-driven shedding avoids both.

IV. EXPERIMENTAL SETUP

We use the LF/C threaded runtime with the MS compiled into the reaction-dispatch path. All runs use reactor-c (tag `ms-eval-v1.0`) and `lfc 0.11.0`, with runtime, harness, and raw data archived online [12], run natively on a Raspberry Pi 5 (quad-core Arm Cortex-A76; 64-bit Raspberry Pi OS with a `PREEMPT_RT` kernel and the performance governor). A synthetic workload releases $HC=2$ and $LC=4$ reactions from per-instance timers at a common period P . Each HC reaction busy-waits a fixed time; each LC reaction busy-waits a time scaled by a load factor L , so increasing L raises only optional LC work; synchronized release is the adversarial case for dispatch reordering (it cannot reduce LC volume), isolating shedding as the lever under test. This synchronized release is the workload's critical instant; we assume HC work alone fits within capacity ($n_{HC}c_{HC} \leq C$, satisfied here), so HC is schedulable there once enough LC is shed. HC reactions model *required* behavior (mandatory output every tag, never degradable); LC reactions are *degradable* (self-contained busy-work, no reaction reads their outputs or state), satisfying the contract of Sec. III. A reaction misses if its physical completion exceeds its logical release plus the relative deadline. We compare **baseline** (MS disabled), **MS** (ready-set dispatch, no shedding), and **degrade** (MS with pressure-driven per-tag budget), sweep workers $\in \{1, 2\}$ and load L (the four cores leave headroom for the runtime and MS threads), and report means with 95% confidence intervals over $n=15$ runs per condition.

We set P above the platform's jitter floor so the overload onset is sharp: worst-case scheduling latency on the Pi is $\leq 3\mu s$ (`cyclictst`), and $P=2ms$ is a realistic control period, with proportionally scaled reaction times and deadline. The mechanism itself is period-independent by construction (the budget is per tag), so P sets the time scale, not the qualitative outcome; we revisit this in Sec. VI. Metrics: HC deadline-miss rate and tail latency; LC completion ratio; degradation actions, fallbacks, and MS/runtime pick *mismatches* (the runtime executing a different ready reaction than the MS's pick: a different valid intra-tag order, i.e., mechanism intrusions, not semantics violations); and, per tag, the *tag-overflow ratio* (completion exceeds P) and the *longest overrun run*. All measurements are user-space; the MS provides no OS-level preemption.

V. RESULTS

A. HC protection and worker interaction

Table I and Fig. 2 summarize HC timing. Without overload, every mode meets every HC deadline. Under overload they diverge. In Fig. 2(a) the dispatch-only MS curve lies on top of the baseline at both worker counts and all loads: reordering alone does not help, as the work-conserving argument predicts. Fig. 2(b) shows degradation: at two workers and the overload onset ($L=4$) the baseline misses every HC deadline, while degradation keeps HC-miss near 0% with tail latency close to

TABLE I
HC DEADLINE-MISS RATE AND 95TH-PERCENTILE HC LATENCY, BASELINE VS. DEGRADE, PER WORKER COUNT AND LOAD L . BUDGET $b=2$; DEGRADE RETAINS 50% OF LC IN EVERY ROW; MISMATCHES $< 1/RUN$ (BENIGN); DISPATCH-ONLY MS TRACKS BASELINE (FIG. 2(A)).
BOLD: TWO-WORKER OVERLOAD ROWS.

Workers	L	HC miss (%)		HC p_{95} (ms)	
		Base	Degr	Base	Degr
1	2	100	0.0	186	1.8
1	3	100	99.6	459	53
1	4	100	100	733	190
1	5	100	100	1006	327
1	6	100	100	1279	464
2	2	0	0.0	1.4	0.9
2	3	1	0.0	2.6	1.2
2	4	100	0.0	85	1.4
2	5	100	0.1	222	1.8
2	6	100	1.6	358	3.2

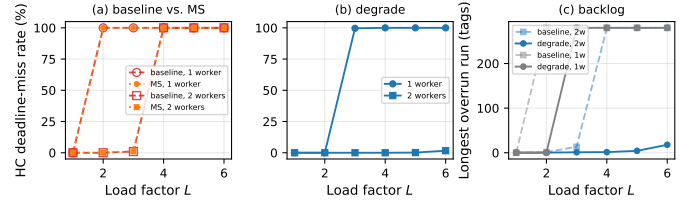


Fig. 2. (a) Baseline (red, dashed) and dispatch-only MS (orange, dotted) coincide: reordering does not reduce per-tag work. (b) Degradation suppresses overload-induced HC misses where capacity permits (clearest at two workers). (c) Longest overrun run: baseline backlog is unbounded under overload; degradation bounds it.

one period (non-overlapping confidence intervals). The benefit depends on worker count: one worker is protected only at $L=2$, because shedding half of the LC work still leaves $W_b > C$ at higher loads; two workers are protected across the entire overloaded range. Table I also reports the HC 95th-percentile latency, which collapses with degradation wherever HC is protected (two workers, $L=6$: $358 \rightarrow 3$ ms) and tracks the backlog otherwise.

The runtime counters also check the semantics argument on every observed execution: the guards never fired (zero fallbacks); no HC or non-degradable reaction was shed (degrade retains exactly the budget fraction, 50%, of LC and 100% of HC); and the only intrusions were < 1 benign pick mismatches per run (a different valid intra-tag order, Sec. III). Every observed schedule is thus one stock LF could have produced with the shed reactions never triggered.

B. Backlog propagation

Fig. 2(c) reports the longest run of overrun tags. Under overload the baseline overruns essentially every tag, so the longest run spans the entire trace—physical time never recovers. Degradation reduces per-tag work below the period, so the longest overrun run collapses from the full trace to single digits near the onset (two workers: ≈ 1 tag at $L=4$) and to ≈ 18 tags at the highest tested load ($L=6$). Degradation does

not merely reorder work; it prevents unbounded backlog from propagating across logical-time tags.

C. Policy (budget) sensitivity

We swept the per-tag budget b at two workers ($L \in \{3, 4\}$). The budget sets the retained LC fraction exactly (b/LC) and trades completeness for overload tolerance monotonically: at the onset ($L=4$) a smaller budget sheds more and protects HC ($b=1,2$, retaining 25%/50% LC, give $\approx 0\%$ HC-miss), whereas $b=3$ (retain 75%) no longer removes enough work and HC misses ($\approx 99\%$). Mismatches stay < 1 per run across the sweep and LF determinism holds by construction, so b is a safe, tunable knob rather than a correctness-affecting parameter.

D. A capacity model for the protected region

The worker dependence follows a simple capacity argument that the data confirm. With m workers, period P , n_{HC} HC reactions of cost c_{HC} , and budget b retaining b LC reactions of cost c_{LC} at load L , the retained per-tag work and the per-tag capacity are

$$W_b = n_{HC} c_{HC} + b c_{LC} L, \quad C = mP. \quad (1)$$

Backlog stays bounded, and HC is protected, when $W_b \leq C$, i.e. for $L \leq L^*(m, b) = (mP - n_{HC} c_{HC}) / (b c_{LC})$. For our parameters ($c_{HC}=0.36$, $c_{LC}=0.24$, $P=2$ ms, $n_{HC}=2$, $b=2$) this gives $L^*(1) \approx 2.7$ and $L^*(2) \approx 6.8$, matching the observed thresholds in Table I (one worker protected only at $L=2$, two workers across $L \leq 6$). The budget sweep agrees: at $L=4$, $m=2$, budgets leaving headroom ($b \leq 2$) protect HC while $b=3$ does not, so the model is a design-time predictor that bounds the feasible region ($L \leq L^*$) before deployment, with b sized for a capacity margin, not merely $W_b < C$. Repeating the two-worker comparison at $P \in \{1, 2, 10\}$ ms (work and deadline scaled with P) gives the same outcome, confirming P -independence; only 1 ms at the extreme load shows a $\approx 10\%$ residual, where dispatch overhead consumes the margin.

VI. DISCUSSION AND LIMITATIONS

The capacity model turns budget selection into a *margin* target: rather than $W_b < C$, one should size b so that $W_b \leq \rho C$ for a safety factor $\rho < 1$ that absorbs dispatch overhead; our data are consistent with $\rho \approx 0.7$ (protection held at $W_b \leq 0.66 C$ but not at $0.9 C$). Two parts are replaceable without touching the semantics argument or safe fallback: *which* LC to shed (Algorithm 1 sheds arbitrary degradable reactions; a value- or priority-aware rule is a drop-in), and the overload detector: our lag/queue signal is a runtime proxy for $W_b > C$, and any monotone pressure signal works. Because shedding is user-space and reduces *offered work*, it is orthogonal to OS-level methods that reallocate *capacity* (RT priority, core partitioning); the two compose.

Runtime vs. application-level shedding. The guard of Algorithm 1 could instead be called at the top of each LC body, returning early under pressure. We shed at the ready-set boundary because the budget is a per-tag, *cross-reaction* quantity: “at most b of this tag’s LC reactions” needs

state shared across reactions and workers plus the runtime’s pressure signal, which no single body sees; boundary shedding also removes work before dispatch, so accounting is exact and applications only declare degradability. The levels compose: application logic (e.g., bounding consecutive skips) can refine *which* LC to shed, as with the replaceable policy above.

Limitations. The results are user-space measurements on one LF/C configuration; absolute latencies and the exact overload onset are platform-dependent, which is why we report worker and load sweeps rather than single points. On physical hardware we confirmed period-independence at $P \in \{1, 2, 10\}$ ms; cyclictst reports worst-case scheduling latency $\leq 3 \mu s$, so the operating point is not jitter-limited. The semantics evidence of Sec. V covers the observed executions; it is not a machine-checked proof. The workload is synthetic with synchronized releases and fixed per-class execution times; production LF applications with data-dependent costs and richer dependencies remain to be evaluated.

VII. CONCLUSION

Controlled degradation (a pressure-driven, per-tag LC budget at the LF ready-set boundary) sheds degradable reactions under overload while preserving LF determinism (shedding is observationally equivalent to non-triggering). Reordering alone cannot prevent HC misses once per-tag work exceeds capacity; only shedding can. On a PREEMPT_RT Raspberry Pi at 2 ms, degradation protects HC at the overload onset, bounds backlog to tens of tags, and gives a tunable LC/timeliness trade-off. Future work: relaxing synchronized release and combining user-space shedding with OS-level placement.

DISCLOSURES

The authors declare no conflict of interest. Claude (Anthropic) assisted with drafting, editing, figure/table generation, scripting, and data analysis. The authors reviewed and edited all AI-generated content and take full responsibility for this publication.

REFERENCES

- [1] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee, “Toward a Lingua Franca for deterministic concurrent systems,” *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 4, pp. 1–27, 2021.
- [2] M. Lohstroh *et al.*, “Reactors: A deterministic model for composable reactive systems,” in *Cyber Physical Systems. Model-Based Design (CyPhy/WESE)*, ser. LNCS, 2019.
- [3] Y. Matsubara, W. K. Huang, and A. Iwai, “A semantics-preserving master scheduler for mixed-criticality control in Lingua Franca,” in *Proc. JSAC Spring Conv.*, 2026.
- [4] J. W. S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung, and W. Zhao, “Algorithms for scheduling imprecise computations,” *Computer*, vol. 24, no. 5, pp. 58–68, 1991.
- [5] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, “Elastic scheduling for flexible workload management,” *IEEE Trans. Computers*, vol. 51, no. 3, pp. 289–302, 2002.
- [6] G. Bernat, A. Burns, and A. Llamas, “Weakly hard real-time systems,” *IEEE Trans. Computers*, vol. 50, no. 4, pp. 308–321, 2001.
- [7] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in *Proc. 28th IEEE Real-Time Syst. Symp. (RTSS)*, 2007, pp. 239–243.
- [8] A. Burns and R. I. Davis, “Mixed criticality systems – a review,” Dept. Comput. Sci., Univ. of York, Tech. Rep., 13th ed., 2022.

- [9] W. K. Huang, Y. Terazawa, Y. Matsubara, and A. Iwai, "Software-defined vehicles: Challenges and orchestrating mixed-criticality services using Lingua Franca," *IEEE Embedded Syst. Lett.*, vol. 18, no. 3, pp. 237–240, 2026.
- [10] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips," in *Proc. 16th IEEE Real-Time Syst. Symp. (RTSS)*, 1995, pp. 110–117.
- [11] S. Baruah and J. Haritsa, "Scheduling for overload in real-time systems," *IEEE Trans. Computers*, vol. 46, no. 9, pp. 1034–1039, 1997.
- [12] Y. Matsubara, "LF master-scheduler evaluation programs and raw data (evaluated runtime pinned as a submodule at `ms-eval-v1.0`), archived on Zenodo," 2026. doi: 10.5281/zenodo.21695545. <https://github.com/ertlnagoya/lf-ms-evaluation>