AI and Data Science Journal
Online ISSN: 3078-7394
DOI: https://doi.org/10.61784/adsj4046

SLO-AWARE CPU RESOURCE ISOLATION AND ADAPTIVE SCHEDULING FOR MIXED WORKLOADS IN CLOUD DATABASE SYSTEMS

YiHan Wang[1*], SuChuan Xing[2]
[1]*University of Pennsylvania, Philadelphia, PA, USA.*
[2]*Duke University, Durham, NC, USA.*
[*]*Corresponding Author: YiHan Wang*

Abstract: Cloud database services increasingly co-locate latency-critical transactional (OLTP) requests and throughput-oriented analytical (OLAP) queries on the same virtual cores, which makes CPU isolation a first-order determinant of both tail latency and utilization. We first quantify this tension on a real testbed: when TPC-H queries on DuckDB share a core with a TPC-C-like SQLite workload under the default Linux fair scheduler, OLTP p99 latency grows by 5.4–7.6×, whereas the SCHED_IDLE class preserves the tail but lengthens transactions that start on a just-harvested core by 15% and lowers analytical efficiency to 0.87. Building on these measurements, we propose SLICE, an SLO-aware CPU isolation and adaptive scheduling framework that selects, every second, the combination of dedicated OLTP cores, dedicated OLAP cores and idle-class harvesting that maximizes analytical capacity subject to a calibrated p99 bound. SLICE couples an M/G/k tail-latency model with online interference estimation and damped load forecasting, and wraps the model in a Mondrian conformal PID calibrator that drives the long-run frequency of bound violations to a target risk budget; a 100-ms guard absorbs residual bursts. Trace-driven experiments with three Azure production request traces, calibrated by the testbed measurements, show that SLICE keeps SLO-violating seconds at 0.11–0.67%, the lowest among seven policies, while delivering 2.01× the analytical throughput of peak-provisioned partitioning and 26% more than a PARTIES-style controller that violates the SLO 2.2× as often.
Keywords: Cloud databases; HTAP; Performance isolation; Tail latency; CPU scheduling; Conformal prediction

1 INTRODUCTION

Cloud database services consolidate heterogeneous work on shared hardware to keep costs low. Hybrid transactional/analytical processing (HTAP) engines execute OLTP transactions and OLAP queries over fresh data within one system [1-3], and production analyses show that the load of cloud database instances varies strongly over time in both volume and type [4]. On every node of such a service, the CPU is the resource on which the two classes collide most directly. Transactions are short, latency-critical and judged by their tail latency [5], while analytical queries are long, parallel, throughput-oriented and able to absorb any spare cycle.

Existing mechanisms resolve this tension only partially. Engine-level reservations for database-as-a-service guarantee CPU shares but do not target tail-latency objectives [6-7]. Static core partitioning protects transactions but strands cores whenever the transactional load is below its peak. Proportional sharing under the Linux fair scheduler keeps cores busy but lets analytical threads delay transactional wake-ups by entire time slices [8]. Feedback controllers such as Heracles and PARTIES resize partitions after latency has degraded [9-10], which is too late when load surges within a second, and quota-based control as in Autothrottle inherits the wake-up delays of fair sharing [11]. Priority-based harvesting with the SCHED_IDLE class removes most of these delays, yet a transaction that starts on a just-harvested core runs with cold caches, and the analytical footprint keeps competing for shared caches and memory bandwidth.

This paper argues that the right isolation mechanism depends on the operating point—load level, SLO tightness and interference strength—and that a controller should choose among mechanisms continuously, with an explicit and verifiable bound on the risk it takes. Real measurements on a TPC-C-like/TPC-H co-location testbed (Section 2) support three observations: (i) fair sharing inflates OLTP p99 by 5.4–7.6×; (ii) idle-class harvesting keeps the tail close to isolation on a lightly loaded core but costs 15% longer execution after each harvested interval and 13% of the harvested cycles' analytical value; and (iii) queueing amplifies these per-request costs sharply near saturation, so a mechanism that is safe at night can violate the SLO at the daily peak.

We present SLICE (SLO-aware Isolation with Conformal Estimation), a CPU isolation and adaptive scheduling framework for mixed database workloads. Every second, SLICE forecasts the next-second transactional load, evaluates a queueing-theoretic tail-latency model for candidate allocations of dedicated OLTP cores, dedicated OLAP cores and harvested cores, and applies the allocation that maximizes analytical capacity among those whose calibrated p99 bound meets the SLO. The calibration layer adapts conformal PID control to closed-loop resource allocation: it scores the realized p99 of the allocation that was actually applied, calibrates separately per utilization regime, and drives the long-run fraction of periods whose p99 exceeds the bound to a user-chosen risk budget ε without distributional assumptions [12]. A 100-ms guard escalates the allocation when a burst escapes the forecast. The main contributions are as follows.

© By the Author(s) 2024, under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0).

1) A measurement study on a real testbed that quantifies wake-up delays, cold-start inflation and analytical efficiency under fair sharing and idle-class harvesting, and a validated trace-driven node simulator calibrated with these measurements.
2) A risk-constrained formulation of CPU isolation that jointly selects partitions, harvesting and analytical throttling, built on an M/G/k tail model with online interference estimation.
3) A Mondrian conformal PID calibrator for closed-loop allocation with a long-run guarantee on bound violations, combined with a two-timescale control loop.
4) An evaluation with three Azure production request traces against six baselines [13], including sensitivity, ablation and calibration studies.
Trace-driven experiments with three Azure production traces show that SLICE has the lowest rate of SLO-violating seconds of seven policies on every trace (0.11%, 0.67% and 0.28%), delivers 2.01× the analytical throughput of peak-provisioned partitioning, and improves analytical throughput by 26% and 17% over PARTIES-style and Heracles-style controllers while reducing their violating seconds by 55% and 94%. The calibrated bound behaves as designed: its realized violation frequency stays at or below the risk budget, which gives operators an interpretable dial between transactional safety and analytical efficiency.

2 BACKGROUND AND MOTIVATION

2.1 Mixed Workloads on Cloud Database Nodes

A cloud database node that serves both classes hosts two kinds of CPU work. Transactions consume tens to hundreds of microseconds of CPU in an in-memory engine, arrive in an open-loop fashion from many client sessions, and are governed by a p99 latency service-level objective (SLO). Analytical work—reports, dashboards, extract-transform-load jobs and ad-hoc queries—consists of CPU-heavy, highly parallel queries whose throughput determines the value extracted from the node. Mixed benchmarks such as the CH-benCHmark run both classes concurrently on the same data [14], and modern engines execute analytical pipelines elastically on whatever cores they are given.

2.2 CPU Isolation Mechanisms

Cloud database nodes typically rely on Linux control groups, which expose four mechanisms used by SLICE and the baselines: (i) cpuset partitions pin each class to disjoint cores; (ii) cpu.weight shares cores proportionally under the fair scheduler, which implements EEVDF since Linux 6.6 [15]; (iii) cpu.max caps a class at a CPU quota per enforcement period; and (iv) cpu.idle places a class under the SCHED_IDLE policy, so that it runs only when no normal-priority thread is runnable, the basis for harvesting idle cores safely [16]. Kernel-bypass runtimes reallocate cores at microsecond granularity [17-18], but they require applications to adopt a custom threading runtime and are therefore not available to unmodified database engines.

2.3 Testbed Measurements

Setup. We measure the per-core cost of each sharing mode on a 1-vCPU cloud virtual machine (Intel Xeon at 2.3 GHz, 3 GB memory) running a recent Linux kernel with the EEVDF scheduler. The transactional workload is a TPC-C-like implementation on in-memory SQLite (2 warehouses, 10 districts per warehouse, 1,000 customers per district, 50,000 items) with the standard NewOrder/Payment/OrderStatus/Delivery/StockLevel mix of 45/43/4/4/4%. A single worker serves an open-loop Poisson stream in FCFS order and records arrival, dispatch and completion times of every transaction. The analytical workload runs the 22 TPC-H queries at scale factor 1 (6.0 M lineitem rows, generated with the official dbgen) back-to-back on DuckDB with one thread [19]; their single-thread CPU demands range from 9 to 199 ms (1.37 s per stream). Each configuration runs for 45 s.

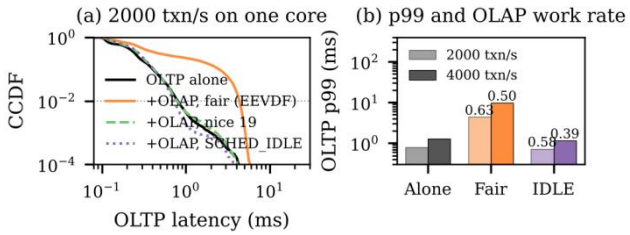


Figure 1 Real Co-Location Measurements on One Core

Note: (a) Latency CCDF of TPC-C-like transactions at 2,000 txn/s when TPC-H runs on the same core under different policies. (b) OLTP p99 at 2,000 and 4,000 txn/s; labels give the analytical work rate (single-thread TPC-H work per second; 1 when running alone).

Findings. Figure 1 and Table 1 summarize the results. Under the fair scheduler, OLTP p99 grows from 0.8 to 4.36 ms at 2,000 txn/s and from 1.28 to 9.75 ms at 4,000 txn/s, because a woken transactional worker waits for the analytical slice to expire (p99 excess dispatch delay 3.74 ms) and keeps losing slices during busy periods. Lowering the analytical priority

restores the tail: 0.78 ms with nice 19 and 0.72 ms with SCHED_IDLE at 2,000 txn/s, and 1.15 ms with SCHED_IDLE at 4,000 txn/s. Harvesting is not free, however. A transaction that starts on a core that has just run analytics executes 15% longer because of cold caches, and the analytical thread converts a harvested CPU-second into only 0.87 s of useful work. These per-request costs are small on a lightly loaded core, but queueing amplifies them near saturation, and at node scale the analytical footprint also interferes through shared caches and memory bandwidth [9, 20]. No single mechanism is therefore best across load levels and SLOs.

Table 1 Per-Core Cost of Sharing Measured on the Testbed

Quantity	Fair (EEVDF)	SCHED_IDLE
OLTP p99 @ 2k txn/s (alone 0.8 ms)	4.36 ms	0.72 ms
OLTP p99 @ 4k txn/s (alone 1.28 ms)	9.75 ms	1.15 ms
Exec. inflation after analytics ran (cold)	1.2×	1.15×
Exec. inflation, back-to-back (warm)	1.04×	1.01×
p99 excess dispatch delay (cold / warm)	3.74 / 2.99 ms	0.03 / 0.01 ms
Analytical efficiency of shared cycles	0.93	0.87

3 SLICE DESIGN

3.1 Overview and Problem Formulation

Figure 2 shows the architecture. SLICE runs as a user-space agent next to the database engine and controls three knobs through cgroups: the number k of cores in the OLTP cpuset, the number a of cores in the OLAP cpuset, and a harvesting flag h that, when set, places OLAP threads in the idle class on the OLTP cores so that they absorb cycles left idle by transactions. The remaining $C - k - a$ cores stay idle, which throttles the analytical footprint when interference dominates. Decisions are taken at two timescales: a 1-s decision period for model-based allocation and a 100-ms guard.

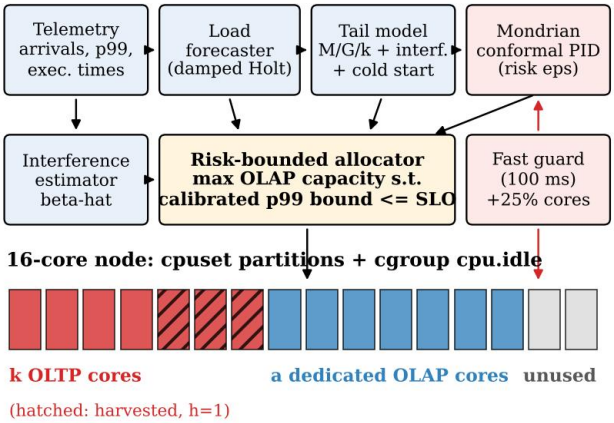


Figure 2 SLICE Architecture

Note: A risk-bounded allocator chooses (k, a, h) every second from a calibrated tail-latency bound; a 100-ms guard escalates the allocation when a burst escapes the forecast.

Let λ_t denote the OLTP arrival rate in period t, m the mean transactional CPU demand, L the p99 latency SLO, and $x_t = (k_t, a_t, h_t)$ the allocation. With η_h the analytical efficiency of harvested cycles, the analytical capacity of an allocation is

$$\Gamma(x) = a + h\,\eta_h\,(k - \hat{\lambda}\,m)^+, \tag{1}$$

and SLICE solves in every period

$$\max_{x} \Gamma(x) \;\text{ s.t. }\; U_t(x) \le L,\; k + a \le C, \tag{2}$$

where $U_t(x)$ is a calibrated upper bound on the p99 latency that x will produce in period t. The difficulty lies in making U_t trustworthy under non-stationary load and model error; Sections 3.2 to 3.4 construct it.

3.2 Tail-Latency Model

SLICE models the OLTP partition as an M/G/k queue whose service times are inflated by cross-core interference and, on harvested cores, by cold starts. Interference grows with the analytical footprint

$$\iota(a_e) = 1 + \beta\,a_e / C,\;\; a_e = a + h\,(k - \lambda\,m)^+. \tag{3}$$

For the empirical service-time distribution S (offline profile), the mean effective service time is $\bar{m} = \iota\, m\,[P_W + (1 - P_W)\, f_h]$, where P_W is the Erlang-C waiting probability of k servers at offered load $\lambda\ \bar{m}$ and f_h the cold-start factor ($f_h = 1$ when $h = 0$) [21]. Following the Allen–Cunneen approximation of the M/G/k queue, the conditional wait is exponential with rate

$$\theta = (k / \bar{m} - \lambda) \cdot 2 / \left(1 + c_S^2\right), \tag{4}$$

where c_S^2 is the squared coefficient of variation of S. The response-time tail mixes requests that start immediately and pay the cold-start factor with requests that wait on a warm core:

$$\mathrm{P}(R > x) = (1 - P_W)\,\mathrm{P}(f_h\,\iota S > x) + P_W\,\Psi(x), \tag{5}$$
$$\Psi(x) = \mathrm{E}\big[\mathbf{1}\{\iota S > x\} + \mathbf{1}\{\iota S \le x\}\,\mathrm{e}^{-*\theta*(*x* - *\iota S*)}\big].$$

The predicted p99, $\hat{p}_{99}(\lambda, x)$, solves $\mathrm{P}(R > x) = 0.01$ by bisection over a 384-point quantile summary of S.

3.3 Online Estimation and Forecasting

Interference. The slope β is unknown and hardware-dependent. Every 100 ms, SLICE observes the mean execution time $\bar{y}$ of back-to-back (warm) transactions together with the analytical footprint a_e of the previous tick, and estimates β by least squares with the intercept pinned to the offline mean,

$$\widehat{\beta} = \Sigma_\tau\, w_\tau\, u_\tau\, v_\tau \,/\, \Sigma_\tau\, w_\tau\, u_\tau^2, \;\; u = a_e \,/\, C, \;\; v = \bar{y}\,/\,m - 1, \tag{6}$$

with exponential forgetting ($w_\tau = 0.999^{*t*-*\tau*}$) and clipping to 0–2. Pinning the intercept avoids the covariance wind-up that a two-parameter recursive least-squares estimator suffers when the footprint varies little. *Load.* The next-period peak arrival rate is forecast with Holt's linear smoothing (level weight 0.5, trend weight 0.2) applied to the per-period peaks of 100-ms rates. The forecast is bounded below by the last observed peak and capped at 1.5× that peak, which prevents trend overshoot after ramps.

3.4 Mondrian Conformal PID Calibration

Model (5) is approximate and the forecast is uncertain. Instead of hand-tuned safety margins, SLICE calibrates the model online. After each period it computes the score

$$s_t = \log\big(\,\mathrm{p99}^{\mathrm{obs}}_{*t*} \,/\, \hat{p}_{99,*t*}\,\big), \tag{7}$$

where $\hat{p}_{99,*t*}$ is the prediction for the allocation applied in period t, evaluated with the forecast used at decision time, and $\mathrm{p99}_{*t*}^{\mathrm{obs}}$ is measured on the ticks governed by that allocation (up to the first guard intervention). Model error depends on the operating regime, so scores are partitioned in the Mondrian style into bins b by predicted OLTP-core utilization $u = \hat{\lambda}\, m \,/\, k$, with boundaries {0.5, 0.65, 0.8} [22]. Each bin runs a conformal PID controller [12, 23]:

$$q_b \leftarrow q_b + \eta\,(err_t - \varepsilon), \;\; err_t = \mathbf{1}\{s_t > o_t\}, \tag{8}$$
$$o_b = \hat{s}_b(z_t) + q_b + K_I \tanh\big(\,\Sigma_{\tau \in \mathrm{W}}\,(err_\tau - \varepsilon)\,/\,S_I\,\big), \tag{9}$$

where ε is the risk budget, $\hat{s}_b$ is a recursive-least-squares scorecaster on features z_t (relative load trend, intra-period burstiness, harvesting flag and k/C), and the integral term uses a window W of 200 periods. Offsets are made monotone in utilization, $o_b \leftarrow \max_{*b'*\le *b*} o_{b'}$, and the calibrated bound in (2) is $U_t(x) = \hat{p}_{99}(\hat{\lambda}, x)\,\exp(o_{b(x)})$.

Proposition 1. If scores are bounded, $|s_t| \le B$, and $|o_b - q_b| \le B'$, then for every bin with T_b scored periods, $|(1/T_b)\,\Sigma_t\, err_t - \varepsilon| \le (B + B' + \eta)\,/\,(\eta\, T_b)$. Consequently, the long-run frequency of bound violations equals ε, and the long-run frequency of SLO violations among periods governed by a feasible SLICE decision is at most ε.

Proof sketch. Summing (8) over a bin gives $q_b^{(*T*)} - q_b^{(0)} = \eta\,\Sigma_t(err_t - \varepsilon)$. If $q_b > B + B'$, then $o_b > B \ge s_t$, so $err_t = 0$ and q_b decreases; symmetrically, q_b increases when $q_b < -B - B'$. Hence q_b stays in $[-B - B' - \eta,\, B + B' + \eta]$, which bounds the sum [12, 23]. An SLO violation implies $\mathrm{p99}^{\mathrm{obs}} > L \ge U_t$, i.e., $err_t = 1$. Scores are bounded because latencies are recorded on a finite histogram (1 µs to 10 s), and B' is finite because the scorecaster output and the integral term are bounded. ∎

3.5 Allocation and Fast Guard

Algorithm 1 summarizes one decision. For each harvesting mode $h \in \{0, 1\}$ and analytical throttle $j \in \{0, \ldots, 3\}$ idle cores, SLICE finds the smallest k that satisfies the calibrated bound by binary search; $\hat{p}_{99}$ decreases in k because every added OLTP core removes an analytical core. With harvesting, it also keeps the OLTP-core utilization below $u_H = 0.65$, since a harvested OLTP core costs only the fraction $1 - \eta_h$ of its analytical value. Among the candidates SLICE applies the allocation with the largest Γ; if none is feasible it reverts to full protection ($k = C$, $a = 0$, $h = 0$). At most two cores are released per period, while acquisitions take effect immediately.

Algorithm 1. SLICE decision in period t (1 s)

Score the previous allocation on its pre-guard ticks with (7); update its bin with (8)–(9).
Update $\hat{\beta}$ with (6); forecast $\hat{\lambda}$ (peak-aware damped Holt); compute monotone offsets o_b.
  for h ∈ {0, 1} and j ∈ {0, …, 3} do
    $k^\star \leftarrow \min\{\, k : \hat{p}_{99}(\hat{\lambda}, k, C - k - j, h)\, e^{*o_-\{b(k)\}} \le L \,\}$ (binary search)
    if h = 1 then $k^\star \leftarrow \max(k^\star, \lceil\, \hat{\lambda}\, m \,/\, u_H \,\rceil)$
    $\Gamma \leftarrow (C - k^\star - j) + h\,\eta_h\,(k^\star - \hat{\lambda}\, m)$; keep the candidate with the largest Γ
  if no candidate is feasible then (k, a, h) ← (C, 0, 0)
  $k \leftarrow \max(k, k_{t-1} - 2)$; apply cpuset and cpu.idle settings
Guard (every 100 ms): if $\mathrm{p99}_{\mathrm{tick}} > L$ or $Q > 4k$ then $k \leftarrow \min(C, k + \lceil\, k/4 \,\rceil)$; h ← 0 if persistent; freeze releases for 2 periods

Fast guard. Every 100 ms, SLICE checks the p99 of the last tick and the OLTP queue length Q. If p99 > L or $Q > 4k$, it adds $\lceil\, k/4 \,\rceil$ OLTP cores at once, disables harvesting if the condition persists for two consecutive ticks, and freezes core

releases for two periods. Because the score (7) only uses the ticks before the first guard intervention, guard actions do not hide model errors from the calibrator.

Complexity. A decision evaluates at most $2 \times 4 \times (\lceil \log_2 C \rceil + 1)$ tail models, each with 32 bisection steps over 384 service quantiles. The mean decision time of our implementation is 1.67 ms, negligible relative to the 1-s period.

4 EXPERIMENTAL METHODOLOGY

4.1 Trace-Driven Node Simulator

Because the testbed exposes a single core, node-level policies are evaluated with an event-driven simulator of a 16-core database node, calibrated with the measurements of Section 2. OLTP requests arrive as a non-homogeneous Poisson process, wait in one FCFS queue and are dispatched to the most recently released idle OLTP core. Every core carries a sharing mode: dedicated, harvested under SCHED_IDLE, or fair-shared, optionally under a quota. Dispatches on shared cores draw the excess dispatch delay from the measured empirical distributions and multiply execution by the measured cold or back-to-back factors; on fair-shared cores the transactional worker additionally yields one measured analytical slice after every 4 ms of CPU, i.e., one tick at 250 Hz. Cross-core interference follows (3) with a ground-truth $\beta = 0.3$ by default, which SLICE does not know and which Section 5.3 varies. The analytical side runs backlogged TPC-H streams as malleable, morsel-driven work with the measured single-thread demands [24], progressing at rate 1 on dedicated cores and at the measured efficiency on shared cycles.

Fidelity. We replayed the single-core testbed configurations in the simulator (Figure 3). For the alone and SCHED_IDLE modes, the simulated p50, p95 and p99 deviate from the measurements by 7.4% on average. For fair sharing the deviation is 17.6%: the simulator is accurate at 2,000 txn/s and pessimistic at 4,000 txn/s (p99 of 12.5 vs. 9.75 ms). The fair-sharing baselines violate a sub-millisecond SLO by a wide margin already in the accurately modeled regime, so this bias does not affect their ranking.

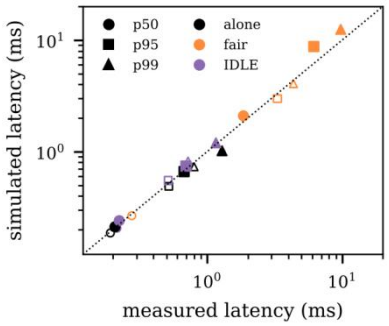


Figure 3 Simulator Fidelity: Simulated vs. Measured Latency Percentiles for the Single-Core Testbed Configurations (Hollow: 2,000 txn/s; Filled: 4,000 txn/s)

4.2 Workloads and Traces

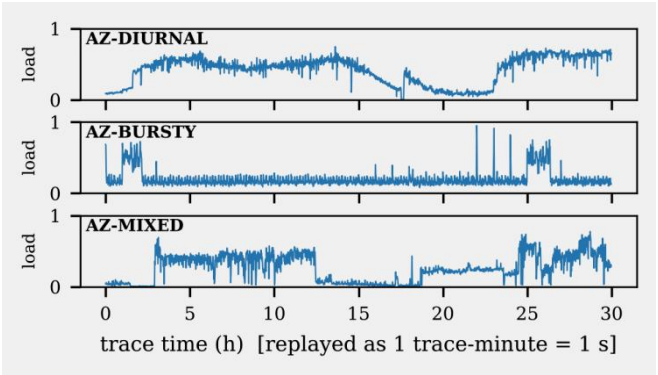


Figure 4 OLTP Load Profiles Derived from Three Always-On HTTP Applications of the Azure Functions 2019 Trace (Fraction of Node Capacity at ρ = 0.7)

OLTP load is driven by the Azure Functions 2019 trace [13], which records per-minute invocation counts of every function over 14 days. We use the HTTP-triggered applications of days 1–4 (8,844 applications, 5,760 minutes), which carry user-facing request/response traffic, and select three always-on applications with explicit criteria. AZ-DIURNAL is the application with the strongest smooth diurnal swing (hourly p95/p5 ratio 19.5, detrended CV 0.14); AZ-BURSTY has the largest minute-scale surges (peak-to-mean ratio 4.73 in the window); and AZ-MIXED combines a diurnal swing with noisy bursts (detrended CV 0.59). Each profile is a 30-h window (1,800 minutes starting at 06:00 of day 1) normalized to its 99.5th percentile (Figure 4). A profile value of 1 corresponds to a peak OLTP load of $\rho = 0.7$ of the node, i.e., 103.6 ktxn/s with the measured mean service time of 0.108 ms. One trace minute is replayed per simulated second, so every run

covers 1,800 s and minute-scale surges become second-scale bursts, a deliberately adversarial setting for controllers. Transactions draw their service times from 256,397 executions measured on the testbed (mean 0.108 ms, SCV 1, p99 0.53 ms).

4.3 SLO, Metrics and Baselines

Following prior colocation studies [9-10], the SLO derives from the tail latency of the transactional workload running alone at peak load: the isolated p99 at $\rho = 0.7$ on 16 cores is 0.532 ms, and we set $L = 0.7$ ms (1.32×). We report (i) SLO-violating seconds, the fraction of 1-s windows whose p99 exceeds L; (ii) the overall p99; (iii) OLAP throughput as the fraction of node capacity turned into useful analytical work; and (iv) effective machine utilization (EMU), the useful OLTP plus OLAP work divided by node capacity [9]. The first 5 s of every run are excluded; the main comparison uses two seeds and the sweeps one seed.

Six baselines run in the same simulator. *Shared* places both classes on all cores under the fair scheduler. *Idle-Harvest* runs OLAP in the idle class on all cores (work-conserving priority). *Static-Peak* is a cpuset partition sized offline, with full knowledge of the trace, as the smallest k whose modeled p99 at the 99th-percentile load meets L. *Heracles-style* uses a 1-s period [9], disables analytics on a violation (5-s cool-down) or above 85% load, reclaims two cores when the slack is below 5% and grants one core when it exceeds 10%. *PARTIES-style* uses a 0.5-s period [10], adds one core on a violation and then freezes down-sizing for 10 s, and releases one core when p99 < 0.8L. *Autothrottle-style* runs a contextual ε-greedy bandit with decaying exploration that chooses an OLAP CPU quota in {0, 2, …, 16} cores for each of four load contexts every second [11], rewarding analytical share minus a violation penalty, while OLAP shares all cores under the fair scheduler subject to the quota. SLICE uses $\varepsilon = 0.1$, $\eta = 0.1$, $K_I = 0.1$, $S_I = 5$ and $u_H = 0.65$ unless stated otherwise.

5 EVALUATION

5.1 Overall Comparison

Table 2 Main Results at the Default Operating Point: SLO-Violating Seconds (%) and OLAP Throughput (% of Node)

Policy	DIURNAL		BURSTY		MIXED		EMU
	Viol.	OLAP	Viol.	OLAP	Viol.	OLAP	(%)
Shared	100	53.6	100	67.5	100	62.9	88.0
Idle-Harvest	28	43.5	91	64.3	56	58.7	84.8
Static-Peak	0.3	18.8	1.6	25.0	0.3	18.8	50.2
Heracles-style	7.9	26.3	4.0	44.5	5.7	36.0	65.0
PARTIES-style	0.6	27.9	0.9	33.4	0.8	38.7	62.7
Autothrottle-style	53	22.5	9.9	8.6	15	5.5	40.0
SLICE	**0.1**	**34.1**	**0.7**	**54.0**	**0.3**	**37.4**	**71.2**

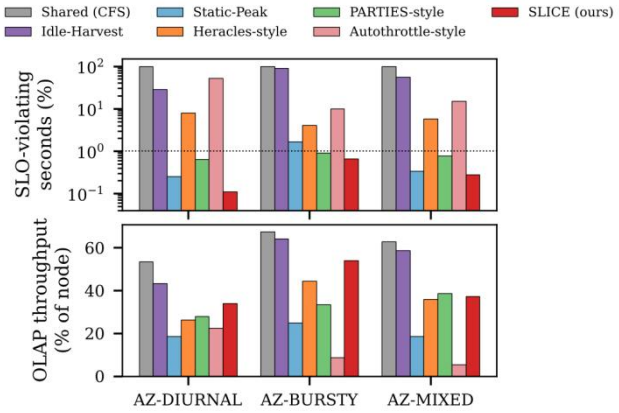


Figure 5 SLO-Violating Seconds (Top, Log Scale; Dotted Line: 1%) and OLAP Throughput (Bottom) of All Policies on the Three Traces

Table 2 and Figure 5 summarize the comparison at the default operating point. The two fair-sharing configurations violate the 0.7-ms SLO in most seconds: Shared misses it in every second because transactional wake-ups wait for analytical slices, and Autothrottle-style, which rations the analytical quota but still shares cores, violates it in 9.9–53.0% of the seconds while its bandit pushes the quota toward zero (12.2% OLAP throughput on average). Idle-Harvest attains the highest OLAP throughput but violates the SLO in 28.1–90.6% of the seconds: at a 0.7-ms SLO the cold-start inflation of every transaction that lands on a harvested core, compounded by cross-core interference, lifts the p99 above the target whenever the load is moderate or high. Static-Peak meets the SLO in most seconds (0.25–1.64%), but provisioning for the 99th-percentile load keeps 12–13 cores reserved for transactions and limits OLAP throughput to 18.8–25.0% of the node.

Among the adaptive controllers, Heracles-style violates the SLO in 4.0–7.9% of the seconds because its slack-driven, one-core-at-a-time growth cannot follow second-scale surges, and PARTIES-style violates it in 0.64–0.92% of the seconds because it learns the required allocation only after a violation. SLICE achieves the lowest violation rate on every trace, 55% fewer violating seconds than PARTIES-style and 94% fewer than Heracles-style on average, while delivering 2.01× the OLAP throughput of Static-Peak, 26% more than PARTIES-style and 17% more than Heracles-style. The advantage is largest on AZ-BURSTY (+62% OLAP throughput over PARTIES-style), where model-based allocation returns cores to analytics within seconds after a surge, and smallest on AZ-MIXED, where SLICE matches the throughput of PARTIES-style (37.4% vs. 38.7%) with 64% fewer violating seconds. Overall, SLICE raises the effective machine utilization from 50.2% (Static-Peak) and 62.7% (PARTIES-style) to 71.2%.

5.2 Behavior under Bursts

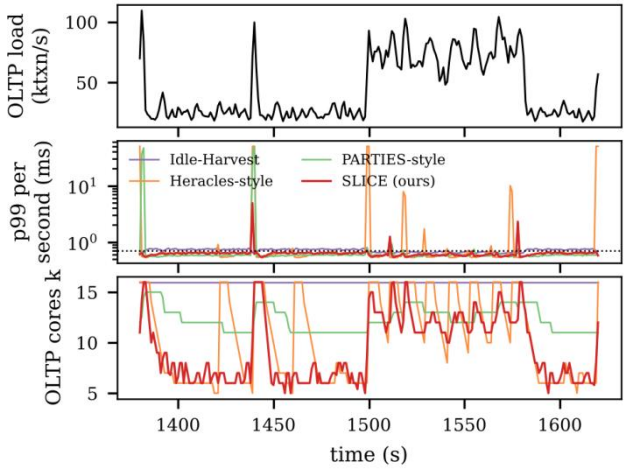


Figure 6 Load Surge on AZ-BURSTY: OLTP Load, Per-Second p99 (Dotted: SLO) and OLTP Cores for Four Policies

Figure 6 zooms into the largest surge of AZ-BURSTY: at t = 1500 s the OLTP load jumps from about 28 to 104 ktxn/s and fluctuates at that level for about 81 s. During the surge and the 5 s that follow, Heracles-style violates the SLO in 10 seconds with a peak p99 of 172.8 ms, because it grows the OLTP partition one core at a time after the backlog has formed. Idle-Harvest violates it in 6 seconds although all cores are available to transactions, since the cold-start and interference costs persist at high utilization. PARTIES-style violates it in 4 seconds (peak 0.93 ms) and SLICE in 2 seconds (peak 2.30 ms); in SLICE the guard enlarges the OLTP partition within 100 ms of the first slow tick and the forecaster keeps it sized for the plateau. The two controllers differ after the surge: SLICE releases cores at two per period and returns to 6 OLTP cores, whereas PARTIES-style still holds 11 cores 30 s later because its down-sizing is frozen after violations, which explains most of the OLAP-throughput gap between the two on this trace (Table 2).

5.3 Sensitivity

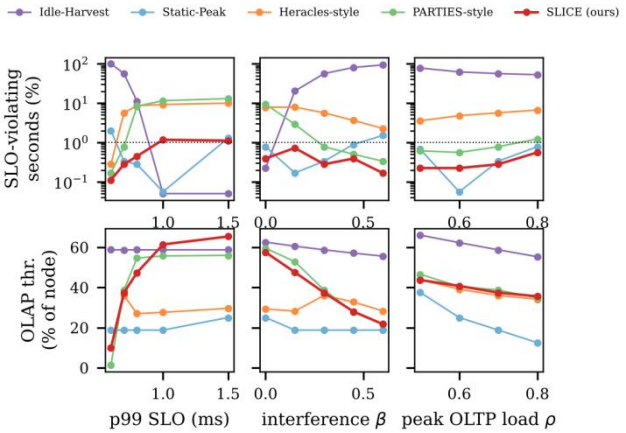


Figure 7 Sensitivity on AZ-MIXED to the SLO (Left), the Interference Strength β (Middle) and the Peak OLTP Load ρ (Right)

Figure 7 (left) varies the SLO from 0.6 to 1.5 ms on AZ-MIXED. Idle-Harvest complies only when the SLO is at least 1.0 ms; for tighter SLOs, cold-start inflation and interference make it violate the SLO in up to 99.9% of the seconds. PARTIES-style becomes less safe as the SLO loosens (8.4–13.0% violating seconds for SLOs of 0.8 ms and above), because its down-sizing threshold is easily met and it shrinks the OLTP partition until a surge causes a queueing collapse; Heracles-style stays between 5.7% and 10.0%. SLICE keeps violations at or below 1.2% throughout and changes its

mechanism mix with the SLO; the share of periods in which it harvests is 0% at 0.6 ms, 4% at 0.7 ms, 20% at 0.8 ms, 81% at 1.0 ms, 99% at 1.5 ms. At loose SLOs, dedicated analytical cores combined with harvesting yield more OLAP throughput than Idle-Harvest itself (65.3% vs. 58.7% at 1.5 ms), because dedicated cores avoid the efficiency loss of harvested cycles.

Figure 7 (middle) varies the ground-truth interference strength β. Without interference, Idle-Harvest is close to ideal (0.22% violating seconds, 62.5% OLAP throughput), and SLICE attains 92% of its throughput while harvesting in 53% of the periods. As β grows to 0.6, Idle-Harvest's violations rise to 93.5%, whereas SLICE learns β online, stops harvesting and throttles the analytical footprint, keeping violations between 0.17% and 0.72%; PARTIES-style violates the SLO in 9.4% and 2.9% of the seconds at $\beta = 0$ and 0.15. Figure 7 (right) varies the peak OLTP load ρ from 0.5 to 0.8. SLICE has the lowest violation rate of the adaptive policies at every load (0.22–0.56% vs. 0.56–1.23% for PARTIES-style and 3.6–6.7% for Heracles-style) with OLAP throughput within ±6% of PARTIES-style, while the throughput of Static-Peak falls from 37.5% to 12.5% because its partition must grow with the peak.

5.4 Ablation Study

Table 3 Ablation of SLICE Components (Mean of AZ-MIXED and AZ-BURSTY)

Variant	Viol. (%)	p99 (ms)	OLAP (%)	EMU (%)	Avg. k
SLICE (full)	**0.45**	**0.62**	**45.6**	**68.7**	**8.8**
w/o conformal	0.45	0.64	47.2	70.3	9.3
w/o Mondrian bins	0.42	0.62	45.1	68.2	9.1
w/o scorecaster	0.33	0.62	44.9	68.0	9.0
w/o forecast	0.42	0.62	45.5	68.6	8.8
w/o harvesting	0.31	0.61	45.0	68.1	8.8
w/o guard	3.57	3.55	44.9	68.0	9.0
w/o guard & conformal	13.70	51.32	59.2	82.3	7.2
w/o interference model	0.36	0.60	35.6	58.7	10.5
w/o interference model & conformal	1.64	0.62	27.7	50.8	15.7

Table 3 removes one component at a time (and two pairs) on AZ-MIXED and AZ-BURSTY. The fast guard is the main safety mechanism against second-scale surges: without it, violating seconds grow from 0.45% to 3.57%, and removing the conformal calibration as well raises them to 13.70%. The interference model is the main efficiency mechanism: without it, the calibrator still keeps SLICE safe (0.36%) by absorbing the systematic error into larger offsets, but OLAP throughput drops from 45.6% to 35.6%; removing the calibrator too leaves the model unprotected and violations reach 1.64%. With an interference-aware model and the guard in place, the remaining components (calibration, Mondrian bins, scorecaster, forecaster) change violations by at most 0.12 percentage points and OLAP throughput by at most 1.6 points at this operating point: the calibrated model is already accurate here, and the calibration layer pays off as a safeguard against model misspecification and as an explicit risk dial (Section 5.5). Harvesting is rarely selected at a 0.7-ms SLO, so disabling it changes little here, whereas it is chosen in most periods at looser SLOs (Section 5.3).

5.5 Calibration and Risk Budget

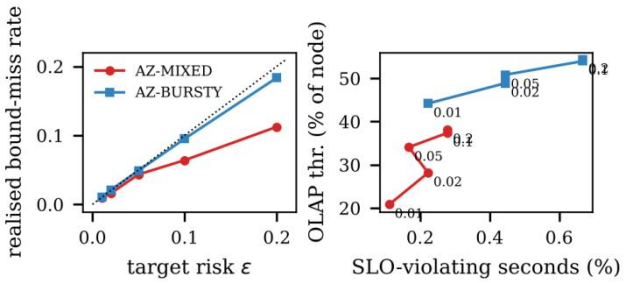


Figure 8 Conformal Calibration

Note: Left: realized frequency of bound violations vs. the risk budget ε (dotted: ideal). Right: the resulting trade-off between SLO-violating seconds and OLAP throughput (labels: ε).

Figure 8 varies the risk budget ε from 0.01 to 0.2. On AZ-BURSTY the realized frequency with which the observed p99 exceeds the calibrated bound follows the budget closely (0.011, 0.021, 0.049, 0.095, 0.184 for ε = 0.01, 0.02, 0.05, 0.1, 0.2). On AZ-MIXED it matches small budgets (0.010 at ε = 0.01) and stays below larger ones (0.064 at 0.1 and 0.112 at 0.2), i.e., the deviation is on the safe side: offsets are monotone in utilization, so high-utilization bins inherit larger offsets, and with a few hundred scored periods per bin the finite-horizon term of Proposition 1 is still visible. The budget acts as an interpretable dial: on AZ-BURSTY, raising ε from 0.01 to 0.2 increases OLAP throughput from 44.2% to 54.4% while violating seconds move from 0.22% to 0.67%; on AZ-MIXED the corresponding ranges are 20.8–38.1% and 0.11–0.28%. SLO violations remain far below ε because a bound miss turns into a violation only when the chosen allocation sits at the SLO boundary, and the guard intervenes within 100 ms.

Overhead. A SLICE decision takes 1.67 ms on average in our Python/Numba implementation, about 0.2% of the 1-s period, and the guard check is a constant-time comparison per tick. SLICE reconfigures the cpusets 28.6 times per minute on average (PARTIES-style: 1.8; Heracles-style: 27.2); each change rewrites two cgroup files and migrates at most a few analytical threads, which morsel-driven engines absorb at morsel boundaries.

6 RELATED WORK

Performance isolation in database services. SQLVM introduced resource reservations inside a multi-tenant database engine [6], and Das et al. designed CPU schedulers that provide absolute reservations without static allocation [7]. These mechanisms guarantee CPU shares rather than tail-latency objectives and keep a fixed mechanism regardless of the operating point; fleet analyses show that the temporal variability they leave unexploited is substantial [4].

HTAP and analytical scheduling. HTAP systems isolate the two classes through replication and dedicated replicas or stores [2-3, 25], and a recent survey identifies resource scheduling as a central open challenge [1]. Raza et al. schedule CPU and memory elastically between OLTP and OLAP engines to meet freshness requirements [26]. Morsel-driven parallelism and self-tuning query scheduling make analytical execution elastic in the number of cores [24, 27], which SLICE exploits when it resizes the OLAP partition every second. SLICE is complementary to these engines: it decides how many cores each class receives under an explicit tail-latency risk budget.

Colocation and tail latency. Bubble-Up predicts interference from memory-subsystem pressure [20]; Heracles and PARTIES coordinate isolation mechanisms with feedback [9-10]; Leverich and Kozyrakis identified the Linux scheduler as a major source of tail inflation [8]; Shenango and Caladan reallocate cores at microsecond scale with kernel-bypass runtimes [17-18]; SmartHarvest predicts the core demand of primary virtual machines to harvest idle cores safely [16]; and Autothrottle couples learning-based SLO feedback with CPU throttling for microservices [11]. SLICE differs in two respects: it treats partitioning, harvesting and throttling as one per-second choice driven by a queueing model, and it calibrates its latency bound with a distribution-free guarantee.

Conformal prediction. Conformal methods provide distribution-free coverage [22]; adaptive conformal inference and conformal PID control extend coverage to non-stationary sequences [12, 23]. SLICE carries these guarantees into closed-loop resource allocation by scoring the allocation actually applied and by partitioning the calibration by operating regime.

7 CONCLUSION

Mixed transactional and analytical workloads make CPU isolation a moving target: the mechanism that is safe and efficient at one load level, SLO and interference strength violates the tail or wastes cores at another. Measurements on a real testbed quantified the costs of fair sharing and idle-class harvesting, and SLICE turns them into an adaptive, risk-bounded controller that combines a queueing model, online interference estimation, Mondrian conformal PID calibration and a fast guard. Across three Azure production traces, SLICE keeps SLO-violating seconds within 0.11–0.67%, delivers 2.01× the analytical throughput of peak-provisioned partitioning, and automatically moves from partitioning to harvesting as the SLO loosens or interference weakens. The risk budget ε gives operators an interpretable dial between transactional safety and analytical efficiency. Future work will extend the formulation with cache and memory-bandwidth partitioning and evaluate it on multi-socket servers with simultaneous multithreading.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Zhang C, Li G, Zhang J, et al. HTAP databases: A survey. IEEE Transactions on Knowledge and Data Engineering, 2024. DOI: 10.1109/TKDE.2024.3389693.
[2] Huang D, Liu Q, Cui Q, et al. TiDB: A Raft-based HTAP database. Proceedings of the VLDB Endowment, 2020, 13(12): 3072-3084.
[3] Wang J, Li T L, Song H Z, et al. PolarDB-IMCI: A cloud-native HTAP database system at Alibaba. Proceedings of the ACM on Management of Data, 2023, 1(2): 199.
[4] van Renen A, Horn D, Pfeil P, et al. Why TPC is not enough: An analysis of the Amazon Redshift fleet. Proceedings of the VLDB Endowment, 2024, 17(11): 3694-3706.
[5] Dean J, Barroso L A. The tail at scale. Communications of the ACM, 2013, 56(2): 74-80.
[6] Narasayya V, Das S, Syamala M, et al. SQLVM: Performance isolation in multi-tenant relational database-as-a-service. In: Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR), 2013.
[7] Das S, Narasayya V R, Li F, et al. CPU sharing techniques for performance isolation in multi-tenant relational database-as-a-service. Proceedings of the VLDB Endowment, 2013, 7(1): 37-48.
[8] Leverich J, Kozyrakis C. Reconciling high server utilization and sub-millisecond quality-of-service. In: Proceedings of the 9th European Conference on Computer Systems (EuroSys), 2014: 4.
[9] Lo D, Cheng L, Govindaraju R, et al. Heracles: Improving resource efficiency at scale. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA), 2015.

[10] Chen S, Delimitrou C, Martínez J F. PARTIES: QoS-aware resource partitioning for multiple interactive services. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019: 107-120.
[11] Wang Z, Li P, Liang C-J M, et al. Autothrottle: A practical bi-level approach to resource management for SLO-targeted microservices. In: Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2024.
[12] Angelopoulos A N, Candès E J, Tibshirani R J. Conformal PID control for time series prediction. In: Advances in Neural Information Processing Systems (NeurIPS), 2023, 36.
[13] Shahrad M, Fonseca R, Goiri Í, et al. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: Proceedings of the USENIX Annual Technical Conference (ATC), 2020.
[14] Cole R, Funke F, Giakoumakis L, et al. The mixed workload CH-benCHmark. In: Proceedings of the 4th International Workshop on Testing Database Systems (DBTest), 2011.
[15] Stoica I, Abdel-Wahab H. Earliest eligible virtual deadline first: A flexible and accurate mechanism for proportional share resource allocation. Department of Computer Science, Old Dominion University, Norfolk, VA, USA, Technical Report TR-95-22, 1995.
[16] Wang Y, Arya K, Kogias M, et al. SmartHarvest: Harvesting idle CPUs safely and efficiently in the cloud. In: Proceedings of the 16th European Conference on Computer Systems (EuroSys), 2021.
[17] Ousterhout A, Fried J, Behrens J, et al. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In: Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2019.
[18] Fried J, Ruan Z, Ousterhout A, et al. Caladan: Mitigating interference at microsecond timescales. In: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020: 281-297.
[19] Raasveldt M, Mühleisen H. DuckDB: An embeddable analytical database. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2019: 1981-1984.
[20] Mars J, Tang L, Hundt R, et al. Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2011: 248-259.
[21] Harchol-Balter M. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge, U.K.: Cambridge University Press, 2013.
[22] Vovk V, Gammerman A, Shafer G. Algorithmic Learning in a Random World. New York, NY, USA: Springer, 2005.
[23] Gibbs I, Candès E. Adaptive conformal inference under distribution shift. In: Advances in Neural Information Processing Systems (NeurIPS), 2021, 34.
[24] Leis V, Boncz P, Kemper A, et al. Morsel-driven parallelism: A NUMA-aware query evaluation framework for the many-core age. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2014: 743-754.
[25] Makreshanski D, Giceva J, Barthels C, et al. BatchDB: Efficient isolated execution of hybrid OLTP+OLAP workloads for interactive applications. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2017: 37-50.
[26] Raza A, Chrysogelos P, Anadiotis A C, et al. Adaptive HTAP through elastic resource scheduling. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2020: 2043-2054.
[27] Wagner B, Kohn A, Neumann T. Self-tuning query scheduling for analytical workloads. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2021.