Towards Practical Latency SLOs on Cloud Data Warehouses

Markos Markakis
MIT CSAIL
Cambridge, MA, USA
markakis@mit.edu

Tim Kraska
MIT CSAIL
Cambridge, MA, USA
kraska@mit.edu

ABSTRACT

Modern cloud data warehouses decouple compute from storage, letting multiple compute clusters access the same underlying data. Organizations often use this feature for performance isolation, in order to help satisfy the latency service-level objectives (SLOs) of diverse workloads. For example, interactive dashboards, ad hoc analysis, and batch jobs can each run on separate clusters. However, this multi-cluster approach requires continuous resource management to adapt to workload evolution, with over-provisioning wasting resources and under-provisioning risking SLO violations.

In this work, we outline AutoSLO, a latency-SLO-aware workload management framework for multi-cluster cloud data warehouses. AutoSLO includes three key components: (i) a periodic Policy Tuner that proactively plans resources using workload forecasts; (ii) an SLO-aware reactive Autoscaler that adjusts the active cluster set based on the observed workload; and (iii) an online Query Router that reacts to concurrent query load when routing.

In preliminary experiments on TPC-DS-derived workloads, the Query Router and Autoscaler outperform alternatives, respectively delivering SLO violation rate reductions of 47.8% and 93.7% on average. We also find the current implementation efficiency of these two components sufficient for their operational timescales. This extended abstract presents work in progress.

VLDB Workshop Reference Format:
Markos Markakis and Tim Kraska. Towards Practical Latency SLOs on Cloud Data Warehouses. VLDB 2026 Workshop: Applied AI for Database Systems and Applications (AIDB 2026).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mmarkakis/autoslo.

1 INTRODUCTION

The Problem. Modern cloud data warehouses such as Amazon Redshift [1] and Snowflake [11] decouple compute from storage, enabling multiple compute clusters to access the same underlying data. Customers have embraced this feature for coarse-grained performance isolation [12], placing workloads with different latency service-level objectives (SLOs) onto different clusters (e.g. an interactive dashboard vs. ad hoc internal analytics).

Proceedings of the VLDB Endowment. ISSN 2150-8097.

However, even with dedicated clusters for different workloads, current systems do not allow SLOs to be specified directly; administrators have to experiment with each cluster's settings until the relevant SLO is met. Even then, there is no control over query interference *within* each workload, which can significantly impact performance as acknowledged by recent work [6].

In this work, we instead treat latency SLOs as explicit inputs and use the mechanisms exposed by cloud data warehouses to meet them. Instead of cross-workload performance isolation for its own sake, we aim to meet the SLOs *of individual queries*, keeping the overall SLO violation rate under a user-provided target τ while minimizing infrastructure cost. This requires deciding which clusters to use, when to scale them, and where to route each incoming query.

What Success Looks Like. Per Table 1, addressing this problem requires three **key desired behaviors** at different timescales:

- **Plan ($\approx$ 24 hrs):** Enterprise workloads often exhibit patterns [12]. By extracting and anticipating these patterns, the system can make cluster-management and knob-tuning decisions proactively, ensuring that appropriately configured clusters are available before demand materializes.
- **Adjust ($\approx$ 5 mins):** Forecasts are imperfect, and workloads can evolve over time. The system must be able to respond by spinning up or tearing down clusters in response to unexpected bursts, lulls, or shifts in workload mix.
- **React ($\approx$ 1 s):** When a query arrives, the system must determine which active cluster to execute it on to best meet latency SLOs cost-efficiently. This decision must account for the query's own SLO, its potential impact on other running queries, and the implied infrastructure cost.

How Prior Work Falls Short. Several automatic workload management mechanisms have been proposed in recent years. However, as shown in Table 1, none exhibits all three **key desired behaviors**.

Some systems can only **plan** for a known or forecasted workload, using replays [16] or modeling [5], but cannot adjust resources to deal with workload shifts. At the other extreme, some systems can only **adjust** resources online using various forms of short-term performance modeling/assumptions accounting for different available resources [2, 4, 7–10, 15]. However, spinning up new clusters takes time, so purely reactive approaches can transiently leave the system under-provisioned at each transition.

Finally, existing systems route each query based on its own features or estimated resource requirements [5, 6, 9, 10, 15]. They do not **react** to live concurrent query load by considering which routing option maximizes overall SLO adherence. This is where RAIS [6] also falls short: by only exposing a *qualitative* price-performance slider, it remains unaware of latency SLOs, so it cannot provide SLO-aware query placement.

Table 1: No prior work exhibits all key desired behaviors.

Key Desired Behavior	**Plan** resources based on history	**Adjust** resources based on demand	**React** to live concurrent load
Timescale	≈ 24 hrs	≈ 5 mins	≈ 1 s
ResTune [16]	✓ *Replay-based knob tuning on replica*	✗	✗
WiseDB [5]	✓ *Tree model for fixed per-template routing*	✗	✗
Das et al. [2]	✗	✓ *Container sizing w/ token bucket*	✗
SLAOrchestrator [7–9]	✗	✓ *Utilization- or simulation-based pool resizing*	✗
BRAD [4, 15]	✗	✓ *Blueprint beam search*	✗
Auto-WLM [10]	✗	✓ *Queueing-based horizontal scaling*	✗
RAIS [6]	✓ *Multi-forecast parameter tuning*	✓ *Queueing-based horizontal scaling*	✗
AutoSLO	✓ *Multi-forecast spinup planning & tuning*	✓ *Simulation-based best-cluster spinup*	✓ *Concurrency-aware routing*
Component	Policy Tuner	Autoscaler	Query Router
Details in...	Section 2.3	Section 2.2	Section 2.1

Our Approach. In response to the landscape above, we propose AutoSLO, which exhibits each key desired behavior through a key component: the Policy Tuner, the Autoscaler, and the Query Router.

The Policy Tuner can **plan** using the historical workload. It generates a set of forecasted workloads, which can be simulated to determine appropriate proactive cluster management decisions. By ensuring that the right resources are available at the right time (shortly before demand materializes), it reduces dependence on reactive scaling and mitigates cluster spinup delays.

The Autoscaler can **adjust** the active cluster set when necessary. Its key sub-components include the Scaling Triggers, which identify opportune moments for resource adjustments, and the Spinup Size Selector, which uses short-term simulations to estimate the SLO and cost implications of different scaling actions.

Finally, the Query Router can **react** by selecting which active cluster to execute each incoming query q on by efficiently managing a multi-way tradeoff among (1) the risk that q will violate its SLO; (2) the risk q poses to the SLO adherence of running queries; and (3) the resource cost implied by each routing decision.

Contributions. In summary, we make the following contributions:

- We introduce AutoSLO, a framework that provides the three key desired behaviors needed for SLO-aware multi-cluster workload management: planning from historical workload patterns, adjusting resources when workloads deviate from the forecast, and reacting to live concurrent load when routing individual queries.
- We outline the components of AutoSLO: a forecast-driven Policy Tuner, a simulation-based Autoscaler, and a load-aware Query Router.
- We present a promising preliminary evaluation of the Query Router and the Autoscaler on TPC-DS-derived workloads.

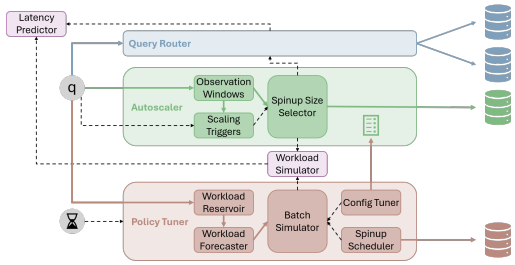


Figure 1: The architecture of AutoSLO.

2 AUTOSLO OVERVIEW

We will now dive deeper into the architecture of AutoSLO, as seen in Figure 1. We assume that each incoming query carries a latency SLO, possibly inherited from its workload, in the form of a maximum acceptable latency. We will present the components in the order a query encounters them, starting with the Query Router.

2.1 Query Router

The Query Router routes each incoming query q to one of the active clusters $c \in C$. Its objective is to balance three competing concerns: (1) the ability of q to meet its SLO, (2) the impact of q on the SLO adherence of currently running queries, and (3) the cost implications of the routing decision.

To estimate these effects, the Query Router leverages the Latency Predictor, an adapted version of the LSTM-based concurrency-aware Iconq [14]. We extend Iconq's query featurization to account for cluster size and use it to evaluate possible query placements.

For each active cluster c, the Query Router hypothetically places q on c and predicts the latency of both q and any queries currently running on c. These predictions are used to estimate both the resulting SLO violations and the projected cost, since longer execution times will increase c's billed time. The Query Router then computes the marginal change in aggregate SLO violation and cost relative to the current system state (without q) and selects the cluster that minimizes this objective, prioritizing SLO adherence and using cost as a secondary criterion.

By explicitly reasoning about query interference, SLO violation risk, and resource cost, the Query Router can make decisions that balance performance and efficiency in real time.

2.2 Autoscaler

While the Query Router selects the best active cluster for each incoming query, the active cluster set itself may become suboptimal as workload conditions change. To address this challenge, the Autoscaler runs in a background thread, continuously monitors SLO adherence, and can dynamically adjust the active cluster set.

The Autoscaler maintains rolling Observation Windows with recently submitted queries and their routing decisions (Arrival Window), as well as recently completed queries (Completion Window). When the SLO violation rate among running and recently completed queries exceeds a configurable threshold, the Scaling Triggers fire and the Autoscaler evaluates whether to spin up an additional cluster. Conversely, clusters that remain idle for a sufficiently long period are detected by the same Scaling Triggers for removal, allowing AutoSLO to reduce its footprint when demand subsides.

To evaluate potential scaling actions, the Spinup Size Selector runs the Workload Simulator over a short-term workload forecast, constructed by repeatedly replaying copies of the Arrival Window, preserving the relative query arrival times. For each candidate cluster size that a new cluster could have, it simulates the execution of this projected workload, routes queries using the Query Router, and estimates the resulting SLO violation rate and cost. The simulation explicitly accounts for the spinup time δ of a new cluster, evaluating its benefit only on forecasted queries arriving after this time has elapsed and the new cluster can be genuinely expected to be available. The Spinup Size Selector then selects the cluster size that achieves the best SLO adherence, breaking ties in favor of lower cost. A cluster of that size is spun up, only if having it would outperform the simulated option of taking no scaling action at all.

By using the recent workload to evaluate spinup actions, the Autoscaler can dynamically adapt the active cluster set to workload conditions and sustain high SLO adherence, while keeping cost low.

2.3 Policy Tuner

While the Autoscaler reacts to workload changes online using recent observations, additional optimization opportunities become available once AutoSLO has accumulated a richer history of workload behavior. To exploit these opportunities, AutoSLO includes a Policy Tuner that periodically optimizes system behavior using forecasted workloads and an event-driven simulator.

The Policy Tuner focuses on two objectives. First, it identifies recurring workload patterns that justify proactively provisioning additional clusters before congestion occurs. Second, it tunes the configuration parameters of the Autoscaler to improve the tradeoff between SLO adherence and cost for reactive autoscaling.

To support these tasks, the Workload Forecaster generates a set of forecasted workloads from the Workload Reservoir. Our default implementation groups historical queries by day-of-week and hour-of-day and samples query texts and inter-arrival times with recency weighting within each day/hour bin.

The Policy Tuner then invokes the Spinup Scheduler, which searches for recurring periods of congestion. It simulates the forecasted workloads and identifies intervals where multiple forecasts experience elevated SLO violation rates. It then greedily evaluates candidate cluster spinups before these intervals begin, accounting for the cluster spinup delay δ, and assesses their benefit on downstream SLO adherence and cost. Beneficial spinups are proactively scheduled for the corresponding point in the future. This process is made more efficient by using the Batch Simulator to explore multiple options in parallel.

Finally, the Configuration Tuner optimizes the parameters of the Autoscaler. It generates candidate configurations, simulates the forecasted workloads on them, and retains the most promising one. Throughout this process, configurations are ranked according to a target-τ-aware objective that prioritizes lowering the SLO violation rate below τ before minimizing resource cost.

By offering workload forecasting, proactive provisioning and simulation-driven configuration tuning, the Policy Tuner allows AutoSLO to adapt to recurring workload patterns.

3 PRELIMINARY EVALUATION

We use a machine with two 20-core 2.10 GHz Intel Xeon Gold 6230 CPUs [3] and 256 GiB of memory, running Linux 6.9.7-arch1-1. From this machine, we use `psycopg2` to execute queries against Amazon Redshift Serverless. We use `boto3` to spin up and tear down workgroups and namespaces. For continuity, we will refer to workgroups as "clusters" below. We use the current Amazon Redshift Serverless price for `us-east-1` ($0.375/RPU/$h$), where an RPU is the unit of cluster size. For TPC-DS, we use SF 1000 (1 TB).

3.1 Query Router Deep Dive

We begin by assessing the performance of the Query Router.

3.1.1 Effectiveness. We compare three query routing approaches. Round-Robin alternates between the available clusters, as a naive baseline. Stage + Cost Opt. uses Stage [13] to derive concurrency-unaware latency predictions on each active cluster and then routes each query to a cluster where its SLO will be met, breaking ties by cost. Ours (Iconq + Cost Opt.) represents our approach described in Section 2.1, where the concurrency-aware Iconq model is used and the risk to the SLOs of running queries is also assessed.

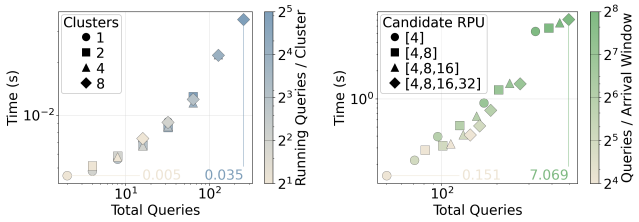
We use a workload based on TPC-DS, with 3 copies of each benchmark query (for a total of 297 queries) in shuffled order. We first execute this workload in closed-loop mode on a 16-RPU cluster as a baseline. We then evaluate each approach on two open-loop versions of the workload, modeling query arrival times arrivals as a Poisson process with rate parameter λ, where we set λ to either 0.5 or 0.2. For each query, we define an SLO as κ times its maximum latency in the closed-loop run, where we set κ to either 4 or 2. Note that the SLOs are unused by the Round-Robin approach. Lastly, we either route between two clusters with 16 RPU each ($C = [16, 16]$), or between one cluster each with 32 RPU and 16 RPU ($C = [32, 16]$).

The results are presented in Table 2. As evident, our approach provides excellent performance across the 9 workloads, delivering a mean SLO violation rate reduction of 47.8% compared to the naive Round-Robin approach. It also reduces cost by a mean of 12.9%, since the Query Router's decisions are also influenced by cost considerations. Using the concurrency-aware Iconq model is vital, since the Stage + Cost Opt. method can only reduce the SLO violation rate by a mean of 24.3% compared to Round-Robin.

3.1.2 Efficiency. The query router is invoked in the critical path each incoming query, so it needs to be highly efficient. We assess this by timing its decision making while sweeping two variables: the number of active clusters, and the number of currently-running queries per cluster. For each combination of values, we repeat the experiment 10 times, plotting the median in Figure 2a. As evident, the Query Router is highly efficient, producing decisions in 35 ms even with 8 active clusters executing 32 queries each.

Table 2: Query Router effectiveness evaluation. VR = SLO Violation Rate. Mean improvement is relative to Round Robin.

λ	κ	C	Round Robin		Stage + Cost.Opt.		Iconq + Cost.Opt.	
			VR	Cost	VR	Cost	VR	Cost
0.1	4	[32, 16]	0.0269	$15.78	**0.0135**	$11.01	0.0168	$11.17
0.1	4	[16, 16]	0.0976	$10.40	0.0370	$7.99	**0.0303**	$8.91
0.1	2	[32, 16]	0.0842	$15.78	0.0808	$10.14	**0.0505**	$12.28
0.1	2	[16, 16]	0.2088	$10.57	0.1414	$7.43	**0.1246**	$8.77
0.2	4	[32, 16]	0.1414	$7.92	0.0707	$7.56	**0.0202**	$7.60
0.2	4	[16, 16]	0.3064	$5.41	0.2761	$4.81	**0.1919**	$5.29
0.2	2	[32, 16]	0.1919	$8.13	0.1919	$6.92	**0.1044**	$7.71
0.2	2	[16, 16]	0.4512	$5.48	0.5118	$5.20	**0.3300**	$4.98
Mean Δ			—	—	-24.3%	-19.3%	**-47.8%**	-12.9%



(a) Query Router **(b) Autoscaler**

Figure 2: Efficiency evaluation of key AutoSLO components.

3.2 Autoscaler Deep Dive

We now move on to assessing the performance of the Autoscaler, focusing in particular on the Spinup Size Selector.

3.2.1 Spinup Size Selector Effectiveness. We compare four autoscaling approaches. No-Op makes no changes to the active clusters. Horizontal spins up an additional cluster of the same size as the existing one, using our Query Router to route between the two. Vertical w/ Ours uses the same simulation-based decision-making as our method, but instead of considering what single cluster to *add*, it considers what single cluster to *have* (i.e. it tears down pre-existing clusters once the new one becomes available). Add Cluster w/ Ours represents our approach described in Section 2.2.

We create 8 experimental scenarios using the values of λ and κ from Section 3.1.1, but we start with only a single cluster, of size either 8 or 16 RPU. We execute 20% of the workload queries, at which point we forcibly trigger autoscaling. We then let the following 60% of the workload elapse, for scaling to take effect and any congestion-affected queries around the autoscaling point to drain, and report the steady-state SLO violation rate and cost on the last 20% of the workload.

The results are presented in Table 3. Our cluster size selection approach (Add Cluster w/ Ours) provides the most reliable performance, reducing the SLO violation rate of No-Op by a mean of 93.7%. Using our algorithm for vertical autoscaling (Vertical w/ Ours) is also highly effective, reducing the SLO violation rate of No-Op by a mean of 83.6%. However, Add Cluster w/ Ours performs best on average, making effective use of the pre-existing cluster.

Table 3: Autoscaler effectiveness evaluation. VR = SLO Violation Rate. Mean improvement is relative to No-Op.

λ	κ	C	No-Op		Horizontal		Vertical w/Ours		Add Cluster w/Ours	
			VR	Cost	VR	Cost	VR	Cost	VR	Cost
0.1	4	[8]	0.9833	$2.81	0.8167	$1.99	0.2000	$1.79	**0.0167**	$2.16
0.1	4	[16]	0.1000	$1.65	**0.0000**	$2.67	0.0833	$1.59	**0.0000**	$2.35
0.1	2	[8]	0.9833	$2.86	1.0000	$2.04	**0.0167**	$3.40	0.1000	$2.09
0.1	2	[16]	0.3667	$1.74	0.0667	$3.14	**0.0167**	$3.40	0.0833	$2.81
0.2	4	[8]	1.0000	$3.27	1.0000	$3.55	**0.0333**	$3.11	**0.0333**	$2.88
0.2	4	[16]	0.8333	$2.21	0.1167	$1.83	0.0333	$2.14	**0.0000**	$2.71
0.2	2	[8]	1.0000	$3.31	0.9500	$3.34	**0.0667**	$3.12	**0.0667**	$3.15
0.2	2	[16]	0.9000	$2.05	0.4167	$2.26	0.0667	$2.29	**0.0500**	$2.75
Mean Δ			—	—	-42.7%	+11.0%	-83.6%	+9.1%	**-93.7%**	+11.8%

3.2.2 Spinup Size Selector Efficiency. The Autoscaler does not operate on the critical path of each query; it instead runs in a background thread, as described in Section 2.2. Still, it must be reasonably efficient for the scaling decisions to remain timely. The bulk of the Autoscaler's computational time is spent in the simulation step, so we assess the efficiency of this step while sweeping two variables: the family of additional cluster sizes considered and the arrival rate of queries in the Arrival Window. Both of these variables affect the number of concurrently running queries during the simulation, which affects the complexity of routing additional queries in their presence. For each combination of values, we repeat the experiment 10 times, plotting the median in Figure 2b. As evident, the hypothetical replay is efficient, requiring around 7 s even in the most challenging case.

4 CONCLUSIONS AND FUTURE WORK

We presented AutoSLO, a framework for cost-efficiently meeting latency SLOs on a multi-cluster cloud data warehouse deployment, comprised of a concurrency-aware Query Router, a simulation-driven reactive Autoscaler and a workload-pattern-aware Policy Tuner making proactive scaling decisions. On preliminary experiments, the components of AutoSLO are shown to be effective and efficient. This extended abstract presents work in progress; we are continuing to improve the algorithmic and implementation efficiency of individual components, before subjecting AutoSLO to a full suite of end-to-end experiments involving realistic workloads.

ACKNOWLEDGMENTS

This research was supported by Amazon, Google, and Intel as part of the MIT Data Systems and AI Lab (DSAIL). This research was also sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Amazon Web Services. 2026. Amazon Redshift. https://aws.amazon.com/redshift/. Retrieved June 1, 2026.

[2] Sudipto Das, Feng Li, Vivek R. Narasayya, and Arnd Christian König. 2016. Automated Demand-driven Resource Scaling in Relational Database-as-a-Service. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1923–1934. https://doi.org/10.1145/2882903.2903733

[3] Intel Corporation. 2019. *Intel Xeon Gold 6230 CPU*. Intel Corporation. Retrieved July 31, 2024 from https://ark.intel.com/content/www/us/en/ark/products/192437/intel-xeon-gold-6230-processor-27-5m-cache-2-10-ghz.html

[4] Tim Kraska, Tianyu Li, Samuel Madden, Markos Markakis, Amadou Ngom, Ziniu Wu, and Geoffrey X. Yu. 2023. Check Out the Big Brain on BRAD: Simplifying Cloud Data Processing with Learned Automated Data Meshes. *Proc. VLDB Endow.* 16, 11 (jul 2023), 3293–3301. https://doi.org/10.14778/3611479.3611526

[5] Ryan Marcus and Olga Papaemmanouil. 2016. WiSeDB: a learning-based workload management advisor for cloud databases. *Proc. VLDB Endow.* 9, 10 (Jun 2016), 780–791. https://doi.org/10.14778/2977797.2977804

[6] Vikram Nathan, Vikramank Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan (Murali) Narayanaswamy, and Tim Kraska. 2024. Intelligent scaling in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3626246.3653394

[7] Jennifer Ortiz, Victor Teixeira De Almeida, and Magdalena Balazinska. 2015. Changing the Face of Database Cloud Services with Personalized Service Level Agreements. In *CIDR*.

[8] Jennifer Ortiz, Brendan Lee, and Magdalena Balazinska. 2016. PerfEnforce Demonstration: Data Analytics with Performance Guarantees. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 2141–2144. https://doi.org/10.1145/2882903.2899402

[9] Jennifer Ortiz, Brendan Lee, Magdalena Balazinska, Johannes Gehrke, and Joseph L Hellerstein. 2018. SLAOrchestrator: Reducing the Cost of Performance SLAs for Cloud Data Analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 547–560.

[10] Gaurav Saxena, Mohammad Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan Narayanaswamy. 2023. Auto-WLM: Machine learning enhanced workload management in Amazon Redshift. In *Companion of the 2023 International Conference on Management of Data*. 225–237.

[11] Snowflake. 2026. Snowflake AI Data Cloud. https://www.snowflake.com/en/. Retrieved June 1, 2026.

[12] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC is Not Enough: An Analysis of the Amazon Redshift Fleet. *Proc. VLDB Endow.* 17, 11 (July 2024), 3694–3706. https://doi.org/10.14778/3681954.3682031

[13] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD/PODS '24)*. Association for Computing Machinery, New York, NY, USA, 280–294. https://doi.org/10.1145/3626246.3653391

[14] Ziniu Wu, Markos Markakis, Chunwei Liu, Peter Baile Chen, Balakrishnan Narayanaswamy, Tim Kraska, and Samuel Madden. 2025. Improving DBMS Scheduling Decisions with Accurate Performance Prediction on Concurrent Queries. *Proc. VLDB Endow.* 18, 11 (July 2025), 4185–4198. https://doi.org/10.14778/3749646.3749686

[15] Geoffrey X. Yu, Ziniu Wu, Ferdi Kossmann, Tianyu Li, Markos Markakis, Amadou Ngom, Samuel Madden, and Tim Kraska. 2024. Blueprinting the Cloud: Unifying and Automatically Optimizing Cloud Data Infrastructures with BRAD. *Proc. VLDB Endow.* 17, 11 (Aug 2024), 3629–3643. https://doi.org/10.14778/3681954.3682026

[16] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 international conference on management of data*. 2102–2114.