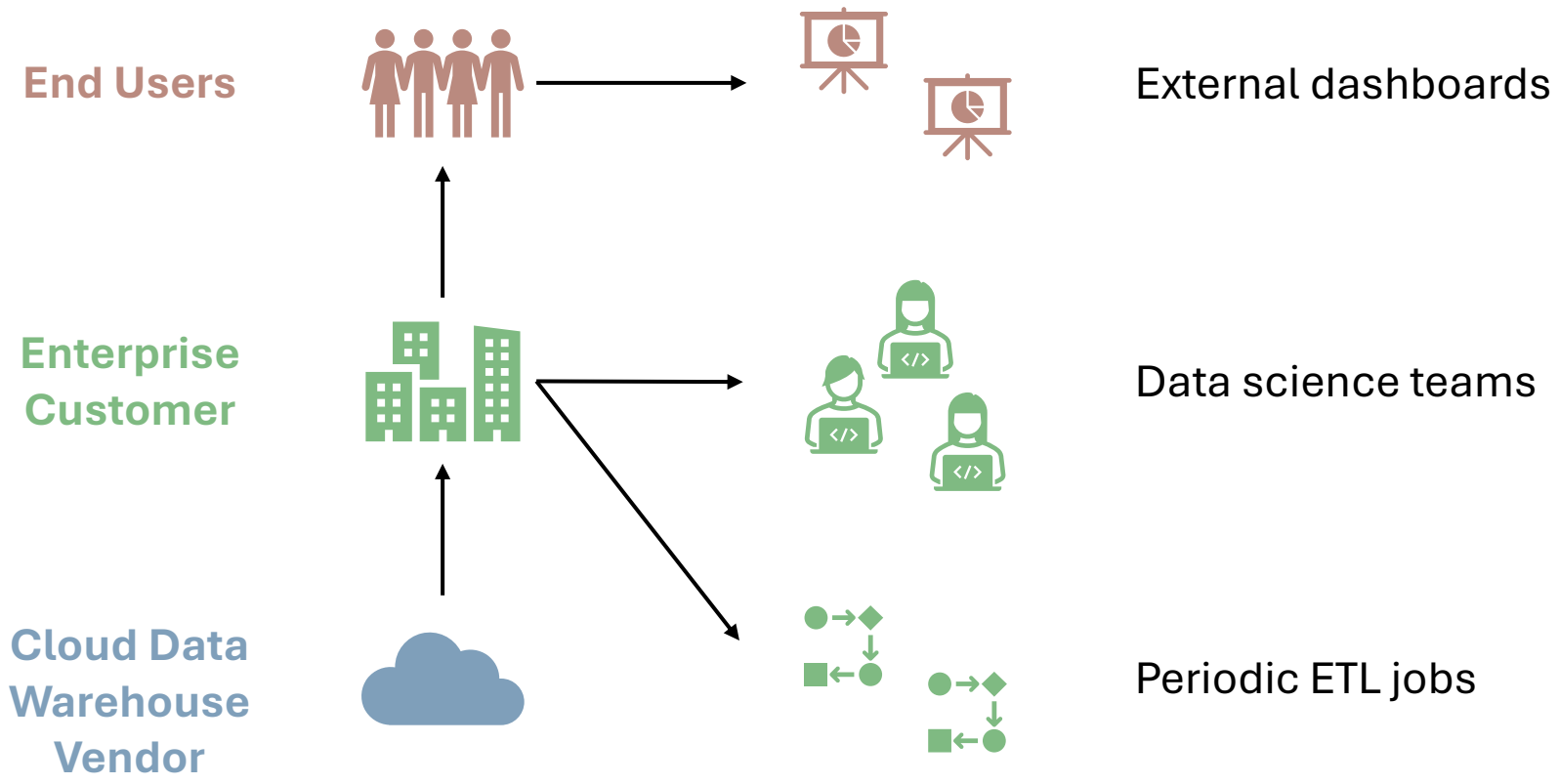
Towards Practical Latency SLOs on Cloud Data Warehouses

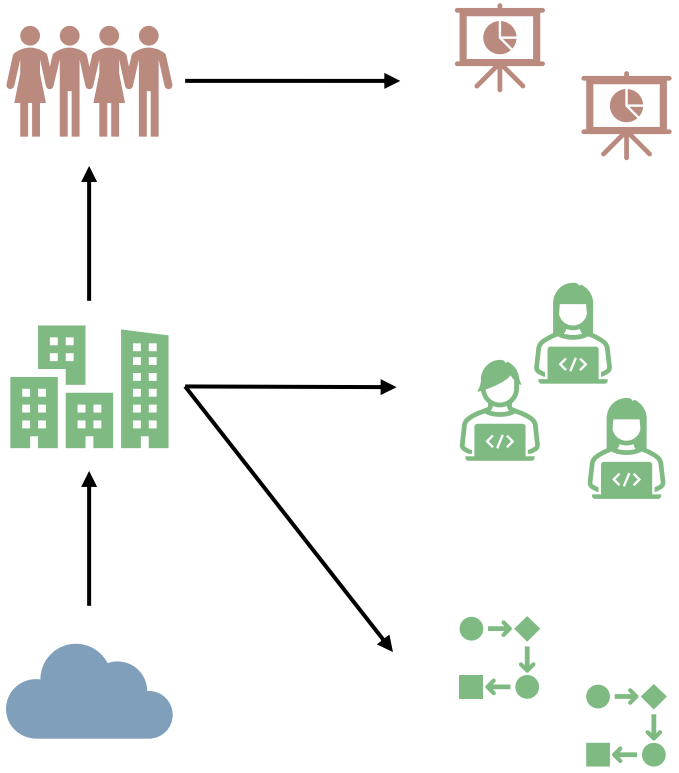
Markos Markakis, Tim Kraska

August 31, 2026

Cloud data warehouses see many workloads



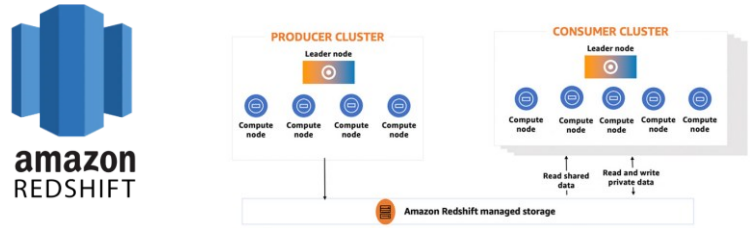
Each workload has a different latency SLO

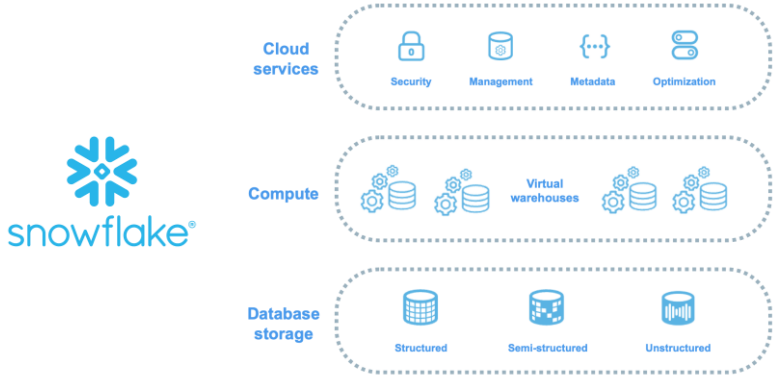


	At least x% of queries complete within...
External dashboards	A few seconds
Data science teams	Seconds to minutes
Periodic ETL jobs	Minutes to hours

Compute-storage separation can help isolate

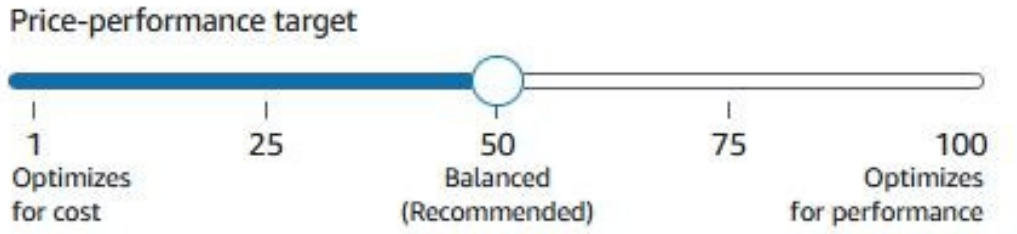
- Modern cloud data warehouses let multiple compute clusters query the same data.
- Offers a crude way to meet latency SLOs through performance isolation:
  - Map each workload to a separate cluster.
  - Manage each of these clusters separately.





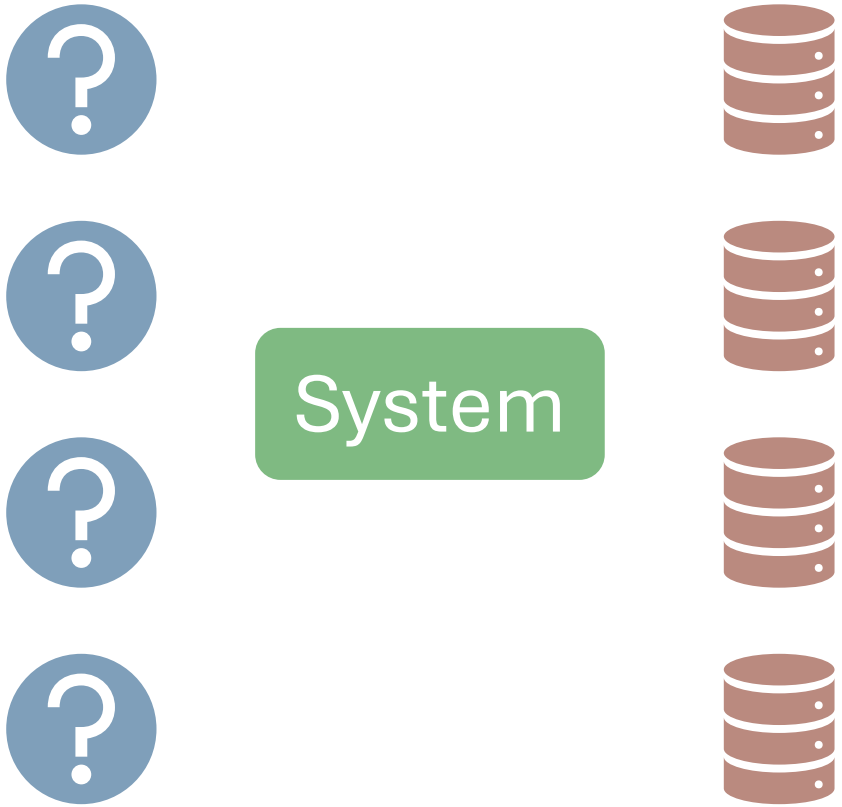
Meeting SLOs using separation is only implicit

- Per-workload clusters must be tuned independently until performance is acceptable.
  - Too small: Too many SLO violations vs. acceptable threshold
  - Too large: unnecessary costs
- Current auto-tuning solutions are not driven by explicit SLOs.
- Even if we tune each cluster, intra-workload contention is not managed.



What if we don't silo each workload?

- "Workloads" are not a necessary intermediate level. In practice:
  - There is a stream of queries
  - Each query has a latency SLO
  - The SLO may be inherited from somewhere, but it does not matter.
- On the other end, we can have one or more clusters running.
- We can then individually route each query to meet its SLO.



Managing this setup requires 3 key behaviors

Key Desired Behavior	Plan resources based on history	Adjust resources based on demand	React to concurrent live load
Timescale	≈ 24 hrs	≈ 5 mins	≈ 1 s

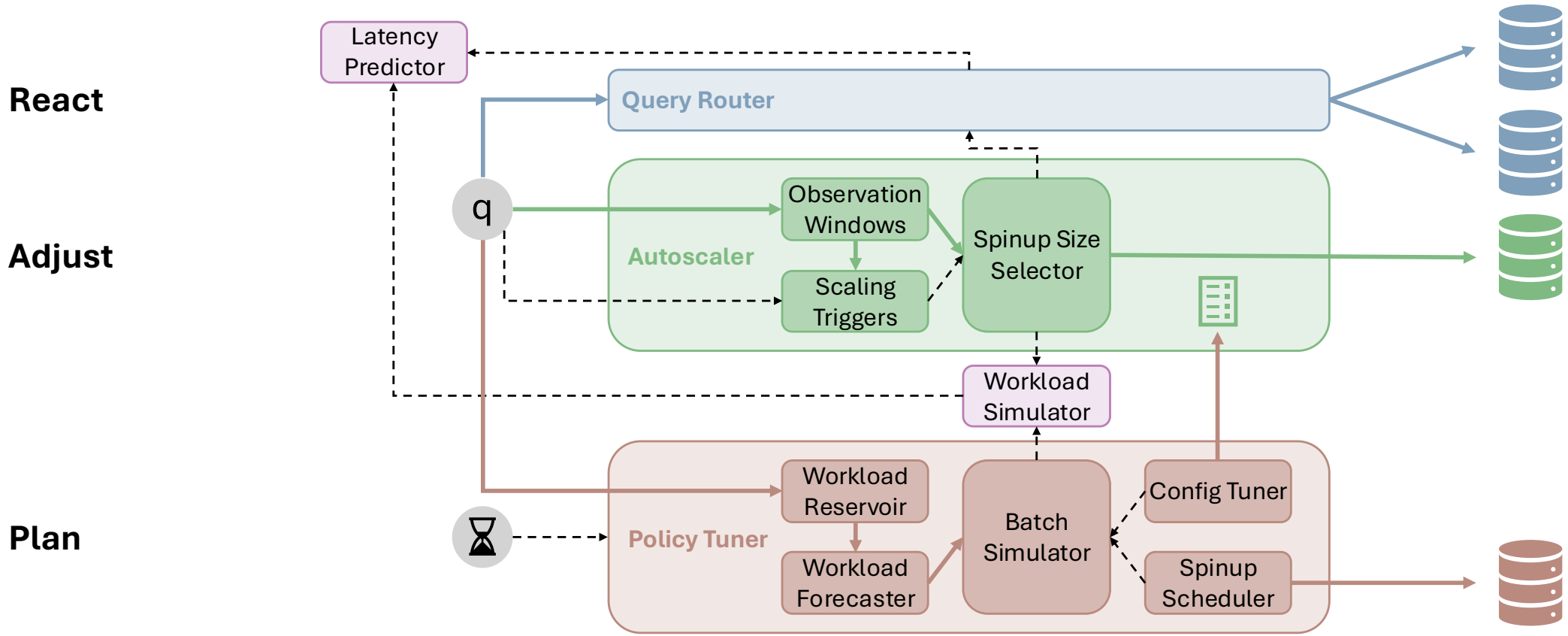
- Some workload elements are repeating/predictable; the system should plan resources to execute them.
- Online variability cannot be eliminated; the system should adjust resources to it.
- Each query is now routed individually; the system can and should react to concurrent load to limit SLO violations.

No prior work offers all 3 key behaviors

- Some works only plan for a given workload but cannot handle deviations.
- Other works can only adjust online, but SLO violations can accrue by the time new resources spin up.
- Across the board, no work can react, because no work treats SLOs as a first-class input.
- We now have the latency models and the system primitives to achieve that.

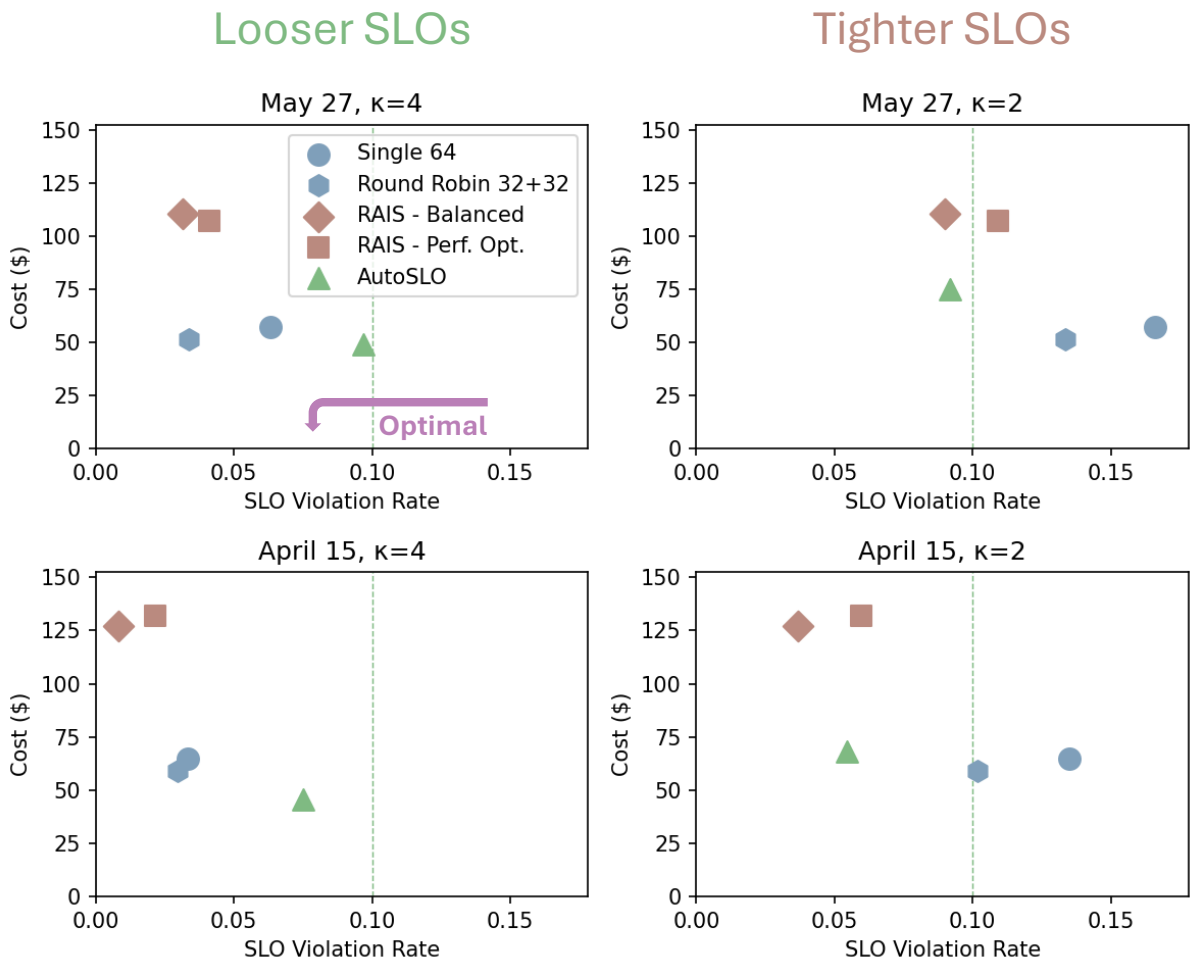
Key Desired Behavior	**Plan** resources based on history	**Adjust** resources based on demand	**React** to concurrent live load
Timescale	≈ 24 hrs	≈ 5 mins	≈ 1 s
ResTune	✓ *Replay-based knob tuning on replica*	✗	✗
WiseDB	✓ *Tree model for fixed per-template routing*	✗	✗
Das et al.	✗	✓ *Container sizing w/ token bucket*	✗
SLAOrches-trator	✗	✓ *Utilization- or simulation-based pool resizing*	✗
BRAD	✗	✓ *Blueprint beam search*	✗
Auto-WLM	✗	✓ *Queueing-based horizontal scaling*	✗
RAIS	✓ *Multi-forecast parameter tuning*	✓ *Queueing-based horizontal scaling*	✗

AutoSLO assigns a key component to each



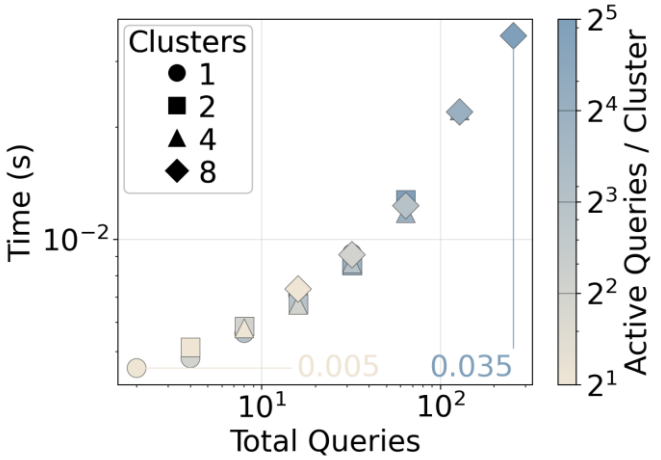
AutoSLO effectively meets SLOs end-to-end

- Manual single- or dual-cluster configurations are cheap but ineffective as SLOs tighten.
- Current autoscaling with RAIS is ~2x expensive.
- Meets SLOs more often, but cannot control it when κ=2.
- AutoSLO is always the cheapest method that meets the SLO target.
- It reduces cost by a mean of 26.4% vs. per-scenario next-best.
- Compared to the only baseline that always meets the target, it reduces cost by a mean of 49.6%.

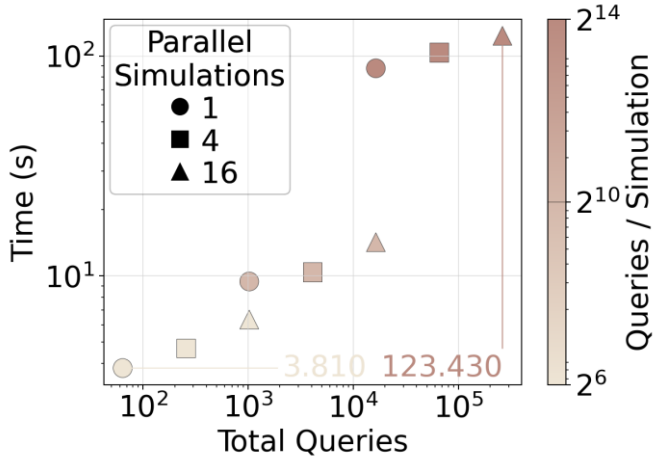


Each component is efficient for its timescale

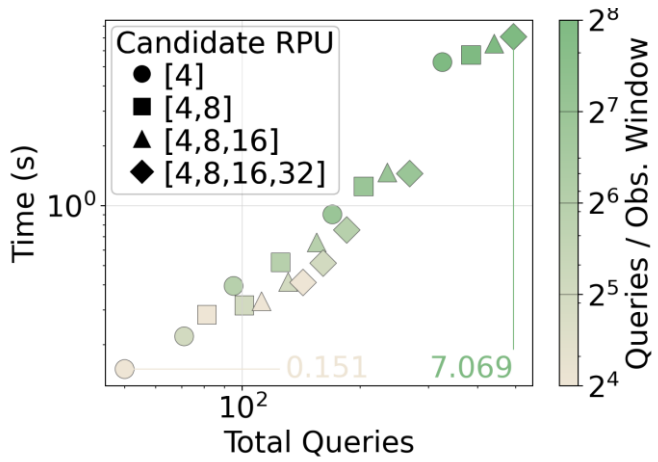
Route



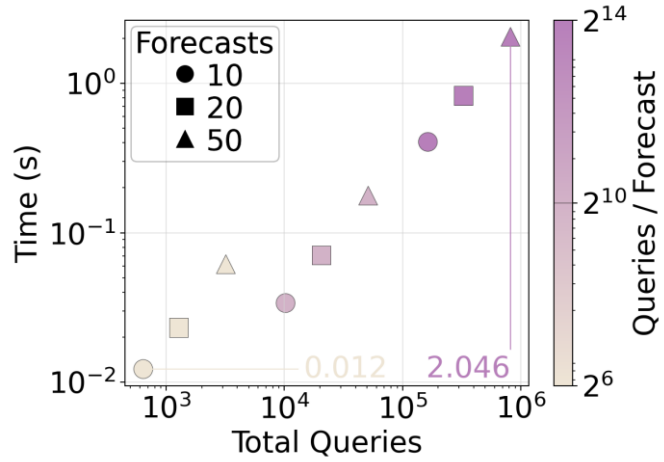
SimulateBatch



FindBestSpinupSize



FindGoodSpinupTime



Key Takeaway

Extended abstract

AutoSLO can scale resources and route queries to meet latency SLOs cheaply

Full preprint