Adaptive Resource Allocation in Stateful Stream Processing
A *Control-Based Autoscaling Policy for the Styx Runtime*

Author: Mihai-Valentin Nicolae
m.v.nicolae@student.tudelft.nl
Supervisor: Oto Mráz
Responsible Professor: Asterios Katsifodimos

1 Introduction

Stream processing systems power real-time, latency-sensitive applications:

- payments, inventory, reservations, business workflows

Challenges in production:

- **Workloads fluctuate** (spikes, gradual ramps)
- **Static resource allocation** cannot adapt at runtime
- **Over-provisioned** → wasted cost
- **Under-provisioned** → high latency, backpressure

This work designs a **control-based** policy that monitors system signals and adjusts Styx's configuration dynamically.

2 Research Question

*How can a **control-based autoscaling policy** be designed and tuned to optimize **performance** and **resource efficiency** of Styx under variable workloads?*

3 Background

Styx - transactional stateful stream processor (SFaaS):

- **Epoch-based execution:** workers sync at each epoch boundary
- **Targets:** serializable transactions, low latency, exactly-once
- **In-development autoscaler:** uses PID controller + Chronos forecaster

Existing Autoscalers (Dhalion, DS2):

- **Not directly applicable to Styx:** designed for non-transactional streaming operators **(Heron, Flink)**

4 Policy Design

▲ **Scale up**	backlog-PID ⩾ 1.0 **OR** epoch latency ⩾ 250 ms
▼ **Scale down**	backlog ≈ 0 (30 s) **AND** worker CPU < 40%
⏱ **Cooldown**	30 s between actions

5 Results

Setup: Cosine YCSB-T load (~0–14 k TPS, 300 s). **Reactive policy** (4–10 workers) vs. static baselines (**4**-worker **under-provisioned** and **9**-worker **over-provisioned**), the ideal manual schedule, and Styx's in-development autoscaler.

- **−26% cost vs 9-workers (within 3% of ideal)**
- **6 ms latency recovery between peaks**
- **3.1× less leftover backlog than the existing autoscaler**

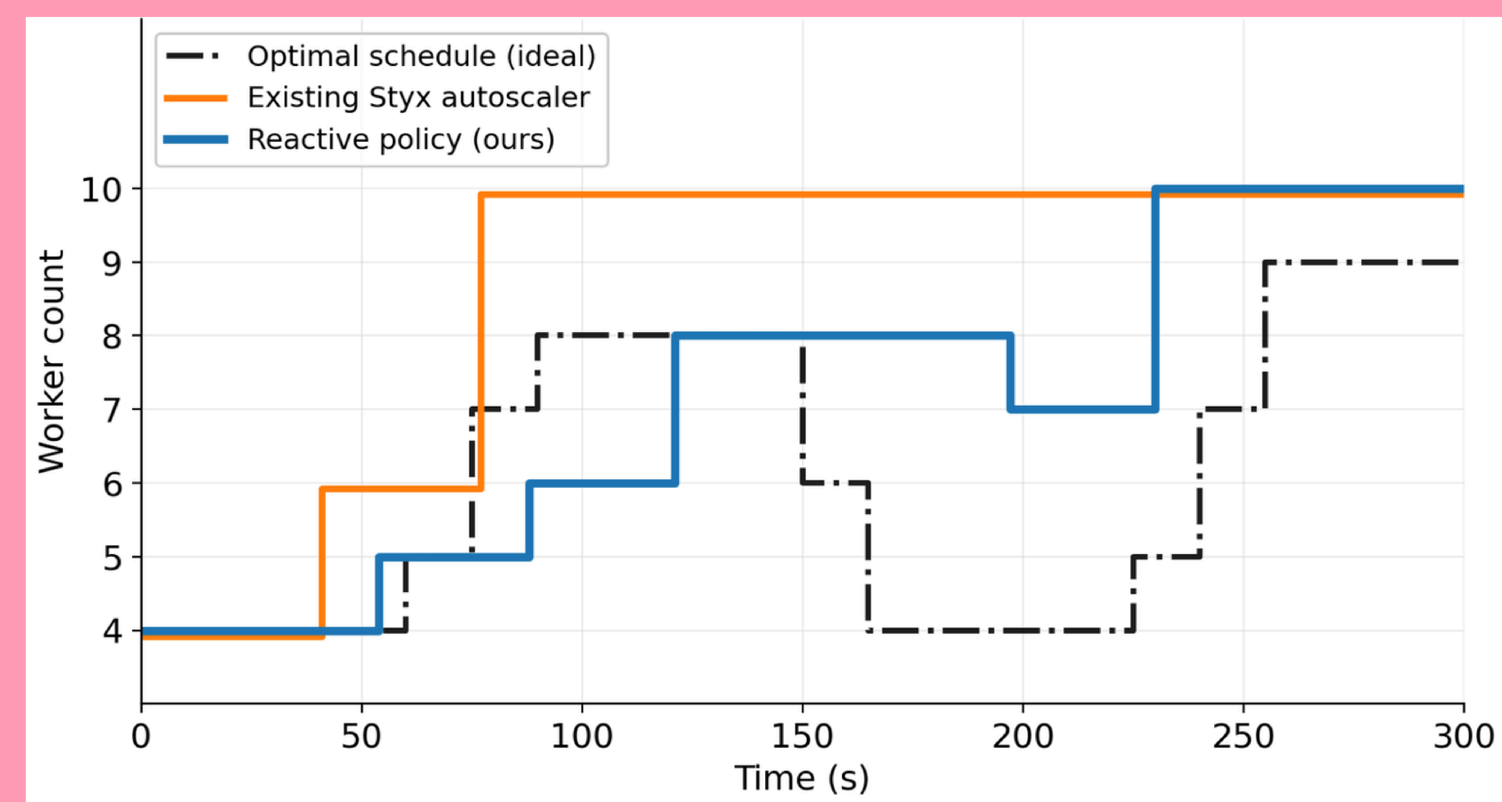

Figure 1: Worker-count decisions over one cosine cycle. The reactive policy follows the ideal schedule up and down; the existing autoscaler overshoots to the cap and never scales back.

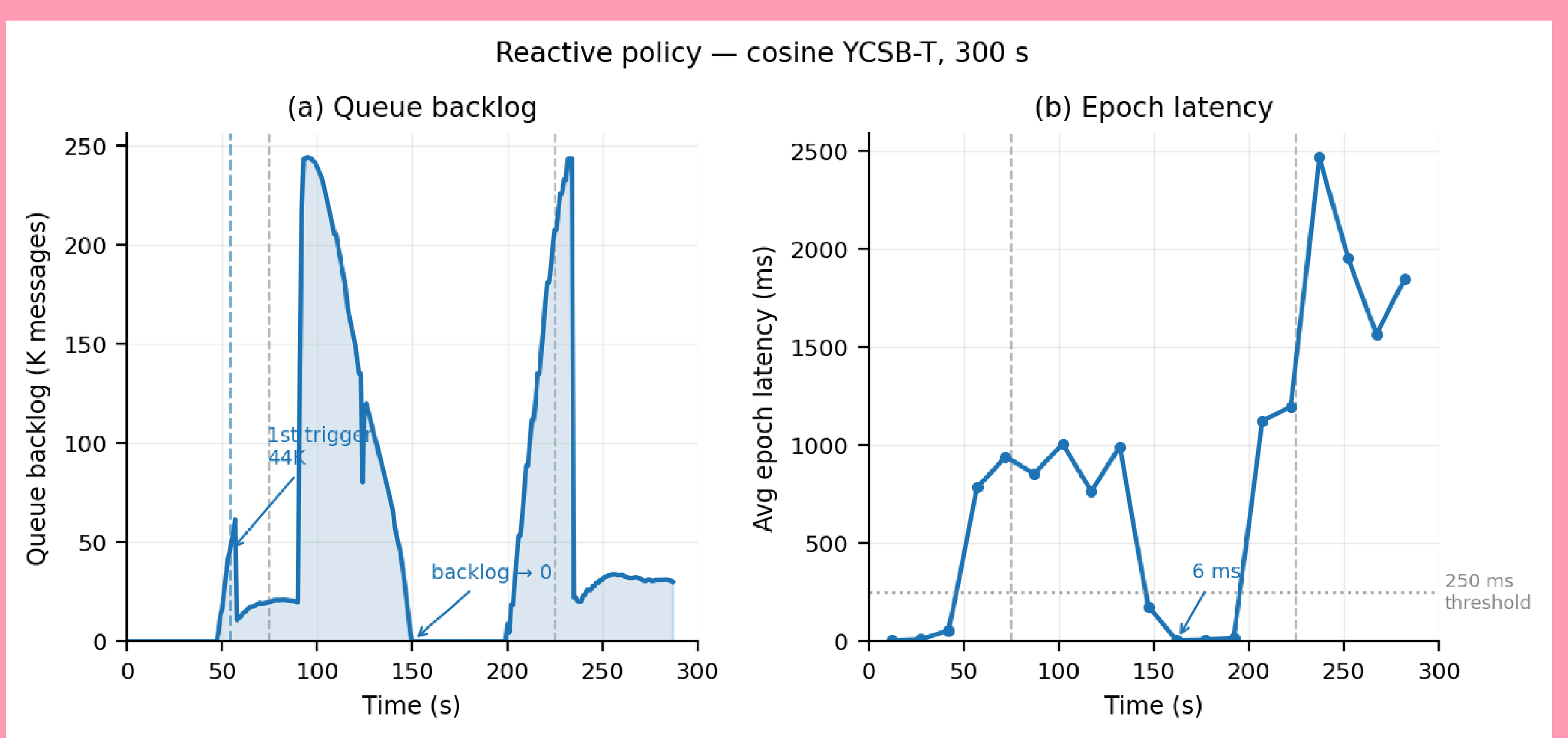

Figure 2: Reactive policy: backlog and epoch latency. Epoch latency crosses 250 ms and triggers scale-up while the backlog is still small; backlog then drains to zero and latency recovers to 6 ms between peaks.

Configuration	Worker-seconds	Peak backlog	End backlog
Under-provisioned (4w)	1,200	251 K	~200 K
Over-provisioned (9w)	2,700	0	0
Optimal manual	1,950	0	0
Reactive policy	**1,999**	**244 K**	**30 K**

6 Discussion

By using epoch latency as a leading scale-up signal plus reactive scale-down based on backlog and worker usage, a purely reactive policy can reach close to ideal cost.

- **Epoch latency leads overload -** Styx's epoch barrier makes an overloaded worker raise latency before backlog builds
- **The existing autoscaler overshoots** to the 10-worker cap and never scales down → cannot adapt to the next cycle

7 Limitations

- **Autoscaler reacts, does not predict -** the policy adjusts only ***after*** overload appears, so **backlog** and **latency** still spike at peaks (~244 K, ~2,468 ms)
- **Simplified cost metric -** worker-seconds, not real cloud pricing
- **Narrow evaluation -** one benchmark (YCSB-T), one workload shape (cosine)

8 Future Work

- **Add prediction -** optimize the reactive policy together with a predictive forecaster to scale ***before*** load peaks happen
- **Realistic cost model -** use instance-hour cloud pricing instead of worker-seconds
- **Broader workloads -** test using step/irregular loads and compute-heavy benchmarks like TPC-C