

Visual Insights into Agentic Optimization of Pervasive Stream Processing Services

Boris Sedlak*, Víctor Casamayor Pujol*, Schahram Dustdar*[†]

* Distributed Intelligence and Systems-Engineering Group, Universitat Pompeu Fabra, Barcelona, Spain

Email: {boris.sedlak, victor.casamayor, schahram.dustdar}@upf.edu

[†] Distributed Systems Group, TU Wien, Vienna, Austria

Abstract—Processing sensory data close to the data source—often involving Edge devices—promises low latency for pervasive applications, like smart cities. This commonly involves a multitude of processing services, executed with limited resources; this setup faces three problems: first, the application demand and the resource availability fluctuate, so the service execution must scale dynamically to sustain processing requirements (e.g., latency); second, each service permits different actions to adjust its operation, so they require individual scaling policies; third, without a higher-level mediator, services would cannibalize any resources of services co-located on the same device. This demo first presents a platform for context-aware autoscaling of stream processing services that allows developers to monitor and adjust the service execution across multiple service-specific parameters. We then connect a scaling agent to these interfaces that gradually builds an understanding of the processing environment by exploring each service’s action space; the agent then optimizes the service execution according to this knowledge. Participants can revisit the demo contents as video summary and introductory poster, or build a custom agent by extending the artifact repository.

Index Terms—Stream Processing, Autoscaling, Service Level Objectives, Elasticity, Edge Computing, Regression Analysis

I. INTRODUCTION

Sensory data is used for fueling and optimizing pervasive applications, from autonomous driving [1] to smart cities [2]. This is supported by the growing computational power of embedded devices and Edge servers that support low-latency processing close to the data source. The precise requirements *how* this processing must be done are specified through Service Level Objectives (SLOs); real-time applications—like point cloud mapping [3]—might specify a maximum target latency. Yet, resources on Edge servers are limited, whereas client demand is fluctuating; this inevitably leads to situations where resources do not suffice to satisfy SLOs across multiple competing clients and applications. To ensure SLO fulfillment, autoscaling solutions—like Kubernetes [4]—have specialized in adjusting applications according to varying demand; however, the default mechanism is provisioning additional resources. Also, it cannot be assumed that computation can be offloaded to nearby devices [5]. As the context changes dynamically and services cannot rely on predefined mechanisms (e.g., offloading or resource scaling), the processing services must autonomously find actions that optimize their SLO fulfillment.

To facilitate the transition to flexible and context-aware autoscaling of processing services, we developed a two-fold approach [6]: MUDAP—our Multi-Dimensional Autoscaling

Platform—supports fine-grained adjustments of the service execution and the allocated resources; notably, this permits dynamic adjustments to service-specific parameters, like the size of Machine Learning (ML) models or input tensors. Second, we presented RASK, a scaling agent that uses Regression Analysis of Structural Knowledge to interpret the effect of different parameter assignments on the SLO fulfillment, and then infer optimal scaling actions. Together, they enable flexible processing services that scale different parameters according to the context—a behavior called *multi-dimensional elasticity* [7]. Thus, services can trade off less critical aspects (e.g., client experience) to sustain critical SLOs (e.g., latency).

This demo first introduces the architecture of MUDAP and RASK; next, we design a scaling agent that uses these interfaces to optimize the performance of three stream processing services, co-located on an Edge device. To provide insights into this operation, we visualize the agent’s understanding of the processing environment and show how its internal model and the SLO fulfillment improve parallel. We complement this with an **introductory poster** [8] for quickly conveying the high-level idea; additionally, we host the **demo application** at a public URL, provide a **video summary** [9] to it, and share an **artifact repository** [10] for revisiting the demo contents.

II. SOLUTION ARCHITECTURE

In the following, we present an architecture for context-aware autoscaling of stream processing services, involving two components: MUDAP and RASK. MUDAP exposes service-specific parameters for fine-grained adjustments of the processing environment, while RASK uses the interfaces for interpreting and optimizing the environment. Later, we visualize the RASK agent’s internal models and show how increasingly accurate world models improve decision-making.

A. Autoscaling Platform (MUDAP)

The MUDAP platform is introduced in Figure 1 in four steps: ① It streams and buffers sensory data (e.g., video frames) at a nearby device, where multiple containerized processing services run. ② The data is processed, e.g., by running video inference. ③ It continuously exports processing metrics to a time-series DB; this includes metrics about service executions (e.g., *latency* or *data quality*) and the associated resources (e.g., *CPU limit*). These variables describe a service’s state space; those variable that can be directly adjusted

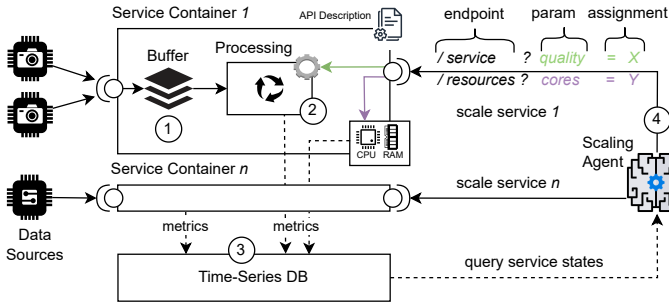


Fig. 1: Architecture of the MUDAP platform [6]: sensor data is ① buffered and ② processed by containerized services; ③ service and container states (i.e., processing metrics) are collected in a time-series DB. Lastly, ④ a scaling agent interprets these states, develops a policy, and adjusts service configurations and their containers through a REST API.

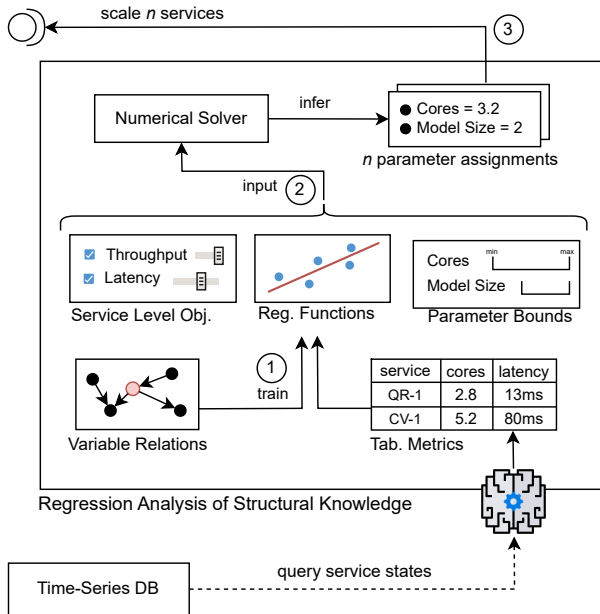


Fig. 2: Conceptual sequence of RASK algorithm [6]: ① create a tabular structure from time-series data and train regression functions; ② supply functions, SLOs, and parameter bounds to numerical solver; ③ optimize parameter assignments for all monitored services and adjust values through MUDAP API.

form the action space. For example, video resolution (i.e., *data quality*) can be scaled dynamically. To invoke actions for a service (e.g., change its *data quality*) we offer a REST API in the container. ④ It optimizes service execution by coupling an agent to these interfaces. This allows arbitrary implementations of autoscalers; in our case, the RASK agent.

B. Autoscaling Agent (RASK)

To optimize the execution of pervasive stream processing services, we present RASK alongside Figure 2 in three steps: ① The agent models the behavior of the processing environment by fitting regression functions, using tabular metrics from the time-series data and domain knowledge about variable

relations¹. This tells the agent *how* its interventions (e.g., changing *data quality* or *resources*) affect processing (e.g., *inference latency*). ② For all services executed on the device, the agent collects the regression models, their SLOs, and the parameter bounds. ③ The agent combines these factors into a global optimization function, uses a numerical solver to infer parameter assignments for all service, and adjusts the values through the API provided by MUDAP.

By using this architecture, autonomous scaling agents can optimize the execution across multiple co-located processing services, thus supporting decentralized and robust operation. In the demo setup we will bring up more details on *how* the agent explores the solution space to develop an accurate understanding of the variable relations.

III. DEMO CONTENTS

The demo contains a multitude of contents that allow the participants to interact with it during the demo session, as well as afterwards. In the following, we present the contents in the same order in which participants should consume them.

A. Introductory Poster

Ideally, participants read the **introductory poster** [8] before switching to the visual demo of the agentic service optimization. Thus, they are already roughly aware of the problem domain, the solution's objectives and architecture, and the organization of the visual demo contents. Also, they can read at their own pace, ask questions that come up, or fall back to the poster when in doubt of the presented visual contents.

B. Visual Demo of Operation

Afterward, participants move to the visual demo of the scaling agent's operation, with three main parts:

1) *Experimental Setup*: The visual demo covers an autoscaling agent that optimizes the execution of three stream processing services: we implement a QR code reader, a Computer Vision (CV) service using Yolov8, and a Point Cloud (PC) mapper; Figure 3 shows an exemplary service output in the top row. To achieve SLO fulfillment, the scaling agent adjusts the resource allocation between services, and each service's parameters. As shown in Table I, variables have viable bounds and a step size. For all three services, the agent can adjust the *data quality* and the allocated CPU resources from a limited global budget of 8 *cores*; the CV service also allows changing the Yolov8 *model size*. Lastly, the *completion rate* of processed items cannot be directly set, but depends on the other variables—this is the domain knowledge we supply to the agent. According to variables' importance to the system operation, we assign SLO thresholds and weights to them.

The demo covers a duration of 600s; in the first 300s, the scaling agent randomly explores the solution space to create an understanding of the processing environment, in the second 300s, the agent exploits the best configurations known. The

¹For simplicity, we supply variable relations according to expert knowledge. However, the relation can equally be extracted through structural learning.

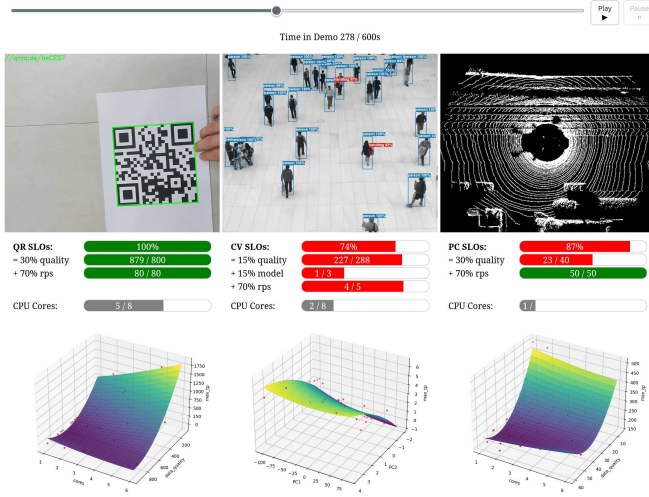


Fig. 3: Snapshot of the visual demo: For three processing services, we display their current service output, their SLO fulfillment, and the regression model that the RASK agent learns through interventions with the autoscaling platform.

TABLE I: Service variables for the QR, CV, and PC service; SLO targets and their weights (i.e., importance) included.

Service	Variable	Descript.	Bounds	Step	SLO	w
QR	<i>cores</i>	CPU quota	(0, 8)	float	—	—
	<i>data quality</i>	Image size	$[10^2, 10^3]$	± 1	≥ 800	0.5
	<i>completion</i>	Rate finish	$[0, 1]$	—	≥ 1.0	1.0
CV	<i>cores</i>	CPU quota	(0, 8)	float	—	—
	<i>data quality</i>	Image size	$[128, 320]$	± 32	≥ 288	0.2
	<i>model size</i>	Yolov8[n./.]	$[1, 4]$	± 1	≥ 3	0.2
	<i>completion</i>	Rate finish	$[0, 1]$	—	≥ 1	1.0
PC	<i>cores</i>	CPU quota	(0, 8)	float	—	—
	<i>data quality</i>	Lidar range	$[6, 60]$	± 1	≥ 40	0.5
	<i>completion</i>	Rate finish	$[0, 1]$	—	≥ 1	1.0

agent operates in cycles—every 10s, it collects metrics, builds the regression models, and infers an autoscaling policy.

2) *Visual Animation*: Figure 3 shows the demo screen: on the top, the current time in the experiment and the control for the playback are displayed. To create a fluent animation, the playback is accelerated by $\times 10$; hence, the entire demo lasts 60s, and every second the configuration and SLO fulfillment are updated. Each column shows the service output, the SLO fulfillment, and the agent’s regression model—its understanding of how to achieve high *completion* by adjusting the available parameters. Interested readers can access the public **demo application** [11] to track the agent’s progress.

3) *Results*: The experimental demo shows how operating the RASK agent for 300s—equivalent to 30 interventions in the environment—suffices to improves global SLO fulfillment from 56% to 98%. This is considerably sample-efficient when compared with contemporary RL approaches, like Q-learning. In the later 300s, the agent keeps stable, high SLO fulfillment by adjusting the parameters along the Pareto front of optimal assignments; this behavior is also tracked in the video.

C. Explanatory Video

The **video summary** [9] contains a technical explanation of the paper’s methodology using the introductory poster, and a showcase of the visual demo progress. This allows conference participants and other researchers to revisit the contents.

D. Artifact Repository

The **artifact repository** [10] (published under CC BY-NC-SA license) invites researchers to reuse the existing processing environment—including the three services—and couple their custom implementation of a scaling agent. Also, to support researchers in creating their own autoscaling platform, it shows how to expose arbitrary scaling parameters for their services.

IV. CONCLUSION

This paper presented a solution for context-aware autoscaling of stream processing applications under strict resource limits. Our approach lets scaling agents explore services’ action space, build a model of the processing environment, and gradually optimize the service execution. Scaling agents run directly on the Edge device, thus increasing the autonomy and robustness of pervasive processing services. We created a visual demonstration of the execution of our scaling agent: in under 30 iterations, it develops an accurate environmental model, fulfilling SLOs of three co-located services over 98%. We accompany the demo with an introductory poster, an explanation video, and an artifact repository—researchers can revisit and extend these contents at their own pace.

ACKNOWLEDGMENT

This work is partially supported by CNS2023-144359 and the European Union NextGenerationEU/PRTR under MI-CIU/AEI/10.13039/501100011033.

REFERENCES

- [1] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, “Vehicular edge computing and networking: A survey,” *Mobile networks and applications*, vol. 26, no. 3, pp. 1145–1168, 2021.
- [2] M. De Donno, K. Tange, and N. Dragoni, “Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog,” *IEEE Access*, vol. 7, pp. 150 936–150 948, 2019.
- [3] Z. Liu, Q. Li, X. Chen, C. Wu, S. Ishihara, J. Li, and Y. Ji, “Point Cloud Video Streaming: Challenges and Solutions,” *IEEE Network*, vol. 35, no. 5, pp. 202–209, Sep. 2021.
- [4] K. Labs. (2023, May). [Online]. Available: <https://kubernetes.io/blog/2023/05/12/in-place-pod-resize-alpha/>
- [5] G. Quattrocchi, E. Incerto, R. Pincioli, C. Trubiani, and L. Baresi, “Autoscaling Solutions for Cloud Applications Under Dynamic Workloads,” *IEEE Transactions on Services Computing*, May 2024.
- [6] B. Sedlak, P. Raith, A. Morichetta, V. C. Pujol, and S. Dustdar, “Multi-Dimensional Autoscaling of Stream Processing Services on Edge Devices,” Oct. 2025. [Online]. Available: <https://arxiv.org/abs/2510.06882>
- [7] B. Sedlak, A. Morichetta, P. Raith, V. C. Pujol, and S. Dustdar, “Towards Multi-dimensional Elasticity for Pervasive Stream Processing Services,” in *2025 IEEE PerCom Workshops*, 2025.
- [8] B. Sedlak, “Percom Demo Introductory Poster.” [Online]. Available: <https://borissedlak.github.io/uploads/poster-percom-demo.pdf>
- [9] —, “Explanation Video for Demo Showcase.” [Online]. Available: <https://borissedlak.github.io/percom-demo-2026/video/>
- [10] —, “Elastic Workbench Artifact Repository.” [Online]. Available: <https://github.com/borissedlak/elastic-workbench/tree/percom-demo>
- [11] —, “Elastic Workbench Public Demo Application.” [Online]. Available: <https://borissedlak.github.io/percom-demo-2026/>