TransScale: Combined-Approach Elasticity for Stream Processing in Fog Environments

Alessio Pagliari, Guillaume Pierre

TransScale: Combined-Approach Elasticity for Stream Processing in Fog Environments

Alessio Pagliari
Univ Rennes, Inria, CNRS, IRISA
alessio.pagliari@irisa.fr

Guillaume Pierre
Univ Rennes, Inria, CNRS, IRISA
guillaume.pierre@irisa.fr

***Abstract*—Real-time data processing is a standard requirement in Fog Computing. Dynamically adapting data stream processing frameworks is an essential functionality to handle time-varying workloads efficiently and to optimize resource consumption. However, horizontal scaling alone, by adapting the parallelism and number of provisioned nodes, faces limits when available compute resources are scarce. We propose TransScale, a combined-approach auto-scaler that combines horizontal scaling to approximation computing, controlling it through transprecision computing. We design TransScale to make the approximation method transparent to the system and support context-specific requirements through QoS-driven re-configuration decisions. Based on the policy's objective, we show that it can reduce re-configuration occurrences, optimize resource utilization and sustain high workloads in resource-constrained environments.**

***Index Terms*—Data stream processing, elasticity, transprecision computing, fog computing**

I. Introduction

The increasing popularity of IoT applications and the consequent large number of connected devices results in massive volume of continuously generated data that needs low-latency real-time processing. To reduce processing latency and congestion, Fog Computing shifts processing in the vicinity of data sources using limited geo-distributed computational nodes [1]. Data Stream Processing (DSP) platforms are often deployed in fog clusters to accommodate the necessity of real-time processing [2], [3]. A non-trivial challenge in fog environments is for streaming platforms to sustain time-variable workloads [4] while being constrained by resource scarcity, as the input streams are susceptible to fluctuations depending on the time of the day (e.g., working hours in a factory) or are subject to sudden spikes (e.g., exceptional short-term events).

Runtime adaptation is a standard technique to adjust DSP systems to workload variations [3]. The main idea is to monitor the execution environment and responsively re-configure the system when a change happens. A very popular approach is horizontal scaling [5], [6], which dynamically adapts the application's tasks parallelism or the number of assigned nodes where to distribute the stream operators.

However, horizontal scaling faces several challenges in Fog Computing. *Firstly*, available resources may be limited. It is common to perform heavy operations in the fog such as ML inference [7] and real-time video analysis [8]. Hence, such heavy operators may require more processing capacity than those provided by a fog node. It is thus likely to reach the cluster limits when scaling a streaming application. *Secondly*, re-configuring a streaming application once deployed is time-consuming as it requires checkpointing the application's state, stopping and re-deploying it with a new configuration. The incoming data are buffered during a re-configuration, waiting for the application to restart to resume processing. These offline times, reduce application availability and increase response latency, a priority for life-critical applications as healthcare monitoring [9]. *Finally*, several constraints may force the application to a static configuration. For instance, the developer's design choices or the use of specific libraries may not support parallel execution. In these situations, the system cannot scale horizontally.

When scaling is infeasible, an alternative solution is to adapt the algorithm, e.g., through approximated computing [10]. Switching to a lower-precision version of the deployed algorithm reduces the resources consumed and speed-up processing at the cost of lower result accuracy. For instance, an application collecting statistics on user usage may lower the computation precision when the number of simultaneous usages reaches unexpected peaks. However, adapting the processing approximation also has its drawbacks. *First*, an overuse of approximated computing may result in significant accuracy degradation. A less-precise computation for critical applications may even be entirely unaffordable (e.g., anomaly detection in health monitoring). *Second*, the choice of approximation methods strictly depends on the application characteristics, and only some applications can support one. For instance, record sampling was proven effective but not applicable for failure detection use-cases [11].

Application- and context-dependent constraints make the choice of the best approach non-trivial. The variability and heterogeneity in Fog Computing and different QoS necessities require combining the two techniques to optimize resource utilization and application performance. There are various examples of the usage of elastic approximation computing for stream processing [3] and fog computing [12]. However, most works focus on a single one of the two techniques [13]. Those proposing a combined approach design it with a fixed approximation method [14], limiting the solution to a specific context. Hence, we need a solution that can: (i) combine both techniques; (ii) transparently support approximation regardless

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-19-CHR3-0002 (DiPET CHIST-ERA).

its implementation; and (iii) choose the suitable technique to automatically re-configure the system based on QoS policies dictated by the context-specificity.

We propose a combined approach which exploits horizontal scaling and approximation to optimize resource utilization and application algorithm adaptation when scaling is infeasible. We exploit Transprecision Computing [15], a novel paradigm built upon the principles of approximation computing, which allows to control *when* and *where* the approximation is applied, to integrate approximation control in an elastic auto-scaler.

We present TransScale, a combined-approach auto-scaler for streaming applications in Fog environments that leverages transprecise components to overcome resource and application limitations. The combined-controller architecture, in addition to infrastructure and parallelism scaling, allows transprecision management. Given the application-dependent nature of approximation methods, we envision the implementation of the approximated code at the operator-level so TransScale may transparently support any application-dependent implementation. We exploit two performance prediction models, for parallelism and transprecision, to optimize the re-configuration phase. TransScale supports implementing different auto-scaling policies defining different precision and resource efficiency tradeoffs. We further investigate two representing scenarios. One focuses on a situation where we have constraints on resource utilization. The second investigates a use case where the bottleneck operator cannot scale. Firstly, we show the advantages of transprecision computing and the combined approach regarding resource optimization and re-configuration times reduction. Secondly, we illustrate how transprecision can overcome parallelism limitations while sustaining peaks in the incoming flow in particular scenarios.

To summarize, we make the following main contributions:

- We introduce a combined-approach auto-scaler that exploits horizontal scaling and transprecision computing to overcome the limitations of fog environments.
- We propose a transprecise auto-scaler that can control the approximation, regardless of the user-defined method.
- We implement a combined performance prediction component to optimize the re-configuration considering both approaches' impact on performance.
- We define five QoS-based scaling strategies to decide the best re-configuration, each optimizing a different metric.

In the remainder of this paper we introduce the background in Section II; in Section III, we present TransScale's design and components; we evaluate it in Section IV; we review related works in Section V; finally, we conclude in Section VI.

II. Background

A. Transprecision Computing

Approximated computing solutions use low-precision processing forms of the applications to speed-up processing and reduce resource consumption by producing less accurate results [16]. These works often propose full approximate algorithm versions, deploying them statically. Elastic solutions exploit algorithm switching [3] to change between those versions in real time by redeploying the entirety of the application [13].

The Transprecision Computing paradigm [15] extends the principles of approximation computing with the key idea to dynamically apply approximation by controlling *"space and time"*, paying with result accuracy loss to achieve energy efficiency and reduce resource consumption. It envisions a control loop system that monitors the runtime job and decides *when* and *where* the approximation is applied. By the former, it intends to control the moments when low-precision should run instead of statically deploying it, i.e., similarly to algorithm switching. By the latter, the paradigm introduces partial approximation of the algorithm, defining different approximated parts of the algorithm which can be switched individually.

As intended, the paradigm includes approximation at hardware and software layers [15]. This paper focuses on the software layer as we implement transprecision at the task level (Section II-B). Transprecise applications can vary depending on the context. In some cases, one may prefer low precision to reduce consumption; in others, results accuracy is a priority. Hence, the control loop can exploit precision adaptation to reduce or increase it. An Intrusion Detection System (IDS) may run in low precision to reduce resource consumption in a Fog cluster and enable higher precision only when anomalies are detected to perform deeper packet analysis. An online machine learning algorithm may need higher precision to train its model and permit it to switch to lower precision when workload fluctuations are not sustainable [17]. How a transprecise framework implements and controls approximation is highly application- and context-dependent.

B. Transprecision for Stream Processing

The introduced control loop system finds a parallel in traditional DSP auto-scalers. They continuously monitors the application performance and, given the current resource availability, decides *when* to scale the application and infrastructure. Re-scaling a streaming application has a significant cost given by the system re-configuration time [18]. Streaming platforms, e.g. Flink [19], must stop the job, re-configure the underlying infrastructure and application, and finally resubmit it; meanwhile, it saves and retrieves the checkpointing state.

We introduce transprecision at the application layer to manage the *where* at the task level, keeping the approximation transparent to the streaming platform. Ways to implement transprecision can vary from computationally lighter and less precise algorithms (e.g., heuristic algorithms) to exploiting data sampling (e.g., load shedding [14]) or adapting system parameters such as window size. Even though its implementation is highly application-dependent, it is safe to assume that runtime algorithm switching does not add re-configuration cost, allowing for seamless application code-switching.

Nevertheless, it is possible to define two main methods to adapt transprecision at the application level depending on the requirements and the application characteristics: *switching* or by *level*. The former, turns on and off the transprecise portion of the code to substitute the high-precision one or vice-versa, replicating common algorithm-switching solutions.

Instead, the latter define the aggressiveness of the transprecise computation. A set of algorithms or the number of sampled records can describe different precision levels, introducing the advantage of adapting the precision weight to sustain higher workloads and postpone eventual horizontal scaling.

C. Gesscale Overview

We implement TransScale (Section III) by extending with transprecision components an existing horizontal auto-scaler, namely Gesscale[20]. Gesscale is a MAPE-based auto-scaler for geo-distributed environments. Its leading characteristics are an empirical performance prediction model [21] to optimize the resource scaling decision and network awareness to optimize placement in geo-distributed Fog environments.

Architecture: Gesscale runs on a container-based infrastructure, where Kubernetes [22] is used to deploy Flink. To let a task use the maximum resources, it places only one Flink task manager container per node. Furthermore, Gesscale extends the Kubernetes scheduler with a custom algorithm to prioritize the nodes with less communication latency to reduce the network impact on application performance (Section III-B1).

Workflow: Gesscale is a throughput-driven auto-scaler aiming to maximize the performance metric. It periodically monitors throughput and backpressure at runtime and uses the two metrics to detect when a rescaling is required. When backpressure is high, it scales up the infrastructure. If there is no backpressure and the current throughput is lower than the Maximum Sustainable Throughput (MST) reachable in the current configuration, it requests a scale-down.

Prediction Model: A key feature of Gesscale is the performance prediction model [21] used to estimate the optimal number of required nodes. When the system is under backpressure, the auto-scaler uses the MST to train the prediction model and scales up by one node. The unit-wise scaling is due to the impossibility of measuring the actual input rate. However, the controller exploits the model to optimize the scale-down phase by estimating the parallelism level required to sustain the current throughput. This way, Gesscale reduces the number of re-configurations during scale-down.

III. TransScale

A. Combined-Controller Design

We design TransScale with a combined-controller architecture (Fig. 1). The extended controller simultaneously considers both parallelism and transprecision adaptation when choosing the required resource configuration, i.e., the couple parallelism and transprecision level targets. We exploit Kubernetes to manage a container-based cluster that hosts the Flink deployment.

1) MAPE Workflow. We adopt the MAPE paradigm. TransScale ***monitors*** the system's metrics for application throughput and backpressure. It ***analyzes*** the retrieved values: high backpressure triggers the scale-up, while low backpressure means a possible scale-down. It exploits the two parallelism and transprecision *performance prediction models* (Section III-B1) to estimate the MST for every configuration. The two models are combined to build the *prediction matrix* (Section III-B2), used

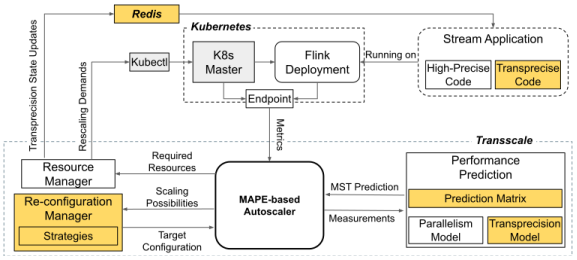


Fig. 1: TransScale Architecture. Yellow components highlight the extensions to support transprecision computing.

to extract the list of configurations that can achieve the target throughput. We define a threshold in addition to the target throughput that the new configuration must exceed to trigger the scaling to avoid a "ping-pong" effect common in step-wise approaches. Moreover, transprecision can reduce infrastructure re-configurations occurrences during sudden bursts.

TransScale ***plans*** the re-configuration by invoking the *re-configuration manager* upon scaling necessity notification. The component receives the possible configurations, i.e., parallelism and transprecision level combinations, and applies the policy defined by the running QoS-based *re-configuration strategy* (Section III-C) to choose the most suitable configuration given application- and context-dependent requirements. Finally, the controller ***executes*** the re-configuration manager's decision. If horizontal scaling is required, it demands the targeted number of nodes through Kubernetes APIs. If there is a change in the Transprecision status, it is updated in a Redis [23] database for the application to query. Depending on the running strategy, both can happen simultaneously.

2) Transprecision Integration. In distributed DSP frameworks, it is impractical to define a globally shared parameter for transprecision state at the application level. A custom record sent to the task requires high queuing times before processing, especially if the system is under backpressure. Instead, we employ a Redis database to store the transprecision state, enabling a faster response to transprecision changes. The tasks queries it to update their internal state when it varies. From a scaling performance perspective, querying the Redis database has no significant impact, given the initial assumption of working in a limited resource environment and hence limited number of tasks.

The application implements transprecision status as a task-level parameter. It impacts the behavior of the processing task, e.g., sampling the data from the incoming flow or changing the processing algorithm to one with different complexity. Implementing transprecision at the task level allows for a seamless adaptation at each status change and transparent support for user-defined approximation methodologies.

B. Performance Prediction

A primary concern of every horizontal auto-scaler is to minimize the number of re-configurations given the cost of such operation. A standard solution is to estimate the number of nodes needed to sustain the current workload during the scale-down phase [20]. In TransScale we use two models, one

for transprecision and one for parallelism, to estimate the MST of every scaling configuration. The predictions are combined to fill the prediction matrix.

1) Performance Prediction Models

Parallelism Model: The following experimentally-validated model [21] predicts the application parallelism (i.e., nodes' number) needed to achieve the required throughput:

$$MST_n = \alpha \times n^{\beta} - \gamma \times ND_{max} \quad (1)$$

where MST_n is the *Maximum Sustainable Throughput* for a given parallelism n and ND_{max} is the largest network distance between the n nodes. The model has three parameters: α, which characterizes the MST of a single node; β describes the Flink parallelization inefficiency; and γ, the impact of network latency on throughput. We derive the values of the three parameters via experimental measurements. Exploiting the complete model (Eq. (1)) requires three measurements, and when those are not yet available, it uses simplified versions of the model fixing the parameters β to 1 with two or less measurements and γ to 0 with only one measurement.

Transprecision Model: Following the previous methodology, we introduce a performance prediction model for transprecision. We exploit the model to estimate the necessary transprecision level for a given operator at its current parallelism to sustain the target throughput. Thus, keeping the number of nodes fixed, the network latency is irrelevant. Accordingly, the model for transprecision will be:

$$MST_t^n = \alpha \times t^{\beta} \quad (2)$$

where MST_t^n is the *Maximum Sustainable Throughput* at parallelism n with a transprecision level t, regulated by the two parameters α and β, respectively the MST without transprecision enabled and the approximation inefficiency. Similar to the previous model, it requires a minimum of two measurements to work. Likewise, we use a simplified version with $\beta = 1$ when only one measurement is available.

2) Performance Prediction Matrix

The prediction matrix is an $N \times T$ matrix, with N and T the maximum parallelism and transprecision levels achievable by the system. It contains the MST values for every parallelism-transprecision's configuration pair. To update the matrix, we use a queue Q of incomplete rows and columns to be predicted. Each entry is a quadruple $E : (res, lvl, \#real, \#pred)$ formed by the resource res, i.e. parallelism or transprecision, its level lvl for which we have to fill the row (or column), and the number of real ($\#real$) and predicted ($\#pred$) measurements for the resource res at level lvl.

It orders the entries by the amount of real measurements, and by predicted values in case of equality, using as many available measurements to maximizes the prediction correctness. Then, it uses the appropriate prediction models to fill the entire row (or column). From experimental results, the predicted values have an average error at most of 4% compared to real measurements. Finally, it provides a list of possible configurations which can achieve the targeted performance to the re-configuration manager, which strategy will choose the one fitting the QoS objectives.

3) Model Calibration and Matrix Update

The system calibrates the models and updates the prediction matrix during the scale-up phase. That is when backpressure limits throughput and the only moment we are sure the application reaches its MST. The auto-scaler keeps the measurements in an $N \times T$ matrix, also used to initialize the prediction matrix. The parallelism model (transprecision is specular) uses all the measurements for different parallelism levels for a fixed transprecision level.

C. Re-configuration Strategies

When TransScale needs re-scaling, the re-configuration manager applies the chosen strategy to decide the best configuration among those computed by the prediction matrix. The user's objectives and SLAs largely depend on the application and execution environment. The combination of horizontal scaling and transprecision forces a trade-off between *resource optimization*, *accuracy*, and *availability*. I.e., using transprecision to reduce resource consumption will decrease processing accuracy; horizontal scaling maintains the accuracy but increases offline times and resources provisioned.

We propose five scaling strategies to cover a range of QoS characteristics, each aiming to optimize a different metric (see Table I). The combined approach aims to benefit the variety of divergent QoS requirements contexts. The first strategy (*Par-only*) represents standard horizontal scaling [20] without transprecision control. It is the base ground for comparison. The four remaining exploit transprecision, applying it based on different QoS conditions. Three (*Par-prio,Transp-prio,Transp-only*) apply a single-resource priority technique, where one between parallelism and transprecision is preferred. The last one (*Res-Prio*) fully exploits the combined controller by performing a *combined-resource* choice towards a QoS objective.

Parallelism-only Strategy (Par-only). This first strategy is representative of standard horizontal scalers' behavior. It only manages parallelism without considering transprecision. The primary use case is when the importance of the results cannot allow a loss in accuracy. In case the application manages the approximation internally, for example if the decision is not based on runtime metrics but rather on computational results, using this strategy the auto-scaler can be used as a horizontal scaler.

Transprecision-only Strategy (Transp-only). In a variety of scenarios, the deployed applications can be heavily resource-constrained. Hence, the user can require minimal resource utilization for cost reasons or in resource-shared clusters where one may want to limit a specific application to prioritize more critical ones. In others, some tasks can require a single instance aggregation. In such a resource-constrained context, automatically enabling transprecision allows to sustain heavier workloads without changing the parallelism. The *transp-only* strategy only controls transprecision, keeping physical resources unaltered.

TABLE I: Re-configuration strategies comparison.

Strategy	Scaling		QoS Characteristics		
	Horiz.	Trans.	Resource Opt.	Accuracy	Availability
Par-Only	✓	✗	○○○○○	●●●●●	●●○○○
Transp-Only	✗	✓	●●●●●	○○○○○	●●●●●
Par-Prio	✓	✓	●○○○○	●●●●○	●●○○○
Transp-Prio	✓	✓	●●●○○	●●○○○	●●●●○
Res-Prio	✓	✓	●●●●○	●●○○○	●○○○○

Parallelism-priority Strategy (Par-prio). Critical applications necessitate high-precision results. However, at the same time, they can require maintaining a low latency response. Hence, when the system reaches the maximum capacity in Fog's node-limited clusters, Transprecision helps to avoid data buffering, keeping low end-to-end latency and reducing the risk of records loss. The *par-prio* strategy favors high-precision computation by prioritizing resource adaptation. When it reaches the maximum available nodes and the input rate increases, it enables and scales transprecision.

Transprecision-priority Strategy (Transp-prio). A standard requirement in Fog Computing is to reduce resource consumption due to their scarcity. To sustain sudden workload peaks, Transprecision scaling postpones resource adaptation. Moreover, some applications prioritize processing availability to reduce offline times at the cost of lower result precision. Transp-prio favors scaling transprecision until a maximum user-defined level, after which it scales horizontally. Consequently, it reduces the number of infrastructure re-configurations and gives a faster response to data bursts.

Other auto-scalers [20] cannot accurately determine the target throughput due to the backpressure limiting at the MST. While TransScale's combined approach enables **scale-up optimizations**. Transp-prio strategy reconfigures transprecision before parallelism, allowing to reach a higher MST closer to the actual ingestion flow. When transprecision reaches its upper limit, the strategy disables transprecision while adding n nodes instead of one at a time. This approach has a twofold advantage: it reduces the low-precision processing time *and* the number of re-configurations.

Resource-Priority Strategy (Res-prio). The previous strategies follow a *single-resource-driven* QoS objective, prioritizing the adaptation of either parallelism or transprecision. *Res-prio*, instead, is a *QoS-driven* strategy aiming for resource consumption minimization. To achieve the objective, the strategy considers combined-resource re-configurations.

During **scale-up**, it favors transprecision to minimize the number of nodes. However, a limitation of transp-prio is the forced reset of transprecision at every parallelism scale-up. The risk is to have a configuration with a lower MST than the currently achieved throughput when reaching the parallelism limit. The Res-Prio strategy avoids such occurrences by scaling to a combined parallelism-transprecision configuration. Instead, to **scale down**, it chooses the configuration with minimum parallelism, whether the transprecision. Thus, it prioritizes its QoS objective rather than a single resource, even if it means re-scaling both resources simultaneously or increasing the number of re-configurations.

IV. Evaluation

A. Implementation and Experimental Setup.

As proof of concept to show the benefits of the combined approach, we implement TransScale in Python, to control and scale the bottleneck task. Its modularity supports the definition of different re-configuration strategies.

We perform our experimentation in a cluster of nine Raspberry Pi 4 nodes, where we emulate a geo-distributed network with latencies from 5 to 200 ms between the nodes, set through the Linux TC command (traffic control). We deploy Flink 1.12, configured with a TaskSlot per TaskManager. We set up six nodes for Flink TaskManagers' deployment (one per node), one for Flink JobManager, one as a Controller node (with the Kubernetes master), and the last one for deploying additional services as Prometheus and Graphana for monitoring. We rely on an external host to deploy the auto-scaler, the Redis database, a Kafka broker to ingest data on the system, and the data producer publishing to it.

B. Evaluation Metrics

We evaluate TransScale using the metrics introduced by the SPEC Research Group [24] to describe Elastic Cloud Systems' performance. We consider only the metrics not exclusively coupled to physical resources, e.g., provisioning accuracy is meaningless when employing transprecision.

Provisioning Cost describes the amount of provisioned resources using *"a simple model where each replica is charged one cost unit per minute of execution"* [20].

Excess Computation Time: in our experiments, the data stream has a finite amount of records which eventually completes processing. If the system cannot keep up with the input rate, records are buffered and not processed in "real-time." The *excess computation time* metric describes the time needed by the system to complete the processing of the incoming records after the producer stops sending new data. Results are normalized by the duration of the measurement period. The lower the excess time, the better.

Number of Re-configurations: the objective of any auto-scaling strategy is to minimize the re-configuration occurrences to reduce offline times. This metric is a count of how many re-configurations the auto-scaler enacts.

Availability: the metric shows how much time during execution the job can process records. It is complementary to the re-configuration number: with fewer re-configurations, the system has more time to process records. *Availability* is the amount of time the application can process input records, normalized by the total measurement time.

These metrics do not evaluate the effect of transprecision. However, we exclude a processing accuracy metric as we do not aim to propose the optimal transprecise application but rather a system that can support many forms of approximation. Instead, we introduce three additional metrics to evaluate the impact of transprecision regardless of the algorithm under test.

Transprecision Timeshare: transprecise execution increases the probability of approximated results. Hence, the less

TABLE II: UC1 Evaluation Metrics.

Strategy	excess time	#reconf	avail.	prov. cost	transp. cost	time-share$_{TP}$	record-share$_{TP}$	proc. records
Par-Only	0.006	22.000	0.856	1133.433	0.000	0.000	0.000	1.000
Par-Prio	0.007	20.800	0.859	1147.093	53.760	0.185	0.264	0.868
Transp-Prio	0.006	3.800	0.970	757.816	326.693	0.798	0.921	0.474
Res-Prio	0.009	26.400	0.810	587.200	308.870	0.683	0.919	0.454

frequently it is enabled, the more accurate will be the final result. The *transprecision timeshare* metric evaluates the period the system runs in a transprecise configuration as the ratio of time running transprecise code to the total measurement time – the lower the value, the less the impact of transprecise computing on the final result.

Transprecision Recordshare: the system commonly enables transprecision during high-rate events. Hence, the same time range equals higher produced records. *Transprecision recordshare* evaluates how many records the transprecise code processes, defined as the ratio of records processed under transprecision to the total amount of processed records.

Transprecision Cost: with different transprecision levels, timeshare and recordshare do not express the weight of the transprecise execution. The *transprecision cost* defines such weight by charging each minute of transprecise execution by the executed level of transprecision – the lower the cost, the lower the impact on the computation precision.

C. Use-cases

We test our auto-scaler in two common uses-cases to show the benefits of transprecise computing in resource-constrained environments. The first one (**UC1**) envision a geo-distributed cluster where it can exploit horizontal scaling and show the different QoS approaches. In the second one (**UC2**), a non-scalable streaming application has bottleneck operator constrained by the design choice of a non-parallelizable library.

1) UC1: Statistical Computation. UC1 uses load shedding [14], a standard and recognized approximation method implementing transprecision through data sampling. We define transprecision levels to represent the sampling frequency: a level t will process one record every t, discarding the rest. We implement a statistical computation application as a characterizing example of stream application where load shedding can be applied, keeping result relevance. UC1 computes the carbon footprint of 2-day trips of New York city taxis [25]. The application outputs the sum aggregate to a Redis DB. We added the 16th Fibonacci number computation to make the task more resource-demanding.

We use **Par-Only** as our baseline, to represent standard resource scaling solutions, as Gesscale [20]. We compare it to **Par-Prio**, **Transp-Prio**, and **Res-Prio**, to cover a range of typical QoS requirements of Fog environments. We exclude *Transp-Only*, as it would not be significant to adapt only the accuracy when the system can exploit horizontal scaling. We instead show its benefits in the more constrained UC2.

2) UC2: Online ML. The second application validates TransScale's flexibility to transparently support approximation methods. Hence, we implement transprecision through an algorithm alternative exploited to lower the computational precision. We comprise only one transprecision level, i.e., an on/off system. We implement the ClusTree clustering algorithm [26] in Flink using the MOA Java library [27]. MOA's sequential API further motivates the deployment of the model trainer on a single node. Transprecision switches the tree descent strategy between *breadth-first* for high-precision, which shows the highest average purity [26], and *depth-first* as low-precision, giving faster processing at the cost of lower purity. The stream application processes RBF data generated following the timestamp trace of 1-day Azure's VM creation events [28].

We leverage **Transp-Only** to sustain workload fluctuations. We evaluate the application in three configurations: high-precision, low-precision, and auto-scaled. We exclude the other strategies as the use case does not support horizontal scaling.

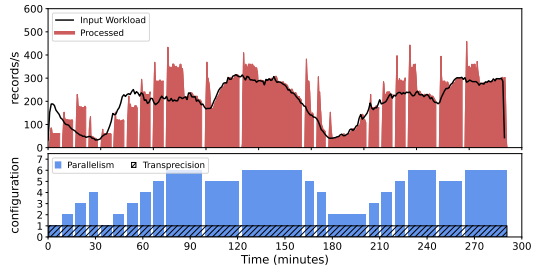
D. Results

UC1: Table II shows the metrics results averaged from five runs, small system variations can trigger differently the auto-scaler. The transprecise-aware strategies show different improvements (Fig. 2). **Par-prio** avoids the re-configuration ping-pong when the workload exceeds the cluster capabilities. **Transp-prio** obtains the most significant improvement in terms of re-configurations, reducing them to less than 5, thanks to the scale-up optimization. It also affects the availability time, where only 3% of runtime is used for reconfiguration compared to more than 14% for the other strategies. The best benefit in terms of resource consumption is given by **res-prio**, as per QoS objective. Paying with the number of re-configurations it achieves almost half of used resources.

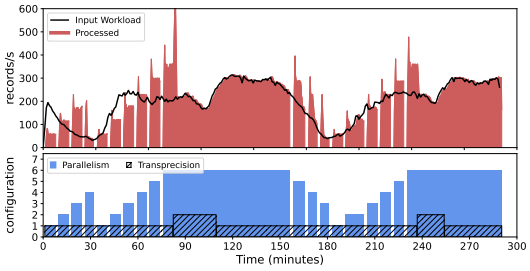
In opposition, we see the **impact of transprecise computing** on processing accuracy. The transprecision cost is significantly higher with transp-prio and res-prio, where the transprecision timeshare and recordshare are more than 68% of execution, up to 79 for transp-prio, and more than 90% of records. We show a detailed comparison in Fig. 3 displaying the processing time and processed records at each transprecision level. With the sampling performed by the transprecise execution, the amount of low-precision processed records is more than 86% of the total input workload for the par-prio strategy and drops below 50% using transp- and res-prio.

As stated in the motivations, choosing a strategy is strictly context-dependent, and the impact on precision derives from the user-defined implementation. However, we show that each strategy optimizes a specific metric targeting a QoS objective. We show that the proposed approach can significantly reduce the re-configuration times, can increase the reaction times to fluctuations, and can significantly reduce the resources used.

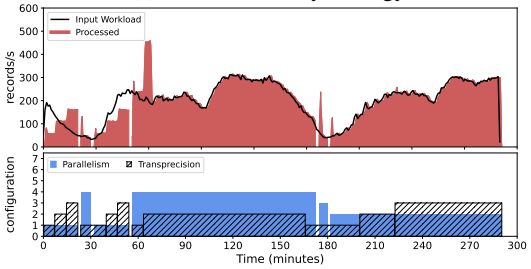
UC2: Table III presents the relevant metrics. The high-precision algorithm (Fig. 4a) cannot sustain the incoming rate, having the excess time to double the total execution



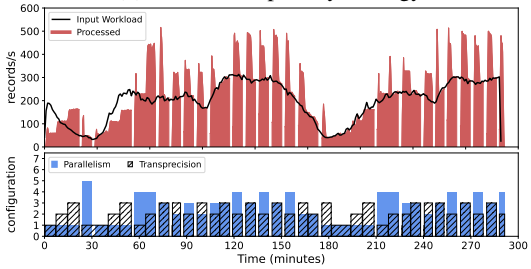
(a) Parallelism-only strategy.



(b) Parallelism-priority strategy.

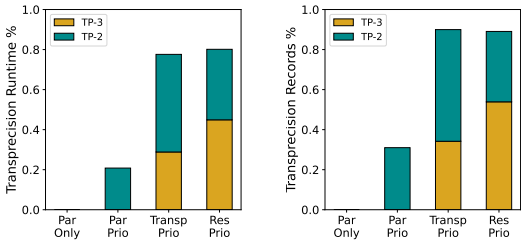


(c) Transprecision-priority strategy.



(d) Resource-priority strategy.

Fig. 2: CarbonFootprint example runs of the four strategies: throughput and resources' configurations.



(a) Time-share. (b) Record-share.

Fig. 3: UC1 processing per transprecision (TP) level.

TABLE III: UC2 Evaluation Metrics.

Configuration	excess time	prov. cost	transp. cost	time-share$_{TP}$	record-share$_{TP}$
High prec.	0.429	419.067	0.000	0.000	0.000
Transprec.	0.008	240.833	240.833	1.000	1.000
Autoscaled	0.005	240.800	53.5	0.221	0.291

time. Therefore, using a single node, it doubles the resource cost. Longer node utilization results in higher costs and energy consumption. Meanwhile, running the low-precision code (Fig. 4b) improves resource utilization by minimizing its consumption and reducing execution time, paying with a high approximation cost. Depending on the user-defined approximation, it can result in non-negligible precision loss.

When we exploit the auto-scaler (Fig. 4c), the application obtains the same provisioning cost as the low-precision configuration, minimizing it. On the other hand, it increases the overall computation accuracy by reducing the low-precision execution to 22% of runtime, and the transprecise cost by almost 80%, impacting 30% of processed records.

We show the benefits of transprecise auto-scaling in static and heavily constrained resource scenarios. It enables the application to sustain input bursts by dynamically adapting the precision, it maximizes the high-precision execution time and eliminates record buffering and long response times.

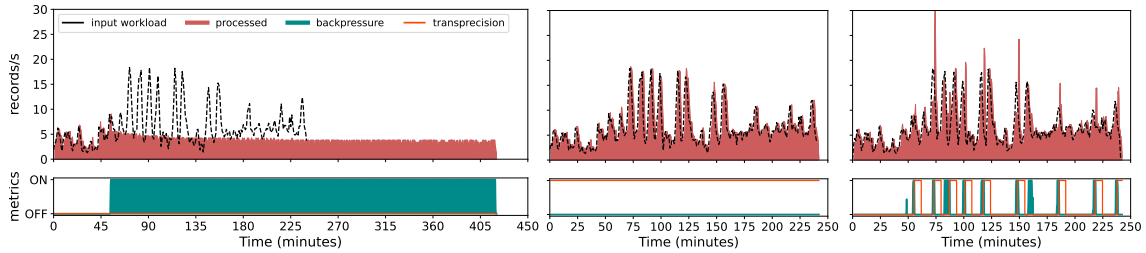
V. Related Work

Streaming systems' runtime adaptation is a widely researched field [2], [3]. Standard solutions either adapt the operator's allocated resources, i.e., vertical scaling [29], or the parallelism of the operators or the physical nodes assigned to the application, i.e., horizontal scaling [30], or a combination of the two [31]. In Fog Computing, it is common to adapt the resources and infrastructure at runtime [20], [32].

While resource adaptation has a vast literature, we focus this section on the approximation state-of-the-art, as the main challenge discussed in this paper is resource scarcity, where horizontal scaling is heavily limited. StreamApprox [10] exploits data sampling, i.e., load shedding, to reduce the computation load and adapt to the input flow fluctuations. Spear [33] implements processing acceleration by an incremental sampling system, adapting the result precision loss to respect SLAs. AWStream [34] propose user-defined degradation functions to approximate the input data. It exploits approximation based on bandwidth utilization to optimize processing latency. These works either focus on Cloud environments [10], [33] or consider offloading to them [34], not directly tackling the Fog computing challenges. Moreover, they do not combine approximation to other elastic techniques.

Qin [13] implements a framework to manage algorithm switching through different application versions. However, they need to stop the application, synchronize the unprocessed data, and redeploy the algorithm's new version. It presents two critical limitations: the approximation granularity is at the application level, and it requires re-configuration times for redeployment. Instead, TransScale implements transprecision at operator-level to benefit a seamless switch at runtime.

Similarly to TransScale, DRS+ [14] combines approximation to resource scaling. While DRS+ defines the approximation as load shedding, TransScale supports transparently user-defined methodologies, avoiding context-specific limitations, e.g. applications that cannot support data sampling [11].



(a) High precision. (b) Low precision. (c) Auto-scaled.

Fig. 4: ClusTree run in different configurations: throughput and system metrics.

Traub et al. [11] assert the context-specificity of adaptive sampling and introduces query-based sampling through user-defined functions (UDFs). Similarly, TransScale introduces user-defined approximation in the operator. While it applies approximation at the sensor level, our work focuses on the DSP platform, envisioning an application-level approximation.

In Fog/Edge, stream processing can exploit adaptive sampling to reduce the network and hence the energy consumption of battery-based edge devices [35], [12]. More generally, approximation is exploited in Fog/Edge environments to optimize edge tasks execution [36], [16]. Approximation improves the energy efficiency of edge devices and speeds up the task execution in computationally-constrained environments.

VI. Conclusion

TransScale is a combined-approach auto-scaler for Fog environments which integrates horizontal scaling and approximation control to dynamically switch the algorithm's precision in response to workload runtime fluctuations. The approach mitigates re-configuration times and resource constraints. TransScale quickly reacts to variations of the incoming flow, reducing offline times and overall used resources. It can also reach higher throughput while minimizing the number of low-precision processed records when it cannot scale horizontally. For future work, we plan to extend TransScale to support the task-dependency of streaming applications.

References

[1] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, 2019.

[2] H. Röger *et al.*, "A comprehensive survey on parallelization and elasticity in stream processing," *ACM Computing Surveys*, vol. 52, no. 2, 2019.

[3] V. Cardellini *et al.*, "Run-time adaptation of data stream processing systems: The state of the art," *ACM CSUR*, 2022.

[4] U. Tadakamalla, "Characterization of IoT workloads," in *Proc. EDGE*, 2019.

[5] F. Lombardi *et al.*, "Elastic symbiotic scaling of operators and resources in stream processing systems," *IEEE TPDS*, vol. 29, no. 3, 2017.

[6] T. De Matteis and G. Mencagli, "Elastic scaling for distributed latency-sensitive data stream operators," in *Proc. Euromicro PDP*, 2017.

[7] L. Zeng *et al.*, "Fograph: Enabling real-time deep graph inference with fog computing," in *Proc. ACM TheWebConf*, 2022.

[8] C. Fathy and S. N. Saleh, "Integrating deep learning-based iot and fog computing with software-defined networking for detecting weapons in video surveillance systems," *MDPI Sensors*, vol. 22, no. 14, 2022.

[9] J. Kaur *et al.*, "Importance of fog computing in healthcare 4.0," *Springer Fog Computing for Healthcare 4.0 Environments*, 2021.

[10] D. L. Quoc *et al.*, "StreamApprox: Approximate computing for stream analytics," in *Proc. ACM/IFIP/USENIX Middleware*, 2017.

[11] J. Traub, "Optimized on-demand data streaming from sensor nodes," in *Proc. ACM SoCC*, 2017.

[12] E. Oliveira *et al.*, "Latency and energy-awareness in data stream processing for edge based iot systems," *Springer Journal of Grid Computing*, vol. 20, no. 3, 2022.

[13] C. Qin, "Quality-aware algorithm switching framework for adaptive stream processing systems," Ph.D. dissertation, Stiftung Universität Hildesheim, 2022.

[14] K. Tang, "DRS+: Load shedding meets resource auto-scaling in distributed stream processing," in *IEEE HPCC/SmartCity/DSS*, 2020.

[15] A. C. I. Malossi *et al.*, "The transprecision computing paradigm: Concept, design, and applications," in *Proc. DATE*, 2018.

[16] A. Younis *et al.*, "Energy-latency-aware task offloading and approximate computing at the mobile edge," in *IEEE MASS*, 2019.

[17] J. Lee *et al.*, "TOD: Transprecise object detection to maximise real-time accuracy on the edge," in *Proc. ICFEC*, May 2021.

[18] S. Vanneste *et al.*, "Distributed uniform streaming framework: an elastic fog computing platform for event stream processing and platform transparency," *Future Internet*, vol. 11, no. 7, 2019.

[19] Apache Flink, https://flink.apache.org/, 2022.

[20] H. Arkian *et al.*, "Model-based stream processing auto-scaling in geo-distributed environments," in *Proc. ICCCN*, 2021.

[21] H. Arkian *et al.*, "An experiment-driven performance model of stream processing operators in fog computing environments," in *Proc. ACM SAC*, 2020.

[22] Kubernetes, https://kubernetes.io/, 2022.

[23] Redis, https://redis.io/, 2022.

[24] N. Herbst *et al.*, "Ready for rain? a view from SPEC research on the future of cloud metrics," *arXiv preprint*, 2016.

[25] B. Donovan and D. Work, "New York City taxi trip data (2010-2013)," Univ. Illinois Urbana-Champaign, Tech. Rep., 2014.

[26] P. Kranen *et al.*, "The ClusTree: indexing micro-clusters for anytime stream mining," *Knowledge and information systems*, vol. 29, no. 2, 2011.

[27] A. Bifet *et al.*, "MOA: Massive online analysis, a framework for stream classification and clustering," in *Proc. workshop on applications of pattern analysis*, 2010.

[28] E. Cortez *et al.*, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. SOSP*, 2017.

[29] T. De Matteis and G. Mencagli, "Proactive elasticity and energy awareness in data stream processing," *Elsevier Journal of Systems and Software*, vol. 127, 2017.

[30] H. Röger *et al.*, "Combining it all: Cost minimal and low-latency stream processing across distributed heterogeneous infrastructures," in *Proc. ACM/IFIP Middleware*, 2019.

[31] V. Marangozova-Martin *et al.*, "Multi-level elasticity for data stream processing," *IEEE TPDS*, vol. 30, no. 10, 2019.

[32] V. Cardellini *et al.*, "Decentralized self-adaptation for elastic data stream processing," *Future Generation Computer Systems*, vol. 87, pp. 171–185, 2018.

[33] N. R. Katsipoulakis *et al.*, "SPEAr: Expediting stream processing with accuracy guarantees," in *Proc. IEEE ICDE*, 2020.

[34] B. Zhang *et al.*, "AWStream: Adaptive wide-area streaming analytics," in *Proc. ACM SIGCOMM*, 2018.

[35] D. Trihinas, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Trans. on Services Computing*, 2018.

[36] W. Yu *et al.*, "Combination of task allocation and approximate computing for fog-architecture-based iot," *IEEE IoT Journal*, vol. 8, no. 9, 2020.