A Multi-Model Predictive Framework for Adaptive Resource Management in Stream Processing Systems

Daniel Wladdimiro, Nicolás Hidalgo, Alessio Pagliari, Luciana Arantes, Pierre Sens, Erika Rosas, Victor Reyes

▶ **To cite this version:**

Daniel Wladdimiro, Nicolás Hidalgo, Alessio Pagliari, Luciana Arantes, Pierre Sens, et al.. A Multi-Model Predictive Framework for Adaptive Resource Management in Stream Processing Systems. Future Generation Computer Systems, 2026, pp.108461. ⟨10.1016/j.future.2026.108461⟩. ⟨hal-05534279⟩

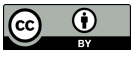
HAL Id: hal-05534279

https://hal.sorbonne-universite.fr/hal-05534279v1

Submitted on 3 Mar 2026

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY 4.0 - Attribution - International License

A Multi-Model Predictive Framework for Adaptive Resource Management in Stream Processing Systems

Daniel Wladdimiro[a,c,*], Nicolás Hidalgo[b], Alessio Pagliari[c], Luciana Arantes[c], Pierre Sens[c], Erika Rosas[d], Victor Reyes[b]

[a]*De Vinci Higher Education, De Vinci Research Center, Paris, France*
[b]*Universidad Diego Portales, Av. Manuel Rodríguez Sur 415, Santiago, Chile*
[c]*Sorbonne Université, CNRS, LIP6, 4 Place Jussieu, Paris, 75005, France*
[d]*DISCA-GRC, Universitat Politècnica de València, Valencia, Spain*

Abstract

Stream Processing Systems (SPSs) are designed to process continuous streams of events, often under highly variable input rates. Although prior work has explored dynamic operator replication, many existing approaches lack generalizability and perform suboptimally across diverse scenarios. In this article, we present MMP-SPS, a predictive self-adaptive framework that extends extends our prior PA-SPS system by introducing a multi-window control loop, support for multiple prediction models, and online model selection via RMSE-based evaluation and multi-armed bandits. Targeting environments with high-volume and fluctuating data streams, such as social media analytics and network traffic monitoring, the framework dynamically selects the most suitable model based on real-time workload characteristics using a reinforcement learning (RL) strategy. To prove the validity of our system, we deployed an implementation of MMP-SPS based on Apache Storm, and we evaluated it on Google Cloud Platform against real-world datasets. Experimental results show substantial improvements in latency, throughput, and resource utilization compared to static and single-model baselines. These findings underscore the potential of multi-model predictive adaptation for scalable and robust stream processing under dynamic conditions.

Keywords: Stream processing, Resource Management, Adaptive Systems, Prediction Models, Reinforced Learning

1. Introduction

The exponential growth of data generated by modern systems, including real-time stock trading platforms, multiplayer online games, and live social media streams (e.g., Instagram, Facebook, X), has created an urgent demand for systems capable of processing continuous data streams efficiently. Stream Processing Systems (SPSs) have become critical infrastructure in finance, cybersecurity, and scientific research, where real-time insights are essential. These systems are designed to maximize throughput while ensuring low latency and high-quality output.

SPSs are typically organized as directed acyclic graphs, in which vertices represent processing operators and edges represent data streams connecting them [1]. To achieve scalability and parallelism, each operator is instantiated with multiple replicas that concurrently process incoming data. Many existing SPS implementations rely on static resource allocation strategies, where the number of replicas per operator is fixed at system initialization. While effective under stable workloads, such static configurations are inadequate in dynamic environments characterized by volatile data rates. For example, surges in trading activity during market volatility, spikes in social media interactions during live events, or traffic bursts during security incidents can overload operators, resulting in processing delays, increased latency, or event loss. Similarly, in post-disaster scenarios, the influx of emergency messages and social media posts can significantly elevate data traffic, making timely processing vital for effective response. Conversely, excessive allocation during low-activity periods results in suboptimal resource utilization.

To address these challenges, various approaches have been proposed for runtime adaptation in SPSs, including techniques that employ machine learning and predictive modeling to improve adaptation accuracy [2, 3, 4]. However, streaming applications often operate across heterogeneous environments with highly variable traffic patterns. Traffic spikes do not follow consistent patterns in frequency or intensity across applications or deployment contexts. Consequently, a single predictive model trained for one scenario may generalize poorly to others. This observation motivates the need for a multi-model strategy that enables dynamic selection of the most suitable predictive model in response to evolving data characteristics.

We propose an adaptive replica management framework that extends our prior single model and fixed control window work PA-SPS [5] by introducing (i) a multi-window control loop, (ii) support for multiple prediction models, and (iii) online model selection strategies based on RMSE and Multi-Armed Bandits (MAB). The framework incorporates multiple predictive models, including: linear regression, Fast Fourier Transform (FFT),

*Corresponding author.
Email addresses: daniel.wladdimiro@devinci.fr (Daniel Wladdimiro), nicolas.hidalgoc@mail.udp.cl (Nicolás Hidalgo), alessio.pagliari@lip6.fr (Alessio Pagliari), luciana.arantes@lip6.fr (Luciana Arantes), pierre.sens@lip6.fr (Pierre Sens), esrosoli@upvnet.upv.es (Erika Rosas), victor.reyes1@mail.udp.cl (Victor Reyes)

artificial neural networks, random forests, Bayesian models, stochastic gradient descent (SGD), ridge regression, and support vector machines (SVMs). Supported by an MAB strategy, the framework dynamically selects the most appropriate model for forecasting input rates based on real-time characteristics of the incoming data stream, enabling fine-grained adjustment of operator replicas. Furthermore, the prediction mechanism employs an asynchronous multi-time-window strategy to improve accuracy and scalability under dynamic conditions.

The key contributions of this paper are:

1. An adaptive replica management system that integrates multiple predictive models for dynamic replica allocation, guided by an online selection strategy.
2. An asynchronous multi-time-window approach to provide accurate input rate predictions.

We implemented our framework on top of Apache Storm [6], extending the adaptive version introduced in [5]. We evaluated it using real-world datasets, including Twitter data streams and DNS traffic traces, deployed on the Google Cloud Platform (GCP). Experimental results demonstrate significant improvements in throughput, latency, and resource utilization when compared to baseline configurations. Based on these foundations, we present detailed results and comparisons that show the system's ability to dynamically adapt to complex, high-velocity data environments.

The remainder of this paper is organized as follows. Section 2 presents foundational concepts related to Stream Processing Systems. In Section 3, we introduce our proposed framework, providing both a theoretical overview and implementation details. Section 5 describes the experimental setup and evaluates the framework's performance using real-world datasets. Section 6 reviews relevant literature on predictive modelling and adaptive techniques in stream processing. Finally, Section 7 concludes the article and discusses directions for future research.

2. Stream processing systems

Stream Processing Systems (SPSs), such as Storm [6], Spark Streaming [7], Flink [8] and Kafka Streams [9],are designed to process large volumes of data in real time. These systems typically abstract applications as Directed Acyclic Graphs (DAGs), where directed edges represent the flow of data between operators, and each node (i.e., operator) performs core processing functions (e.g. counting, filtering, and aggregation) on the data incoming from the upstream flow and forwarding the results downstream to the next operator. In Fig. 1 we illustrate an example DAG, featuring two source nodes that ingest input data and two operators, O_A and O_B, where O_A is replicated three times.

To support scalability and improve throughput (number of messages processed per unit of time), operators can be parallelized and distributed via replication, enabling concurrent processing across multiple instances. Operators may be either stateless, processing each event independently, or stateful, maintaining persistent state to incorporate historical context into current operations. This distinction is crucial in scenarios requiring state-aware logic and consistency.

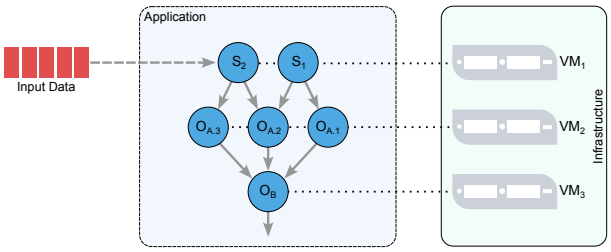


Figure 1: Example of SPS architecture.

The processed data is typically structured as key-value tuples and partitioned among replicas using different strategies, the most common are key-based partitioning, to assure the same key is sent always to the same operator instance, and load balanced partitioning, to equally distributed the keys over the instances. However, naive or random allocation strategies may lead to workload imbalance, where certain replicas are overloaded. To mitigate this, advanced partitioning techniques, including hash-based, partial-key, and executor-centric methods, have been developed to improve load balancing and scalability in high-traffic environments [10, 11].

For execution, the logical DAG must be mapped onto physical resources—such as machines, CPU cores, or virtual environments—within distributed infrastructures like clusters or cloud platforms. Figure 1 illustrates a mapping example of the DAG to a cluster of virtual machines (VMs). Scheduling algorithms must consider both the computational complexity of operators and the heterogeneity of resources. Even in seemingly homogeneous environments, disparities in the distribution of operators can arise due to differences in computational demands [12].

3. MMP-SPS Overview

This section introduces MMP-SPS, an adaptive stream processing system designed to efficiently manage high-throughput, fluctuating data streams with both agility and scalability. The system dynamically adjusts the number of logical processing resources allocated to a running application based on real-time analysis of the input stream behavior.

To anticipate workload variations, MMP-SPS continuously monitors operator-level performance metrics and employs a predictive mechanism to forecast incoming traffic rates. This mechanism is based on a multi-model prediction strategy complemented by a model selection process.

The use of multiple prediction models enhances forecasting accuracy by leveraging diverse modelling techniques suited to varying traffic patterns. Additionally, a multi-windowed MAPE model is implemented to support concurrent execution of prediction models over different observation intervals. This design provides finer control over metric sampling, model evaluation frequency, and the scaling granularity, enabling the system to respond more precisely to dynamic workload conditions.

This proposal builds upon the architecture of PA-SPS introduced by [5], which already extends Apache Storm with two key enhancements: a replica pool and a load-aware event grouping strategy. The replica pool aims to reduce the reaction time of

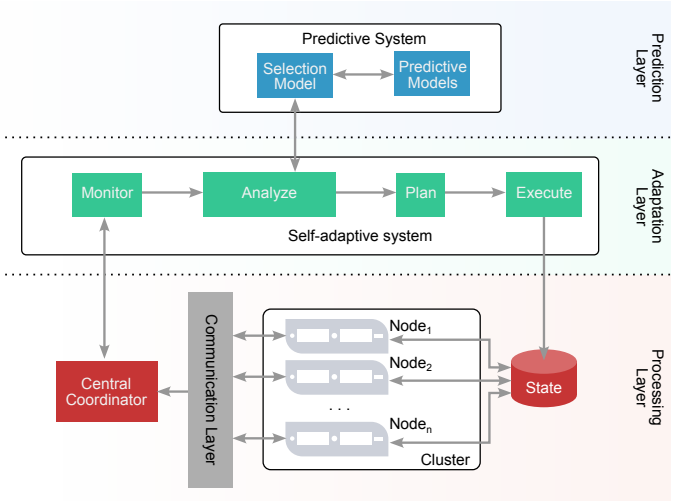


Figure 2: MMP-SPS's Architecture.

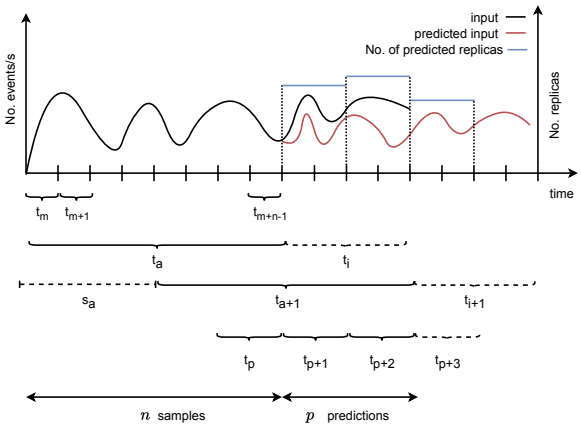


Figure 3: Example of the time windows scheme implemented by the adaptation layer

adaptation mechanisms, thereby minimizing event processing latency by instantiating idle operator instances. Meanwhile, the load-aware grouping strategy distributes events among active replicas based on their current load, improving overall system balance and throughput.

MMP-SPS improves PA-SPS in three main ways. First, it replaces the sequential MAPE loop, where prediction and adaptation were tightly coupled within a monolithic controller, with a layered, asynchronous, multi-window MAPE architecture in which prediction, model selection, and adaptation are implemented as distinct components. Multiple prediction–decision windows can progress concurrently and are evaluated independently, which improves responsiveness to rapid workload changes and enables finer attribution of decisions to individual windows. Second, at the prediction layer, MMP-SPS replaces the single-model approach with a multi-model framework equipped with alternative selector strategies. In addition to a simple RMSE-based selector, it provides Multi-Armed Bandit (MAB) selectors, including both offline and online variants, which adaptively choose the most suitable predictor under changing workload regimes. Finally, MMP-SPS expands the predictor set to include SVM, SGD, Bayesian, and Ridge models, improving robustness across heterogeneous workloads. Together, these advances provide a unified and more resilient elasticity framework that enables stronger adaptation and performance under highly variable real-world data streams.

The proposed architecture, shown in Fig. 2, is organized into three functional layers: the processing layer, the adaptation layer, and the prediction layer.

3.1. Processing layer

The processing layer manages the deployment and execution of the stream processing application. A major limitation of many Stream Processing Systems (SPSs) is their inability to scale operator replicas without requiring an application restart, leading to downtime and service disruption. To address this, the approach in [5] introduces a strategy that dynamically activates and deactivates pre-allocated operator replicas at runtime. While active replicas consume computational resources proportional to the workload, inactive replicas have been shown to incur negligible CPU usage [13].

Furthermore, the implementation incorporates a specialized event grouping strategy. We adopt the Load-Aware Grouping (LAG) mechanism proposed in [3], which considers the load status of each replica over a sliding time window. This method facilitates a more balanced distribution of incoming events among the available replicas, improving processing efficiency and responsiveness under variable load conditions.

3.2. Adaptation layer

The adaptation layer is responsible for the runtime reconfiguration of processing resources. It employs a multi-time window MAPE (Monitor, Analyze, Plan, Execute) control loop to manage adaptations dynamically. Each of the four MAPE stages is executed asynchronously at distinct time intervals, enabling responsiveness to both short-term spikes and long-term trends in input data rates.

Given the inherently fluctuating nature of streaming workloads, accurate prediction of input rates is essential for timely and effective adaptation. To this end, our approach incorporates a multi-window MAPE model in which the four control stages operate as independent modules, each tuned to a specific observation window (represented in Fig. 3). This layered design enables fine-grained resource management, ensuring that the application remains responsive and accurate under diverse and evolving traffic conditions.

For the rest of the paper, we will use the notation t_x to refer to a specific window (i.e., a window iteration) and td_x to its duration, where x specifies the module of the window. The parameter notations are summarized in Table 1.

Monitor

This first module is responsible for collecting the necessary metrics (i.e., latency, input, output) to analyze the system status and adapt the number of resources (i.e., number of active replicas). The module triggers a collection every time window t_m

Parameter	*Description*
O_i	operator i
t_m	time window of the *Monitor* module
td_m	time window size of the *Monitor* module
t_a	time window of the *Analyze* module
td_a	sliding time window size of the *Analyze* module
s_a	slide interval of the *Analyze* module
t_i	time window of the prediction input rate
td_i	time window size of the prediction input rate
t_p	time window of the *Plan* module
td_p	time window size of the *Plan* module
n	number of samples used for the predictive model
p	number of predictions performed by the predictive model
$\widehat{\lambda}(t+1)$	predicted number of events to process by O_i during $t+1$
et_i	average execution time of one event by O_i
$r_i(t+1)$	number of active replicas of O_i computed at the end of t

Table 1: Parameters notation and their description.

(see Fig. 3). Unlike traditional MAPE systems, in our approach, the monitor phase does not automatically trigger the analysis phase; instead, the analysis phase is triggered based on its own time window interval. Each data collection acts as a sample, with multiple samples being used as an input to predict the input rate during the analysis phase.

The integration with the data processing system's monitoring interface enables real-time access to system metrics. A query retrieves the current state of the system, coordinated by a central component that monitors processing nodes and collects statistics from operator replicas. This integration provides centralized access to detailed performance data, including metrics such as the number of events processed by each operator, average execution time per event, and queue lengths.

Analyze

Following the metric collection, the *Analyze* module will evaluate those samples to predict the future behavior of the input data stream. This analysis is performed by a sliding time window t_a with a slide interval s_a, a window size td_a and with $td_a > td_m$. This module exploits also the prediction layer (Fig. 2) to detect patterns that might signal upcoming changes in input rates or the states of active replicas in the pool.

It uses the last n samples collected by the monitor module to run predictive models on the input stream, forecasting behavior over the next time window t_i (Fig. 3). The time window t_i is composed by p predictions made by the prediction module, where $td_i = p \times td_m$ and $td_i < td_a$. In summary, with $s_a = td_i$ and $td_a > td_i$, the sliding window acts as a forward-moving mechanism that sequentially covers prediction intervals.

Fig. 3 illustrates the time window t_a at which the input prediction is conducted. At each t_a, the *Analyze* module sends the last n samples collected during each time window t_m to the prediction layer, which utilizes these datasets for the prediction models. The models will then generate the forecasted data stream for time window t_i. This forecast will be used to determine the necessary number of active replicas for each streaming operator.

The system's modular design allows for the deployment or updating of multiple prediction models within the prediction layer without affecting the adaptation layer. This configuration facilitates the simultaneous introduction of various approaches, including machine learning algorithms, time series forecasting, and statistical methods that use regression to predict input rate behavior. Details about the models included in our solution can be found in Section 3.3.

Once the predicted input rate is obtained from the prediction layer, the *Analyze* module calculates the number of predicted events for the next time window t_i, denoted as $\widehat{\lambda}_i$ [5]. This value is computed based on three components: (1) the predicted input rate for time window t_i, (2) the dependencies among operators, and (3) the queue size of the operators during t_a.

The first component is derived from the *Analyze* module, which utilizes predictive models to forecast future input rates. The second component comes from the topology of the streaming application, represented as a Directed Acyclic Graph (DAG) where operators are interconnected, and data streams flow through a sequence of these operators. Consequently, an increase in the processing capacity of an upstream operator directly influences the number of events that the downstream operator must process.

Finally, the third component considers the queued events that still need to be processed from the previous time window. Since the system may scale up in response to performance issues, not all received events may be fully processed within the adaptation time of that window. To prevent data loss, these events are queued by the streaming system and must be addressed in the upcoming time window. This consideration is essential for determining the number of replicas needed.

Plan

This module determines the scaling policy for the next planning time window $t_p + 1$ at the end of each t_p. Based on the behavior of the predicted input rate, the module selects the optimal number of active replicas for each operator O_i. For every planning time window t_p, it takes into account the prediction from the last valid time window t_a, as provided by the analysis module, which corresponds to the time window t_i illustrated in Figure 3.

The number of active replicas r_i for an operator O_i for the $t_p + 1$ window is computed using Eq. (1) [5]. The calculation of $\widehat{\lambda}_i(t_p+1)$ is carried out by the *Analyze* module, and it is important that the duration of the time window satisfies $td_p \leq td_i$ in order to accurately determine the number of active replicas within the predicted range.

$$r_i(t_p + 1) = \frac{\widehat{\lambda}_i(t_p + 1) \times et_i}{td_p} \tag{1}$$

The scaling policy is established using the procedure outlined in Algorithm 1. The pseudocode illustrates how the *Plan* module determines whether an operator O_i needs to adjust the number of active replicas. The method *computeReplicas*, which calculates the necessary number of active replicas, is executed based on Eq. (1).

The method *getReplicas(O_i)* retrieves the current number of active replicas for the operator O_i. This information serves as a basis for comparing the current number of replicas with the predicted amount. If the prediction indicates an impending increase in workload for the next time window, the *Plan* module will activate additional replicas for that operator, thereby enhancing

its processing capacity. Conversely, if the prediction forecasts a decrease in input rate, the number of active replicas will be reduced by deactivating those that are no longer necessary, thus conserving computing resources.

Algorithm 1 Adaptive Plan algorithm for operator O_i [5].

Require: Statistics Operator O_i in time window t_p.
Ensure: Modifying the current number of active replicas of operator O_i.

```
r_i(t_p + 1) ← computeReplicas(λ̂_i(t_p + 1) , et_i, td_p)
k_i ← r_i(t_p + 1) - getReplicas(O_i)
if k_i > 0 then
    Add k_i active replicas to O_i
else if k_i < 0 then
    Remove k_i active replicas from O_i
end if
```

Execute

The last module is responsible for implementing changes in the number of active operator replicas, as determined by the plan module. Once this module decides whether to increase or decrease the number of replicas of a specific operator O_i, the execute module carries out these changes within the system.

While the previous modules operate in parallel across different time windows, the execute module is the only one that depends on the output of the preceding plan module. As a result, it is automatically triggered after each execution of the plan at each time window t_p.

To manage the number of active replicas in a distributed environment, the system employs a dedicated state component. This module acts as a centralized coordination mechanism, maintaining the current state of all replicas across the distributed cluster. As shown in [13], the overhead introduced by this component is minimal, having negligible impact on the system's overall resource consumption while enabling efficient and consistent replica management.

3.3. Prediction layer

The *prediction layer* focuses on forecasting the incoming data rates to guide the adaptation process. By decoupling prediction from the adaptation logic, the system reduces overhead and enhances responsiveness. The modularity allows us to run in parallel multiple prediction models and choose the most suitable for each prediction window.

The ability to accurately predict the input rate enhances the system's adaptability regarding application performance and resource management. A prediction model can be tailored to specific applications and contexts, ensuring optimal accuracy for that particular use case. However, input data flows can vary significantly across different contexts, and unexpected events, such as those encountered in disaster response, can create fluctuations that the model may not be able to recognize. Therefore, employing multiple prediction models in parallel increases the range of possibilities for improving prediction accuracy.

Since the input stream may vary significantly depending on the environment of our application, it is essential to have a variety of predictive models that can adjust to different patterns in traffic stream. By having multiple predictive approaches, the system is able to select the most appropriate model for each stream behavior, thus increasing adaptability and predictability in the calculation of resources assigned to each operator. Hence, in each time window t_a the *Analyze* module sends the last n samples and triggers the predictions, obtaining as return the predicted input rate for the next time window according to the selected prediction model. The prediction model is chosen by the selection model, which is executed every two t_a windows.

To make a prediction, it is necessary to gather a number of n samples. The prediction model returns a set of p predictions using the data set of the n samples submitted for training, both represented by discrete variables. In our work, the number of samples (n) is 30, so as to be statistically representative [14, 15] and not degrade the response time of the adaptation. The number of predictions (p) is set to 10 to balance the prediction horizon and the accuracy of the model, thus minimizing the loss of reliability [16]. This limit captures short-term data flow variations with minimal estimation deviation, which is essential for real-time processing in restricted sample windows.

In our solution we integrate nine different prediction models to study their behavior under different traffic conditions. Based on the strict requirements of latency and throughput, we have only selected simple prediction techniques that we briefly describe in the following.

- *Basic*: Consider that the behavior of n samples will be the same with respect to p predictions, based on the assumption that the current time window behavior will equal the next time window. This model was used in [3].

- *Linear Regression*: It is a statistical technique used to model the relationship between a dependent variable and one or more independent variables [17]. It assumes a linear relationship between the variables, where the dependent variable can be expressed as a linear combination of the independent variables. The objective of linear regression is to estimate the coefficients (parameters) of this linear model by minimizing the sum of the squared differences between the observed values and the predicted values (i.e., minimizing the residuals).

- *Fast Fourier Transform*: This method performs prediction by transforming time domain data (i.e., the sequence of input rates over time) into the frequency domain [18]. In this context, the input rate refers to the frequency with which events or data points arrive at the system in a given time period. The key idea is to identify periodic patterns or regular fluctuations in the input rate that are not immediately apparent in the time domain, but are evident in the frequency domain.

- *Artificial Neural Network*: The Multilayer Perceptron (MLP) is a type of artificial neural network that learns the relationship between past and future input rates [19].

MLPs are particularly suitable for capturing complex, non-linear patterns in data, making them effective for predictions in scenarios where the input rate exhibits irregular or fluctuating behavior that simpler models might not capture. Once trained, the MLP can be used to predict future input rates based on new data. The non-linear transformations in the hidden layers allow the MLP to capture more complex patterns than traditional linear models, making it well-suited for unpredictable or erratic input rate patterns.

- *Random Forest*: It is a machine learning technique that leverages on an ensemble of decision trees [20]. Each tree is trained on different subsets of the data, and their outputs are averaged to produce a more robust prediction. Random Forest Regression excels at capturing complex, non-linear relationships in the data, making it particularly useful when input rates exhibit irregular or unpredictable behavior.

- *Bayesian*: Bayesian regression is a statistical approach that enhances traditional regression frameworks by incorporating prior distributions, reflecting previous knowledge or assumptions about the model parameters. Unlike ordinary least squares regression, which estimates parameters as fixed values by minimizing residual errors, Bayesian regression treats these parameters as random variables, assigning them probabilistic distributions, known as priors. These priors, when combined with the likelihood of observed data, yield a posterior distribution through Bayes' theorem, allowing for more flexible parameter estimation and the incorporation of uncertainty into the model.

- *Stochastic Gradient Descent*: Stochastic Gradient Descent (SGD) regression is a linear model that uses the iterative optimization technique known as stochastic gradient descent to minimize a loss function, usually the mean square error (MSE) in regression tasks [21]. SGD is a highly efficient and scalable method for input rate prediction, particularly suited for real-time applications where large volumes of data need to be processed. By iteratively adjusting model parameters based on small subsets of the data, SGD can quickly learn patterns in input rate fluctuations and make accurate predictions.

- *Ridge*: This proposal is a linear regression technique that addresses the problems of multicollinearity and overfitting by introducing a regularization term in the loss function [22]. This regularization penalizes large coefficient values, helping to reduce them towards zero, which improves the generalisability of the model. In the context of input rate prediction, Ridge regression can be used when there are multiple input features (such as past input rates or related time series variables) and the objective is to ensure that the model does not over-fit the training data, especially when these features are highly correlated.

- *Support Vector Machine*: Support Vector Machine Regression (SVR) is a regression technique that extends the principles of Support Vector Machines (SVM) to predict continuous outcomes [23]. Instead of minimizing the sum of squared errors, SVR attempts to fit a model within a tolerance range. Data points within this margin do not contribute to the loss function, making SVR robust to outliers and noise in the data. The ability to customize the kernel function and control the flexibility of the model makes it a versatile option for dynamic and unpredictable input rate behavior.

Since the best choice of a predictive model in a given interval time can depend on a large amount of variables, such as the type of application or flow behavior, we propose a selection mechanism that chooses the most suitable predictive model at runtime. To this end, we have studied two selection models based on the *Root Mean Square Error or RMSE* and *Multi-Armed Bandit or MAB*.

3.3.1. RMSE model

This model employs the root-mean-square error (RMSE) as the performance metric to evaluate and select the most suitable prediction technique for each time window. Specifically, the system selects the predictive model that yields the lowest RMSE over the recent history of prediction intervals t_a. By continuously monitoring and comparing the estimation accuracy of available models, the system dynamically switches to the model with the best performance in terms of minimizing prediction error. To enable RMSE-based selection, all candidate predictors are executed in parallel over each window, allowing their errors to be measured against the observed ground truth. However, only the predictor with the lowest RMSE is used to produce the operational forecast for that window.

This adaptive model selection strategy is particularly effective in the presence of fluctuating or non-stationary input stream behavior, where different predictive models may exhibit varying accuracy depending on the underlying data patterns. By selecting the model with the lowest observed RMSE during recent time windows, the system dynamically adapts to changing data characteristics and minimizes forecasting error. Reducing prediction error is crucial for maintaining the performance of stream processing operators and ensuring efficient resource utilization, particularly under conditions of workload variability.

3.3.2. Multi-Armed Bandit model

The Multi-Armed Bandit (MAB) is a Reinforcement Learning (RL) technique that excels in dynamic environments, such as machine learning and traffic prediction [24]. MAB is designed to adapt and learn from its environment, striking a balance between exploration and exploitation. A key advantage of RL models, such as the MAB algorithm, is their ability to make decisions without requiring prior knowledge of the environment or model performance. This allows them to adapt effectively in dynamic and uncertain conditions.

In a MAB scenario, an agent is presented with several options (or arms), each associated with an unknown reward distribution. The goal is to maximize the cumulative reward over time by choosing which arm to select at each step. Despite the agent's lack of knowledge about the reward associated with each

action, denoted as the value of the action, MAB's adaptability allows it to thrive in such uncertain conditions.

In our k-armed bandit problem, each k option (or action) has an expected value given by the action selected. We denote A_t, an action selected in a time step t, and the reward obtained as R_t. The value of an arbitrary action a in time step t is then denoted by $q_*(a)$, representing the expected reward given that a is selected:

$$q_*(a) = \mathbb{E}[R_t | A_t = a] \tag{2}$$

Assuming that we do not know the action values with certainty, we may have estimates. We denote the estimated value of an action a at time step t as $Q_t(a)$. Where $Q_t(a)$ is close to $q_*(a)$.

Exploration and exploitation are key concepts for MAB. Notice that exploitation attempts to maximize the reward, while exploration attempts to produce the greater total reward in the long run. When maintaining estimates of the action values, at any time step, there is at least one action whose estimated value is the greatest (the greedy action). When one of these action values is selected, MAB exploits the current knowledge of the actions. If instead, a non-greedy action is selected, then we said that MAB is exploring as a consequence of improving estimates of the non-greedy action's value.

To control the exploration and exploitation trade-off, we apply classical methods: epsilon-greedy (ϵ-greedy) and Upper Confidence Bound (UCB):

- *ϵ-greedy*: The ϵ-greedy algorithm is a simple yet effective approach to balance exploration and exploitation. At each time step, with a small probability ϵ (i.e., 0.1), the algorithm explores by randomly selecting a prediction model (or arm) to test, regardless of its previous performance. With a high probability of 1 - ϵ, it exploits by choosing the model that has historically delivered the highest reward (i.e., the best traffic prediction). The value of ϵ controls the degree of exploration: a higher ϵ means more frequent exploration, while a lower ϵ emphasizes exploitation. The computational efficiency of this method and the ease of implementation make it a popular choice for real-time applications like dynamic traffic prediction, providing reassurance about its practicality.

- *Upper Confidence Bound (UCB)*: The UCB algorithm improves upon the exploration-exploitation trade-off by using statistical confidence intervals. Select the prediction model based on its past performance and considers the uncertainty (or confidence) around that performance. Specifically, UCB selects the arm with the highest upper bound on its estimated reward, where the upper bound is calculated as the sum of the empirical mean reward and a term that grows inversely with the number of times that arm has been chosen.

  The key idea behind UCB is to select actions based not only on the average reward obtained so far but also on how uncertain the estimate of that reward is. For each action, the UCB algorithm selects the action that maximizes the following expression:

$$A_t = \arg\max_a \left(Q_t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}} \right) \tag{3}$$

  Where:

  - $Q_t(a)$: is the estimated mean reward for action a.
  - $N_t(a)$ is the number of times the action a has been chosen prior to time t.
  - c is a constant that controls the degree of exploration.
  - t is the total number of times any action has been chosen,

  The second term in the equation, $c \cdot \sqrt{\frac{\ln t}{N_t(a)}}$, grows larger for actions that have been chosen less frequently, encouraging exploration of those actions. The first term, $Q_t(a)$, ensures that actions with higher rewards are preferred. This balance enables UCB to explore highly uncertain actions while systematically favoring those that perform well [25].

  This approach is beneficial in reinforcement learning environments, where collecting data from underexplored actions can lead to better long-term performance. UCB is particularly effective in environments with nonstationary rewards, such as dynamic traffic prediction or stream processing systems, where optimal strategies may change over time.

In our MAB implementation, we have used a k-arm bandit model, where the arms represent the predictors to evaluate. Initially, each arm has the same probability to be chosen. We implement both exploration and exploitation techniques: *ϵ-greedy* and *UCB*.

We have applied an α parameter to track non-stationary behavior. In this case, it makes sense to give more weight to recent rewards rather than to long-past rewards.

The reward model implements a weighted linear combination of normalized performance metrics, in which each arm (predictor) is evaluated based on three distinct metrics. The model incorporates two key components: reward computation and adaptive weight optimization.

The reward associated with selecting action a at time step t is defined as:

$$R_a(t) = w_{a,1}(t)\,(1 - m_{a,1}(t)) + w_{a,2}(t)\,(1 - m_{a,2}(t)) + w_{a,3}(t)\,m_{a,3}(t) \tag{4}$$

where:

- $m_{a,1}(t)$, $m_{a,2}(t)$, and $m_{a,3}(t)$ denote the normalized performance metrics, scaled to the range $[0, 1]$ using min–max normalization;
- $w_{a,1}(t)$, $w_{a,2}(t)$, and $w_{a,3}(t)$ are the corresponding weighting coefficients, subject to the constraint $\sum_{j=1}^{3} w_{a,j}(t) = 1$;
- the accuracy loss metric ($m_{a,1}$) and the computational cost metric ($m_{a,2}$) are inverted as $(1 - m_{a,j})$, since lower values indicate better performance;

- the stability metric ($m_{a,3}$) is used directly, as higher values correspond to better performance.

This reward $R_a(t)$ is then used to update the Q-value estimate:

$$Q_{t+1}(a) = Q_t(a) + \alpha(R_a(t) - Q_t(a)) \tag{5}$$

where $\alpha = 0.1$ represent the learning rate for Q-value updates. As $N_t(a) \to \infty$, we have $Q_t(a) \to q^*(a)$.

The model employs online gradient ascent to continuously adapt the weights for each action based on observed performance. After each action selection, the weights are updated using:

$$\mathbf{w}_a(t+1) = \Pi_{\mathcal{W}}\left[\mathbf{w}_a(t) + \alpha_w \cdot (R_{\text{target}}(t) - R_a(t)) \cdot \mathbf{m}'_a(t)\right] \tag{6}$$

where:

- $\alpha_w = 0.05$ is the learning rate for weight optimization
- $R_{\text{target}}(t) = \max_{a'} Q_t(a')$ is the maximum Q-value among all actions (representing the best expected reward at time t)
- $\mathbf{m}'_a(t) = [1-m_{a,1}(t), 1-m_{a,2}(t), m_{a,3}(t)]^T$ is the transformed metric vector used in reward computation
- $\Pi_{\mathcal{W}}$ is the projection operator onto the probability simplex $\mathcal{W} = \{\mathbf{w} : w_j \geq 0, \sum_j w_j = 1\}$

The projection $\Pi_{\mathcal{W}}$ is performed by:

$$\Pi_{\mathcal{W}}[\mathbf{w}] = \frac{\max(\mathbf{w}, \mathbf{0})}{\sum_{j=1}^{3} \max(w_j, 0)} \tag{7}$$

This ensures weights remain non-negative and normalized to sum to 1, maintaining valid probability distributions. The gradient ascent mechanism allows each predictor to develop specialized weight configurations that reflect its relative strengths across the three performance metrics, enabling the bandit to learn action-specific reward functions tailored to different traffic conditions.

We trained the bandit model both offline and online. In the online phase, collected metrics were used to update predictor values continuously. Offline training utilized three distinct traffic patterns and evaluation metrics. The trained model is integrated into the adaptive system via the probability distribution derived from evaluated techniques. In this selection mechanism, a probability distribution is applied to the set of choices (prediction techniques), reflecting the likelihood of each being selected. The mechanism uses a *roulette wheel* analogy, where each choice is assigned a segment size proportional to its probability. A pseudo-random number generator then provides a value within a fixed range, which corresponds to a position on this probabilistic wheel. The segment in which this random value falls determines the chosen one. This approach effectively integrates probabilistic weighting with controlled randomness, allowing the system to make adaptive selections. This mechanism is particularly useful in dynamic environments, where selection probabilities may shift over time, enabling flexible and statistically guided decisions.

4. MAB Theoretical Analysis

In this section, we present a theoretical analysis of our MAB-based approach. Based on the empirical observations of our implementation executions, we examine convergence properties, regret bounds, and performance characteristics under different traffic patterns.

4.1. Convergence of the Weight Adaptation Mechanism

Our implementation employs an online weight adaptation mechanism that optimizes metric weights for each predictor (action) through gradient ascent. For action a, the weights are updated according to:

$$\mathbf{w}_a(t+1) = \Pi_{\mathcal{W}}\left[\mathbf{w}_a(t) + \alpha_w \cdot (R_{\text{target}}(t) - R_a(t)) \cdot \mathbf{m}_a(t)\right] \tag{8}$$

where $\alpha_w = 0.05$ is the learning rate for weight updates, $R_{\text{target}}(t) = \max_{a'} Q_t(a')$ is the best achievable reward (maximum estimated Q-value across all actions), $R_a(t)$ is the observed reward for action a, $\mathbf{m}_a(t)$ is the metric vector, and $\Pi_{\mathcal{W}}$ projects onto the probability simplex (non-negative weights summing to 1).

This update rule implements stochastic gradient ascent that reduces the squared difference between the observed reward and the target reward. The key convergence properties are:

1. *Gradient Direction:* The update moves weights in the direction that increases the reward. When $R_a(t) < R_{\text{target}}(t)$, the gradient $(R_{\text{target}} - R_a) \cdot \mathbf{m}_a$ is positive, increasing weights for metrics with higher values and thereby improving future rewards.
2. *Learning Rate:* With $\alpha_w = 0.05$ and bounded metrics ($m_{a,j} \in [0, 1]$), the learning rate is sufficiently small to ensure stable convergence. The effective step size per update is bounded by $\alpha_w \cdot |R_{\text{target}} - R_a| \cdot \|\mathbf{m}_a\| \leq 0.05 \cdot 1 \cdot \sqrt{3} \approx 0.087$.
3. *Projection Stability:* The projection onto the simplex ensures weights remain valid probability distributions. This projection is non-expansive, meaning it does not amplify errors and preserves convergence properties.
4. *Convergence Rate:* Under stationary metric distributions, the weights converge to locally optimal values at an approximate rate of $O(1/N_t(a))$, where $N_t(a)$ is the number of times action a is selected by time t. In our experiments with $T = 1000$ rounds, frequently selected actions (e.g., $N_T(a) > 400$) have sufficient samples for weight convergence.

Our multi-seed experiments show that the weights converge to configurations reflecting each predictor's strengths across the three metrics. This convergence is guaranteed by the stabilization of $Q_t(a)$ over time.

4.2. Regret Bounds

Regret quantifies the cumulative loss from not always selecting the optimal predictor. Our UCB-based selection mechanism

balances exploration and exploitation. Actions are selected according to:

$$a_t = \arg\max_a \left\{ Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right\} \quad (9)$$

with exploration constant $c = 2$. This Upper Confidence Bound balances exploiting actions with high estimated rewards ($Q_t(a)$) and exploring actions with high uncertainty (large exploration bonuses).

Q-Value Updates: The estimated Q-values are updated using exponential smoothing:

$$Q_{t+1}(a) = Q_t(a) + \alpha(R_a(t) - Q_t(a)) \quad (10)$$

with learning rate $\alpha = 0.1$. This produces an exponentially weighted moving average that tracks each action's mean reward. As $N_t(a) \to \infty$, we have $Q_t(a) \to q^*(a)$.

Regret Definition:

$$\mathcal{R}(T) = \sum_{t=1}^{T} (q^*(a^*) - q^*(a_t)) \quad (11)$$

Regret Analysis: Our approach experiences two sources of regret:

1. *Selection Regret:* The UCB algorithm with fixed reward distributions achieves logarithmic regret $O(K \ln T)$. For $K = 3$ and $T \approx 1000$, this yields approximately $O(3 \ln 1000) \approx$ 21 units of regret.
2. *Adaptation Regret:* The weight adaptation introduces non-stationarity, producing additional regret early in training. As weights converge, typically within $O(100)$ rounds, this component diminishes.

Combining both components:

$$\mathbb{E}[\mathcal{R}(T)] = O(K \ln T) + O(\sqrt{T}) \quad (12)$$

The $O(\sqrt{T})$ term dominates for moderate horizons, and the average regret per round approaches zero as $T \to \infty$.

4.3. Discussion

Our implementation uses $\alpha = 0.1$, $\alpha_w = 0.05$, $\epsilon = 0.1$, and $c = 2$. These values balance convergence speed and stability. Higher learning rates would accelerate adaptation but risk instability; lower values improve stability but slow adaptation. The UCB constant c is set to $c = 2$ as proven to yield near-optimal exploration [26].

The reward function is the same previously described in Eq. (4).

This transformation ensures all components contribute positively to reward, simplifying optimization and accelerating convergence.

For instance, with $K = 3$ predictors and 3 metrics per predictor, the computational complexity per round is $O(K)$ for selection and $O(3)$ for weight updates—a negligible overhead for streaming systems. The approach scales naturally to more predictors or metrics.

5. Performance Evaluation

This section presents the performance evaluation of MMP-SPS, focusing on its ability to adapt to variations in the input event stream without compromising the event processing rate. Section 5.1.1 describes the experimental testbed and the data stream analysis use case employed for validation. To prove the validity of our solution, we implement MMP-SPS on top of Apache Storm 2.8. The source code and deployment scripts are publicly available in an open-source repository[1].

The evaluation focuses on three main aspects: (1) the performance of predictive models using a social media streaming use case; (2) the performance of predictive models under a DNS traffic scenario; and (3) a comparative analysis of model selection strategies, including the Multi-Armed Bandit approach.

5.1. Experimental Setup

Experiments were conducted on the Google Cloud Platform (GCP) using eleven virtual machines (VMs): three for ZooKeeper, seven for Storm Supervisors, and one for the Nimbus node, which also hosted Storm UI along with the adaptation and prediction layers. Three VM types were employed: `n1-standard-1` (1 vCPU @ 2.2 GHz, 3.75 GB RAM) for ZooKeeper; `n1-stan dard-4` (4 vCPUs @ 2.2 GHz, 15 GB RAM) for Nimbus; and `n1-highcpu-8` (8 vCPUs @ 2.2 GHz, 7.2 GB RAM) for Supervisors.

The adaptation and prediction layers, along with Storm UI, were containerized and deployed on the Nimbus node. Containerization decouples services from the host OS, facilitating portability and modularity. Storm's internal coordination was managed by ZooKeeper. Inter-layer communication between adaptation, prediction, and processing components was implemented using RESTful HTTP APIs. Performance metrics were collected through Storm UI, and the prediction layer was exposed as a Flask-based REST service.

Each experiment was conducted using a distinct combination of input workload and stream processing application, with a fixed execution duration of 20 minutes per run. All reported metrics are averaged over multiple independent runs; when Bandit variants are involved, each run is initialized with a different random seed.

5.1.1. Datasets and Topologies

To evaluate our system, we implemented two realistic applications: classification of COVID-19 tweets and classification of DNS traffic.

Social Media - COVID-19:. The social media dataset used in our experiments is derived from X (formerly Twitter) traffic and consists of 237 million COVID-19-related tweets, as documented in [27]. A subset of this dataset was selected from periods exhibiting high traffic variability, including prominent peaks and troughs, to simulate dynamic input conditions for evaluation. The traffic spikes used for testing were constructed

[1] https://github.com/dwladdimiroc/adaptive-sps-storm

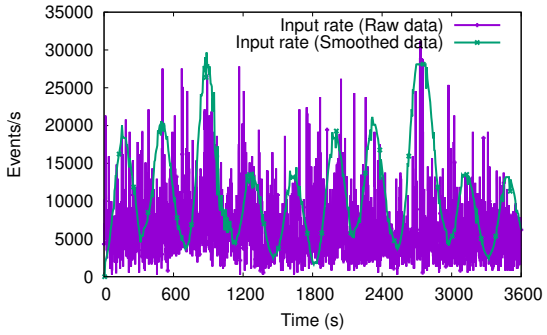


Figure 4: Traffic shape of Covid Social network dataset.

following the methodology outlined in [28]. Figure 4 illustrates the dataset: the purple line represents the original tweet stream, while the green line indicates the synthesized traffic pattern employed in the experiments.

For this dataset, we developed two applications: Information Behavior Analysis (IBA) and Information Classification (IC), as shown in Fig. 5 and Fig. 6, respectively. IBA categorizes events collected through the Twitter API based on tweet content and user identity, and stores the classified tweets in a downstream database. IC analyzes the Twitter API for information such as news or opinions and splits the stream according to the type of content. The results are then stored in a database.

Figure 5: Social Network Application for Analyzing Information Behavior (IBA).

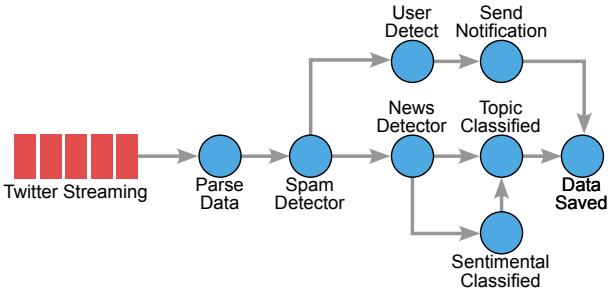


Figure 6: Social Network Application for Information Classification (IC).

DNS Traces: . The second dataset, obtained from [29], consists of DNS traffic traces that support the evaluation of the system's adaptability under diverse traffic conditions. To simulate realistic dynamic input with controlled variability, we applied the same data generation methodology from the social media dataset. This setup allows for a consistent comparison across use cases and a rigorous assessment of model performance under varying workloads.

For this use case, a linear processing topology was implemented, as shown in Fig. 7. The topology comprises four operators that sequentially process, analyze, and classify events extracted from DNS traffic, enabling efficient real-time stream analysis.

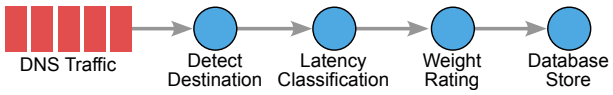


Figure 7: DNS application.

5.1.2. Evaluation Metrics

We use five significant evaluation metrics that are well-established in the literature [30] [3] [5]:

- *Latency* measures the average time an event takes to travel from the application's entry point to the final output generated by the last operator (end-to-end latency). This metric is crucial in stream processing systems (SPSs), where real-time responsiveness is essential.
- *Throughput degradation* assesses system stability by comparing input and output processing rates. Defined as the ratio $\frac{|input_{rate} - output_{rate}|}{input_{rate}}$, it measures how effectively the system processes incoming data. A value near 0 indicates stable, real-time processing, while a value approaching 1 signals instability, where the system fails to keep up with the input rate, leading to data buffering.
- The *saved resources* metric quantifies resource savings (in terms of active replicas) relative to a statically over-provisioned configuration. It is calculated as $1 - \frac{r}{r_{over}}$, where r represents the current number of active replicas, and r_{over} is the estimated number of replicas required to handle peak input rates. A metric value close to 1 indicates significant resource savings.
- *Error estimation input* represents the mean absolute percentage error between the actual input rate and the predicted input rate for each interval t_m.
- *Error estimation replica* is the mean absolute percentage error between the actual number of replicas required to process all events and the predicted number of replicas for each interval t_m.

5.1.3. Parameters Configuration

The system parameter values were set based on our previous works [3, 5]. These parameters offer an effective balance between the number of processed events, saved resources, and end-to-end latency. For calculating the *Saved resources* metric, we have fixed $r_{over} = 32$ (i.e., $r_i = 8$) for *Social media* scenario and $r_{over} = 12$ (i.e., $r_i = 3$) for *DNS* scenario.

The system uses the following parameter values: $td_m = 1, \text{s}$, $td_a = 30, \text{s}$, $s_a = 10, \text{s}$, $td_i = 10, \text{s}$, $n = 30$ samples, and $p = 10$ predictions.

5.2. RMSE Model Evaluation

In this section, we evaluate the Root Mean Square Error (RMSE)-based selection strategy and compare its performance with that of each of the nine individual prediction models. This method allows us to assess the accuracy of each model's predictions and select the best-performing models (with the lowest RMSE).

Additionally, we aim to determine a key parameter of our model, namely the size of the planning time window, denoted as t_p. To achieve this, we analyze the model's performance using three different values for td_p: 1, 5, and 10 seconds. The evaluation will be conducted across the two previously discussed scenarios: social media and DNS traffic, allowing us to assess the impact of different planning window sizes on model performance in varying contexts.

5.2.1. Parameter td_p

This section evaluates the impact of td_p's value in the five metrics in the context of social media and DNS data streams scenarios. Let us remember that for metrics such as latency, throughput degradation, and errors, lower values indicate better performance. Conversely, for the resource savings metric, higher values are preferable as they indicate better results.

Our first experiment will study the behavior of using a single model in comparison with *MMP-SPS*. In this experiment td_p was set to 1 second. Table 2 shows the performance of the various predictive models with $td_p = 1s$ under the social media scenario and the IBA application (see Fig. 5), while Table 3 shows the performance of various predictive models with $td_p = 1s$ under the DNS scenario and the corresponding application.

The results presented in Table 2 and Table 3 indicate that *SGD* achieves the lowest latency, making it well-suited for scenarios requiring rapid responses. However, this advantage comes at the cost of poor resource estimation accuracy. In contrast, methods such as *Ridge*, *Bayesian*, and *SVM* exhibit higher latencies, limiting their suitability for real-time applications despite offering more stable estimations.

While *SGD* delivers lower latency and reduced throughput degradation, its inaccuracies in workload prediction lead to suboptimal resource usage. In comparison, *MMP-SPS* demonstrates up to 10% improvement in resource efficiency.

Notably, the dynamic selection strategy implemented by *MMP-SPS* enables it to maintain a better overall balance across key metrics. By continuously adapting and selecting the most suitable prediction model based on recent performance, it effectively responds to the varying characteristics of input traffic.

In our second experiment we have increased the value of td_p to 5 seconds, in order to analyze the effect of this time window over the adaptive behavior of the system. Table 4 presents the performance of predictive models under a $td_p = 5s$.

The results presented in Table 4 and Table 3 exhibit a behavior similar to the previous scenario. *SGD* continues to deliver the lowest latency and minimal throughput degradation, yet it remains inefficient in terms of resource utilization.

Although *MMP-SPS* experiences an increase in latency compared to the $td_p = 1s$ scenario, it still achieves a more favorable balance across all metrics relative to individual predictive models.

When using $td_p = 5s$, *MMP-SPS* improves resource savings, though this comes at the cost of higher latency and throughput degradation. Nevertheless, it continues to offer the best overall trade-off, maintaining stable latency, efficient resource usage, and low replica estimation errors.

To further investigate the effect of planning window size, we increase td_p to 10 seconds. Table 6 summarizes the performance of the prediction models under this configuration, evaluated using the same set of performance metrics.

The results presented in Table 6 and Table 7 show that *SGD* continues to exhibit the lowest latency confirming its consistency in low-latency environments. However, *MMP-SPS* shows a significant latency increase, up 78% from $td_p = 5s$, reflecting the impact of longer planning intervals. High-latency models like *Basic* and *Ridge* become increasingly unsuitable for real-time applications as the interval grows.

As the planning time window increases, latency and throughput degradation tend to rise. While models such as *SGD* and *ANN* maintain relatively stable performance with respect to these metrics, *MMP-SPS* and *SVM* exhibit more noticeable increases in latency. Nonetheless, resource efficiency remains stable for both, highlighting their resilience in maintaining optimal resource usage regardless of the time window size.

Compared to *MMP-SPS*, the other models exhibit significant trade-offs that compromise overall system performance across the evaluated metrics. Some models introduce high latency, rendering them unsuitable for real-time processing, while others are more prone to inefficient resource usage or throughput degradation due to higher errors in traffic prediction.

In conclusion, *MMP-SPS* provides a versatile and adaptive solution suitable for both stationary and highly dynamic environments. Although *SGD* and *ANN* demonstrate superior performance in specific individual metrics, *MMP-SPS* stands out for its adaptability and overall balance, making it a strong candidate for real-time stream processing under fluctuating traffic conditions. Moreover, based on the observed results, a planning window of $td_p = 5$ seconds offers the best compromise between planning overhead and system performance. This setting enables responsive adaptation while avoiding excessive computational cost.

5.3. MMP-SPS vs. PA-SPS vs. ARiSTO

We compare our system to two state-of-the-art solutions: ARiSTO [31] and PA-SPS [5], to evaluate the performance and the efficiency of our multi-model approach. For a coherent comparison, the two systems were reimplemented on the latest release of Storm, the same used to implement MMP-SPS. In particular, we implemented a lightweight version of ARiSTO including the online scaling rules, but not the offline ML models or their training. The parametrization was performed using the default values specified in their source code. We compare the systems under two different applications: IBA (Fig. 5) and IC (Fig. 6).

With IBA, MMP-SPS provides a balanced compromise between performance and resource efficiency, the results are shown

Pred. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
MMP-SPS (RMSE)	1062.776	0.055	0.626	0.097	0.075
ANN	1038.300	0.065	0.574	0.405	0.582
Basic	2415.100	0.071	0.658	0.095	0.153
Bayesian	2983.638	0.066	0.653	0.057	0.069
FFT	1047.883	0.079	0.658	0.091	0.128
Linear Regression	1138.717	0.066	0.662	0.055	0.070
Random Forest	3093.460	0.080	0.664	0.075	0.098
Ridge	3556.117	0.092	0.658	0.060	0.070
SGD	475.276	0.051	0.521	0.254	0.528
SVM	4318.789	0.085	0.644	0.211	0.211

Table 2: Performance metrics for the social media scenario with a planning window of $td_p = 1$ second.

Pred. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
MMP-SPS (RMSE)	1547.901	0.678	0.858	0.186	1.123
ANN	765.672	0.529	0.801	0.430	2.602
Basic	1669.908	0.784	0.844	0.193	1.421
Bayesian	2155.933	0.913	0.851	0.148	1.192
FFT	1073.807	0.563	0.770	0.774	3.187
Linear Regression	1863.005	0.829	0.849	0.173	1.256
Random Forest	2537.592	0.832	0.851	0.155	1.186
Ridge	2028.921	0.841	0.849	0.228	1.264
SGD	1241.702	0.672	0.820	0.316	2.035
SVM	3104.350	0.943	0.860	0.110	1.013

Table 3: Performance metrics for the DNS scenario with a planning window of $td_p = 1$ second.

Pred. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
MMP-SPS (RMSE)	1325.234	0.073	0.635	0.097	0.099
ANN	760.023	0.049	0.564	0.405	0.571
Basic	3682.322	0.086	0.631	0.095	0.149
Bayesian	1144.947	0.060	0.667	0.057	0.069
FFT	1621.010	0.092	0.653	0.091	0.121
Linear Regression	871.475	0.066	0.660	0.055	0.070
Random Forest	3064.964	0.089	0.634	0.075	0.097
Ridge	2226.574	0.070	0.655	0.060	0.066
SGD	701.779	0.062	0.550	0.254	0.406
SVM	2594.393	0.071	0.645	0.211	0.212

Table 4: Performance metrics for the social media scenario with a planning window of $td_p = 5s$ second.

in Table 8. When using MMP-SPS as the reference, PA-SPS achieves approximately 12% lower latency and 52% lower throughput degradation, reflecting its focus on maintaining high data-processing performance. However, this improvement comes at the cost of increased resource consumption: its saved resources decrease by 74% compared to MMP-SPS. This higher resource usage is evident in Fig. 8(a), where PA-SPS tends to overestimate the number of replicas. ARiSTO shows a contrasting behavior, with 44% higher latency and 30% lower throughput degradation than MMP-SPS. Its saved resources decrease by 82%, indicating a slower and more conservative replica adaptation, also visible in Fig. 8(a). Overall, MMP-SPS achieves a balanced trade-off between performance and resource utilization, whereas PA-SPS prioritizes processing performance through increased resource use, and ARiSTO adapts its replica count more slowly during execution.

Similar results can be seen also when running the ICapplication, as shown in Table 9. In these experiments we set $r_{over} = 40$. MMP-SPS continues to offer a balanced compromise between performance and resource utilization. Using MMP-SPS as the

Pred. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
MMP-SPS (RMSE)	1836.949	0.820	0.841	0.186	1.430
ANN	707.091	0.558	0.798	0.430	2.620
Basic	2107.195	0.894	0.852	0.193	1.419
Bayesian	1743.718	0.828	0.853	0.148	1.183
FFT	1016.788	0.572	0.778	0.774	3.117
Linear Regression	1500.950	0.779	0.847	0.173	1.258
Random Forest	1956.137	0.806	0.848	0.155	1.201
Ridge	1557.761	0.775	0.849	0.228	1.256
SGD	1040.004	0.559	0.807	0.316	2.604
SVM	3236.510	0.921	0.862	0.118	1.006

Table 5: Performance metrics for the DNS scenario with a planning window of $td_p = 5s$ second.

Pred. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
MMP-SPS (RMSE)	2365.743	0.070	0.651	0.097	0.156
ANN	813.541	0.073	0.556	0.405	0.607
Basic	3953.643	0.085	0.653	0.095	0.152
Bayesian	1194.473	0.068	0.650	0.057	0.068
FFT	1895.470	0.116	0.665	0.091	0.119
Linear Regression	1484.619	0.069	0.661	0.055	0.064
Random Forest	1325.234	0.073	0.635	0.075	0.099
Ridge	2928.735	0.094	0.649	0.060	0.106
SGD	1971.986	0.091	0.665	0.254	0.088
SVM	494.966	0.052	0.532	0.211	0.494

Table 6: Performance metrics for the social media scenario with a planning window of $td_p = 10s$ second.

Pred. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
MMP-SPS (RMSE)	2923.685	0.913	0.843	0.186	1.285
ANN	720.730	0.587	0.790	0.430	2.618
Basic	2009.827	0.825	0.849	0.193	1.449
Bayesian	2136.632	0.864	0.853	0.148	1.194
FFT	726.800	0.524	0.736	0.773	3.120
Linear Regression	2215.867	0.762	0.845	0.173	1.271
Random Forest	2359.535	0.840	0.850	0.155	1.205
Ridge	1486.952	0.697	0.851	0.228	1.255
SGD	2377.899	0.735	0.816	0.316	1.946
SVM	2852.009	0.897	0.860	0.118	1.013

Table 7: Performance metrics for the DNS scenario with a planning window of $td_p = 10s$ second.

reference, PA-SPS achieves 68% lower latency and 52% lower throughput degradation, consistent with a strategy that prioritizes sustained data processing. This improvement relies on greater resource consumption: its saved resources decrease by 25% relative to MMP-SPS. The higher replica allocation driving this behavior is evident in Fig. 8(b).

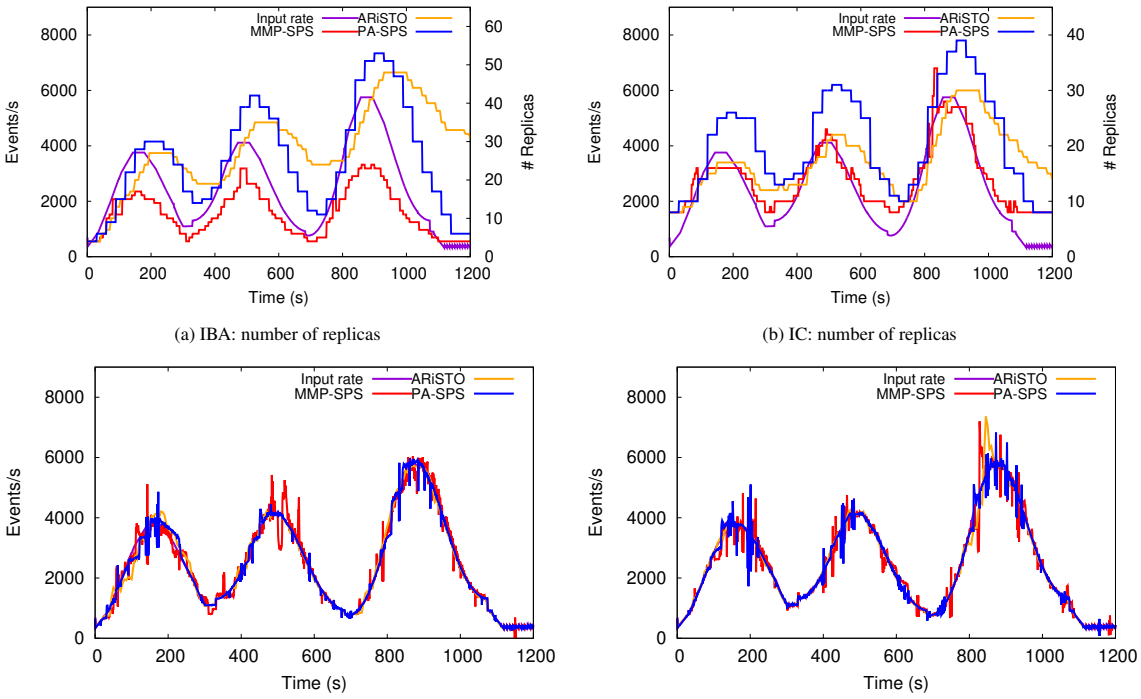
Consistent results are also obtained from ARiSTO, where it shows 19% higher latency and 28% lower throughput degradation than MMP-SPS. Its saved resources decrease by 11%, indicating a more conservative pace in replica adjustments, also visible in Fig. 8(b). Notably, latency spans a wider range in this scenario, reflecting the intrinsic difficulty of automatic adaptation over second-scale windows, to promptly optimize while maintaining high saved resources without building up event queues remains challenging in complex topologies. In this context, MMP-SPS shows balanced performance and resource utilization, whereas PA-SPS emphasizes processing performance through increased resource use, and ARiSTO adapts its replica count more gradually during execution.

Sel. Model	Latency (ms)	Thrp. Degr.	Saved Resources	Error Est. Replica
MMP-SPS	1325.234	0.073	0.635	0.099
ARiSTO	1913.069	0.051	0.113	0.652
PA-SPS	1160.711	0.035	0.165	0.539

Table 8: Performance metrics on social network scenario with IBA application comparing MMP-SPS, ARiSTO and PA-SPS.

Sel. Model	Latency (ms)	Thrp Degr.	Saved Resources	Error Est. Replica
MMP-SPS	1087.619	0.064	0.648	0.107
ARiSTO	1293.409	0.046	0.575	0.582
PA-SPS	343.577	0.031	0.485	0.571

Table 9: Performance metrics on social network scenario with IC application comparing MMP-SPS, ARiSTO and PA-SPS.



(a) IBA: number of replicas

(b) IC: number of replicas

(c) IBA: procesing throughput

(d) IC: procesing throughput

Figure 8: Performance metrics on social network scenario with IBA and IC comparing MMP-SPS, ARiSTO, and PA-SPS.

CPU overhead analysis. We compare the CPU consumption of the prediction mechanism of the three systems to understand the overhead of our multi-model solution. To isolate the prediction component, the measurements are taken exclusively on the VM hosting the adaptation layer, excluding those of the machines hosting the processing layer and other system components.

The average CPU consumption for MMP-SPS, PA-SPS, and ARiSTO is 16.96%, 12.34%, and 9.37%, respectively. Compared to ARiSTO, PA-SPS and MMP-SPS exhibit 72.27% and 81.0% higher CPU usage, respectively. Although MMP-SPS consumes slightly more CPU than PA-SPS, this difference does not significantly affect the adaptation process and the decisional reactivity, as shown in previous experiments. Moreover, since the adaptation–prediction and processing layers operate on separate VMs, the multi-model prediction computation has negligible impact on overall processing performance.

5.4. Multi-Armed Bandit Model

We compare the previous method (RMSE) with a reinforcement learning-based predictor selection approach for the next system analysis window.

We employ a reinforcement-learning selector based on the multi-armed bandit (MAB) paradigm to choose the prediction model used by MMP-SPS at runtime. The selector is evaluated using system-level metrics: latency, throughput degradation, and saved resources. We implement three multi-armed bandit (MAB). Two of them operate in an offline setting, using Greedy and UCB algorithms, where the selection weights are computed in advance from historical data and remain fixed during execution. The third version operates in an online setting, also based on Greedy, where model selection and weight updates are performed in real-time, allowing the system to adapt dynamically to changing runtime conditions.

We perform the offline bandit training over three predictors, Artificial Neural Network (ANN), Bayesian, and Stochastic Gradient Descent (SGD), chosen for their consistent performance in prior experiments. The selector object is reduce cold-start regret. This phase uses a subset of traces from the described scenarios to produce a probability vector, sampled at runtime via a roulette-wheel mechanism. Each arm receives a slice proportional to its probability, and a pseudo-random draw selects the arm. Probabilities are averaged over five seeds for reproducibility. This

prior is not a fixed policy, but an initialization that accelerates convergence when workloads resemble the training traces.

The online bandit operates over the complete predictor set used by the RMSE model. The bandit makes one decision per analyse window and receives one reward per window. The reward combines the three metrics: latency and throughput degradation and saved resources.

We evaluate the MAB variants and an RMSE-based selector in two scenarios (social media and DNS). Tables 10 and 11 summarize results. When enabled, the offline prior is computed over five seeds and used only for initialization; runtime adaptation overrides it as evidence accumulates.

Social Media Workload. Table 10 shows that Bandit-Greedy reduces latency by about 19% and lowers throughput degradation by about 21%, with a modest 8% drop in saved resources and a marked rise in input and replica estimation errors. Bandit-UCB shows a similar pattern: latency decreases by about 16% and degradation by about 32%, while saved resources decrease by about 9% and estimation errors increase. In both cases, the error bias tends toward overestimation, which allocates extra replicas, reduces queueing, and improves latency and stability at a small efficiency cost.

We observe that the Bandit Online approach yields the worst performance: latency and performance degradation increase significantly, while resource savings decrease by approximately 21% compared to the RMSE baseline. This behaviour stems from the online nature of the method, which starts from non-optimal parameter values and gradually converges over time. In the considered scenario, the convergence time exceeds the duration of the experiment; as a result, the reported average metrics primarily reflect the transient phase, leading to degraded overall performance.

An opposite trend is observed in the following experiment with the DNS workload. We acknowledge that average values are not always well suited to highlight the benefits of online multi-armed bandit (MAB) approaches. Due to the finite experimental horizon, we do not attempt a formal convergence analysis of the online bandit; therefore, the reported metrics reflect transient behavior over the experiment duration.

DNS Workload. Table 11 summarizes the results of Bandit and RMSE in the DNS scenario. Despite the dynamic nature of the environment, the Bandit offline approach continues to perform effectively compared to the RMSE approach. In this scenario, Bandit reduces overall latency by more than 53%, at a slight increase in throughput degradation compared to RMSE. Prediction error increases relative to the previous scenario, while saved resources remain nearly unchanged. Specifically, for Bandit online, latency decreases by 36.6%, throughput degradation decreases by 17.6%, saved resources decrease marginally by 0.6%, while input- and replica-estimation errors increase by 10.7% and 29.5%, respectively.

In this scenario, the Bandit online approach significantly outperforms the RMSE, reaching near-optimal parameter configurations more rapidly. This behavior indicates that, over sufficiently long execution horizons, the Bandit online can surpass RMSE in terms of overall performance. However, these gains come with a trade-off between convergence time and short-term performance during the learning phase.

Logs Workload. Considering the results obtained with the ϵ-greedy MAB, we build a generic probability distribution vector that is intended for reuse across scenarios. Specifically, we average the action-selection weights learned in our two previous scenarios and then use this averaged vector as a scenario-agnostic policy. The objective is to test whether such a policy can generalize to previously unseen workload dynamics.

To evaluate this model, we introduce a new highly dynamic workload, the *log* scenario [32], which exhibits spiky traffic patterns. We use a linear application with four operators. For comparison, we report results for the new model and compare it against both the Bandit online and the RMSE-based model. The results are summarized in Table 12.

The bandit offline approach reduces end-to-end latency by over 50%, minimizes throughput degradation, and conserves system resources. These results underscore its effectiveness in advancing self-adaptive processing models. However, compared to RMSE, bandit online approach shows a 52.6% increase in latency, a 1.2% rise in throughput degradation, a 42.4% drop in resource savings, and higher estimation errors; 26.0% for input and 27.3% for replica.

6. Related Work

Elasticity in stream processing systems (SPSs) has been tackled through a variety of strategies, including predictive modeling, adaptive scheduling, and operator-level control. We structure our review around three core capabilities: workload-aware prediction, resource adaptation, and fine-grained operator reconfiguration.

MMP-SPS substantially extends our previous PA-SPS framework [5] along all three dimensions.

6.1. Elastic Scaling Strategies

Elastic frameworks aim to maintain low latency and high resource efficiency under bursty workloads. Reactive controllers like AutoFlow [33] and Emma [34] focus on container-level elasticity using hotspot-aware scheduling and multi-resource allocation. TransScale [35] combines horizontal scaling with transprecision techniques for resource-constrained environments.

Gkolemis et al. [31] propose ARiSTO, which blends rule-based logic with model-guided configuration using genetic algorithms. Xu and Palanisamy [36] present MBLinUCB, a contextual bandit-based autoscaler pretrained with queueing models, achieving rapid convergence and robust latency control in edge environments.

While effective in specific contexts, these strategies often require manual configuration, rely on static heuristics, or assume a fixed operating environment. Our work offers a unified, generalizable approach that integrates forecasting and model selection.

Sel. Model	Lat. (ms)	Thrp. Degr.	Saved Resources	Error Est. Input	Error Est. Replica
Bandit-UCB	1110.823	0.050	0.581	0.291	0.424
Bandit-Greedy	1067.886	0.058	0.587	0.330	0.478
Bandit-Online	2991.327	0.435	0.501	0.344	0.386
RMSE	1325.234	0.073	0.635	0.096	0.099

Table 10: Performance metrics on social network scenario when comparing Bandit and RMSE selection model.

Sel. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
Bandit-UCB	1245.263	0.647	0.797	0.486	2.596
Bandit-Greedy	847.212	0.544	0.799	0.427	2.593
Bandit-Online	1164.811	0.676	0.836	0.249	1.852
RMSE	1836.949	0.820	0.841	0.225	1.430

Table 11: Performance metrics on DNS scenario when comparing Bandit and RMSE selection model.

Sel. Model	Latency (ms)	Throughput Degradation	Saved Resources	Error Est. Input	Error Est. Replica
Bandit-Greedy (Avg)	1379.600	0.436	0.581	0.730	1.372
Bandit-Online	4293.718	0.596	0.403	0.393	0.928
RMSE	2814.763	0.589	0.699	0.312	0.729

Table 12: Performance metrics over Logs scenario. A comparison of bandit and RMSE selection models when using an average probability distribution.

6.2. Predictive Modeling for Elasticity

A key trend in recent SPS research is the use of predictive models to anticipate workload fluctuations and proactively allocate resources. Early works such as Truong et al. [37] apply queueing theory to estimate system utilization and throughput, while Russo et al. [38] introduce MEAD, which uses Markovian Arrival Processes (MAPs) to drive vertical CPU scaling. Similarly, Heinze et al. [39] propose an empirical latency-aware model to minimize disruption during operator migration.

Time series forecasting has also gained traction; Pfister et al. [40] leverage ARIMA models per worker to enable lightweight horizontal scaling, while Geldenhuys et al. [41] combine ARIMA with multi-objective Bayesian optimization to adapt configurations under varying load. ContTune [42] applies Bayesian search to tune autoscaling parameters while satisfying SLAs.

However, these methods typically rely on a single model tailored to static workload profiles. Benchmarks such as [43] highlight the brittleness of such approaches under non-stationary traffic. In contrast, our framework incorporates online model selection through Multi-Armed Bandits (MAB), enabling real-time adaptation across diverse conditions without retraining.

6.3. Operator-Level and Fine-Grained Elasticity

Fine-grained elasticity addresses the need to reconfigure resources at the operator level, which remains a major challenge in live systems [44]. Hadian et al. [45] apply reinforcement learning to detect and relocate bottlenecks, while utility-driven controllers [46] scale operators based on local performance metrics. Hierarchical controllers [47] coordinate decisions across system layers but may introduce overhead or centralized dependencies.

Our approach decentralizes elasticity by embedding model selection directly at the operator level. It enables local, lightweight decisions without the need for system-specific configuration. This aligns with recent calls for modular, reconfigurable elasticity frameworks capable of reacting to dynamic and heterogeneous workloads.

6.4. Multi-Armed Bandit Approaches for Model Selection

Multi-Armed Bandit (MAB) algorithms are effective for balancing exploration and exploitation in real-time adaptive decision-making. In multi-access edge computing (MEC), Chu et al. [48] employ a constrained MAB approach for real-time service caching, reducing latency by adaptively caching services at the network edge. Li et al. [49] apply a batched MAB strategy to improve task offloading efficiency in large-scale, multi-user MEC systems, ensuring optimal server selection under dynamic conditions. Similarly, Mardi et al. [50] utilize an ARIMA-enhanced MAB approach for resource allocation in LoRaWAN networks, maximizing packet delivery rates and maintaining SLAs through predictive adaptation.

These studies demonstrate MAB's utility in handling dynamic environments by quickly adjusting to fluctuating demands and optimizing resource allocation. Inspired by these approaches, our framework leverages MAB algorithms to dynamically select the best predictive model from a pool based on real-time performance. This MAB-based strategy ensures that the most appropriate model is chosen to match current traffic characteris-

tics, effectively balancing short-term accuracy with long-term adaptability.

By incorporating MAB for model selection, our framework achieves enhanced flexibility and responsiveness in complex, high-variability environments. Unlike traditional fixed-policy approaches that may struggle with unseen traffic patterns, our MAB-based selection continuously refines model choice, maintaining optimal performance as conditions evolve. This adaptive selection mechanism distinguishes our framework from other SPSs that lack a model selection strategy tailored to dynamic data patterns, making it more resilient and efficient in real-time processing scenarios.

7. Conclusion and Future Work

This article presented MMP-SPS, an adaptive stream processing system that combines a multi-model prediction strategy with dynamic model selection to improve resource efficiency and responsiveness in highly dynamic workloads. By predicting input traffic and adapting operator parallelism accordingly, MMP-SPS enables efficient and timely scaling decisions.

The proposed architecture follows a modular three-layer design: Processing, Adaptation, and Prediction, which supports scalable execution and seamless integration of heterogeneous predictive models. Experiments on social media and DNS workloads show that MMP-SPS consistently improves resource utilization, reduces latency, and stabilizes throughput compared to single-model baselines. Overall, MMP-SPS reduces resource usage by up to 10–25% while maintaining competitive latency and throughput. Our evaluation also indicates that a planning window of 5 seconds provides an effective balance between planning overhead and prediction accuracy.

A central contribution of this work is the model selection mechanism. We compared a static RMSE-based selector with a dynamic Multi-Armed Bandit (MAB) approach. While RMSE relies on historical prediction accuracy, the MAB-based selector adapts to changing conditions by balancing exploration and exploitation, enabling dynamic model switching at runtime. As a result, MMP-SPS achieves strong performance under diverse and fluctuating workloads, though this advantage comes with an important caveat.

It is also worth nothing that, while the proposed architecture is designed to be conceptually applicable to other stream processing systems, the current implementation and experimental evaluation are specific to Apache Storm. This represents a limitation that we intend to address in future work.

Future work will focus on extending MMP-SPS to dynamic cloud infrastructures, reducing the overhead of maintaining multiple models, and improving robustness. In particular, we plan to investigate self-tuning of control parameters, dynamic VM-pool scaling, and fault-tolerant coordination mechanisms to eliminate single points of failure. We also plan to validate and extend our approach to other SPSs, such as Apache Flink or Spark Streaming.

In summary, MMP-SPS provides a flexible and efficient framework for adaptive stream processing, well suited to modern data-intensive applications with highly variable workloads.

8. Acknowledgement

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-22-CE25-0008-01 (SkyData project). Victor Reyes and Nicolas Hidalgo want to thank to project the STIC-AmSud ITERATION-D project number 24-STIC-13, Enlace UDP. This material is based upon work supported by the Google Cloud Research Credits program with the award GCP19980904.

References

[1] S. Chakravarthy, Q. Jiang, Stream Data Processing: A Quality of Service Perspective - Modeling, Scheduling, Load Shedding, and Complex Event Processing, volume 36 of *Advances in Database Systems*, Kluwer, 2009. doi:10.1007/978-0-387-71003-7.

[2] V. Cardellini, F. L. Presti, M. Nardelli, G. R. Russo, Runtime adaptation of data stream processing systems: The state of the art, ACM Computing Surveys 54 (2022) 1–36. doi:10.1145/3514496.

[3] D. Wladdimiro, L. Arantes, P. Sens, N. Hidalgo, A predictive approach for dynamic replication of operators in distributed stream processing systems, in: 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Bordeaux, France, November 2-5, 2022, IEEE, 2022, pp. 120–129. doi:10.1109/SBAC-PAD55451.2022.00023.

[4] F. Rossi, V. Cardellini, F. L. Presti, M. Nardelli, Dynamic multi-metric thresholds for scaling applications using reinforcement learning, IEEE Transactions on Cloud Computing 11 (2022) 1807–1821. doi:10.1109/TCC.2022.3163357.

[5] D. Wladdimiro, L. Arantes, P. Sens, N. Hidalgo, Pa-sps: A predictive adaptive approach for an elastic stream processing system, Journal of Parallel and Distributed Computing 192 (2024) 104940. doi:10.1016/j.jpdc.2024.104940.

[6] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, D. Ryaboy, Storm@twitter, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 147–156. doi:10.1145/2588555.2595641.

[7] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, M. Zaharia, Structured streaming: A declarative api for real-time applications in apache spark, in: Proceedings of the 2018 International Conference on Management of Data, Association for Computing Machinery, 2018, pp. 601–613. doi:10.1145/3183713.3190664.

[8] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink™: Stream and batch processing in a single engine, IEEE Data Eng. Bull. 38 (2015) 28–38.

[9] K. Padmanaban, T. R. Ganesh Babu, K. Karthika, B. Pattanaik, D. K, C. Srinivasan, Apache kafka on big data event streaming for enhanced data flows, in: 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2024, pp. 977–983. doi:10.1109/I-SMAC61858.2024.10714884.

[10] M. A. U. Nasir, G. D. F. Morales, D. Garcia-Soriano, N. Kourtellis, M. Serafini, The power of both choices: Practical load balancing for distributed stream processing engines, in: ICDE, IEEE Computer Society, 2015, pp. 137–148. doi:10.1109/ICDE.2015.7113279.

[11] L. Wang, T. Z. J. Fu, R. T. B. Ma, M. Winslett, Z. Zhang, Elasticutor: Rapid elasticity for realtime stateful stream processing, in: SIGMOD Conference, ACM, 2019, pp. 573–588. doi:10.1145/3299869.3319868.

[12] J. Xu, Z. Chen, J. Tang, S. Su, T-storm: Traffic-aware online scheduling in storm, in: 2014 IEEE 34th International Conference on Distributed Computing Systems, IEEE Computer Society, 2014, pp. 535–544. doi:10.1109/ICDCS.2014.61.

[13] D. Wladdimiro, L. Arantes, P. Sens, N. Hidalgo, A multi-metric adaptive stream processing system, in: NCA, IEEE, 2021, pp. 1–8. doi:10.1109/NCA53618.2021.9685871.

[14] Y. N. Rimal, N. Sharma, A. Alsadoon, The accuracy of machine learning models relies on hyperparameter tuning: student result classification using random forest, randomized search, grid search, bayesian, genetic, and

optuna algorithms, Multim. Tools Appl. 83 (2024) 74349–74364. doi:10.1007/S11042-024-18426-2.
[15] V. Cerqueira, L. Torgo, I. Mozetic, Evaluating time series forecasting models: an empirical study on performance estimation methods, Mach. Learn. 109 (2020) 1997–2028. doi:10.1007/S10994-020-05910-7.
[16] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, G. M. Ljung, Time series analysis: forecasting and control, John Wiley & Sons, 2015.
[17] D. C. Montgomery, E. A. Peck, G. G. Vining, Introduction to Linear Regression Analysis, John Wiley & Sons, 2021.
[18] H. J. Nussbaumer, H. J. Nussbaumer, The Fast Fourier Transform, Springer, 1981.
[19] M. Riedmiller, A. Lernen, Multi layer perceptron, Machine Learning Lab Special Lecture, University of Freiburg (2014) 7–24.
[20] S. J. Rigatti, Random forest, Journal of Insurance Medicine 47 (2017) 31–39.
[21] S. V. Balakrishnan, Fast incremental adaptation using maximum likelihood regression and stochastic gradient descent, in: INTERSPEECH, ISCA, 2003, pp. 1521–1524. doi:10.21437/EUROSPEECH.2003-441.
[22] G. C. McDonald, Ridge regression, Wiley Interdisciplinary Reviews: Computational Statistics 1 (2009) 93–100.
[23] D. Jap, M. Stöttinger, S. Bhasin, Support vector regression: exploiting machine learning techniques for leakage modeling, in: HASP@ISCA, ACM, 2015, pp. 2:1–2:8.
[24] M. Abundo, V. D. Valerio, V. Cardellini, F. L. Presti, Bidding strategies in qos-aware cloud systems based on n-armed bandit problems, in: 2014 IEEE 3rd Symposium on Network Cloud Computing and Applications (ncca 2014), 2014, pp. 38–45. doi:10.1109/NCCA.2014.15.
[25] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, A Bradford Book, Cambridge, MA, USA, 2018.
[26] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, A. Bifet, River: machine learning for streaming data in python, J. Mach. Learn. Res. 22 (2021) 110:1–110:8. doi:doi.org/10.48550/arXiv.2012.04740.
[27] A. Gruzd, P. Mai, COVID-19 Twitter Dataset, 2020. doi:10.5683/SP2/PXF2CU.
[28] P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, D. A. Patterson, Characterizing, modeling, and generating workload spikes for stateful services, in: SoCC, ACM, 2010, pp. 241–252. doi:10.1145/1807128.1807166.
[29] M. MontazeriShatoori, L. Davidson, G. Kaur, A. H. Lashkari, Detection of doh tunnels using time-series classification of encrypted traffic, in: IEEE Intl Conf on Dependable, Autonomic and Secure Computing, IEEE, 2020, pp. 63–70. doi:10.1109/DASC-PICOM-CBDCOM-CYBERSCITECH49142.2020.00026.
[30] F. Lombardi, L. Aniello, S. Bonomi, L. Querzoni, Elastic symbiotic scaling of operators and resources in stream processing systems, IEEE Trans. Parallel Distrib. Syst. 29 (2018) 572–585. doi:10.1109/TPDS.2017.2762683.
[31] E. Gkolemis, K. Doka, N. Koziris, Automatic scaling of resources in a storm topology, in: ALGOCLOUD, volume 10739 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 157–169.
[32] S. He, J. Zhu, P. He, M. R. Lyu, Loghub: A large collection of system log datasets towards automated log analytics, CoRR abs/2008.06448 (2020). URL: https://arxiv.org/abs/2008.06448.
[33] P. Lu, Y. Yue, L. Yuan, Y. Zhang, Autoflow: Hotspot-aware, dynamic load balancing for distributed stream processing, in: Y. Lai, T. Wang, M. Jiang, G. Xu, W. Liang, A. Castiglione (Eds.), Algorithms and Architectures for Parallel Processing. ICA3PP 2021, volume 13157 of *Lecture Notes in Computer Science*, Springer, Cham, 2022, pp. 1–12. doi:10.1007/978-3-030-95391-1_9.
[34] R. Dou, X. Wang, R. T. B. Ma, Emma: Elastic multi-resource management for realtime stream processing, in: IEEE INFOCOM 2024 - IEEE Conference on Computer Communications, 2024, pp. 1581–1590. doi:10.1109/INFOCOM52122.2024.10621313.
[35] A. Pagliari, G. Pierre, TransScale: Combined-Approach Elasticity for Stream Processing in Fog Environments, in: 2023 11th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 17–24. doi:10.1109/MobileCloud58788.2023.00009.
[36] J. Xu, B. Palanisamy, Model-based reinforcement learning for elastic stream processing in edge computing, in: 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC), 2021, pp. 292–301. doi:10.1109/HiPC53243.2021.00043.
[37] M. T. Truong, A. Harwood, R. O. Sinnott, S. Chen, Performance analysis of large-scale distributed stream processing systems on the cloud, in: IEEE CLOUD, IEEE Computer Society, 2018, pp. 754–761. doi:10.1109/CLOUD.2018.00103.
[38] G. R. Russo, V. Cardellini, G. Casale, F. L. Presti, Mead: Model-based vertical auto-scaling for data stream processing, in: CCGRID, IEEE, 2021, pp. 314–323. doi:10.1109/CCGrid51090.2021.00041.
[39] T. Heinze, Z. Jerzak, G. Hackenbroich, C. Fetzer, Latency-aware elastic scaling for distributed data stream processing systems, in: DEBS, ACM, 2014, pp. 13–22. doi:10.1145/2611286.2611294.
[40] B. J. J. Pfister, D. Scheinert, M. K. Geldenhuys, O. Kao, Daedalus: Self-adaptive horizontal autoscaling for resource efficiency of distributed stream processing systems, in: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024, London, United Kingdom, May 7-11, 2024, 2024, pp. 130–141. doi:10.1145/3629526.3645042.
[41] M. K. Geldenhuys, D. Scheinert, O. Kao, L. Thamsen, Demeter: Resource-efficient distributed stream processing under dynamic loads with multi-configuration optimization, in: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024, London, United Kingdom, May 7-11, 2024, 2024, pp. 142–153. doi:10.1145/3629526.3645048.
[42] J. Lian, X. Zhang, Y. Shao, Z. Pu, Q. Xiang, Y. Li, B. Cui, Conttune: Continuous tuning by conservative bayesian optimization for distributed stream data processing systems, Proc. VLDB Endow. 16 (2023) 4282–4295. doi:10.14778/3625054.3625064.
[43] G. Siachamis, G. Christodoulou, K. Psarakis, M. Fragkoulis, A. van Deursen, A. Katsifodimos, Evaluating stream processing autoscalers, in: Proceedings of the 18th ACM International Conference on Distributed and Event-Based Systems, Association for Computing Machinery, 2024, pp. 110–122. doi:10.1145/3629104.3666036.
[44] M. Fragkoulis, P. Carbone, V. Kalavri, A. Katsifodimos, A survey on the evolution of stream processing systems, VLDB J. 33 (2024) 507–541. doi:10.1007/S00778-023-00819-8.
[45] H. Hadian, M. Farrokh, M. Sharifi, et al., An elastic and traffic-aware scheduler for distributed data stream processing in heterogeneous clusters, Journal of Supercomputing 79 (2023) 461–498. doi:10.1007/s11227-022-04669-z.
[46] M. Humayoo, Y. Zhai, Y. He, B. Xu, C. Wang, Operator scale out using time utility function in big data stream processing, in: Z. Cai, C. Wang, S. Cheng, H. Wang, H. Gao (Eds.), Wireless Algorithms, Systems, and Applications, Springer International Publishing, Cham, 2014, pp. 54–65. doi:10.1007/978-3-319-07782-6_6.
[47] G. R. Russo, V. Cardellini, F. L. Presti, Hierarchical auto-scaling policies for data stream processing on heterogeneous resources, ACM Trans. Auton. Adapt. Syst. 18 (2023) 14:1–14:44. doi:10.1145/3597435.
[48] W. Chu, X. Zhang, X. Jia, J. C. S. Lui, Z. Wang, Online optimal service caching for multi-access edge computing: A constrained multi-armed bandit optimization approach, Computer Networks 246 (2024) 110395. doi:10.1016/j.comnet.2024.110395.
[49] H. Li, L. Shi, X. Zhong, X. Wang, Y. Ji, S. Zhang, A multi - user effective computation offloading mechanism for mec system: Batched multi-armed bandits approach, in: 2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), 2023, pp. 750–757. doi:10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00109.
[50] F. Z. Mardi, Y. Hadjadj-Aoul, M. Bagaa, N. Benamar, Resource allocation for lorawan network slicing: Multi-armed bandit-based approaches, Internet of Things 26 (2024) 101195. doi:10.1016/j.iot.2024.101195.