Lightweight LSTM-Based Adaptive Kafka Tuning for Predictive IoT Data Streams

Yangyang Wang*, Praveen Kumar Donta‡, Lauri Lovén†, Schahram Dustdar§, Naser Hossein Motlagh*
*Department of Computer Science, University of Helsinki, Finland
‡Department of Computer Systems and Sciences, Stockholm University, SE-106 91, Sweden
†Center for Ubiquitous Computing, University of Oulu, Finland
§Distributed Systems Group, TU Wien, Vienna 1040, Austria & ICREA, UPF Barcelona, 08002, Spain
{wang.yangyang,naser.motlagh}@helsinki.fi
Lauri.Loven@oulu.fi, praveen@dsv.su.se, dustdar@dsg.tuwien.ac.at

2025 IEEE International Conference on Quantum Software (QSW) | 979-8-3315-6720-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/QSW67625.2025.00038

***Abstract*—The Internet of Things (IoT) is a fundamental element of the computing continuum, characterized by data streams that exhibit predictable patterns primarily driven by sensor configurations and deployment strategies. On the other hand, Apache Kafka, renowned for its high-throughput and fault-tolerant data streaming capabilities, is well-suited for managing IoT data streams. However, static Kafka configurations often result in inefficiencies such as suboptimal batching, increased consumer lag, and underutilization of system resources. To address these challenges, we propose a dynamic reconfiguration approach that leverages short-term historical data to forecast message rates and adjust Kafka parameters in real time using a lightweight Long Short-Term Memory (LSTM) model. This adaptive approach optimizes the configuration of Kafka producers and consumers for IoT environments, achieving a prediction accuracy of 91.42% with minimal computational overhead. Experimental evaluations demonstrate substantial improvements in consumer lag reduction, throughput stability, and CPU utilization across heterogeneous IoT workloads, with the system requiring only brief observation periods to effectively tune performance.**

***Index Terms*—Publish/subscribe systems; Kafka adaptive tuning; IoT; Data Streams; and quality of service**

I. Introduction

INTERNET of Things (IoT), as a foundational layer of the computing continuum [1], plays a crucial role in bridging the physical and digital worlds by enabling real-time data collection through sensors, supporting limited local processing and interaction at the edge, and facilitating integration with higher layers for large-scale computation and analysis [2]. IoT data streams are typically more structured and predictable due to fixed sensor configurations set during hardware programming. This inherent predictability is particularly advantageous for stream processing frameworks such as Kafka[1], which rely on stable data flows to ensure efficient buffering, throughput optimization, and scalable performance. For instance, temperature sensors might generate a data point every ten minutes, while $\mathtt{PM}_{2.5}$ sensors may report once per hour. The data volume depends on factors such as sensor type and deployment scale. Geographical characteristics also influence data flow—larger regions typically require more sensors, while smaller areas may need fewer. Moreover, failure or addition of a single sensor results in only a minor variation in overall message rate, making short-term monitoring effective for accurately predicting long-term message flows [3].

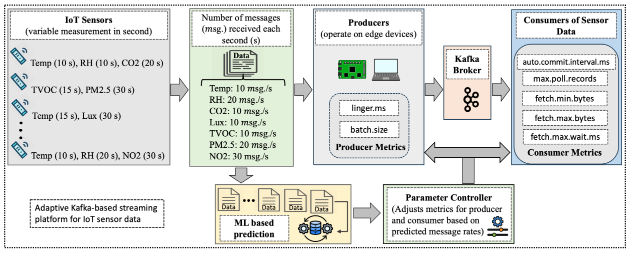


Fig. 1: General structure of the proposed Lightweight LSTM-based Adaptive Kafka tuning for IoT sensor data streams.

IoT data transmission to Kafka typically follows two paths. For sensors with network capabilities, data can be sent directly to the MQTT broker using the MQTT protocol[2], and subsequently transferred to Kafka using the Kafka Connect MQTT Source connector. On the other hand, when sensors lack built-in communication capabilities, nearby edge devices collect and forward their messages. These devices aggregate data from multiple sensors and create producer instances to efficiently forward the aggregated data to Kafka brokers. Despite the predictable nature of IoT data streams, Kafka's default settings are often suboptimal for IoT workloads, which are characterized by periodic traffic patterns and heterogeneous sensor distributions. Static configurations can result in inefficient batching, increased consumer lag, and suboptimal utilization of system resources [4], [5].

To address these challenges, we propose a lightweight Long Short-Term Memory (LSTM)-based approach that dynamically tunes Kafka configuration based on predicted data rate trends. We consider a scenario involving multiple environmental sensors (as shown in Fig. 1), each capturing a different set of variables and sending data to producers, typically deployed on edge devices, at varying times. The overall sending rate of the producer depends on the number of sensors aggregated by

[1]https://kafka.apache.org/

[2]https://mqtt.org/

the edge device. As mentioned before, the failure or addition of a single sensor only results in a small incremental change in the total message flow. Therefore, short-term monitoring can accurately predict the message rate and volume over an extended period. These characteristics allow the edge device to estimate the optimal sending rate for each Kafka producer and the receiving rate for the consumer. Once the predicted message rate is received, the system adjusts the relevant parameters for the Kafka producer and consumer accordingly. For the message rate prediction, we use the LSTM model due to its ability to effectively capture sequential patterns and temporal dependencies. To assess the performance of our proposed adaptive configuration mechanism, we conducted an experiment comparing the system's behavior using default and dynamically adjusted Kafka parameters. Real-time monitoring and measurement of key performance metrics were performed using Prometheus and Grafana. Our adaptive configuration enables Kafka-based pipelines to better adapt to workload variations, resulting in reduced consumer latency, improved throughput, and near-optimal CPU utilization.

II. Related Works

Research on distributed messaging systems area has concentrated on two main aspects: managing resources through prediction techniques and improving configuration settings.

To manage system resources efficiently, dynamic autoscaling of Docker containers in response to fluctuating workloads during runtime can be implemented with a machine learning (ML) approach, using LSTM models [6]. Moreover, predictKube [7], a predictive autoscaling mechanism, identifies seasonality patterns and unexpected events after an initial monitoring period, enabling more accurate scaling decisions for varying workloads.

An alternative approach is to focus on a particular resource management aspect, such as the challenge of optimizing consumer-partition relationships in Kafka clusters. That can be formulated as a variable-sized bin packing problem; subsequently introducing the R-score metric to account for rebalancing costs during queue migrations demonstrates how specialized metrics can improve system performance beyond general resource management [8]. Moreover, taking resource optimization into cloud-native environments, Joyce *et al.* [9] proposed an innovative autoscaling policy for Kafka-centric microservices in event-driven architectures. Their approach leverages Kubernetes infrastructure and Reinforcement Learning to dynamically adjust scaling based on message processing rates rather than conventional resource metrics like CPU and memory, representing a significant shift toward workload optimization.

In addition, building effective real-time machine learning pipelines requires well-designed system architectures with integrated data producer layers, messaging and ingestion layers, and multiple downstream consumers for feature extraction, model training, and prediction [10]. Within this framework, reinforcement learning-based methods for scheduling and resource management are becoming increasingly common in real-time pipelines, enabling systems to adapt dynamically to changing conditions [11]. To improve configuration settings, two critical parameters, namely, the number of partitions and brokers in Apache Kafka, can be optimized based on the specific characteristics and constraints of the target applications [12]. Moreover, a queuing-based packet flow model can predict key performance metrics in Kafka cloud deployments [13]. This model incorporates multiple configuration parameters such as the number of brokers, topic partitions, and message batch size, allowing users to assess various impacts on system performance.

In this paper [14], rather than relying on empirical methods to determine optimal parameter settings for Kafka deployments, a comprehensive evaluation of various Kafka configurations and performance metrics is presented. This evaluation helps users avoid bottlenecks, fully utilize Kafka's capabilities, and implement efficient stream processing based on best practices.

Taking configuration optimization to the next level, Bao *et al.* [15] proposed fully automatic configuration tuning methods specifically designed for Distributed Message Systems. Their AutoConfig framework introduces a robust comparison-based model and weighted Latin hypercube sampling for efficiently exploring the vast parameter space to discover optimal configurations for Kafka data streams.

In this context, the study in [16] performed a survey of reactive and proactive auto-scaling strategies for cloud-based IoT applications, outlining their respective strengths and limitations. The study also identifies key performance metrics critical for building effective and autonomous scaling frameworks in IoT-enabled cloud environments. To enhance the scalability of IoT message brokers, the study in [17] explored the use of Apache Kafka and clustered MQTT-based brokers. The study in [18] uses Kafka middleware for fast processing, multi-partition distribution, and high throughput to enable asynchronous communication between services, implementing a high-concurrency, low-latency IoT message subscription system using the Netty framework and a custom communication protocol.

Building upon these works, our approach focuses on a lightweight LSTM-based method tailored to IoT workloads characterized by predictable data patterns. In contrast to previous research, our method leverages the deterministic nature of IoT data streams, where sensors transmit at fixed intervals with consistent message sizes, to enable rapid parameter tuning based on short observation windows.

III. Experiments and Results

A. Experiments environment

The Kafka structure consists of two main components: **Producers** and **Consumers**, as illustrated in Fig. 1. Each of these components is associated with a set of parameters, which are detailed in Table I. In our experiment, first, we deployed the Kafka broker on a cloud-based `standard.xlarge` virtual instance equipped with 6 vCPUs, 15.6 GB RAM, and 80 GB storage, running Ubuntu 24.04.2 LTS. The Kafka cluster (version 3.5.0) utilized ZooKeeper for metadata management, and the Java runtime environment was OpenJDK 17.0.8. Next, we executed the Kafka producer and consumer locally on a MacBook Pro (2023) with Apple M2 Pro processor, 16 GB of memory, and macOS Sonoma 14.5 as the operating system.

B. Experiments

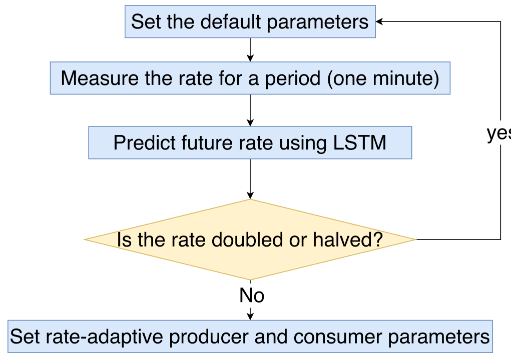
Basically, there are two kinds of producers for sensor data: one is through the MQTT Broker Connect, and the other is through edge devices (e.g., Raspberry Pi or personal computers) that directly create Kafka producers. Both approaches allow for tuning `batch.size` and `linger.ms` to optimize Kafka performance. Since it is rare for a single sensor to send data directly to the cloud with independent network communication, in our experiment, we chose the edge device as the producer to aggregate sensor data before sending it to the cloud, that is, assuming that the data of each sensor is first sent to the edge device via Bluetooth or other low-power communication protocols. As our experiment does not focus on the communication between sensors and edge devices, each sensor type is simulated by a separate process that sends randomly generated measurements to the corresponding edge node process. In our experiment, we simulate seven air quality-related sensor types generating values for variables $\mathtt{CO}$, $\mathtt{PM}_{2.5}$, $\mathtt{PM}_{10}$, $\mathtt{NO}_2$, $\mathtt{O}_3$, $\mathtt{SO}_2$, and $\mathtt{TVOC}$.

Each sensor transmits messages at one of three predefined rates: <u>fast</u> (100 messages/second), <u>medium</u> (50 messages/second), and <u>slow</u> (10 messages/second). The number of sensors at each second is simulated to be different, so the number of sensor data received by the edge node per second is different. Every 10 minutes, the rate rotates in a fixed pattern: <u>fast</u> $\rightarrow$ <u>medium</u> $\rightarrow$ <u>slow</u> $\rightarrow$ <u>fast</u>. This dynamic traffic pattern switching simulates the periodic changes in traffic in real IoT scenarios, such as shift changes in industrial scenarios or day/night mode switching in smart homes.

These seven sensor processes, with each process representing one type of variable such as Temp, RH, and $\mathtt{PM}_{2.5}$ (i.e., one process for $\mathtt{TVOC}$, and one process for $\mathtt{SO}_2$). Then, these processes use socket communication to send the sensor values to seven producer processes. Fig. 2 shows the adaptive parameter adjustment process of the producer process after receiving the message from the sensor process.

As shown in the Fig. 2, in the first minute, all producers and consumers use the default configuration when collecting message size and count statistics. The message rates from the first minute are input into the ML model to predict the message rate of the upcoming time window. For the message size prediction, we choose the LSTM because it can effectively capture sequential patterns and temporal dependencies.

We use a lightweight architecture consisting of two LSTM layers (64 and 32 units, respectively), followed by a dropout layer (rate 0.2) and a dense output layer. For each 10-minute evaluation window, the sensor message statistics for the first minute are aggregated and used to predict the total number and size of messages for the remaining 9 minutes. The prediction has an average accuracy of 91.42% for message count estimates. The inference time for each prediction round is less than 0.3 seconds on a standard MacBook Pro. Based on the predicted message rate, the producer dynamically adjusts its parameters to optimize throughput and latency; and the consumer adjusts its parameters to keep consistent with the message volume and size.



Set the default parameters
Measure the rate for a period (one minute)
Predict future rate using LSTM
yes
Is the rate doubled or halved?
No
Set rate-adaptive producer and consumer parameters

Fig. 2: Adaptive Kafka producer and consumer configuration process using LSTM-based sensor rate prediction.

In the second 10-minute evaluation window, when the message rate is doubled or halved (users can adjust based on their own experiment design) compared to the previous window, the system resets the parameters to the default configuration, collects new message rate data over the next one-minute interval, and uses these data to predict the future message rate and adjust the producer and consumer parameters accordingly. This feedback loop ensures responsiveness to traffic anomalies and dynamic workload changes. Table I presents both the default parameter configurations and the customized settings we applied for the **Producer** and **Consumer** components.

In our experiment, we use the *fast* rate (100 messages/s) as a baseline, while allowing users to define their own baselines according to their individual needs. For lower message rates such as *medium* and *slow* (predicted rates by LSTM), parameters related to throughput (including `batch.size`, `fetch.min.bytes`, and `max.poll.records`) are scaled in direct proportion to the message rate. In contrast, latency-related parameters (including `linger.ms`, `auto.commit.interval.ms`, and `fetch.max.wait.ms`) are scaled inversely. The parameter `fetch.max.bytes` remains unchanged across all rates, as even under the *fast* rate condition, the accumulated message volume remains well below the 5MB default threshold. The producer and consumer parameters for LSTM-predicted message rates are scaled proportionally or inversely based on the *fast* rate baseline, serving as a reference point for multiplicative adjustment.

Our configurations allow scaling the producing and consuming of the sensor data. This scaling approach ensures that Kafka parameters are systematically adjusted based on the predicted message rate, creating an efficient and proportional resource allocation strategy. The initial values set for the fastest message rate can be further configured based on multiple test

TABLE I: The configuration parameters for Kafka components along with their descriptions.

Component	Parameter	Default	Fast (100 msg/s)	Description
Producer	`batch.size (bytes)`	16384	16384	Controls the maximum size of a batch to be sent in a single request, affecting throughput and latency tradeoff.
	`linger.ms`	0	5	Determines how long the producer waits to accumulate messages before sending, impacting batching efficiency.
Consumer	`auto.commit.interval.ms`	5000	3000	Controls how frequently the consumer commits offsets, balancing processing guarantees and overhead.
	`max.poll.records`	500	500	Limits the maximum number of records returned in a single poll, affecting memory usage and processing time.
	`fetch.min.bytes`	1	10000	Sets the minimum amount of data the server should return for a fetch request, optimizing network usage.
	`fetch.max.bytes`	5242880	5242880	Limits the maximum amount of data the server will return for a fetch request, preventing memory overflow.
	`fetch.max.wait.ms`	500	200	Maximum time the server will block before answering the fetch request if insufficient data is available.

results. In real-world deployments, once the configuration for the baseline message rate is established, each parameter can be proportionally scaled up or down based on the ratio.

In our study, we established the following two experimental scenarios for comparison: (1) **Default Configuration**: Where we utilize Kafka's default producer and consumer settings without any optimization; (2) **Adaptive Configuration**: Where we apply our proposed approach using LSTM-based prediction for dynamic parameter adjustment. The entire 60-minute (6 rate changes $\times$ 10 minutes) sessions for both experimental scenarios were monitored using Prometheus(v2.45.0)[3] and Grafana(v10.1.0)[4]. Whereas, we collected three performance metrics including consumer lag, throughput (incoming bytes/outgoing bytes), and CPU utilization at 15-second intervals.

IV. Results and Analysis

To validate the effectiveness of our IoT sensor data rate prediction and parameter adaptation strategy, we conduct a comparative analysis of system performance before and after configuration using the following metrics:

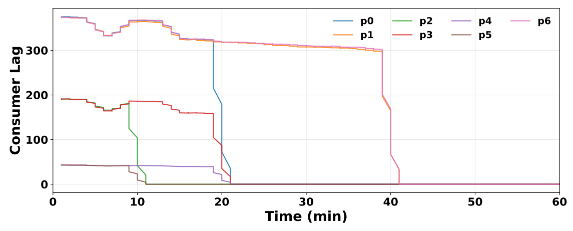
1) Kafka Consumer Lag: As shown in Fig. 3, we compare consumer lag before and after implementing adaptive configuration. In our experiment, data from seven different types of sensors ($\texttt{CO}$, $\texttt{PM}_{2.5}$, $\texttt{PM}_{10}$, $\texttt{NO}_2$, $\texttt{O}_3$, $\texttt{SO}_2$, and $\texttt{TVOC}$) were sent to seven corresponding Kafka partitions ($p0 - p6$) and processed by seven dedicated consumers. Each producer and consumer had its parameters configured according to the predicted message rate from the LSTM model. In the figure, each colored line in the graph represents the consumer lag for a specific partition over time.

Before applying adaptive configuration (Fig. 3a), consumer lag across all partitions continuously accumulated from the beginning (0 min) to 20 minutes. For partitions 0, 1, and 6, which began at fast speed (100 messages/s or 10000 bytes/s), $p1$ and $p6$ maintained consistently high lag values until 40 minutes, and the maximum lag reached more than 300 messages. This high lag would cause significant data processing delays, potential memory pressure due to buffered messages, and could lead to system instability if sustained for long periods. Other partitions ($p0$, $p2$, $p4$, and $p5$) eventually reduced to zero but experienced significant initial delays. After implementing the adaptive strategy (Fig.3b), parameters were dynamically adjusted based on the predicted message rates. This adjustment ensured that consumer configurations remained aligned with real-time message processing demands. As a result, the maximum consumer lag was significantly reduced to 140 messages, representing a 53% reduction. In addition, Fig. 3b shows regular fluctuations corresponding to the 10-minute sensor rate changes, demonstrating the system's responsiveness to workload variations. All partition lags are maintained between 20 and 120 messages. This balanced distribution helps prevent any single partition from becoming a bottleneck in the system, and results in reducing delays to consume messages.

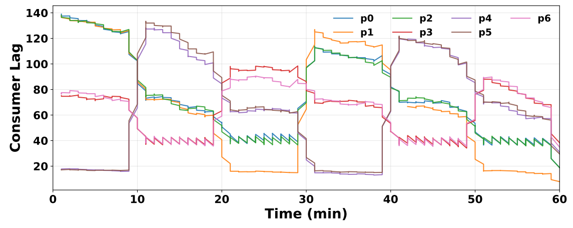
2) Throughput: As shown in Fig. 4, we computed the throughput ratio by dividing `outgoing.byte.rate` by `incoming.byte.rate`. Throughput means how efficiently messages received by the Kafka broker (incoming bytes) are processed and forwarded to consumers (outgoing bytes).

Before applying the adaptive configuration (red line), the system maintained a throughput ratio of around 90% during the first 40 minutes. However, at the beginning, the throughput briefly dropped below 10%, and around the 20-minute, it fell below 80%. Between 40 and 60 minutes, the system experienced frequent and significant throughput fluctuations, with values dropping as low as 20 to 40%. During this period, throughput stabilized somewhat between 60% and 80%. These throughput levels indicate that the system was processing less than 80% of the incoming data stream in real-time, which could lead to potential bottlenecks. After implementing adaptive configuration (green line), the system achieved a higher and more stable throughput ratio that consistently approached or exceeded 95% throughout the entire 60-minute test period. This improvement maximizes bandwidth utilization even under changing workload conditions.

[3]https://prometheus.io/
[4]https://grafana.com/



(a) Consumer lag before adaptive configuration.



(b) Consumer lag after adaptive configuration.

Fig. 3: Consumer lag before and after adaptive configuration.

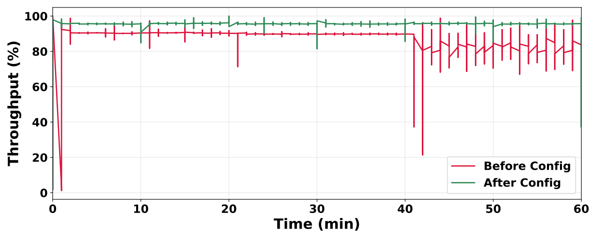


Fig. 4: Throughput ratio before and after configuration.

3) CPU Utilization: Fig. 5 depicts CPU utilization, displayed as a value from 0 to 1, and explains how many CPU resources Kafka processes are consuming. The y-axis labeled "CPU Core Usage" ranges from 0 to 0.175 (0% to 17.5%), indicating that during testing, Kafka processes used a maximum of approximately 17.5% of a single core's processing capacity.

Before adaptive configuration (red line), CPU utilization exhibited considerable volatility and inefficiency. At the beginning, CPU usage spiked to 17.5%, likely due to the initial influx of messages. Next, CPU usage remained relatively high (between 7.5-8.5%) until the 20 minutes. Then, the CPU utilization significantly dropped under 2% after the 40-minute. These fluctuations indicate poor resource management, with the system either over-utilizing CPU resources during periods of high lag accumulation or under-utilizing the resources during periods of lower activity. After implementing adaptive configuration (green line), the CPU utilization maintained a much more consistent pattern throughout for the entire 60-minute test period, generally remaining between 4-6%. This indicates that the adaptive configuration effectively distributed the processing load more balanced over time.

In conclusion, our strategy captures the traffic characteristics of sensors and employs an LSTM-based predictor to forecast the message rate and size for the upcoming time window. This prediction drives dynamic tuning of both producer and consumer parameters. Compared to static settings, our adaptive approach reduces consumer lag, stabilizes system throughput, and optimizes resource usage. The results demonstrate the practical feasibility and robustness of the proposed auto-tuning strategy in handling heterogeneous IoT streaming workloads.

V. Discussion and Future Works

Despite some promising results, our approach has several limitations that warrant consideration and several challenges remain open for future research in this area:

1) Simulated Environment: Our experiments relied on simulated sensor processes rather than actual IoT devices communicating over real wireless protocols. In real-world deployments different wireless communications such LoRaWAN would introduce additional features such as connection latency, packet loss, and signal interference. These would likely cause fluctuations in the per-second message rates, potentially reducing the prediction accuracy of our LSTM model and complicating parameter optimization.

The real IoT systems, in addition to environmental noise, often experience short-lived but intense traffic bursts, where the message rate can temporarily spike from tens to thousands of messages per second due to peak times, sensor anomalies, or collective behavioral shifts. During regular times, the traffic rate usually returns to normal levels. These transient but extreme fluctuations show challenges for adaptive streaming systems that must respond quickly without overreacting to temporary anomalies.

In our future work, we aim to evaluate our approach under such short-lived burst scenarios by simulating unpredictable traffic surges and comparing system responsiveness, lag, and throughput stability. Testing with such load patterns will give a clearer picture of how well the system can handle real-world traffic and help improve our prediction and tuning methods. Moreover, we plan to test with physical sensor deployments over standard IoT protocols to develop more robust prediction models that account for both network variability and traffic volatility.

2) Simplified Topology: Our implementation uses a one-to-one mapping between sensor type, producer, partition, and

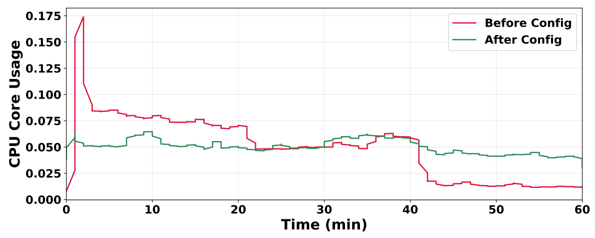


Fig. 5: CPU usage before and after adaptive configuration.

consumer. This simplified architecture may not reflect more complex real-world scenarios where multiple sensors feed into shared topics or where dynamic scaling of consumer groups is required. Particularly for applications with widely disparate message volumes (like financial transactions or e-commerce data), the ability to dynamically add or remove consumers may be limited by architectural constraints, reducing the flexibility of our approach. In the future, we will extend our approach to handle many-to-many topologies with shared topics and dynamic consumer scaling, investigating load-balancing strategies for heterogeneous message volumes across multiple applications.

3) Parameter Baseline Subjectivity: The initial configuration for maximum message rates (fast category) was determined somewhat subjectively rather than through exhaustive testing. The effectiveness of our scaling approach depends heavily on these baseline values being optimal, however we cannot guarantee they represent the true optimal configuration without more comprehensive performance testing across varied hardware environments. In our future work, we will create an automated process to discover optimal baseline parameters for the maximum message rate category would eliminate the subjectivity in our current approach. This could involve an initial calibration phase using techniques like Bayesian optimization to find optimal parameter combinations. In addition, we plan to conduct a systematic parameter sensitivity analysis to quantify how variations in individual Kafka configuration parameters affect system performance metrics. For example, we should identify which parameters are critical and must be fine-tuned and understand the most effective parameter combinations across different load scenarios. This analysis can help assessing the robustness of our scaling strategy and guide the refinement of parameter adaptation functions based on the relative impact of each parameter under different workload conditions. Moreover, sensitivity analysis would also allow us to determine whether extensive initial calibration is truly necessary, or if a lightweight, approximate baseline configuration is sufficient to achieve effective system performance.

VI. Conclusions

This paper presents a dynamic method for optimizing resource utilization in Apache Kafka, developed in the context of IoT data streams, which often exhibit predictable patterns driven by sensor configurations and deployment strategies. The proposed approach enables real-time adjustment of Kafka's producer and consumer configurations based on message rate predictions generated by a lightweight LSTM model. This adaptive tuning mitigates the inefficiencies inherent in static Kafka configurations, such as suboptimal batching, increased consumer lag, and resource under-utilization. The experimental results validate the effectiveness of our proposed approach, showing reductions in consumer lag, stabilized throughput, and improved CPU utilization across heterogeneous IoT workloads. Notably, the adaptive configuration achieved a 53% reduction in lag, maintained throughput consistently above 95%, and reached a prediction accuracy of 91.42% with minimal computational overhead.

Funding

This research is partially funded by the Research Council of Finland through the evoS3 (Grant Number 362594) and the 6G Flagship (Grant Number 369116) projects, and by Business Finland through the Neural Pub/Sub research project (Diary Number 8754/31/2022). Moreover, the research is partially supported by the *Svenska Institutet* (Project No. 31005669).

References

[1] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar, "Exploring the potential of distributed computing continuum systems," *Computers*, vol. 12, no. 10, p. 198, 2023.
[2] N. H. Motlagh, L. Lovén, J. Cao, X. Liu, P. Nurmi, S. Dustdar, S. Tarkoma, and X. Su, "Edge computing: The computing infrastructure for the smart megacities of the future," *Computer*, vol. 55, no. 12, pp. 54–64, 2022.
[3] A. Rebeiro-Hargrave, N. H. Motlagh, S. Varjonen, E. Lagerspetz, P. Nurmi, and S. Tarkoma, "Megasense: Cyber-physical system for real-time urban air quality monitoring," in *2020 15th IEEE conference on industrial electronics and applications (ICIEA)*. IEEE, 2020, pp. 1–6.
[4] A. Saleh, S. Tarkoma, S. Pirttikangas, and L. Lovén, "Publish/subscribe for edge intelligence: Systematic review and future prospects," *Available at SSRN 4872730*, 2025.
[5] P. K. Donta, S. N. Srirama, T. Amgoth, and C. S. R. Annavarapu, "Survey on recent advances in iot application layer protocols and machine learning scope for research directions," *Digital Communications and Networks*, vol. 8, no. 5, pp. 727–744, 2022.
[6] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9745–9760, 2020.
[7] PredictKube, "PredictKube Documentation: FAQ," 2025, [Online]. Available: https://docs.predictkube.com/faq (accessed: Mar. 25, 2025).
[8] D. Landau, X. Andrade, and J. G. Barbosa, "Kafka consumer group autoscaler," *arXiv preprint arXiv:2206.11170*, 2022.
[9] J. E. Joyce and S. Sebastian, "Reinforcement learning based autoscaling for kafka-centric microservices in kubernetes," in *2022 IEEE 4th PhD Colloquium on Emerging Domain Innovation and Technology for Society (PhD EDITS)*. IEEE, 2022, pp. 1–2.
[10] A. Raza, "Real-time machine learning pipelines for big data in cloud environments: Implementing streaming algorithms on apache kafka," *Open Journal of Robotics, Autonomous Decision-Making, and Human-Machine Interaction*, vol. 8, no. 6, pp. 1–11, 2023.
[11] A. B. Sharma and K. D. Forbus, "Graph-based reasoning and reinforcement learning for improving q/a performance in large knowledge-based systems." in *AAAI Fall Symposium: Commonsense Knowledge*, 2010.
[12] T. P. Raptis, C. Cicconetti, and A. Passarella, "Efficient topic partitioning of apache kafka for high-reliability real-time data streaming applications," *Future Gener. Comput. Syst.*, vol. 154, pp. 173–188, 2024.
[13] H. Wu, Z. Shang, and K. Wolter, "Performance prediction for the apache kafka messaging system," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems*. IEEE, 2019, pp. 154–161.
[14] P. Le Noac'h, A. Costan, and L. Bougé, "A performance evaluation of apache kafka in support of big data streaming applications," in *International Conference on Big Data*. IEEE, 2017, pp. 4803–4806.
[15] L. Bao, X. Liu, Z. Xu, and B. Fang, "Autoconfig: Automatic configuration tuning for distributed message systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 29–40.
[16] S. Verma and A. Bala, "Auto-scaling techniques for iot-based cloud applications: a review," *Clust. Comput.*, vol. 24, pp. 2425–2459, 2021.
[17] M. Ismaiel, "Enabling multi-tenant scalable iot platforms," Ph.D. dissertation, Universitätsbibliothek der Universität Stuttgart, 2020.
[18] Z. Yuan, B. Pang, Y. Du, X. Liu, J. Yao, and C. Kong, "Design and implementation of internet of things message subscription system based on kafka," in *11th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2019, pp. 603–606.