ISSN 2224-087X. Електроніка та інформаційні технології. 2024. Випуск 26. С. 46–58
Electronics and information technologies. 2024. Issue26. P. 46–58

UDC 004 DOI: https://doi.org/10.30970/eli.26.5

USAGE OF APACHE KAFKA FOR LOW-LATENCY IMAGE PROCESSING

N. Karpiuk[1], H. Klym[1,2], T. Tkachuk[1]

[1]*Specialized Computer Systems Department,*
Lviv Polytechnic National University,
12 Bandery St., UA-79013, Lviv, Ukraine
karpiuk.nazar@gmail.com, halyna.i.klym@lpnu.ua

[2]*Radielectronic and Computer Systems Department,*
Ivan Franko National University of Lviv,
50 Drahomanova St., UA–79005 Lviv, Ukraine
halyna.klym@lnu.edu.ua

This study addresses the limitations of conventional centralized systems in handling the surge in real-time image processing demands. We propose a distributed architecture employing Apache Kafka to achieve near real-time image analysis. Our approach implements a decoupled workflow for image acquisition, processing, and spreading, facilitating parallel execution across a processing node cluster. Kafka acts as the core communication and data flow infrastructure, ensuring scalability, fault tolerance, and high throughput. Evaluations demonstrate substantial performance gains compared to a centralized system, validating the feasibility, advantages, and limitations of Kafka for distributed image processing. We systematically analyzed the impact of topic partitioning, consumer group configuration, and processing workload on performance. This work presents a robust solution for near real-time image processing tasks, promoting the development of efficient and scalable image analysis applications.

Key words: Apache Kafka, image processing, distributed environment, parallel processing.

Introduction. The ever-growing deluge of data in today's world necessitates real-time information processing capabilities. Conventional batch processing methods struggle to keep pace with high-speed data streams, particularly in image processing applications. This is where event streaming steps in, offering a transformative approach that enables continuous data ingestion, manipulation, and analysis upon arrival. Apache Kafka, a free and open-source distributed streaming platform from the Apache Software Foundation, spearheads this revolution [1-3]. By harnessing Kafka's robust architecture and functionalities, near real-time image processing becomes a reality, opening doors to a multitude of groundbreaking applications across various sectors [4-6].

Kafka boasts a distributed architecture built upon key components that meticulously orchestrate the data flow. Producers, akin to data injectors, continuously feed the system streams of events, in this case, images. These events are published to designated topics, which function as logical channels for categorizing and grouping related data. Brokers receive and manage the events, replicating them across the cluster (as depicted in Fig. 1) to ensure high

availability and fault tolerance [7,8]. Consumers, subscribed to specific topics, actively pull and process the incoming events. This decoupled producer-consumer architecture inherently fosters scalability and flexibility, enabling independent scaling of each component based on specific processing requirements.

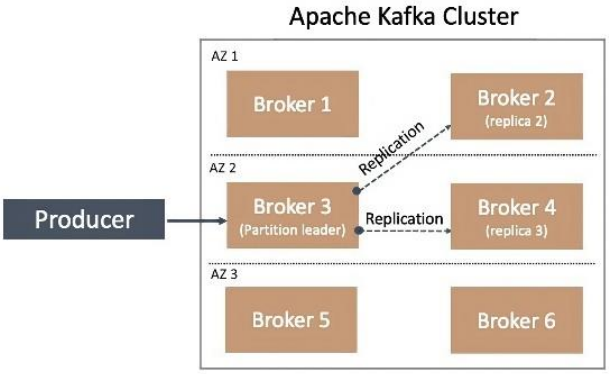

Fig. 1. The process of event replication inside Apache Kafka

Furthermore, Kafka leverages a concept called partitioning to segment topics, thereby facilitating parallel processing for high throughput. Each partition functions as an independent log, storing sequences of events. Consumers can individually assign themselves to partitions, effectively distributing the processing load across multiple nodes. This architectural design empowers the system to scale and handle massive data volumes while preserving near real-time processing capabilities.

However, there are situations where a single event needs to be delivered to multiple consumers. Kafka addresses this challenge by offering two solutions: fan-out and fan-in. In the fan-out approach, a single producer publishes events to multiple topics, allowing different consumers to subscribe based on their specific needs. Conversely, fan-in involves multiple producers contributing to a single topic, enabling aggregation or parallel processing of the combined data stream. These functionalities offer significant flexibility in designing streaming pipelines tailored to address unique use cases.

The true power of event streaming lies in its ability to process data as it is generated, offering substantial advantages over traditional batch-based approaches [10-11]. A prominent example is in the realm of autonomous vehicles. Event streaming enables real-time processing of camera footage captured by the vehicle, allowing for immediate object detection, obstacle avoidance, and dynamic route adjustments, all crucial aspects for safe and efficient autonomous navigation. Imagine a world where machines can “see” and react instantaneously. This is the promise of real-time image processing, a field revolutionized by technologies like Apache Kafka. In the realm of autonomous vehicles, Kafka becomes the backbone for processing video streams. It empowers vehicles to identify obstacles, pedestrians, and traffic signals with split-second precision, paving the way for safer and more efficient transportation [12].

Beyond the automotive industry, Kafka's applications in image processing extend far and wide. In manufacturing, it enables automated quality control by analyzing images on assembly

lines at breakneck speeds. This ensures product integrity and minimizes defects before they reach consumers. Similarly, the healthcare sector can leverage Kafka for near real-time processing of medical images. This facilitates faster diagnoses, streamlines remote consultations, and ultimately personalizes patient care.

The influence of Kafka-powered image processing transcends these core examples. Its reach extends to environmental monitoring, where it empowers researchers to analyze vast amounts of image data in real-time, enabling a deeper understanding of our planet's health. Security surveillance systems can also benefit from Kafka's capabilities, allowing for faster detection of anomalies and improved threat response times. Even scientific research can leverage Kafka to process and analyze large datasets of scientific images, accelerating discovery and innovation [13].

While Kafka reigns supreme in the event streaming domain, it's not without competition. Apache Pulsar [14] and Amazon Kinesis Streams [15] offer similar functionalities. When making a choice, consider factors like project requirements and technical expertise. Kafka boasts a mature and open-source ecosystem, while Pulsar offers a simpler setup and lower operational overhead. Kinesis Streams, on the other hand, seamlessly integrates with other AWS services, making it ideal for cloud-native deployments.

Now, let's delve into the intricate details of implementing real-time image processing within a Kafka architecture. The journey begins with data ingestion. Images captured by cameras or sensors need to be transformed into a format suitable for Kafka. This typically involves encoding the image data and attaching relevant metadata like timestamps. Specialized libraries and frameworks can streamline this process, ensuring a smooth transition from raw images to Kafka events.

Next comes the producer stage, where data sources like cameras push these image events to designated Kafka topics. Choosing the right topics is crucial for efficient downstream processing [16]. Images from different sources or containing diverse content might be published to separate topics for targeted analysis.

With events flowing through Kafka's veins, consumer applications come into play. These applications subscribe to relevant topics and continuously pull and process the image data as it arrives. This opens doors for powerful capabilities like real-time object detection, image feature extraction, and anomaly analysis.

However, complex image understanding often necessitates multi-stage processing or distributed architectures. In such scenarios, Kafka acts as the central nervous system, facilitating communication and data exchange between various consumers. Fan-out and fan-in patterns become instrumental here. For instance, an initial consumer might perform preliminary analysis and then distribute the events to specialized consumers based on the detected content. This enables parallel processing and tailored analysis.

To achieve optimal performance in this real-time processing environment, several factors come into play. Topic partitioning is crucial, as it distributes the processing load across multiple consumers, leading to improved concurrency and throughput. Additionally, consumer group management becomes essential for horizontal scaling [17]. Consumer groups allow for parallel processing within a group, effectively handling large volumes of data. However, striking a balance between concurrency, resource utilization, and efficient load distribution within the group requires careful consideration.

For latency-sensitive applications, buffering strategies and consumer rebalance configurations can be fine-tuned to minimize processing delays. By understanding the trade-

offs between buffer size and latency, we can optimize performance tailored to specific application requirements [18].

Kafka provides a robust foundation for building real-time image processing pipelines. Its inherent scalability, fault tolerance, and flexibility empower developers to design innovative solutions that unlock the true potential of image data. From autonomous vehicles navigating dynamic environments to healthcare professionals making real-time diagnoses, the applications are vast and constantly evolving. However, choosing Kafka requires a thoughtful evaluation of its strengths and limitations, alongside a clear understanding of its role within the broader data processing ecosystem. By harnessing its capabilities effectively and integrating it with the right tools and technologies, we can unlock the full potential of real-time image processing and transform the way we interact with the visual world around us.

Related work. The potential of Apache Kafka for near real-time image processing, a niche compared to text-based streams, is a subject demanding further exploration [19]. While Kafka excels in real-time data management across domains, its image processing capabilities haven't received as much attention. This gap can be attributed to several factors.

Firstly, image data's inherent complexity, demanding significant processing power and bandwidth, pushes the boundaries of Kafka's performance, especially for high-resolution or high-volume streams [20]. This challenge is compounded by the lack of native image processing functionalities within Kafka, necessitating integration with external libraries, introducing development complexity [21].

Despite these hurdles, research is actively investigating Kafka's potential in this domain. One study explored its use with Apache Spark for real-time object detection in video streams, highlighting its potential for video surveillance and anomaly detection [22]. However, the need for optimization and resource management for true scalability and maintaining near real-time performance under heavy workloads was acknowledged.

Another investigation focused on Kafka's feasibility for real-time medical image analysis, particularly cancer detection in mammograms [23]. This research emphasized the benefits of Kafka's fault tolerance and scalability in ensuring uninterrupted data flow and enabling distributed processing of large datasets. However, the need for tailored data encoding and compression techniques to minimize bandwidth and storage requirements was also highlighted.

The realm of steganography, where images hide messages, has also seen research into leveraging Kafka for real-time processing [24]. The study proposes a system utilizing Kafka for steganographic image transmission, capitalizing on its strengths in scalability and fault tolerance for ensuring reliable and efficient transmission of potentially sensitive data.

These studies represent promising advancements, but key challenges remain. Efficient and scalable image preprocessing techniques that seamlessly integrate with Kafka's streaming architecture are crucial. Additionally, research into novel compression and encoding methods specifically designed for image data is essential for bandwidth and storage optimization. Finally, addressing security and privacy concerns related to sensitive image data transmission within the Kafka ecosystem is paramount for wider adoption in critical applications.

In conclusion, while Apache Kafka offers a powerful platform for real-time data processing, its widespread adoption for near real-time image processing is still in its early stages. Overcoming the challenges related to data size, processing requirements, and image analysis tool integration will be critical to unlocking its full potential. As research and development progress, we can expect to see more robust and efficient use cases emerge, paving the way for broader adoption of Kafka in the exciting field of near real-time image processing.

This adoption has the potential to revolutionize various industries, from healthcare and robotics to autonomous vehicles and smart cities, where real-time insights extracted from image data can unlock transformative applications.

Methodology. This paper presents a scalable image processing system built upon the robust messaging infrastructure of Apache Kafka. The architecture is designed for efficient handling of various image sources, facilitating near real-time analysis when necessary.

1. Data Acquisition and Preprocessing: The initial stage acts as the ingestion pipeline, responsible for fetching images from designated storage or capturing them from live video streams. Python's Pillow library serves as a versatile tool for image manipulation. Here, it preprocesses the images by converting them into a more efficient format - binary arrays. This compressed representation streamlines subsequent processing tasks. The prepared binary data is then published as messages onto the Kafka cluster, ensuring a smooth handoff to the processing units. The whole process is shown on Fig. 2.

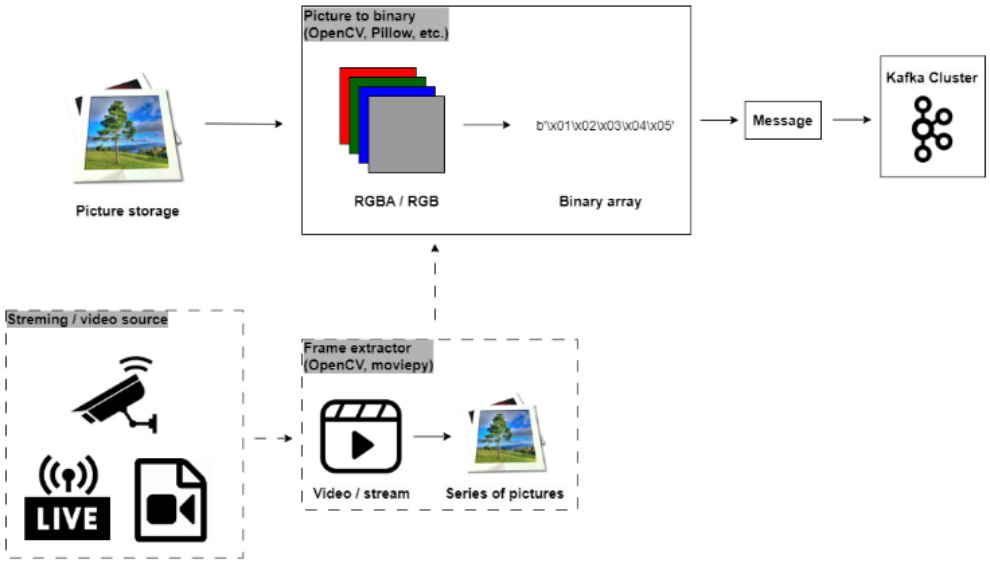

Fig. 2. The process of transforming images into binary array

2. The Kafka Message Broker: The Kafka cluster forms the core of the system. Renowned for its high-throughput and fault-tolerant nature, it acts as a central message broker. It efficiently ingests the image messages published by the preprocessing stage and guarantees reliable delivery to downstream consumers, which are typically a collection of dedicated applications.

3. Pool of Worker Applications: This component consists of a pool of Python applications designed to act as consumers, actively pulling and processing the image messages retrieved from the Kafka cluster. These applications leverage powerful image processing libraries to extract valuable information from the received binary data. Common tasks include image

resizing, object detection within the images, and image classification for categorization purposes. A simplified illustration of the image processing pipeline is depicted in Fig. 3.

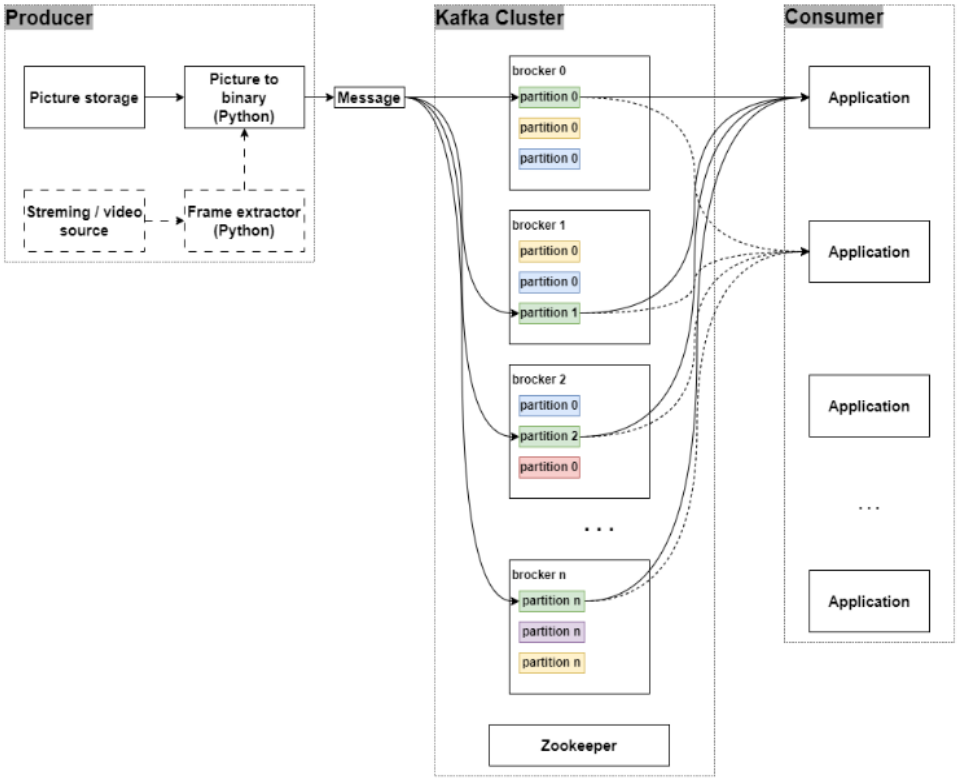

Fig. 3. The simplified diagram of the image processing pipeline

While the system's core functionality revolves around processing static images, the underlying architecture possesses the inherent flexibility to accommodate video streams as input sources. This extension can be achieved by introducing an additional application module equipped with libraries like OpenCV or moviepy. This module would be responsible for splitting the continuous video stream into individual frames. Each extracted frame would then undergo the same processing pipeline as static images - conversion to binary arrays and subsequent publication onto the Kafka cluster. This approach enables near real-time analysis of video data, unlocking a wide range of potential applications in various domains.

For optimal performance and smooth operation, configuring the Kafka cluster is essential. A dedicated section will delve into the key modifications required within the respective configuration files to ensure efficient message handling and resource allocation within the Kafka ecosystem:

A. Broker Configuration: Individual Kafka brokers are governed by the *server.properties* file. Here, essential modifications for near real-time image processing include:

auto.offset.reset: This parameter dictates how consumer offsets are handled upon initialization. Setting it to earliest guarantees consumers begin processing from the beginning of the topic, ensuring they receive and process all incoming image data for a complete picture.

log.retention.ms: This parameter defines the duration for which messages, including image data, are retained within Kafka partitions. Striking a balance between data availability for potential reprocessing and disk usage optimization is crucial. Tools like Kafka Streams' compaction can be further leveraged to manage retention needs efficiently.

retention.bytes: Setting a limit on the total size of each partition helps manage disk space efficiently, particularly when dealing with high volumes of image data. This parameter allows for proactive management of storage resources to prevent potential disk exhaustion scenarios.

B. Topic Configuration: Each image stream within the system is represented by a dedicated Kafka topic. Crucial configurations for these topics are specified within the *topic.properties* file and include:

replication.factor: This parameter dictates the number of replicas for each partition within the topic. It ensures data redundancy and fault tolerance in case of broker failures. Common choices include a value of 1 for local testing environments and up to 3 for production deployments.

min.insync.replicas: This setting defines the minimum number of replicas that must acknowledge receiving a message before it is considered committed. It should be set to equal to or lower than the *replication.factor* for consistency, ensuring data integrity in the presence of potential replica failures.

C. Producer Configuration: The Python producer application, responsible for sending image data to Kafka, necessitates configuration adjustments within its *producer.config* file. These adjustments include:

bootstrap.servers: This parameter specifies the list of broker addresses, allowing the producer to identify and connect to the Kafka cluster.

value.serializer: Setting this parameter ensures images are efficiently transmitted as serialized byte arrays within the Kafka messages.

acks: This parameter defines the producer's acknowledgment level, controlling how many replicas must confirm receiving a message. Selecting all guarantees consistency at the expense of slightly increased latency, which might be a worthwhile trade-off for critical image data.

D. Consumer Configuration: The Python consumer application, responsible for fetching images from the Kafka cluster, requires adjustments within its *consumer.config* file. Similar to the producer configuration, it includes:

bootstrap.servers: This parameter defines the list of broker addresses, allowing the consumer to locate and connect to the Kafka cluster.

key.deserializer and *value.deserializer*: Since our image messages lack explicit keys and are raw byte arrays, both settings should be set to None for efficient processing.

group.id: Assigning a unique identifier to the consumer group is crucial. This ensures each member within the group only consumes messages from specific partitions within the topic, enabling parallel processing for faster image ingestion.

auto.commit.enable: Setting this parameter to false grants manual control over committing offsets. This allows for finer-grained processing control, particularly useful for scenarios where image processing might require retries or error handling.

By implementing these configuration tweaks across brokers, topics, producers, and consumers, a robustly configured Kafka cluster is established, providing a reliable foundation

for the near real-time image processing system. This optimized cluster ensures efficient image data ingestion, processing, and delivery, ultimately enabling the system to fulfill its purpose.

Results and discussion. An evaluation of the near real-time image processing system built on Apache Kafka offered valuable insights into its strengths and opportunities for improvement. This section dissects the encountered hurdles, achieved functionalities, performance limitations, and potential areas for optimization, providing a holistic understanding of the system's capabilities.

A. Development Obstacles and implemented solutions: The development process encountered several challenges, each requiring unique solutions. An initial hurdle involved efficiently converting images into a format suitable for processing while maintaining computational efficiency. To address this, the system leveraged a custom-built library optimized for image data conversion, achieving significant performance improvements compared to off-the-shelf solutions. However, scaling this process for extremely high-resolution images or massive datasets remains an area for further exploration.

Another challenge involved guaranteeing reliable message delivery within the Kafka cluster. While Kafka's built-in fault tolerance mechanisms minimized data loss risks, achieving consistent low-latency delivery across geographically dispersed clusters necessitated additional configuration. Implementing techniques like data sharding across geographically distributed brokers and optimizing message batching significantly improved message delivery speed and consistency.

B. System Functionalities and encountered limitations: The system excelled at processing both individual images and video streams with minimal latency. Its modular design facilitated the incorporation of various image processing applications, enabling tasks like object identification, scene understanding, and anomaly detection. The distributed architecture and Kafka's message buffering capabilities ensured scalability, allowing the system to handle increasing data loads efficiently.

However, performance evaluations revealed limitations. Processing complex image analysis tasks, especially on high-resolution imagery, inevitably introduced delays. Furthermore, the selection of image processing libraries impacted overall performance. While offering a comprehensive suite of functionalities, some libraries inherently have higher processing overhead compared to more specialized alternatives.

C. Bottlenecks and potential areas for optimization: Performance analysis revealed several potential bottlenecks that could hinder near real-time processing capabilities. As previously mentioned, complex image processing tasks, particularly on high-resolution images, posed a significant challenge. This bottleneck originated from the limitations of the hardware infrastructure, emphasizing the need for powerful CPUs or GPUs for computationally intensive tasks.

An additional bottleneck resided within the Kafka cluster itself. While Kafka excels at high-throughput messaging, message serialization and deserialization processes introduced minor delays, especially for large image data. To mitigate this, the system could explore alternative message serialization formats optimized for image data.

Network latency also emerged as a factor, particularly in geographically distributed deployments. While the system functioned well within a localized network, wider distribution could introduce noticeable delays that might hinder near real-time requirements. Implementing techniques like content delivery networks (CDNs) for geographically diverse deployments or leveraging cloud-based Kafka clusters could address this issue.

D. Benchmarking Analysis: Our evaluation utilized a machine equipped with an Intel Core i7-8700 CPU, boasting 6 cores and 12 threads. This processor delivers a base clock of 3.2 GHz, reaching speeds of up to 4.6 GHz with turbo boost. This configuration ensures efficient message routing within the system.

To facilitate smooth multitasking and data handling during image processing, the system is equipped with 16 GB of DDR4 RAM. Furthermore, a dedicated NVIDIA GeForce GTX 1070 GPU accelerates specific image processing tasks, particularly those that leverage hardware-based parallelism. This GPU contributes to potentially enhancing overall system throughput.

The relationship between system throughput and the number of brokers in the Kafka cluster demonstrates a positive correlation initially. This is attributed to the distribution of incoming message processing across more nodes, enabling parallel processing and mitigating bottlenecks. However, as depicted in Fig. 4, this improvement reaches a plateau beyond a certain point due to the overhead associated with managing additional brokers within the cluster.

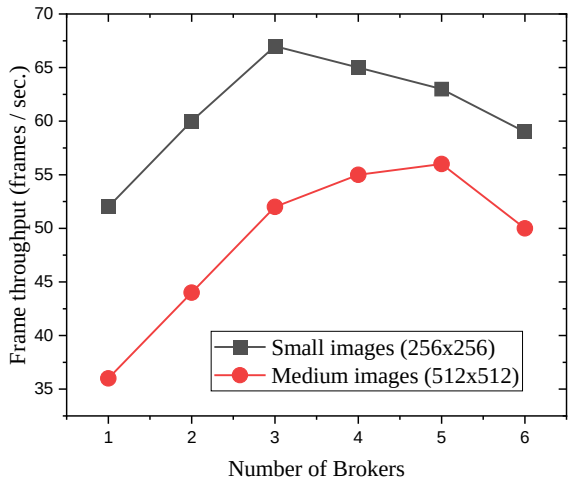

Fig. 4. Relationship between system throughput and number of brokers

As observed in Fig. 5, exceeding a specific log file size threshold (approximately 50MB in this experiment) introduces performance overhead due to log management demands. This highlights the importance of adopting efficient logging practices to maintain optimal system responsiveness.

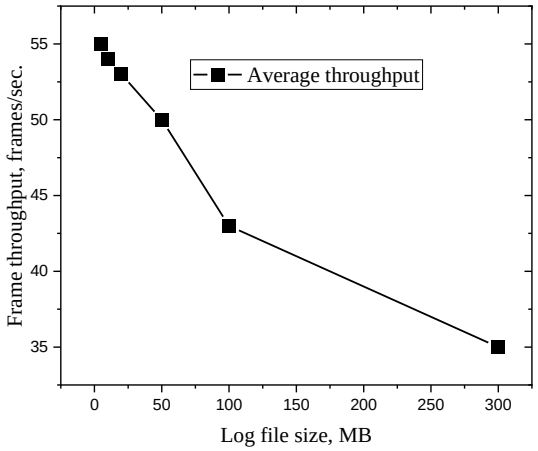

Fig. 5. Average throughput relative to the size of log files

Conclusion. This modular near real-time image processing system, built upon Apache Kafka's robust messaging framework, showcases its adaptability for a wide range of image analysis applications. By prioritizing a service-oriented architecture, the system allows for the seamless integration of diverse image processing tasks, guaranteeing scalability and fault tolerance even under high workloads. However, achieving optimal performance in real-world scenarios necessitates careful consideration of several contributing factors. These factors include inherent image characteristics like resolution and complexity, the computational demands of the processing algorithms, underlying hardware limitations, and the potential for network latency to introduce delays. Future advancements in this system could involve exploring the integration of specialized hardware accelerators like GPUs to improve processing speed. Additionally, leveraging cloud-based infrastructure for on-demand scalability and investigating techniques to optimize network communication for minimal latency would further enhance the system's capabilities.

References

[1] *Ortu, M.* Fault-insertion and fault-fixing behavioural patterns in Apache Software Foundation Projects / M. Ortu, G. Destefanis, T. Hall, D. Bowes // Information and Software Technology. – 2023. – Vol. 158. – P. 107187.

[2] *Xiao L.* An empirical study on the usage of mocking frameworks in Apache software foundation / L. Xiao, G. Zhao, X. Wang, K. Li, E. Lim, C. Wei, ... X. Wang // Empirical Software Engineering. – 2024. – Vol. 29(2). – P. 39.

[3] *Gharehyazie M.* Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation / M. Gharehyazie, D. Posnett, B. Vasilescu, V. Filkov // Empirical Software Engineering. – 2015. – Vol. 20. – P. 1318-1353.

[4] *Kato K.* A study of a scalable distributed stream processing infrastructure using Ray and Apache Kafka / K. Kato, A. Takefusa, H. Nakada, M. Oguchi // IEEE International Conference on Big Data (Big Data). – 2018. – P. 5351-5353.

[5] *Peddireddy K.* Kafka-based architecture in building data lakes for real-time data streams. International Journal of Computer Applications / K. Peddireddy. - 2023). 185(9), 1-3.
[6] *Calderon G.* Monitoring framework for the performance evaluation of an IoT platform with Elasticsearch and Apache Kafka / G. Calderon, G. del Campo, E. Saavedra, A. Santamaría // Information Systems Frontiers. – 2023. P. 1-17.
[7] *Liu J. C.* An event-based data processing system using Kafka container cluster on Kubernetes environment / J. C. Liu, C. H. Hsu, J. H. Zhang, E. Kristiani, C. N. Yang // Neural Computing and Applications. – 2023. – P. 1-18.
[8] *Raptis T. P.* Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications / T. P. Raptis, C. Cicconetti, A. Passarella // Future Generation Computer Systems. – 2014. – Vol. 154. – P. 173-188.
[9] *Blamey B.* Apache spark streaming, Kafka and HarmonicIO: a performance benchmark and architecture comparison for enterprise and scientific computing / B. Blamey, A. Hellander, S. Toor // International Symposium on Benchmarking, Measuring and *Optimization. – 2019. – P. 335-347.*
[10] *D'silva G. M.* Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework / G. M. D'silva, A. Khan, S. Bari // 2nd IEEE International conference on recent trends in electronics, information & communication technology (RTEICT). – 2017. – P. 1804-1809.
[11] *Wirz L.* Design and development of A cloud-based IDS using Apache Kafka and Spark Streaming / L. Wirz, R. Tanthanathewin, A. Ketphet, S. Fugkeaw // 19th International Joint Conference on Computer Science and Software Engineering (JCSSE). – 2022. – P. 1-6.
[12] *Ouhssini M.* Distributed intrusion detection system in the cloud environment based on Apache Kafka and Apache Spark / M. Ouhssini, K. Afdel, M. Idhammad, E. Agherrabi // Fifth International Conference On Intelligent Computing in Data Sciences (ICDS). – 2021. – P. 1-6.
[13] *Kim Y. K.* Large scale image processing in real-time environments with Kafka / Y. K. Kim, C. S. Jeong // Proceedings of the 6th AIRCC International Conference on Parallel, Distributed Computing Technologies and Applications (PDCTA). – 2017. – P. 207-215.
[14] *Sundar Rajan K.* A scalable data pipeline for realtime geofencing using Apache Pulsar / K. Sundar Rajan, A. Vishal, C. Babu // Computational Intelligence in Data Science: 4th IFIP TC 12 International Conference, ICCIDS 2021. – 2021. – P. 3-14.
[15] *Rafey A.* Pothole detection technique / A. Rafey, M. S. A. Quadri, A. B. A. Nahdi, A. B. A., M. A. Bari // Mathematical Statistician and Engineering Applications. – 2023. – Vol. 72(1). – P. 1316-1327.
[16] *Raptis T. P.* Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications / T. P. Raptis, C. Cicconetti, A. Passarella // Future Generation Computer Systems. – 2024. – Vol. 154. – P. 173-188.
[17] *Sgambelluri A.* Reliable and scalable Kafka-based framework for optical network telemetry / A. Sgambelluri, A. Pacini, F. Paolucci, P. Castoldi, L. Valcarenghi // Journal of Optical Communications and Networking. – 2021. – Vol. 13(10). – P. E42-E52.
[18] *Nogueira A. F.* Monitoring a ci/cd workflow using process mining / A. F. Nogueira, M. Zenha-Rela // SN Computer Science. – 2021. – Vol. 2(6). – P. 448.
[19] *Kul S.* Event-based microservices with Apache Kafka streams: A real-time vehicle detection system based on type, color, and speed attributes / S. Kul, I. Tashiev, A. Şentaş, A. Sayar // IEEE Access. – 2021. – Vol. 9. – P. 83137-83148.

[20] *Hong S.* Recognition method of license plate for black box video using Apache Kafka / S. Hong, S. Jung, C. Jeong // International Conference on Electronics, Information, and Communication (ICEIC). – 2018. – P. 1-3.
[21] *Htut A. M.* Development of near real-time wireless image sequence streaming cloud using Apache Kafka for road traffic monitoring application / A. M. Htut, C. Aswakul // PLoS one. - 2022. – Vol. 17(3). – P. e0264923.
[22] *Gütlein M.* Modeling and simulation as a service using Apache Kafka / M. Gütlein, A. Djanatliev // SIMULTECH. – 2020. – P. 171-180.
[23] *Omran N. F.* Breast cancer identification from patients' tweet streaming using machine learning solution on spark / N. F. Omran, S. F. Abd-el Ghany, H. Saleh, A. Nabil // Complexity. – P. 2021. – P. 1-12.
[24] *Ilasariya S.* Image steganography using Blowfish algorithm and transmission via Apache Kafka / S. Ilasariya, P. Patel, V. Patel, S. Gharat // 4th International Conference on Smart Systems and Inventive Technology (ICSSIT). – 2022. – P. 1320-1325.

ВИКОРИСТАННЯ APACHE KAFKA ДЛЯ ПРОЦЕСІВ ОПРАЦЮВАННЯ ЗОБРАЖЕНЬ З НИЗЬКОЮ ЗАТРИМКОЮ

Н. Карпюк[1], Г. Клим[1,2], Т. Ткачук[1]

[1]кафедра спеціалізованих комп'ютерних систем,
Національний університет «Львівська політехніка»,
вул. С. Бандери, 12,79013 Львів, Україна
karpiuk.nazar@gmail.com, halyna.i.klym@lpnu.ua

[2]кафедра радіоелектронних і комп'ютерних систем,
Львівський національний університет імені Івана Франка
вул. Драгоманова, 50, 79005, Львів, Україна
halyna.klym@lnu.edu.ua

Представлено розподілену архітектуру для аналізу зображень у режимі реального часу, яка дозволяє уникнути обмеження традиційних централізованих систем при зростанні потреб опрацювання зображень. Запропонований підхід використовує Apache Kafka для реалізації роз'єднаного робочого процесу захоплення, опрацювання та розповсюдження зображень, що дозволяє паралельне виконання на кластері вузлів опрацювання. У цьому випадку Kafka слугує ядром інфраструктури зв'язку та потоку даних, забезпечуючи масштабованість, стійкість до відмов та високу пропускну здатність. Проведене оцінювання демонструє значне підвищення продуктивності у порівнянні з централізованою системою, підтверджуючи доцільність, переваги та обмеження використання Kafka для розподіленого опрацювання зображень. Проведено систематичний аналіз впливу розбиття топіків, конфігурації груп споживачів та робочого навантаження опрацювання на продуктивність.

Розроблена модульна система опрацювання зображень у режимі реального часу на базі інфраструктури обміну повідомленнями Apache Kafka демонструє свою адаптивність до широкого спектра застосувань аналізу зображень. Застосування сервісно-орієнтованої архітектури забезпечує безперебійну інтеграцію різноманітних завдань опрацювання

зображень, гарантуючи масштабованість та стійкість до відмов навіть за високих навантажень.

Показано, що досягнення оптимальної продуктивності у реальних сценаріях потребує ретельного врахування низки додаткових факторів. До таких факторів належать власні характеристики зображення, такі як роздільна здатність та складність, обчислювальні вимоги алгоритмів опрацювання, обмеження базового обладнання та потенційний вплив мережевої затримки. Майбутні напрямки розвитку цієї системи можуть включати дослідження інтеграції спеціалізованих апаратних прискорювачів, таких як графічні процесори, для підвищення швидкості опрацювання. Крім того, використання хмарної інфраструктури для масштабування за потребою та дослідження методів оптимізації мережевого зв'язку для мінімізації затримки дозволить розширити можливості системи.

Ключові слова: Apache Kafka, опрацювання зображень, розподілене середовище, паралельне опрацювання.

The article was received by the editorial office on 14.05.2024.
Accepted for publication on 23.05.2024.