engineering proceedings

MDPI

Proceeding Paper

Event-Driven Data Orchestration: A Modular Approach for High-Volume Real-Time Processing †

Stanislav Dakov [1] and Megi Dakova [2,*]

[1] Faculty of Mathematics and Informatics, University of Plovdiv "Paisii Hilendarski", 4000 Plovdiv, Bulgaria; stanislav.dakov@uni-plovdiv.bg
[2] Faculty of Physics and Technology, University of Plovdiv "Paisii Hilendarski", 4000 Plovdiv, Bulgaria
* Correspondence: mdakova@uni-plovdiv.bg
† Presented at the 14th International Scientific Conference TechSys 2025—Engineering, Technology and Systems, Plovdiv, Bulgaria, 15–17 May 2025.

Abstract

This article presents a model for orchestrating data extraction, processing, and storage, addressing the challenges posed by diverse data sources and increasing data volumes. The proposed model includes three primary components: data production, data transfer, and data consumption and storage. Key architectures for data production are explored, such as modular designs and distributed processes, each with advantages and limitations regarding scalability, fault tolerance, and resource efficiency. A buffering module is introduced to enable temporary data storage, ensuring resilience and asynchronous processing. The data consumption module focuses on transforming and storing data in data warehouses while providing options for parallel and unified processing architectures to enhance efficiency. Additionally, a notification module demonstrates real-time alerts based on specific data events, integrating seamlessly with messaging platforms like Telegram. The model is designed to ensure adaptability, scalability, and robustness for modern data-driven applications, making it a versatile solution for effective data flow management.

Keywords: big data; producer; consumer; buffer; event-driven

check for updates

Academic Editors: Nikola G. Shakev, Valyo N. Nikolov, Nikolay R. Kakanakov and Sevil A. Ahmed-Shieva

Published: 18 July 2025

Citation: Dakov, S.; Dakova, M. Event-Driven Data Orchestration: A Modular Approach for High-Volume Real-Time Processing. *Eng. Proc.* **2025**, *100*, 48. https://doi.org/10.3390/engproc2025100048

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Data orchestration is the coordinated movement of data from different sources to a single center where it can be stored and processed. This includes collection, pre-processing, quality assurance, and storage of information. For this purpose, various approaches, models, and tools are applied that ensure data security and authenticity. Effective coordination and operation with information enables important judgments and conclusions to be made, leading to successful development. Coordinated management of data extraction, processing, and storage has become paramount for organizations that want to use their information assets successfully. With the proliferation of devices and data sources, a robust model that seamlessly organizes these processes is needed. In this article, we propose a model that we have implemented to meet this challenge. Through it, we can securely, easily, and conveniently control data extraction, processing, and storage. It focuses on the roles of data production modules, buffers, and data stream consumers.

Effective orchestration in data extraction, processing, and storage has been a cornerstone for modern digital systems to unlock the full potential of an organization's information assets. With data volumes, speeds, and types continuing to increase, underpinned by the exponential adoption of IoT devices, cloud services, and enterprise systems, demand

for strong frameworks for data management correspondingly rises [1–3]. Historically, pioneering works have highlighted the need for data organization in order to make decisions and undertake strategic planning. The early works of [4,5] naturally set up the bases for current architectures that would aim at modularity, fault tolerance, and scalability. These early efforts laid the groundwork for contemporary architectures that prioritize modularity, fault tolerance, and scalability [6–8].

This work revisits such principles by proposing an overall model that smoothly integrates data production, buffering, and consumption toward solving contemporary challenges. Unlike previous centralized approaches that suffered from bottlenecks and single points of failure, this model emphasizes modularity, distributed processing, and real-time analytics inspired by recent work in cloud computing and asynchronous processing [6]. It integrates components such as modular adapters, message brokers, and notification systems, among others, to avoid shortcomings like system downtime, wastes of resources, and slow processing.

In addition, notification mechanisms for data changes are integrated into the model, showing its commitment to proactive communication and decision-making. It reflects a shift in focus from static data storage to dynamic data flow, thus fitting well into the real-world demands of e-commerce, healthcare, and finance, among others [9]. Accordingly, this work advances the state of the art in data management from its traditional roots to modern needs for flexibility, efficiency, and fault tolerance in data-driven ecosystems.

In the domain of large-scale data collection and processing, Lambda Architecture and Kappa Architecture [10] have become foundational paradigms. Lambda Architecture combines a batch layer for accurate and complete historical processing with a speed layer for real-time, low-latency updates. While effective for ensuring accuracy and fault tolerance, this dual-layer approach introduces considerable complexity, as it requires maintaining and synchronizing two distinct code paths for the same business logic. To address this, Kappa Architecture was introduced as a simplification, advocating for a unified stream processing model where all data, past and present, is treated as a continuous stream, processed and reprocessed in a single layer. Despite reducing complexity, Kappa Architecture assumes a more uniform data model and may face challenges when accommodating highly heterogeneous data sources or when needing modularity and flexibility in processing logic. By contrast, the proposed event-driven orchestration model advances these paradigms by emphasizing modular adapters for diverse producers, configurable buffering mechanisms, and unified or parallel consumer architectures. This design offers greater adaptability for real-time heterogeneous environments such as e-commerce and IoT systems, where data formats and processing needs can be highly variable. As part of our future work, we plan to conduct a detailed comparative analysis between the proposed model and existing models like Lambda and Kappa Architectures to further validate performance, flexibility, and operational efficiency across diverse application scenarios.

2. Related Work and Background

We consider a basic model for data extraction, transfer, processing, and storage. Its block diagram is shown in Figure 1. It contains three main components:

- The data production module;
- The data transfer module;
- The data processing and storage module.

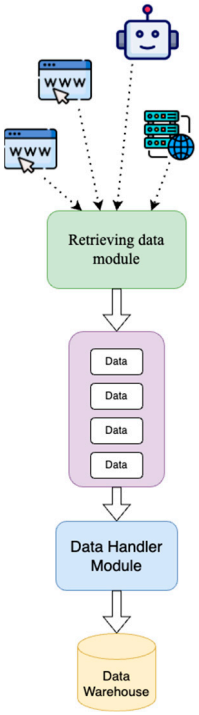


Figure 1. Basic model for data extraction, transfer, processing, and retention.

We will briefly present their essence and their main operations.

2.1. Data Producer Module

The data producer plays a central role in the model. It is a core component that aims to collect data from various devices, applications, and spaces. These devices can range from IoT sensors and mobile applications to enterprise systems and external APIs [11]. The producer service is the main module through which data flows in the system, using protocols and interfaces tailored to the characteristics of each data source. The data producer allows the ability to adapt to different data formats, transmission protocols, and authentication mechanisms, ensuring compatibility with heterogeneous environments.

In this article, we will look at two main data producer architectures:

- Modular Data Producer

This component variant is distinguished by its modular architecture, which includes a flexible adapter framework that integrates with various data sources. This component serves as a unified entry point for data ingestion, abstracting the complexity of device-specific protocols and interfaces. At its core are its adapter modules shown in Figure 2, each of which is tailored to the unique characteristics of different devices and data streams.

These adapters act as modular components [12] within the producer service, enabling asynchronous [13] data extraction based on predefined rules and configurations. This module manages a set of additional adapters. Each adapter is responsible for extracting data from a specific device. Using asynchronous data extraction not only improves performance and scalability but also facilitates real-time processing and analysis of streaming data [14].

The advantage of this modular approach lies in its ability to accommodate a wide range of data sources. Each adapter module is responsible for interacting with a specific type of device or data stream, using protocols and communication mechanisms optimized for efficient data transfer.

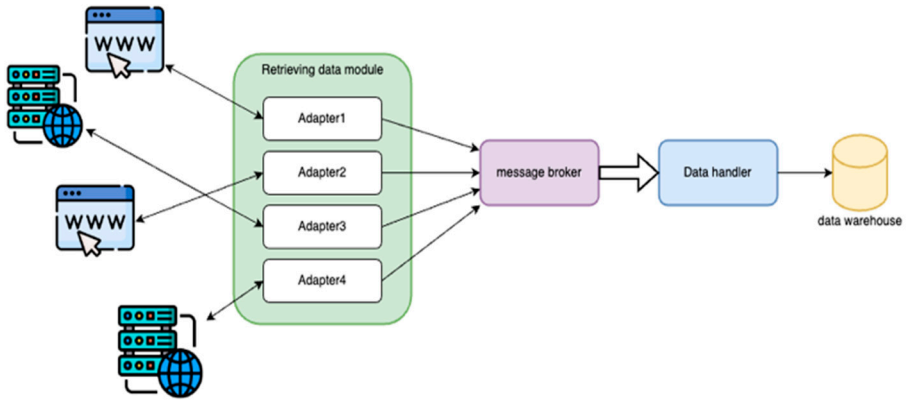


Figure 2. Modular adapter-based data ingestion architecture.

This module architecture not only simplifies management and monitoring but also improves fault tolerance and resilience, since the failure of one adapter module does not affect the operation of others. Although the modular architecture [15] and its diverse adapter modules offer numerous advantages, it also introduces potential disadvantages, especially in terms of system reliability and fault tolerance. One of the biggest disadvantages of this option is the single point of failure. In the event of a service outage or application failure, all data extraction processes stop, resulting in a complete halt in data collection from all devices.

This vulnerability not only disrupts real-time data streams but also jeopardizes critical operations that depend on timely data information. Additionally, the complexity of managing multiple adapter modules within a single application or container increases the risk of cascading failures, where a failure in one module can propagate to others, amplifying the impact on data retrieval capabilities.

- Each data producer in a separate process

Instead of having a single process managing multiple adapters in this case, each adapter is represented as a separate process; as shown in Figure 3, there is no centralization of the adapters.

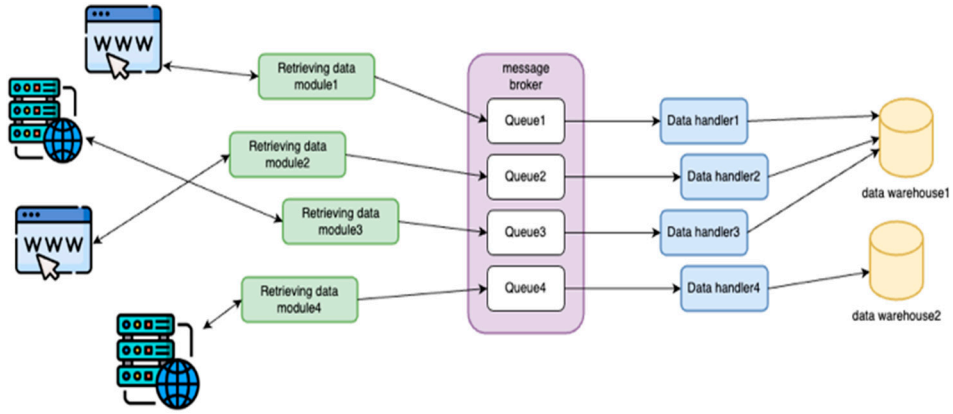


Figure 3. Decentralized architecture for independent data adapters.

These multiple processes data processors, each dedicated to a specific device, introduce a distributed architecture that offers several advantages: increased fault tolerance [16] and resiliency. By isolating the processors for each device in separate processes, a failure in one processor does not affect the operation of the others, ensuring continuous data extraction from unaffected devices. This architecture promotes scalability, as each processor can be

independently scaled based on the volume and bandwidth requirements of the device connected to it.

This allows for efficient resource allocation and optimization, preventing bottlenecks, and ensuring optimal performance across the system. In addition, distributing processors across different processes facilitates easier management and monitoring, allowing for granular control and problem isolation.

This architecture also creates certain challenges. Managing a large number of independent processor processes complicates deployment, configuration, and maintenance. Coordinating interactions between multiple processors and downstream components, such as buffers [17], requires more open connections to the data buffer areas. They are finite in number, and this limits performance.

This multitude of processes impacts system resources, especially when we have a large number of devices or variable volumes of data. Ensuring consistency and synchronization between different production processes can be challenging, especially in distributed environments where network latency [18] and reliability issues can arise.

2.2. Data Buffering Module

To ensure that all data has sufficient time to be processed, we provide a place to temporarily store the data before processing. We call it a buffer for temporary storage. After data is collected by the data collection component, it is passed to the buffer for temporary storage. The buffer serves as a basic intermediary, separating the data consumption process from the downstream processing and storage operations. This separation improves system resilience, scalability, and flexibility by enabling asynchronous data flow and load balancing. The buffer can take various forms, including message brokers [19], databases, files, or cloud storage services, depending on the specific requirements of the system. We suggest using message brokers like Apache Kafka [20] or RabbitMQ [21] for the buffer zone because they offer robust queuing and streaming capabilities, allowing for seamless data transfer and distribution. Databases, on the other hand, provide persistence and query capabilities, allowing for complex transformations and data enrichment. The important thing for this module is that the data that enters and leaves the buffer does not change.

2.3. Data Consumption and Processing Module

Once the data is passed to the buffer, it is retrieved, consumed, and processed by a component responsible for extracting information, performing analytics, and storing the information in structured and unstructured formats [22]. This module uses the buffered data to execute predefined processing pipelines, including tasks such as data cleansing, transformation, aggregation, and enrichment.

The role of the consumer is to process and write the data to a specific repository. These repositories are also called data warehouses [23]. A data warehouse is a centralized repository designed to store and manage large amounts of structured and semi-structured data from multiple sources, optimized for querying, reporting, and analysis, unlike a functional time-sensitive database, personal internal communications processing, or purpose-built data warehouses for decision support. There are three main types of data warehouses: enterprise data warehouses (EDWs), which provide comprehensiveness and organizational data integrated thinking; the operational data warehouse (ODS), which maintains interim operational reports; and data marts, which are small domain-specific data warehouses tailored to individual departments or offices. By implementing validation mechanisms, error handling strategies, and data management policies, the module reduces the risk of erroneous or inconsistent data [24] entering the data store. In addition, this component can have a different set of rules according to which it processes the data.

- The component as a set of processes

Implementing the consumer module as a parallel process for processing and storing data from different queues has several advantages in terms of performance and scalability. One notable advantage is the ability to use parallelism to improve processing performance and reduce latency. By simultaneously processing data from multiple queues, the consumer module can use the available computing resources more efficiently, maximizing processing speed and minimizing processing time. This parallel processing capability allows the system to accommodate large volumes of data and meet strict latency requirements, ensuring timely data ingestion and analysis. The ability to scale horizontally by implementing multiple data processing processes, shown in Figure 3, for each queue, allows for seamless expansion to meet growing demands, further improving the scalability and responsiveness of the system.

However, this approach also creates certain challenges and trade-offs. Managing multiple user processes or handlers, each dedicated to a specific queue, can introduce complexity in implementation, configuration, and monitoring. Coordinating interactions between multiple user instances and downstream storage systems may require robust orchestration mechanisms to ensure efficient data flow and fault tolerance.

In addition, the overhead of managing multiple user processes or handlers can impact resource utilization and impose additional operational costs, especially in terms of memory and processor overhead. In addition, ensuring consistency and synchronization between parallel processors can pose challenges, especially in distributed environments where data ordering and transaction integrity are critical. Therefore, while a parallel user architecture offers performance and scalability advantages, careful consideration must be given to the associated complexities and overhead, balancing efficiency with manageability and reliability.

Using the module with a unified worker process but including separate subscribers for each queue, as shown in Figure 4, offers multiple advantages in terms of efficiency, resource utilization, and manageability. One significant advantage is resource allocation, as a single-worker process reduces costs compared to managing multiple independent processes. This results in optimized memory and processor usage, ensuring efficient use of computing resources and minimizing operating costs.

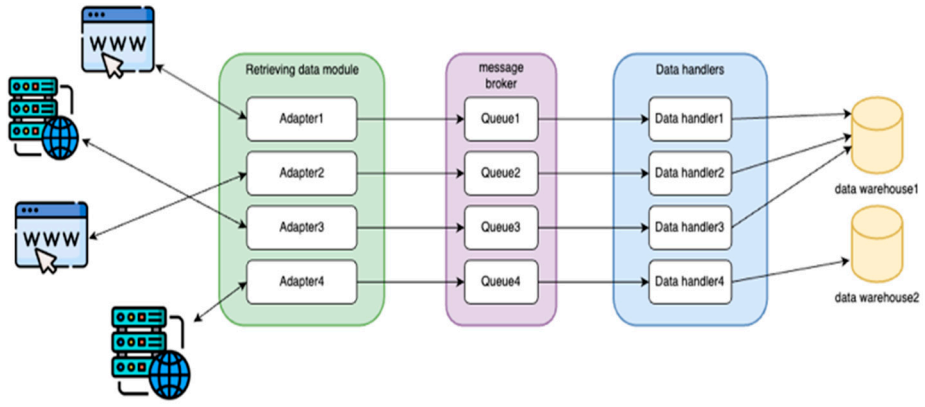


Figure 4. Parallel consumer architecture for high-throughput processing.

In addition, the unified architecture simplifies deployment, configuration, and monitoring, facilitating easier system management and maintenance. We performed a test comparing a single Java [25] module that handles 10 queues and a separate Java module for each queue. It turned out that using a separate application for each queue used eight times more resources. However, this data storage mechanism allows for non-blocking operations, ensuring smooth and uninterrupted data flow even during peak loads or transient

peaks in traffic. This consumer module architecture promotes modularity and extensibility, facilitating easier integration of new data sources or processing logic. Each subscriber can be independently configured and customized to accommodate specific data formats, protocols, or processing requirements, allowing rapid adaptation to changing business needs or technological advancements. This modularity promotes code reuse and maintainability, as common functionality can be encapsulated and shared across different subscribers.

2.4. Event Detection and Notification System

The flexibility of this model makes it easy to implement additional functionality specific to business needs. There is data that the business needs to know about. We successfully implemented a notification module for data changes in only one of the data streams, without affecting the other streams. The consumer module, in addition to its data processing capabilities, plays a crucial role in facilitating real-time notifications to users based on predefined rules and conditions. By integrating with the notification module shown in Figure 5, the consumer enables proactive communication with users, alerting them to significant events or changes detected in the data stream.

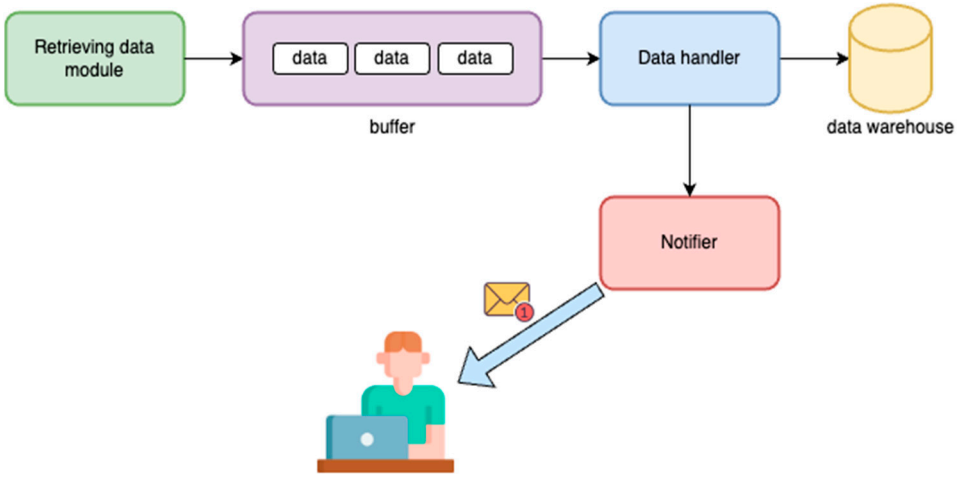


Figure 5. Event-driven notification module for real-time alerts.

One of the main advantages of this integrated approach is its ability to automate the notification process based on user-defined rules and thresholds. For example, the consumer module can monitor incoming data for specific patterns, anomalies, or trends and trigger notifications when predefined criteria are met. This notification module enables users to be informed of critical events or developments in their data, allowing them to respond quickly and effectively. The flexibility of the notification module allows for a diverse range of notification channels to be used, meeting the preferences and requirements of individual users. Whether it is sending email, SMS, or using third-party messaging apps such as Telegram [26], as shown in Figure 6, the notification module ensures that notifications are delivered through the most appropriate and convenient channels.

This multi-channel approach improves accessibility and user engagement by ensuring that notifications reach users in a timely and efficient manner. By leveraging third-party integrations, such as APIs provided by messaging platforms like Telegram, the notification module can extend its capabilities beyond traditional communication channels. This allows users to receive notifications directly on their preferred messaging platforms, leveraging the familiarity and convenience of their existing communication tools. Additionally, by integrating with third-party applications, the notification module can access advanced features and functionalities, such as rich media content, interactive messages, and real-time updates, improving user experience and engagement.

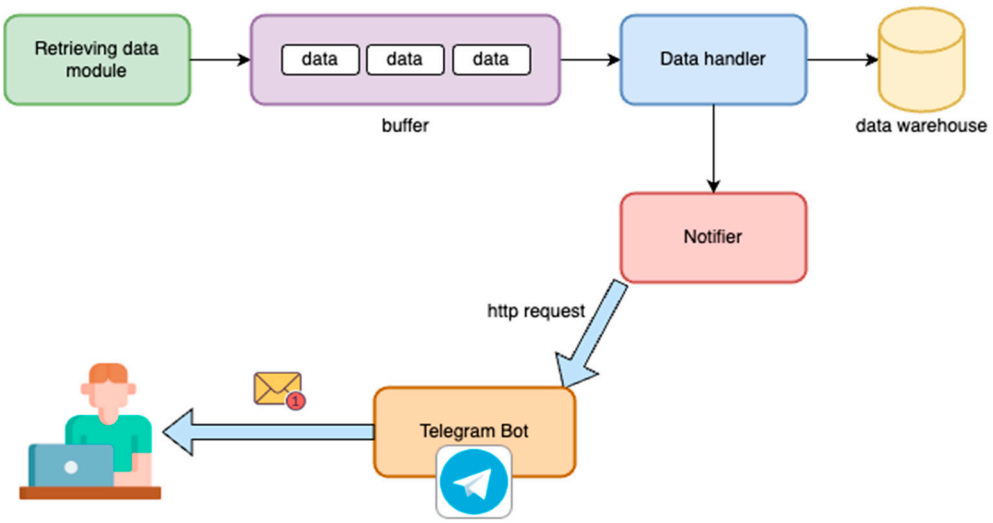


Figure 6. Integration of notification system with Telegram messaging platform.

3. Resource Optimization and Performance Insights

To evaluate the efficiency of different consumer architectures within the proposed data orchestration model, we conducted a comparative benchmark focusing on CPU utilization, memory consumption, latency, and throughput.

3.1. Experimental Data and Scenario

The experimental scenario was designed to evaluate the performance and flexibility of the proposed orchestration model in handling real-world data collection and processing it. Specifically, the scenario focused on collecting and processing data from four distinct simulated data sources, each representing a different origin type and emulating common data generation patterns observed in information, for example. The objective of the scenario was to validate how the system performs under diverse and asynchronous data loads, and to compare the efficiency of the two proposed architectures—parallel and unified—under identical conditions.

For data generation, we developed a dedicated external application responsible for producing randomized objects. These objects contained varied attributes, including identifiers, names, preferences, and behavioral indicators, designed to reflect diverse and semi-structured data formats commonly encountered in dynamic user-centric environments. The data retrieval service, implemented with built-in adapters as part of the producer module, established connections to these external applications to ingest the simulated data in real time.

The simulated environment enabled us to

- Emulate real-world data heterogeneity, as objects featured variations in attributes;
- Test the adaptability of the ingestion layer to dynamic data formats through modular adapters;
- Validate processing and storage efficiency under realistic streaming workloads.

For data persistence, we used a NoSQL database, which was selected based on its suitability for storing semi-structured and schema-flexible data types. NoSQL storage ensured that the variability in incoming objects could be accommodated without the need for rigid schema definitions, aligning well with the principles of scalability and adaptability.

3.2. Performance Benchmarking and Results

The results, summarized in Table 1 and Figure 7, reveal distinct trade-offs between parallel and unified consumption strategies. The Parallel Consumers architecture, while

achieving a slightly higher throughput of 5000 messages/s, incurred a significantly higher CPU usage (75%) and memory consumption (1200 MB). In contrast, the Unified Consumer design maintained a competitive throughput of 4500 messages/s, while demonstrating substantially lower CPU usage (40%) and memory usage (300 MB).

Table 1. Parallel Consumers and Unified Consumer results.

Architecture	CPU Usage (%)	Memory Usage (MB)	Throughput (Messages/s)
Parallel Consumers	75	1200	5000
Unified Consumer	40	300	4500

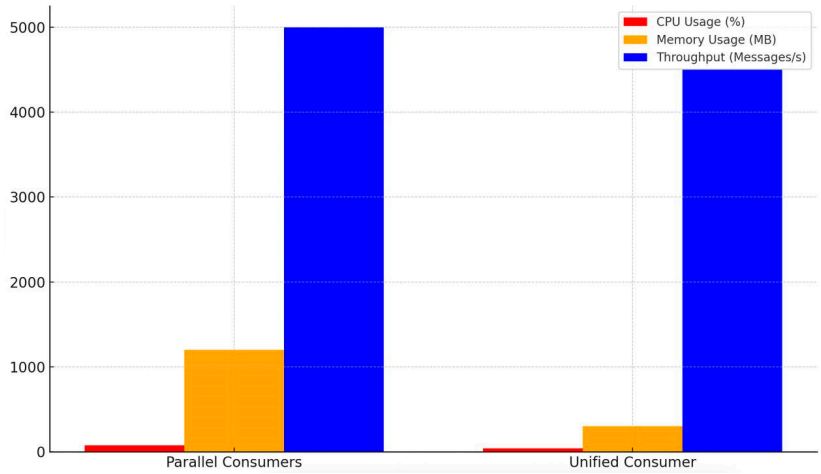


Figure 7. Parallel Consumers and Unified Consumer results.

These results suggest that although the Parallel Consumers model offers marginally better throughput, it does so at the expense of resource efficiency. The unified model emerges as a more sustainable solution, particularly suitable for resource-constrained environments or cloud-native deployments with cost and performance trade-offs. This evaluation validates the architectural decision to support both configurations within the model, enabling system designers to balance performance and efficiency based on domain-specific requirements.

3.3. Results of System Objectives

The implications of these findings are directly related to the model's overarching objectives:

Adaptability: The ability to seamlessly ingest variable User objects using modular adapters and store them in a schema-flexible NoSQL database demonstrates the model's adaptability to diverse data types and application domains.

Scalability: Reduced CPU and memory consumption in the unified model enhances scalability, as additional consumers can be deployed cost-effectively to meet increasing demand without performance degradation.

Robustness: Despite lower resource consumption, the unified model sustained near-equivalent throughput and ensured smooth data flow, thus maintaining robustness under real-time processing conditions.

These results validate the proposed model's capacity to meet the goals of adaptability, scalability, and robustness, making it suitable for modern data-driven applications.

4. Conclusions

In this study, we proposed a robust and adaptable model for orchestrating data extraction, processing, and storage, addressing the growing need for efficient and scalable data management solutions. The model incorporates modular and distributed architectures for data production, ensuring compatibility with heterogeneous data sources while balancing scalability and fault tolerance. A buffering module was introduced as a crucial intermediary to decouple data ingestion and processing, enhancing system resilience and load balancing. Furthermore, the data consumption and storage module demonstrated multiple approaches to optimize performance and scalability, particularly through parallel processing and unified worker processes.

We also showcased the implementation of an integrated notification module, highlighting its ability to provide real-time alerts tailored to specific business needs. This flexible architecture allows seamless integration of additional functionalities without disrupting existing workflows. Despite some challenges, such as managing distributed processes and ensuring consistency, the proposed model effectively addresses the critical demands of modern data-driven systems. It lays a strong foundation for future advancements in data orchestration, offering organizations a scalable, fault-tolerant, and efficient framework for leveraging their data assets.

In the present, defined by complex and pervasive information, using such a data orchestration model would contribute to driving innovation, gaining competitive advantage, and delivering a superior user experience.

Author Contributions: Introduction S.D. and M.D.; Conceptualization S.D. and M.D.; visualization S.D. and M.D.; Related Work and Background S.D.; Module platform S.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Research Fund, MUPD2-2-FTF-023, National Program "Young Scientists and Postdoctoral Fellows—2" at the University of Plovdiv "Paisii Hilendarski".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Data are contained within the article.

Acknowledgments: This article is part of the work on the project NP "Young scientists, doctoral students and postdoctoral fellows"—2, at the University of Plovdiv "Paisii Hilendarski".

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Murad, S. Big data marketing. In *The Essentials of Today's Marketing*; Tarakçı, i.E., Aslan, R., Eds.; Efe Academy Publishing: Muğla, Turkey, 2023; pp. 265–284.
2. Divyeshkumar, V. Big Data in Marketing Strategy. Available online: https://www.igi-global.com/gateway/chapter/359370 (accessed on 1 July 2025).
3. K, S.K. Exploring Real-Time Data Processing Using Big Data Frameworks. *Commun. Appl. Nonlinear Anal.* **2024**, *31*, 620–634. [CrossRef]
4. Inmon, W.H. *Building the Data Warehouse*; Wiley: Hoboken, NJ, USA, 1995.
5. Kimball, R. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*; Wiley: Hoboken, NJ, USA, 1996.
6. Bystrov, S.; Kushnerov, A. Asynchronous Data Processing. Behavior Analysis. 2022. Available online: https://www.researchgate.net/profile/Alexander-Kushnerov/publication/362910934_Asynchronous_Data_Processing_Behavior_Analysis/links/6307535a61e4553b9538f614/Asynchronous-Data-Processing-Behavior-Analysis.pdf (accessed on 15 July 2025).
7. Alfaia, E.C.; Dusi, M.; Fiori, L.; Gringoli, F.; Niccolini, S. Fault-Tolerant Streaming Computation with BlockMon. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015.
8. Pogiatzis, A.; Samakovitis, G. An Event-Driven Serverless ETL Pipeline on AWS. *Appl. Sci.* **2020**, *11*, 191. [CrossRef]

9. Khriji, S.; Benbelgacem, Y.; Chéour, R.; Houssaini, D.E.; Kanoun, O. Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *J. Supercomput.* **2022**, *78*, 3374–3401. [CrossRef]
10. Tatipamula, S. Real-Time vs. Batch Data Processing: When speed matters. *World J. Adv. Res. Rev.* **2025**, *26*, 1612–1631. [CrossRef]
11. Speckhard, D.; Bechtel, T.; Ghiringhelli, L.M.; Kuban, M.; Rigamonti, S.; Draxl, C. How big is Big Data? *Faraday Discuss.* **2024**, *256*, 483–502. [CrossRef] [PubMed]
12. Holmstedt, J. Development of a modular open systems approach to achieve power distribution component commonality. In Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), 2024, Novi, MI, USA, 15–17 August 2023. [CrossRef]
13. Wang, M. Design and implementation of asynchronous FIFO. *Appl. Comput. Eng.* **2024**, *70*, 220–226. [CrossRef]
14. Meixia, M.; Siqi, Z.; Jiawei, L.; Jianghong, W. Aggregatably Verifiable Data Streaming. *IEEE Internet Things J.* **2024**, *11*, 24109–24122. [CrossRef]
15. Arora, D.; Sonwane, A.; Wadhwa, N.; Mehrotra, A.; Utpala, S.; Bairi, R.; Kanade, A.; Natarajan, N. MASAI: Modular Architecture for Software-engineering AI Agents. *arXiv* **2024**, arXiv:2406.11638. [CrossRef]
16. Mainali, D.; Nagarkoti, M.; Dangol, J.; Pandit, D.; Adhikari, O.; Sharma, O.P. Cloud Computing Fault Tolerance. *Int. J. Innov. Sci. Res. Technol. (IJISRT)* **2024**, *9*, 220–225. [CrossRef]
17. Bouvier, T.; Nicolae, B.; Chaugier, H.; Costan, A.; Foster, I.; Antoniu, G. Efficient Data-Parallel Continual Learning with Asynchronous Distributed Rehearsal Buffers. In Proceedings of the 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Philadelphia, PA, USA, 6–9 May 2024.
18. Kim, M.; Jang, J.; Choi, Y.; Yang, H.J. Distributed Task Offloading and Resource Allocation for Latency Minimization in Mobile Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2024**, *23*, 15149–15166. [CrossRef]
19. Dyjach, S.; Plechawska-Wójcik, M. Efficiency comparison of message brokers. *J. Comput. Sci. Inst.* **2024**, *31*, 116–123. [CrossRef]
20. Mager, T. Big Data Forensics on Apache Kafka. In Proceedings of the International Conference on Information Systems Security, Raipur, India, 16–20 December 2023; pp. 42–56.
21. Ayanoglu, E.; Aytas, Y.; Nahum, D. *Mastering RabbitMQ*; Packt Publishing Ltd.: Birmingham, UK, 2016.
22. Banowosari, L.Y.; Purnamasari, D. Approach for Unwrapping the Unstructured to Structured Data the Case of Classified Ads in HTML Format. *Adv. Sci. Lett.* **2016**, *22*, 1909–1913. [CrossRef]
23. Cheruku, S.R.; Jain, S.; Aggarwal, A. Building Scalable Data Warehouses: Best Practices and Case Studies. *Mod. Dyn. Math. Prog.* **2016**, *1*, 116–130. [CrossRef]
24. Qi, Z.; Wang, H.; Dong, Z. Feature Selection on Inconsistent Data. In *Dirty Data Processing for Machine Learning*; Springer: Singapore, 2023.
25. Singh, A.; Prasad, R. Java Classes. 2024. Available online: https://www.researchgate.net/publication/380464284_Java_Classes?channel=doi&linkId=663da12008aa54017af11b2b&showFulltext=true (accessed on 15 July 2025).
26. Setiawan, I.P.E.; Desnanjaya, I.G.M.N.; Supartha, K.D.G.; Ariana, A.G.B.; Putra, I.D.P.G.W. Implementation of Telegram Notification System for Motorbike Accidents Based on Internet of Things. *J. Galaksi* **2024**, *1*, 1–11. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.