ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 2, 2025

ADAPTIVE ORCHESTRATION MECHANISMS FOR EFFICIENT SERVERLESS COLLECTION OF HETEROGENEOUS ENVIRONMENTAL DATA

Oleksandr Demidov, Oksana Honsor

Lviv Polytechnic National University, 12, Bandera Str, Lviv, 79013, Ukraine.
Authors' e-mails: *oleksandr.s.demidov@lpnu.ua oksana.y.honsor@lpnu.ua*

https://doi.org/10.23939/acps2025.02.129

Submitted on 22.09.2025

Abstract: **The rapid expansion of cyber-physical systems (CPS) has intensified the need for scalable and adaptive mechanisms to collect heterogeneous environmental data from numerous unstable external sources. Traditional serverless orchestration frameworks, while elastic and cost-efficient, lack runtime adaptability and feedback-awareness, leading to inefficiencies under dynamic API conditions. This paper presents a novel adaptive orchestration model for serverless data collection pipelines, driven by metadata configuration and continuous feedback control. The proposed system integrates AWS-based components (Lambda, EventBridge, SQS, S3, Athena, MongoDB) to enable autonomous management of data collection processes from OpenAQ, NOAA, NASA GES DISC, and ESA Copernicus. Adaptive behavior has been achieved through feedback-based health scoring and metadata-driven reconfiguration, improving resilience to vendor instability, API schema changes, and rate-limit fluctuations. Experimental validation has demonstrated a 40% reduction in failed invocations, a 22% latency improvement, and a 17% decrease in operational costs compared to static orchestration approaches. The results confirm the feasibility of fully adaptive, serverless data orchestration and establish the groundwork for future AI-assisted autonomous orchestration in heterogeneous CPS environments.**

Index terms**: serverless architecture, adaptive orchestration, metadata-driven configuration, cloud computing, environmental data, AWS.**

I. INTRODUCTION

The enormous growth in the need for data in cyber-physical systems (CPS) has led to a demand for new systems to collect them. These are systems that have resilient and scalable infrastructure to maintain stability under a huge amount of load from various data sources. Not only that, but the fault-tolerance and ability to process heterogeneous (different size, type, and format) data as close to real-time as possible. In the environmental and climate change domain, we have diverse data streams that generate a large amount of data for such metrics as air quality, earth surface analysis, climate observations, and so on. Those include:

- OpenAQ: a network that detects atmospheric pollutants
- NOAA (National Oceanic and Atmospheric Administration): for meteorological and oceanic observations, is also a primary source for weather and climate data
- GES DISC (NASA Goddard Earth Sciences Data and Information Services Center): for atmospheric datasets, responsible for archiving, managing, and distributing Earth science data
- ESA (European Space Agency) Copernicus: for satellite imagery and earth observations.

Together, those sources provide a vast amount of heterogeneous data to analyze in near real-time and represent a great dataset for further analysis, which is perfect for testing such an adaptive serverless system.

The domain paradigm of serverless computing has already proven itself to be independent, elastic, highly scalable, and cost-efficient [1, 2]. Most of the platforms for cloud computing (such as AWS) have already provided event-driven solutions to address the demand for automatically scaling resources, without any manual provisioning. Despite all the advantages, a lot of serverless systems remain highly vulnerable and inefficient due to their static architecture. They depend on fixed schedules, have strict rules, and most importantly, they rely on consistent and stable data sources, which appear not to be the case for environmental data most of the time. Environmental data API's, at least those mentioned in this article, experience frequent changes: in schemas, rate limits, update timings, and temporary unavailability. That leads to missing data points, unnecessary executions, and errors due to rate limits if not addressed correctly [3, 4].

Although some previous studies demonstrate the scalability of serverless architectures for data collection purposes, very little attention was paid to one of the most important topics in this domain – adaptive behavior under constantly changing external conditions. The lack of orchestration and feedback of the system often prevents systems from adjusting. And the topic of bringing together data in various formats and analyzing them in near-real-time environments is also hugely unspoken.

This work expands previous studies in some fields. Most specifically, it focuses on measuring a serverless data-collection system based on metadata-driven and feedback-controlled orchestration mechanisms. While

collecting vendor-specific data, the proposed model continuously tracks and analyzes feedback in such metrics as: latency, failure rate, data novelty, and more to always keep track of each vendor's health state.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Usage of serverless systems has become a default solution for most modern distributed systems and applications. Due to its ease in terms of setting up, managing resources, cost efficiency, and many other advantages. However, for large, long-running operations, orchestration of serverless components remains crucial, as it has been stated in prior studies [5, 6].

There are existing solutions that have been proposed. Most of them are based on AWS Step Functions, Apache Airflow, and Azure Durable Functions. They all seem to be well-thought-out solutions at first, but when they encounter problems with heterogeneous data (such as environmental data), they aren't suitable enough. Those solutions provide robust orchestration capabilities, but only under the assumption of static configuration, predefined triggers, expected working time, and stability of data sources. Some studies tried to improve the described orchestration workflow using standardization and rule-based logic [7, 8], but they all lacked reactiveness and feedback awareness.

Self-regulating, adaptive solutions in serverless systems have advanced a lot recently. They seem very promising and have shown huge potential in terms of integration in orchestration mechanisms and feedback adaptability [9]. Dynamic resource allocation modules and learning-based schedulers have shown measurable growth in performance and stability. However vast majority of those solutions are made for container-based or microservices solutions, not for event-driven, serverless architectures. So, the lack of runtime adaptive mechanisms in serverless architecture remains, especially in such unstable and heterogeneous environments.

The same pace of growth can be seen in the data collection of the environmental data domain. Vendors, such as OpenAQ, NOAA, NASA GES DISC, and ESA Copernicus, expose APIs with different schemes, formats, quotas, schedules, they are even sensitive to different downtime and format changes. Prior works, focused on environmental data collections, often focus on standardization and fusion of data from different sources [10]. But none of them investigates adaptive orchestration for continuous data collection, transformation, fusion, and recovery. Those aspects are crucial in this specific domain and for maintaining real-world CPS pipelines.

Summarizing the previous statements, we currently have a situation where the orchestration of cloud pipelines has advanced significantly, but most of those pipelines have not been ready yet for real-life challenges, as we often see when collecting environmental data. Those systems demonstrate low fault tolerance, inefficiency for heterogeneous data, including excessive retries, and require manual reconfiguration when API changes occur.

The primary objective of the present study is to develop a methodology that facilitates the implementation of autonomous, adaptive coordination mechanisms for the effective collection of heterogeneous data about the environment, utilizing serverless technology.

With primary challenges being:

- Providing mechanisms of runtime feedback communication to detect and address the vendor's instabilities.
- Providing a metadata abstraction layer to enable real-time adjustments of orchestration logic.
- Adding fault-resistant mechanisms that provide efficiency under severe data collection circumstances.

This paper addresses the mentioned challenges and provides a pro**totype of a system based on an adaptive modular orchestration** mechanism, based on metadata and real-time feedback communication. Aiming to increase efficiency, adaptability, fault-tolerance, and cost efficiency.

III. SCOPE OF WORK AND OBJECTIVES

The main field of research is enhancing the effectiveness and fault tolerance of serverless data-collection systems through adaptive orchestration mechanisms. Research is specifically focused on heterogeneous data in dynamically changing, unstable environments, where data comes from different vendors with varying timing, stability, rate limits, and structures.

Implementation is based on AWS infrastructure, involving lots of services and frameworks, but some of the core components are the following: AWS Event Bridge, Lambda, SQS, Athena, and MongoDB. For event management, processing, orchestration, aggregation, and data metadata storing, accordingly. The system's ability was validated using those 4 environmental data vendors:

- OpenAQ: exposes air quality conditions in near real-time.
- NOAA: responsible for climate and meteorological changes.
- NASA GES DISC: atmospheric datasets.
- ESA Copernicus: satellite-based geospatial and land-cover data.

Not only are those datasets well-compatible in terms of usability and value of data combined, they also have different data structures, formats, different rate limits, latency, and amounts of data. That makes those sources great for testing an adaptive orchestration model under severe circumstances. System supports dynamic scheduling, a metadata-driven approach, and automatic recovery after failure. That allows anybody to change system schemes, timing, recovery policy, and so much more, dynamically in real-time.

The main research objectives are as follows:

- To design an adaptive orchestration model of a serverless system, configured and managed by a metadata-driven approach. That system should withstand challenges in the collection of vast amounts of heterogeneous data.
- To make a prototype of a serverless system, using cloud technologies for the whole pipeline.
- To develop a model and test the mechanism of pipeline adaptation based on feedback communication. It should automatically adjust scheduling, retries, and other specific to vendor settings.
- To test the efficiency and fault tolerance of the system using the mentioned data sources (OpenAQ, NOAA, NASA GES DISC, ESA Copernicus).
- To establish the foundation for future research in autonomous, self-regulating orchestration systems that can evolve based on previous experience and feedback-based communication, so that they can upgrade in efficiency and optimize working processes without any manual interventions.

The expected outcome of this research is that the adaptive orchestration model:

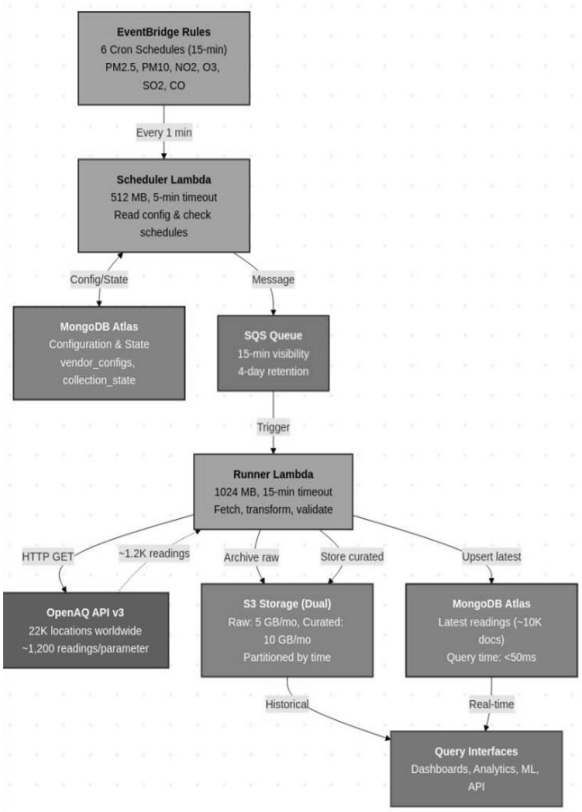
- reduces the amount of redundant calls while the unstable vendor recovers or rate limits are reached,
- improves operational and economic efficiency in comparison to static planning,
- allows real-time integration of data from new endpoints/vendors, and reconfiguration/removal of the old ones,
- makes an initial prototype that can be later evolved into an AI-assisted adaptive orchestration system

IV. ADAPTIVE ORCHESTRATION MODEL AND SYSTEM IMPLEMENTATION

A. SYSTEM OVERVIEW

So, the proposed ecosystem mainly proposes dynamic orchestration based on a metadata-driven approach. It has a lot of elements in it, but the main ones are: AWS Lambda, S3, Athena, Event Bridge, SQS, and MongoDB. That is the list to talk about a basic prototype solution. The simple version of this modular solution can be seen in the diagram (Fig 1). It demonstrates only part of the architecture, with data being collected from the OpenAQ vendor.

Each module of this orchestration process operates as an independent function, responsible for various tasks. Such as: data retrieval, transformation, storage, and so on. These processes are directed by the core controller, it is responsible not only for communication between independent modules, but also for intervals of calls, rate limits tracking, availability and latency check, and data deduplication, based on feedback communication.



Prototype of the proposed architecture

But mainly, the pipeline is separated into 3 main parts:

- Collection: they have the same abstraction, but the implementation is specific to each vendor. They receive necessary data through the REST API or file endpoints (other methods can be easily added later on). In JSON, CSV, NetCDF, and other formats.
- Transformation: Received data points are transformed into one object format with the interface of EnvironmentalReadings. It has all the required properties to bind data, and additionally has metadata, measurements, and timestamps.
- Storage and access: Collected data is stored in AWS S3, with a specific structure (vendors, days, hours, etc.) and separated by raw/curated state, based on their transformation status. This enables efficient querying with the help of Athena.

This architecture allows for collecting all the vendors: OpenAQ, NOAA, NASA GES DISC, and ESA Copernicus data separately. While allowing it to maintain data consistency, it resolves problems and transforms related to each vendor independently, but in the end, we have data available in the same format and structure, keeping operational overhead low.

B. METADATA-DRIVEN CONFIGURATION

At the core of the system, we have metadata for each vendor that we collect data from. This data can be

manually or automatically updated whenever there is a need for a change. Those changes will be applied to the system instantly, without any need for code changes or redeployment. Metadata for each vendor mainly consists of:

- Request type, pagination strategy, request endpoint.
- Rate limits, retrial policy, timeouts.
- Transformation schemes.
- Current working state (for circuit-breaker errors management).

```
{
  "_id": "68eba9982bf63c0808650530",
  "vendor": "openaq",
  "enabled": true,
  "secretName": "openaq",
  "description": "OpenAQ 15-Minute Snapshot
Collection - frequent time-series snapshots",
  "features": [{
    "id": "timeseries-pm25-15min",
    "name": "PM2.5 15-Minute Snapshots",
    "type": "DataCollection",
    "params": { "action": "collectTimeSeries",
"parameter": "pm25" },
    "schedule": {
      "id": "pm25-15min",
      "cron": "0,15,30,45 * * * *",
      "status": "Active",
      "nextRun": "2025-10-22T14:00:00Z"
    }
  }],
  "metadata": {
    "apiUrl": "https://api.openaq.org/v3",
    "endpoint": "/parameters/{id}/latest",
    "collectionStrategy":
"15_minute_snapshots",
    "storage": {
      "raw":      "s3://environmental-data-raw-
*/raw/vendor=openaq/",
      "curated":      "s3://environmental-data-
curated-*/curated/vendor=openaq/",
      "mongo":             ["collection_state",
"latest_readings", "location_metadata"]
    },
    "collectionsPerDay": 96,
    "expectedVolume": "~330,000 readings (~200
MB/day)",
    "notes": "Schedules staggered by 1 min to
avoid API rate limits"
  },
  "updatedAt": "2025-10-22T13:45:27Z"
}
```

Listing 1. Example of Vendor Configuration (OpenAQ)

When the new vendor data source is added to the Mongo database, the orchestrator automatically detects changes on its next run and starts planning tasks for EventBridge according to the configuration.

C. ADAPTIVE FEEDBACK CONTROL MECHANISM

The novelty of this approach is mostly in the orchestration model based on feedback from recent executions. The operational state of each vendor is constantly controlled by continuously gathering metrics:

Success rate (Sv) - proportion of successful requests.

Retry count (Rv) - retries per interval.

Cost factor (Cv) - estimated AWS cost for operation.

Latency (Lv) - average response delay.

A health score is computed periodically as:

$$Hv = \alpha Sv - \beta Rv - \gamma Cv - \delta Lv, \quad (1)$$

where coefficients α, β, γ, δ are weighting factors showing operational priorities (efficiency vs. cost). There is certain threshold, and when Hv falls below it, the orchestrator applies adaptive actions such as:

Increasing the waiting interval between retrials

Temporary disabling vendor collection process (in severe circumstances)

Reducing the number of concurrent collection mechanisms

Triggering notification for manual review or (later) AI analytics

This cycle of feedback communication and adjustments, when defined and optimized correctly, can allow the pipeline to become fully self-regulated, minimizing any human interactions.

D. FAULT TOLERANCE AND CIRCUIT BREAKER

The circuit breaker mechanism provides an additional layer of fault-tolerance and enables efficient handling of any errors in the orchestration mechanism. When a vendor goes into an invalid state a few times in a row (for example, because of server unavailability or rate limits, or even throttling), the system automatically stops making requests and sets some exponential back-off system, then updates metadata accordingly. The default transition of states:

Closed (Working) → Half-Open (First try after failure) → Open (Something failing, no need to retry yet) → Closed (Recovered)

This mechanism prevents the system from wasting resources, keeps the system stable in unstable situations, and reduces pressure on the external server, preventing DDOS attacks on it.

E. IMPLEMENTATION AND EXPERIMENTATION

A prototype of the system was implemented in Node.js (Typescript) and was fully deployed on AWS, using Lambda functions, Event Bridge, SQS, S3, Athena, and other services. Each vendor was provided with a specific config, including data collection rate, retries, formats, and more. For example, frequency is the following:

- OpenAQ: every 15 minutes for more frequent data points (pm25, pm-10), for others - 1 hour (air-quality metrics);
- NOAA: every hour (climate and meteorological data);

- NASA GES DISC: daily (reanalysis and aerosol data);
- ESA Copernicus: every 30 minutes (satellite imagery and environmental indicators).

Solutions comparison

Metric	Static Orchestrati-on	Adaptive Orchestrati-on	Improve-ment
Failed Invocations	127	76	40% fewer
Average Latency	1.42 sec	1.1 sec	22%
AWS Cost	1	0.83	17%
Manual Interventions	6	1	83%

In experimental test runs, system evaluation was made under conditions of simulation of errors, increased latency, and reduced quotas in order to make the test more demonstrative and show how the system behaves in large-scale and severe conditions. You can see the comparison between the proposed solution of the adaptive orchestration module and the baseline static scheduler, which is used in most legacy data collection systems.

These results confirm that adaptive orchestration significantly reduces redundant invocations, recovery times, and operational costs.

F. DISCUSSION AND SCIENTIFIC NOVELTY

The proposed solution demonstrates how to integrate feedback-based communication, adaptive orchestration, and metadata abstraction into a serverless data collection system without introducing new dependencies or complex control flows. Unlike the traditional static approach, the proposed system demonstrates characteristics of autonomous computation, with solutions for orchestrating data collection that are dynamic and occur in real time to address the system's current problems.

G. SCALABILITY AND PERFORMANCE OPTIMIZATION

Since the data size, number of active sources, and data collection frequency are increasing rapidly in the selected niche, scalability and economic effectiveness become more and more important. The proposed system uses a serverless model of elastic scaling using AWS Lambda that can increase or decrease the number of instances, memory, and time in seconds, depending on any application demand. No servers are being provisioned ahead of time. The architecture stays in waiting mode when no data collection is scheduled, hence we have no idle time and unnecessary expenses.

Each data collection task is executed independently, which ensures complete code isolation and fault tolerance, preventing a cascade of overwhelming the server and experiencing a server shutdown. AWS EventBridge is used to manage thunderstorms of simultaneous invocations from different vendors. Internal benchmark showed that the system is capable of gathering more than 25000 data points from all around the world (OpenAQ API) with an average latency of less than 2 seconds per request.

In terms of actual productivity monitoring and checking how the system behaves under different circumstances, AWS CloudWatch is being used. It provides such information in real time, while MongoDB just saves operational metadata, such as circuits for circuit breaker states, timestamps of vendor runs, API rate-limits, saving state, and so on. Those mechanisms provide us high level of traceability and recovery in case of failure. Also, with manual planned review (or AI review later on), we can identify system weak points, such as idle functions, too much memory allocated, duplicated data collected, and so on. This way, we can improve the system incrementally. The more it works, the better it gets.

As for future improvements, the system may be integrated with AWS Step Functions and SageMaker Pipelines to support more advanced working processes. Those being the following: anomaly detection, timing predictions, or removing redundant resources. Those actions will maintain scalability while expanding analytical capabilities.

V. CONCLUSION

This study introduced and validated an adaptive orchestration model for efficient, serverless collection of heterogeneous environmental data. By combining metadata-driven configuration, continuous feedback control, and a fault-tolerant circuit breaker mechanism, the proposed system dynamically adjusts its execution strategies in response to external instabilities. Experimental evaluation confirmed notable improvements in efficiency, fault tolerance, and cost optimization compared with static orchestration methods. The integration of feedback-based metrics and runtime metadata management enables the system to self-regulate and sustain stability in highly variable operational conditions.

The scientific novelty of this work lies in merging adaptive orchestration principles with event-driven serverless architectures, providing a foundation for future self-learning orchestration systems capable of autonomous evolution. Prospective research will focus on incorporating reinforcement learning and predictive analytics for further optimization of orchestration behavior and decision-making in large-scale CPS and IoT environments.

VI. CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

VII. DECLARATION ON GENERATIVE AI

During the preparation of this work, the author(s) used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using this

tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1]. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ... & Patterson, D. A. (2019). Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*. https://doi.org/10.48550/arXiv.1902.03383

[2]. Sirajuddin, M. (2024). Advances in serverless computing: a paradigm shift in cloud application development. *International journal of computer engineering & technology. 15*. 1440-1449. DOI: 10.5281/zenodo.14506248.

[3]. Malyuga, K., Perl, O., Slapoguzov, A., Perl, I. (2020). Fault Tolerant Central Saga Orchestrator in RESTful Architecture. *Proceedings of the XXth Conference of Open Innovations Association FRUCT.26*. 278-283. DOI: 10.23919/FRUCT48808.2020.9087389.

[4]. Werner, S., & Tai, S. (2024). A reference architecture for serverless big data processing. *Future Generation Computer Systems, 155*, 179-192. DOI: https://doi.org/10.1016/j.future.2024.01.029

[5]. Mathew, A., Andrikopoulos, V., Blaauw, F. J., & Karastoyanova, D. (2024). Pattern-based serverless data processing pipelines for Function-as-a-Service orchestration systems. *Future Generation Computer Systems, 154*, 87-100. DOI: https://doi.org/10.1016/j.future.2023.12.026

[6]. Li, Y., Lin, Y., Wang, Y., Ye, K., & Xu, C. (2022). Serverless computing: state-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing, 16*(2), 1522-1539. DOI: 10.1109/TSC.2022.3166553

[7]. Richardson, C. (2018). *Microservices patterns: with examples in Java*. Simon and Schuster, New York. 520 p.

[8]. Sharma, S. K. (2025). Serverless Architectures for Scalable and Cost-Efficient Information Systems in SMEs. *International Journal of Performability Engineering, 21*(8). 438. DOI: 10.23940/ijpe.25.08.p4.438449

[9]. Madhusudanan, J., Geetha, S., Prasanna Venkatesan, V., Vignesh, U., & Iyappan, P. (2018). Hybrid Aspect of Context- Aware Middleware for Pervasive Smart Environment: A Review. *Mobile Information Systems, 2018*(1), 6546501. DOI: https://doi.org/10.1155/2018/6546501

[10]. Bhardwaj, E., Gujral, H., Wu, S., Zogheib, C., Maharaj, T., & Becker, C. (2024, June). Machine learning data practices through a data curation lens: An evaluation framework. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (pp. 1055-1067). DOI: https://doi.org/10.1145/3630106.3658955

[11]. Alatawi, M. N. (2025). Optimizing Multitenancy: Adaptive Resource Allocation in Serverless Cloud Environments Using Reinforcement Learning. *Electronics, 14*(15), 3004. DOI: https://doi.org/10.3390/electronics14153004

[12]. Bordin, M. V., Griebler, D., Mencagli, G., Geyer, C. F., & Fernandes, L. G. L. (2020). Dspbench: A suite of benchmark applications for distributed data stream processing systems. *IEEE Access, 8*, 222900-222917. DOI: 10.1109/ACCESS.2020.3043948

Oleksandr Demidov received a M.S. degree at the Department of Specialized Computer System, at Lviv Polytechnic university. From 2019 till now he has been a Software Engineer at software developing company for web interfaces and serverless backend solutions. Currently, he is a PhD degree student of Computer Engineering at Lviv Polytechnic National University. His research interests include cloud-based architecture, serverless solutions.

Oksana Honsor PhD, Assoc. Professor at the Department of Specialized Computer System, Institute of Computer Technology, Automation and Metrology of Lviv Polytechnic National University. Scientific interests include metrological support in IoT systems and sensor networks, ensuring traceability, processing of large data sets for reproduction and comparison of the results of measurements of physical quantities in remote mode.