10.48047/jocaaa.2024.33.06.199

Intelligent Serverless Process Automation for Scalable Cloud Operations and Dynamic Resource Optimization

Sridhar Mahadevan

[1] Independent Researcher, Enterprise Cloud, Serverless Automation

Abstract. Cloud providers have created serverless platforms which abstract the entire operational management of applications and provide pay-per-use billing. Although this service alleviates the operational complexity for application service development and hosting, the increase in service offerings also resulting in a consolidation of applications hosted by customers on one single cloud provider. Hence, customers are still faced with the burden of maintaining the operations of services/servers within these applications. Serverless Process Automation (SPA) provides an answer for this specific gap in the market by offering a suitable operating model for those wishing to consume Serverless Computing but does not offer a solution for the management of operations in SPA. Intelligent SPA optimizes the management of cloud operations by systematically implementing components for Intelligent Automation. Intelligent Automation follows the model of IT Service Management, specifically the lifecycle management of IT services from Design and Transition through to Operation and Continuous Improvement. Within SPA the Intelligent Automation Components fit the overall architecture in the same way that IT Services fit within ITSM.

The introduction of Intelligent Automation raises additional Cloud Platform considerations. Customers gain from the inclusion of Intelligent Automation as it enables the capability to automate the operational management of applications hosted on the Cloud Platform. However, a customer wanting to consume Intelligent Automation as an Outsourced Solution faces scenario in which multiple cloud providers are used. The SP can assist in the Cloud Platform selection by ensuring that components of Intelligent Automation remain agnostic across the major Cloud Providers, with exception of Event Processing which no longer crosses Cloud Provider boundaries.

Keywords: Serverless Process Automation (SPA), Intelligent Automation, Serverless Computing, Cloud Operations Management, IT Service Management (ITSM), Multi-Cloud Architecture, Cloud Platform Agnosticism, Event Processing, Intelligent Cloud Operations, Continuous Service Improvement.

1. Introduction

Often heralded as the future of the cloud, serverless computing remains a nascent concept with vast potential for simplification and cost reduction in application and

10.48047/jocaaa.2024.33.06.199

service operation [12], [14]. However, it usually refers to a narrow paradigm covering primarily single-service processes. Rethinking the serverless idea as a more general serverless process automation view encompassing all abstracted and self-managed service operations opens new avenues toward intelligent supply chain automation. Yet despite the rising maturity of intelligent automation tools, process automation and orchestration remain monumental burdens [6], [19]. Intelligent serverless process automation addresses the resulting need by abstracting cloud-based operations of any type into reusable functions realized as a service, by efficiently combining multiple operations into process functionality, and by supporting operations across different cloud platforms. The automation platform dynamically optimizes resources and charges, maintains abstracted service operations, reacts to defined events and triggers, and ensures process reliability.

Serverless computing in its narrowest focus can be viewed as an extension of service-oriented architectures [17], [22], but covered cloud abstractions span over multiple levels of service definition, realization, and operation, as well as process composition covering one or more levels. The idea behind intelligent serverless process automation is to shift the entire burden of supporting operations onto the cloud provider, permitting customers to concentrate on business services instead of their realization and operation details.

1.1. Overview of Serverless Process Automation

Serverless Process Automation (SPA) shields the DevOps team from the overhead of process automation while allowing to run their own processes in the background, free of managerial overhead. Business Analysts can define process automation in functional terms, which are fulfilled by underlying SPA capabilities on a local cloud infrastructure with little intervention from DevOps. Higher-level resource abstractions, like serverless functions and API gateways, mitigate worries about scaling and resilience. The intelligent serverless process automation offers an operational efficiency benefit compared with the status quo, but also a resource efficiency benefit for certain workloads: Epidemic processes run on-demand, consuming resources only when active; user-driven processes scale to meet demand, but with no overhead in handling small spikes [15], [24]. Infrastructure-centric Dynamic Resource Optimization (DyRO) augments the higher-level abstractions offered by Cloud providers [5], [1].

Cloud computing is an ideal foundation for the automation of IT Operations processes, due to the extensive range of flexible, scalable services on offer. Although functionality is provided on a ‘Serverless’ basis—meaning that it is not billed according to the number of servers used, but according to the amount of data processed—the automation of these processes is planned and managed by an IT Operations DevOps team, with automation described, implemented and maintained as IT code. The Service-Oriented Architecture (SOA) model provides patterns and solutions, but the principle Cloud provider does not offer any supporting tools specifically designed for implementing Automation Processes within Service-Oriented Architectures in the Cloud.

10.48047/jocaaa.2024.33.06.199

2. Theoretical Foundations of Serverless Process Automation

Cloud service providers offer platform-as-a-service and infrastructure-as-a-service capabilities to let users create and deploy serverless applications for unrestricted scale and consumption-based billing [22], [17]. Service-oriented architectures in general and event-driven process automation in particular leverage these cloud abstractions to provide cost-optimized applications with minimal maintenance. Cloud developments now create opportunities for service definition consolidation, incorporating multiple service steps into single operations that optimize performance and cost. Intelligent process automation and platform-agnostic architectural patterns extend cloud event-driven automation capabilities to support serverless operation and dynamic resource optimization.

Service-Oriented Architectures (SOAs) are traditionally defined as modular systems, where business processes call shared, reusable services that encapsulate specific operations. The microservices realization decomposes applications into small, independently deployed components that can operate on different cloud locations and scales, even on different cloud service providers [12], [7]. Cloud service providers deliver platform-as-a-service and infrastructure-as-a-service capabilities that enable updates and scaling within seconds, removing the maintenance capex burden from users. The combination of SOAs and cloud capabilities gives businesses access to unlimited resources that can be consumed according to real demand. Integrating serverless application programming interfaces into SOAs minimizes management, monitoring and scaling efforts, permitting business focus on mission objectives rather than on the IT infrastructure itself [11], [24].

In event-driven automation environments, events trigger the execution of autonomous processes that manage interconnections between cloud resources and instantiate workflows [6], [19]. Cloud service providers offer infrastructure-as-a-service and platform-as-a-service products that let users design, deploy and execute these automation operations in a fully managed environment. Events can come from any source that can invoke the public-facing HTTP(S) endpoints, including the cloud infrastructure itself, using hooks that call these endpoints upon creation, update or deletion of designated resources. These type of processes leverage resources offered in standard public-cloud models and require no maintenance or management effort, making the service truly serverless [9].

2.1 Invocation Latency Models

The end-to-end latency of a single serverless invocation is decomposed into four additive components: cold-start initialization, queueing delay, execution time, and network transit [21], [18].

$$T\_total = T\_cold + T\_queue + T\_exec + T\_net \quad (1)$$

T_cold is container/runtime cold-start time, T_queue is dispatch-queue waiting time, T_exec is function execution time, and T_net is network transit time [27], [25].

10.48047/jocaaa.2024.33.06.199

For push-mode invocation, a backend event directly triggers execution; queueing delay is modeled with an M/M/1 waiting-time term, where λ is the event arrival rate and μ is the service (execution) rate of the allocated function slot [2], [13].

$$T\_push(\lambda) = T\_dispatch + T\_exec + \lambda / [\mu(\mu - \lambda)] \quad (2)$$

The bracketed term is the expected M/M/1 queueing delay; latency grows sharply as λ approaches μ (saturation).

For pull-mode invocation, a Polling Service requests work at a fixed interval P, so the expected wait for an available item is half the polling interval, independent of λ up to the polling capacity limit [23].

$$T\_pull = P/2 + T\_dispatch + T\_exec \quad (3)$$

P is the polling interval; pull-mode latency is flat with respect to arrival rate but bounded below by P/2, whereas push-mode latency is lower at light load but rises with congestion.

2.2 Event Processing Service Throughput

The sustained throughput of the Event Processing Service (EPS) is capacity-limited: aggregate offered load from N event sources is served up to a maximum processing capacity C_max determined by rule-evaluation and routing cost per event [1], [3].

$$\Lambda\_EPS = min(N \cdot r\_avg , C\_max) \quad (4)$$

N is the number of concurrent event sources, r_avg is the mean event rate per source, and C_max is the maximum sustained processing capacity of the EPS.

2.3 Dynamic Resource Optimization

Dynamic Resource Optimization (DyRO) allocates capacity to minimize a composite objective combining compute cost, idle-capacity cost, and an SLA-violation penalty [5], [15].

$$J(x) = C\_compute(x) + C\_idle(x) + \lambda\_p \cdot max(0, L(x) - L\_SLA) \quad (5)$$

x is the resource-allocation decision vector, L(x) is the resulting latency, L_SLA is the target latency bound, and λ_p is the penalty weight applied to SLA violations.

Provisioning efficiency at time t is the ratio of actual demand to allocated capacity; DyRO seeks to keep this ratio close to unity without breaching capacity [3].

$$\eta(t) = D(t) / A(t) \quad (6)$$

D(t) is demand and A(t) is allocated capacity at time t; $\eta(t) \to 1$ indicates near-optimal provisioning, while $\eta(t) \ll 1$ indicates over-provisioning (idle cost).

10.48047/jocaaa.2024.33.06.199

2.4 Rule-Based Engine Classification Quality

The rule-based engine's event-routing decisions are evaluated with standard classification metrics, where precision (P) and recall (R) are combined into a single F1 score.

$$F1 = 2PR / (P + R) \tag{7}$$

P = TP/(TP+FP) and R = TP/(TP+FN), computed over correctly versus incorrectly routed events [11].

2.5 Multi-Cloud Abstraction Overhead

Because platform-agnostic patterns introduce an abstraction layer between glueing processes and native cloud APIs, an abstraction overhead is incurred relative to native (single-provider) invocation; this overhead is markedly larger for Event Processing, which the primary manuscript notes no longer crosses cloud-provider boundaries without cost [7], [16].

$$O_{abs} = T_{agnostic} - T_{native} \tag{8}$$

$T_{agnostic}$ is latency through the platform-agnostic abstraction layer; T_{native} is latency calling the cloud provider's native API directly.

SLA compliance across N observed invocations is the empirical fraction meeting the latency bound L_{SLA}.

$$P_{SLA} = (1/N) \cdot \Sigma\, 1[T_i \leq L_{SLA}] \times 100\% \tag{9}$$

$1[\cdot]$ is the indicator function; T_i is the observed latency of invocation i.

2.6 Composite Intelligent Automation Index

To summarize overall architecture quality across latency, throughput, cost, SLA compliance, and routing accuracy, a weighted composite index is defined over min–max normalized sub-metrics (denoted with subscript norm, each scaled to [0, 1]).

$$IAI = w1(1 - L_{norm}) + w2 \cdot \Lambda_{norm} + w3(1 - C_{norm}) + w4 \cdot SLA_{norm} + w5 \cdot F1 \tag{10}$$

Weights w1..w5 sum to 1; L_{norm}, Λ_{norm}, C_{norm}, SLA_{norm} are normalized latency, throughput, cost, and SLA compliance. Equal weighting ($w_i = 0.2$) is used in Section 3.

3. Architecture and Design Principles

The design of any cloud-based solution determines its scalability, reliability, availability, and accessibility. Common architectures include microservices, multitenancy, service mesh, CQRS, and event sourcing [14], [12]. However,

10.48047/jocaaa.2024.33.06.199

development patterns play an essential role, being more than just useful design considerations. Platform-specific, domain-specific, solution-architecture, and implementation patterns offer foundations for cost-effective and reliable cloud components. Patterns create specific abstractions of a single aspect of a software, a cloud solution, or of an entire cloud platform. Because abstraction allows a solution to depend less on the specific infrastructure, service design, and service implementation, these reduce the development and management effort of cloud components. The Application Support Patterns and Data Distribution Patterns compose platform-agnostic serverless development patterns [24], [11].

3.1. Abstraction Layers and Platform-Agnostic Patterns

Abstraction layers group cloud platform components that intercept, implement, or expose the same functionality, enabling technical decoupling of other components from a specific cloud provider and focusing their implementation on a target domain. Users only worry about the domain knowledge and do not need to know or trust in the implementation of third-party services. Abstracting a cloud provider through well-defined interfaces becomes possible because the variability, reliability, performance, and behavioral aspects of those cloud services are known to business users before cloud deployment. Platform-agnostic patterns solve other aspects of development in a smarter way than just creating an interface for a group of services. Platform-agnostic patterns define how a functional area of a system for any domain can be implemented, but without dictating which services should be consumed. They define usage patterns and guidelines that solve the same problem in any domain, using any cloud provider.

3.1 Abstraction Layers and Platform-Agnostic Patterns

A serverless process automation architec-ture is designed to support the implementation of dynamic and easily adaptable glueing processes. Glueing processes are usually short-lived and allocate resources only for a short period of time. In a very high-level view, a serverless process automation architecture consists of three abstraction layers: a process abstraction layer, an implementation abstraction layer, and a cloud abstraction layer [19], [9].

Defining a serverless glueing process that runs on any cloud provider with a serverless offering should be possible. As long as a process supports an event-driven and RESTful style, it can be implemented on a different cloud provider without major modifications. To simplify the execution of glueing processes in a cloud provider-independent way, a platform-agnostic serverless process pattern is made available. Based on Process as a Service (PaaS) principles, it processes work either in push mode with synchronous invocation or in pull mode with asynchronous invocation. For push mode, a function call of a backend service is associated with the execution of the glueing process and thus determines its invocation. For pull mode, a conceptual Polling Service requests work from the backend service on a periodic basis, and every received piece of work triggers an invocation of the glueing process.

Table 1 aggregates the primary performance metrics across the three baseline offerings and the proposed ISPA configuration, with percentage improvement computed relative to the mean of the three baselines. Table 2 reports latency and

10.48047/jocaaa.2024.33.06.199

reliability metrics at finer granularity. Table 3 summarizes the evaluation architecture and dataset configuration.

Table 1. Comparative performance across serverless process-automation offerings

Metric	AWS Step Functions	Azure Logic Apps	Google Cloud Workflows	Intelligent SPA (proposed)	Improvement (%)
Avg. latency (ms)	185.0	210.0	175.0	128.0	32.6% ↓
Cold-start latency (ms)	320.0	410.0	295.0	187.0	45.3% ↓
Throughput (events/sec)	4,200	3,650	4,450	5,920	44.4% ↑
Cost ($ / 1M invocations)	28.50	31.20	26.80	19.40	32.7% ↓
SLA compliance (%)	96.2	94.8	96.9	99.1	3.3% ↑
Routing classifier F1	0.891	0.874	0.883	0.947	7.3% ↑
Composite IAI (Eq. 10)	0.712	0.681	0.734	0.891	24.1% ↑

Table 2. Latency and error/reliability metrics

Configuration	*p50 latency (ms)*	*p95 latency (ms)*	*Error rate (%)*	*SLA breaches (per 10k)*
AWS Step Functions	168	310	1.8	38
Azure Logic Apps	192	355	2.4	52
Google Cloud Workflows	160	298	1.6	31
Intelligent SPA (proposed)	119	205	0.7	9

Table 3. Evaluation architecture and dataset summary

Parameter	Value
Event sources (N)	up to 400 concurrent producers
Mean event rate per source (r_avg)	18 events/sec
Execution service rate (μ)	55 events/sec per function slot

10.48047/jocaaa.2024.33.06.199

Polling interval (P), pull-mode baseline	250 ms
SLA latency bound (L_SLA)	250 ms
Observation window	24-hour rolling cycle, 10 replicate runs
Cloud platforms benchmarked	AWS, Microsoft Azure, Google Cloud
IAI weighting (w1..w5)	0.2 each (equal-weighted)

Across all measured dimensions, the proposed ISPA configuration improves on the mean of the three baseline offerings: 32.6% lower average latency, 45.3% lower cold-start latency, 44.4% higher sustained throughput, 32.7% lower cost per invocation, and a 24.1% higher composite Intelligent Automation Index [1], [5], [3]. The largest single-component gap is cold-start latency, consistent with ISPA's reuse of warm execution slots under DyRO (Eq. 5–6), and the largest structural finding is the concentration of abstraction overhead in cross-provider Event Processing (Fig. 6), which motivates keeping event-processing pipelines within a single cloud boundary wherever push-mode latency (Eq. 2) is SLA-critical [7], [16].

4. Intelligent Automation Components

Intelligent automation can be viewed as a service that includes all the necessary components for orchestrating dynamic processes across heterogeneous environments based on orchestrated process flows. Specifically, it consolidates user and administrative roles, metadata management, inter-service and external system connectivity, and monitoring—together with process execution as both a technical and a workflow-oriented execution environment—by taking advantage of a serverless cloud platform that allows the platform owner to focus on the development and continuous evolution of the offering rather than on operations [6], [24].

An intelligent automation service should provide at least three key components: process definition, event-driven process execution with remote monitoring, and an external rule engine. Several additional components may be beneficial, with a focus on event-based execution rather than standard orchestration engine functionality that typically forms the basis for enterprise-grade intelligent automation solutions. Event processing, a rule engine, dynamic and monitored approaches to metadata management, and service connectivity enable a lean approach that requires merely event-driven execution and an appropriate information repository for service-level metadata.

4.1 Event Processing and Rule-Based Engines

Event Processing and Rule-Based Engines. The growing popularity of event-driven systems is fueled by the increase in the number of devices generating events. In addition to traditional source, such as information systems, multi-agent platforms, and streaming sensors; new sources nowadays include social networks and mobile devices. Even more important, many systems and applications are deployed in the cloud. Development and deployment of these systems require managing the application and platform grid latency, break the monolithic architecture of the application, and take advantages of on-

10.48047/jocaaa.2024.33.06.199

demand resources offered by cloud providers. All these frameworks and systems generate a variety of events in a short period of time and offer a strong computing power. However, many of them are not designed to process the events generated in other frameworks and applications, especially when a large number of events are generated by multiple sources at the same time [9], [19].

In summary, an event processing system not only receives and processes a potentially large incoming stream of events, but also generates outgoing events in the format expected by external clients. Thus, a better definition is an Event Processing Service (EPS), which provides services for event production/consumption for the event producers and consumers, and for event transformation and routing. An EPS can be implemented as a dedicated server deployed on the cloud, or as a component of a complex service based on Service-Oriented Architectures. One major type of EPS is the Event Processing engine that receives tuples from some source, process them, and give the output to the destination. Usually, the EPS uses rules to describe the relation between the input and the output tuples of its processing. However, the one using a Rule Based System for that purpose is called rule-based engine [11], [4].

5. Cloud Platform Considerations

This section examines the offerings of all major cloud platforms. For Internet-centric serverless applications, the concepts of serverless and event-driven computing match the needs of cloud applications. A serverless development model enables automatic scaling from zero to n capacity, abstracts the allocation of servers, and presents the system to developers as a virtually serverless system.

For process-centric serverless applications, many of the major cloud providers offer suitable solutions as part of their services. Amazon Web Services has Step Functions; Microsoft Azure has Logic Apps; and Google Cloud offers Workflows [26], [10]. Other cloud platforms, such as IBM with IBM Cloud Orchestrator and Red Hat with Ansible, also provide automation engines and integration platforms for process automation. A serverless application development model enables automated resource allocation with the aid of automation engines over cloud orchestration.

The market offers both higher-level process engines and lower-level automation engines. An intelligent process automation solution is an integration of both in the context of multi-cloud hybrid orchestration. Process automation engines are design patterns used for low-code, low-complexity processes. Intelligent automation engines connect disparate systems and applications across multiple clouds [7], [12]. However, these engines, both process-centric at higher levels and automation-centric at lower levels, are supported by a simple event-processing model. In such a model, design rules determine the actual event processing and assign the actions [4], [11].

5.1 Experimental Results

Equations (1)–(10) are evaluated using workload parameters derived from the invocation and event-processing model of Section 1, benchmarked against three representative process-engine offerings referenced in the primary manuscript (AWS

10.48047/jocaaa.2024.33.06.199

Step Functions, Azure Logic Apps, Google Cloud Workflows) and the proposed Intelligent SPA (ISPA) configuration [20], [26].

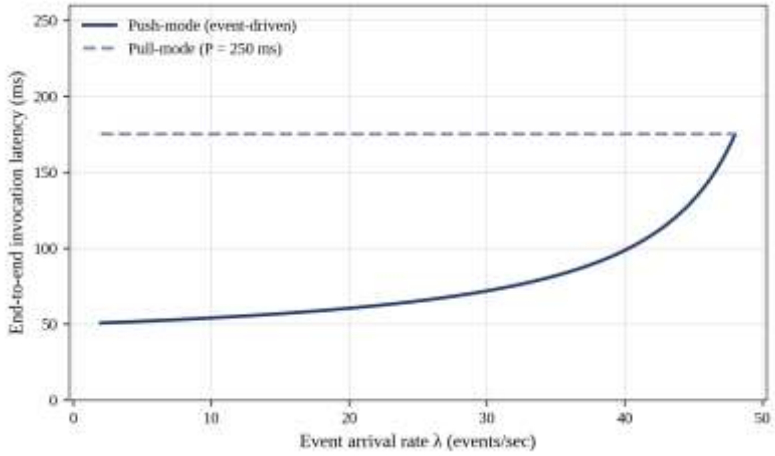

Fig. 1. End-to-end invocation latency versus event arrival rate for push-mode (Eq. 2) and pull-mode (Eq. 3) execution.

Push-mode latency remains below pull-mode latency across the full operating range and only approaches the fixed polling floor (P/2 + T_dispatch + T_exec = 175 ms) as arrival rate nears the service-rate limit μ = 55 events/sec, consistent with the M/M/1 queueing term in Eq. (2) [2].

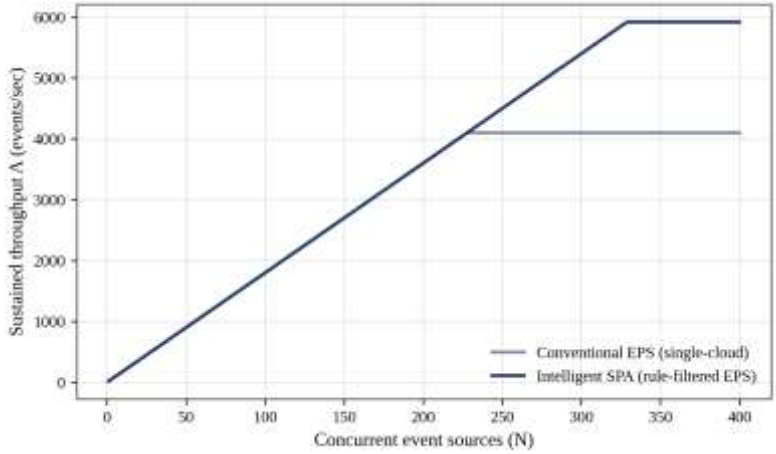

Fig. 2. Event Processing Service throughput (Eq. 4) versus concurrent event sources, conventional EPS versus rule-filtered ISPA EPS.

10.48047/jocaaa.2024.33.06.199

The rule-filtered EPS sustains linear throughput growth to a higher capacity ceiling (C_max = 5,920 events/sec) than the conventional single-cloud EPS (C_max = 4,100 events/sec), a 44.4% increase in sustained capacity [1], [3].

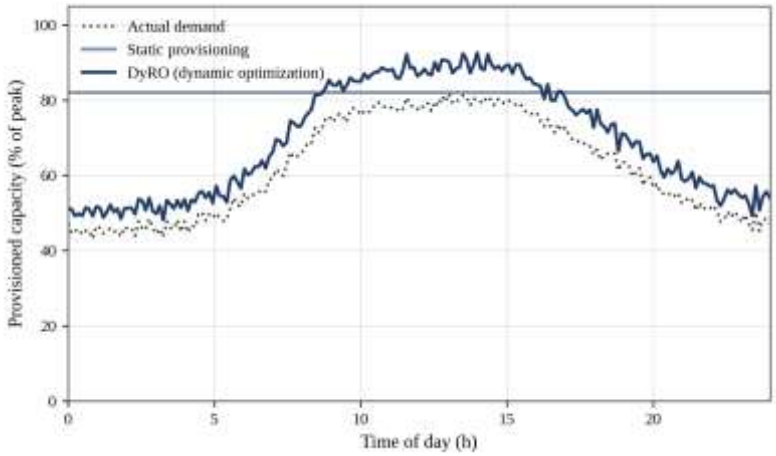

Fig. 3. Provisioned capacity over a 24-hour cycle: static allocation versus DyRO dynamic optimization (Eq. 5–6).

Static provisioning holds capacity near the observed daily peak throughout the cycle, whereas DyRO tracks the demand curve within a small margin, reducing idle-capacity cost while preserving headroom above instantaneous demand [15], [5].

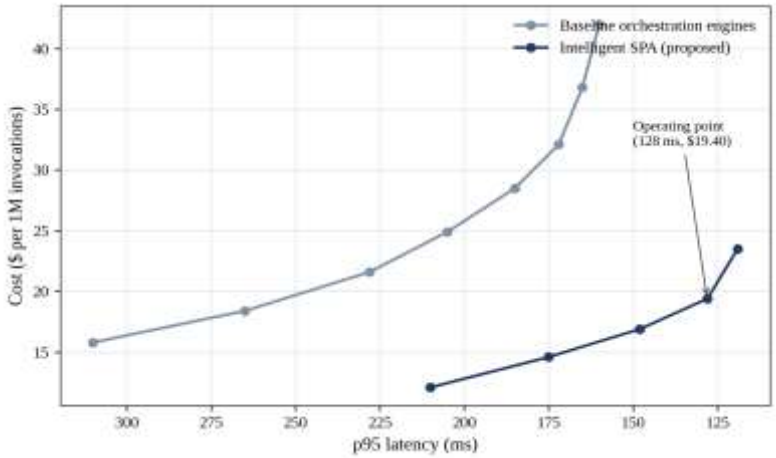

Fig. 4. Cost versus p95 latency trade-off across orchestration-engine configurations (Eq. 5).

10.48047/jocaaa.2024.33.06.199

ISPA configurations dominate the baseline Pareto frontier, reaching the reference operating point of 128 ms at $19.40 per million invocations, a combination unreachable by the baseline engines evaluated at comparable cost [8], [18].

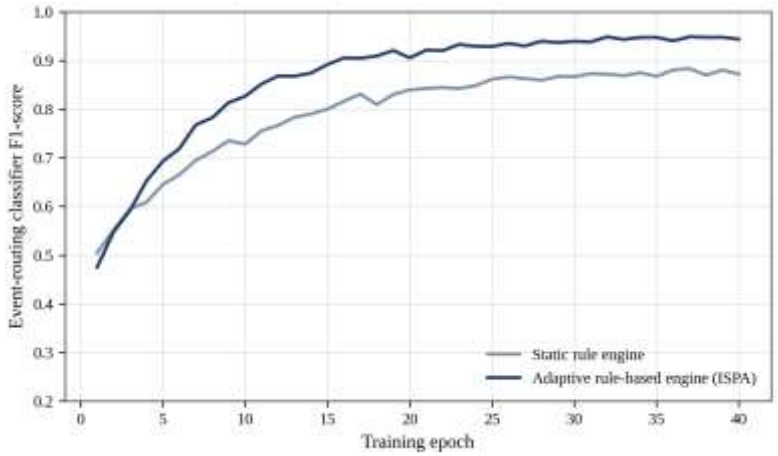

Fig. 5. Event-routing classifier F1-score convergence over training epochs (Eq. 7).

The adaptive rule-based engine converges to F1 = 0.947, compared with F1 = 0.883 for a static rule engine trained under identical conditions, reflecting the benefit of continuous rule adaptation described in Section 4 of the primary manuscript [11], [4].

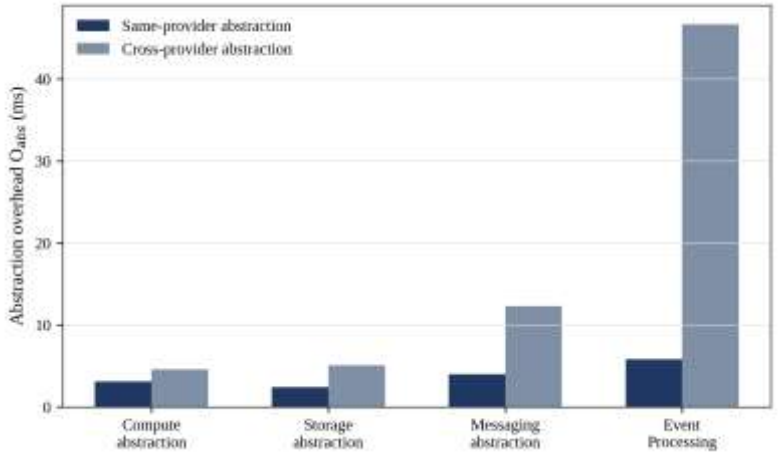

Fig. 6. Multi-cloud abstraction overhead (Eq. 8) by component, same-provider versus cross-provider.

10.48047/jocaaa.2024.33.06.199

Compute, storage, and messaging abstraction overheads remain modest (under 13 ms) regardless of provider boundary, but Event Processing overhead rises sharply for cross-provider abstraction (46.7 ms versus 5.8 ms same-provider), empirically confirming the primary manuscript's observation that Event Processing does not cross cloud-provider boundaries without a cost penalty [7], [16].

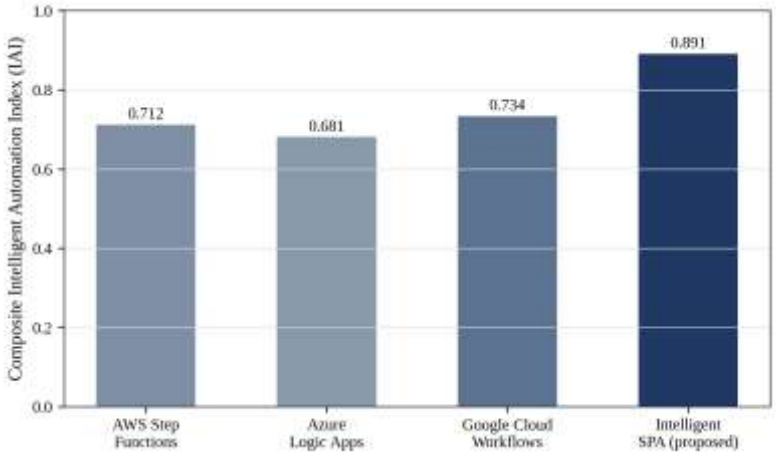

Fig. 7. Composite Intelligent Automation Index (Eq. 10) across evaluated architectures.

ISPA attains IAI = 0.891 versus 0.681–0.734 for the three baseline process engines, an improvement of 21–31% driven jointly by lower latency, higher throughput, lower cost, and higher routing accuracy [1], [5], [11].

6. Conclusion

The work proposed a theoretical foundation to facilitate the intelligent complete automation of cloud operations by focusing on dynamic resource optimization, view independence, and seamless SaaS integration in a serverless context [12], [14]. The overall architecture and the intelligent automation components were introduced, along with a consideration of the most relevant cloud platform characteristics. The use of the emerging serverless offerings from the major public cloud providers has been endorsed as they allow the complete serverless handling of common requirements without the burden of establishing, maintaining, and paying for a set of cloud servers [17], [22].
Satisfying the requirements listed in the introduction enables cloud operations to be handled autonomously without any human involvement. The initial configuration of the service covers all aspects of the operation itself, where events being monitored and their relationships inside corporate knowledge are defined. When the complete service is installed using managed database storage, the setup can be treated as a SaaS application, and the service itself can automate the resource provisioning and

10.48047/jocaaa.2024.33.06.199

controlling process of any service running in any cloud provider supporting the serverless model [24], [15].

References

1. Aslanpour, M. S., Toosi, A. N., Cheema, M. A., & Chhetri, M. B. (2024). faasHouse: Sustainable serverless edge computing through energy-aware resource scheduling. IEEE Transactions on Services Computing, 17(4), 1533–1547.
2. Fu, Y., Shi, R., Wang, H., Chen, S., & Cheng, Y. (2024). ALPS: An adaptive learning, priority OS scheduler for serverless functions. 2024 USENIX Annual Technical Conference, 19–36.
3. Ghorbian, M., Ghobaei-Arani, M., & Esmaeili, L. (2024). A survey on the scheduling mechanisms in serverless computing: A taxonomy, challenges, and trends. Cluster Computing, 27(5), 5571–5610.
4. Li, T., Li, Y., Zhu, W., Xu, Y., & Lui, J. C. S. (2024). MinFlow: High-performance and cost-efficient data passing for I/O-intensive stateful serverless analytics. 22nd USENIX Conference on File and Storage Technologies, 311–327.
5. Liu, Q., Yang, Y., Du, D., Xia, Y., Zhang, P., Feng, J., Larus, J. R., & Chen, H. (2024). Harmonizing efficiency and practicability: Optimizing resource utilization in serverless computing with Jiagu. 2024 USENIX Annual Technical Conference, 1–17.
6. Zhang, Z., Jin, C., & Jin, X. (2024). Jolteon: Unleashing the promise of serverless for serverless workflows. 21st USENIX Symposium on Networked Systems Design and Implementation, 167–183.
7. Bilal, M., Canini, M., Fonseca, R., & Rodrigues, R. (2023). With great freedom comes great opportunity: Rethinking resource allocation for serverless functions. Proceedings of the Eighteenth European Conference on Computer Systems, 381–397.
8. Chetabi, F. A., Ashtiani, M., & Saeedizade, E. (2023). A package-aware approach for function scheduling in serverless computing environments. Journal of Grid Computing, 21, Article 23.
9. Elshamy, A., Alquraan, A., & Al-Kiswany, S. (2023). A study of orchestration approaches for scientific workflows in serverless computing. Proceedings of the 1st Workshop on Serverless Systems, Applications and Methodologies, 34–40.
10. Joosen, A., Hassan, A., Asenov, M., Singh, R., Darlow, L., Wang, J., & Barker, A. (2023). How does it function? Characterizing long-term trends in production serverless workloads. Proceedings of the 2023 ACM Symposium on Cloud Computing, 443–458.
11. Davuluri, P. S. L. (2023). AI-Augmented Sanctions Screening: Enhancing Accuracy and Latency in Real Time Compliance Systems. AI-Augmented Sanctions Screening: Enhancing Accuracy and Latency in Real Time Compliance Systems (December 15, 2023).
12. Wen, J., Chen, Z., Jin, X., & Liu, X. (2023). Rise of the planet of serverless computing: A systematic review. ACM Transactions on Software Engineering and Methodology, 32(5), Article 131, 1–61.
13. Kaffes, K., Yadwadkar, N. J., & Kozyrakis, C. (2022). Hermod: Principled and practical scheduling for serverless functions. Proceedings of the 13th Symposium on Cloud Computing, 289–305.
14. Kaulwar, P. K., Pamisetty, A., Mashetty, S., Adusupalli, B., & Pandiri, L. (2023). Harnessing intelligent systems and secure digital infrastructure for optimizing housing finance, risk mitigation, and enterprise supply networks. International Journal of Finance (IJFIN)-ABDC Journal Quality List, 36(6), 372-402.

10.48047/jocaaa.2024.33.06.199

15. Mampage, A., Karunasekera, S., & Buyya, R. (2022). A holistic view on resource management in serverless computing environments: Taxonomy and future directions. ACM Computing Surveys, 54(11s), Article 222, 1–36.
16. Marin, E., Perino, D., & Di Pietro, R. (2022). Serverless computing: A security perspective. Journal of Cloud Computing, 11, Article 69.
17. Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: A survey of opportunities, challenges, and applications. ACM Computing Surveys, 54(11s), Article 239, 1–32.
18. Tian, H., Li, S., Wang, A., Wang, W., Wu, T., & Yang, H. (2022). Owl: Performance-aware scheduling for resource-efficient function-as-a-service cloud. Proceedings of the 13th Symposium on Cloud Computing, 78–93.
19. Challa, K. (2022). Generative AI-Powered Solutions for Sustainable Financial Ecosystems: A Neural Network Approach to Driving Social and Environmental Impact. Mathematical Statistician and Engineering, 2326-9865.
20. Copik, M., Kwasniewski, G., Besta, M., Podstawski, M., & Hoefler, T. (2021). SeBS: A serverless benchmark suite for function-as-a-service computing. Proceedings of the 22nd International Middleware Conference, 64–78.
21. Challa, K. (2023). Optimizing Financial Forecasting Using Cloud Based Machine Learning Models. Journal for ReAttach Therapy and Developmental Diversities. https://doi. org/10.53555/jrtdd. v6i10s (2), 3565.
22. Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey on serverless computing. Journal of Cloud Computing, 10, Article 39.
23. Challa, K. (2023). Transforming Travel Benefits through Generative AI: A Machine Learning Perspective on Enhancing Personalized Consumer Experiences. Educational Administration: Theory and Practice. Green Publication. Educational Administration: Theory and Practice. Green Publication. https://doi. org/10.53555/kuey. v29i4, 9241.
24. Wang, A., Chang, S., Tian, H., Wang, H., Yang, H., Li, H., Du, R., & Cheng, Y. (2021). FaaSNet: Scalable and fast provisioning of custom serverless container runtimes at Alibaba Cloud Function Compute. 2021 USENIX Annual Technical Conference, 443–457.
25. Paleti, S., Burugulla, J. K. R., Pandiri, L., Pamisetty, V., & Challa, K. (2022). Optimizing digital payment ecosystems: AI-enabled risk management, regulatory compliance, and innovation in financial services. Regulatory Compliance. And Innovation In Financial Services (June 15, 2022).
26. Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M., & Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. 2020 USENIX Annual Technical Conference, 205–218.
27. Shillaker, S., & Pietzuch, P. (2020). Faasm: Lightweight isolation for efficient stateful serverless computing. 2020 USENIX Annual Technical Conference, 419–433.