International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

Adaptive Resource Scheduling in Serverless Architectures for Low-Latency Microservices

Rishabh Agarwal[1]
[1]*Harrisburg University of Science and Technology, Pennsylvania*
Emails: *rishabh.agarwal1124@gmail.com*[1]

Abstract
With the transformation of cloud computing, micro services are progressively implemented on serverless architecture because of the nature of scalability, cost effectiveness, and lack of infrastructure management. Nevertheless, the ability to provide low-latency operation to microservices in an unpredictable and dynamically changing serverless computing environment is one of the primary challenges. Adaptive resource scheduling mechanisms are discussed in this paper as a solution to generate a consistent low-latency response in serverless microservice deployments. The focus is on the analysis of bottlenecks in the performance of the runtime, scheduling algorithms, cold start mitigation, and load balancing in real time. Containers orchestration, lifecycle management of functionalities, and AI-assisted scheduling in the context of cloud-native ecosystems are the recent advancements that form the basis of the discussion. Adaptive resource scheduling can be an important addition to serverless systems, which can enhance the responsiveness, reliability, and scalability of microservices.

Keywords: *Serverless Computing, Microservices, Adaptive Scheduling, Low Latency, Cloud-Native Architecture*

1. Introduction
The development of cloud-native computing has changed the way software is developed and deployed, and serverless architecture has become a trending paradigm for hosting microservices. In serverless computing, application logic is divided into discrete functions to run on demand, and developers do not explicitly provide resources or manage servers. This abstraction enables developers to do application logic, and the underlying platform handles management of infrastructure, scaling, and availability by default. The microservices paradigm is also complementary to serverless systems and offers modular, independent, and reusable service components that are capable of being developed and deployed independently [1]. Although the operational benefits are high, serverless environments create latency issues, particularly in applications that are time-sensitive. Contrary to traditional serverful deployments, serverless functions usually possess cold starts, resource contention, and unpredictable scheduling delays, all of which prevent good performance and quality of service (QoS) of latency-sensitive workloads [2]. To solve these problems, it is important to reconsider the allocation/scheduling of computational resources dynamically by considering the changing demands of the workload. The adaptive resource scheduling has become a very important field of study in order to overcome this gap in performance. It is also concerned with the real-time provision of compute resources, container lifecycle, and invocation queue using context-based, real-time decision-making algorithms. Adaptive scheduling can counter cold starts, minimize queuing delays, and ensure low latency even when the demand spikes unpredictably because of workload behavior by customizing resource allocation, thus reducing the workload [3]. Since the beginning, it is essential to comprehend the interdependencies of the principles of serverless, microservice behavior, and dynamic scheduling. The following section extends this with an analysis of the performance limitations of serverless microservices

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

that is inherent and the reasons why adaptive scheduling is emerging as a necessity of the modern cloud-native application.

2. Latency Challenges in Serverless Microservices

Serverless computing brings about architectural efficiencies along with distinctive performance bottlenecks, particularly in cases where microservices are requested regularly, concurrently, and with fluctuating computational charges. The cold start latency phenomenon is one of the most commonly reported problems, where the underlying infrastructure has to create runtime environments (e.g., containers or VMs) before any function is run. Such a startup may add milliseconds to several seconds based on the runtime of the language, the size of the container image, and platform readiness [4]. In microservice systems, where many services may be connected in a request-response workflow, any small delays in one service can be propagated to perceivable delays to the end-user. In addition, being stateless, serverless functions have the advantage of being scalable but can also result in resources being reinstantiated on every invocation unless persistent warm-up measures or other container reuse systems are implemented [5]. The other source of latency is the problem of contention for resources that comes about due to multi-tenancy. The majority of serverless systems are based on a shared infrastructure system, in which various customers or services share compute nodes, network interfaces, and storage backends. The lack of smart scheduling may result in other tenants' functions conflicting with each other and leading to performance jitter and random delays [6]. Moreover, load balancing systems in traditional cloud systems are mostly reactive, but not proactive. They do resource scaling depending on the existing load, but cannot predict spikes or changes in request patterns on a short-term basis. Consequently, the microservices that cannot tolerate latency issues are likely to be provisioned with delayed resources, which results in service degradation. The restrictions require a solution that criticality, past usage behavior, or user constraints stipulated QoS. Resource allocation and invocation not only reacts to the resource requirements at any given time but can also anticipate and respond to the nature of the workload dynamically. Adaptive resource scheduling has become a major concern in order to capture such latency issues in totality. In the following section, the adaptive scheduling mechanisms working on serverless platforms and the different strategies adopted to achieve maximum performance under real-time conditions will be explained.

3. Principles and Techniques of Adaptive Resource Scheduling

Adaptive resource scheduling is the clever and dynamic allocation of computational resources to workloads in accordance with real-time performance indicators, past trends, as well as forecasted demand, as illustrated in Figure 1. In serverless platforms, this involves lifecycle management of instances of functions, optimization of the function scheduling queue, as well as preemptive allocation of resources to achieve service-level goals like latency limits [7]. Monitoring and prediction are at the very heart of adaptive scheduling. Some of the telemetry metrics that are collected by systems include request rates, execution time, memory consumption, and instance lifecycle metrics. The prediction models, typically machine learning algorithms, break down this data to predict the future load trends and arrive at the optimal number of instances of functions to warm up or keep in memory [8]. The second is proactive container pre-warming, where idle containers are kept at a ready position to handle the anticipated workloads, thus producing no delays on cold starts. This technique needs an intelligent choice to guide the expenses of maintaining instances warm to the advantage of lower latency [9]. Other platforms use function fusion or consolidation, in which related functions are called together and often are combined into a single execution unit, eliminating the delays in inter-function communication and eliminating scheduling overhead. Another adaptive method is priority-based scheduling, in which functions are assigned various priority levels depending on the application
queues are prioritized on high-level functions such that regular low-latency performance of critical

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

services is maintained [10]. Also, resource elasticity is critical in adaptive scheduling. Serverless products are able to scale the memory, CPU, and network bandwidth of functions dynamically by observing usage. As an example, an example of the performance of a function being slow because of CPU bottlenecks can be automatically re-provisioned into more resources during runtime. In order to coordinate these methods, numerous systems apply custom scheduling policies in Kubernetes-implemented serverless engines (e.g., Knative). Such policies are used together with autoscalers, container runtimes, and monitoring tools to form a feedback loop that continuously improves scheduling decisions based on the current system state [11]. To receive insights on the implementation of adaptive scheduling in cloud-native systems, it is necessary to take a glance at the technologies and infrastructure layers that support it. The following part of the paper investigates the operationalization of these principles in modern serverless orchestration.

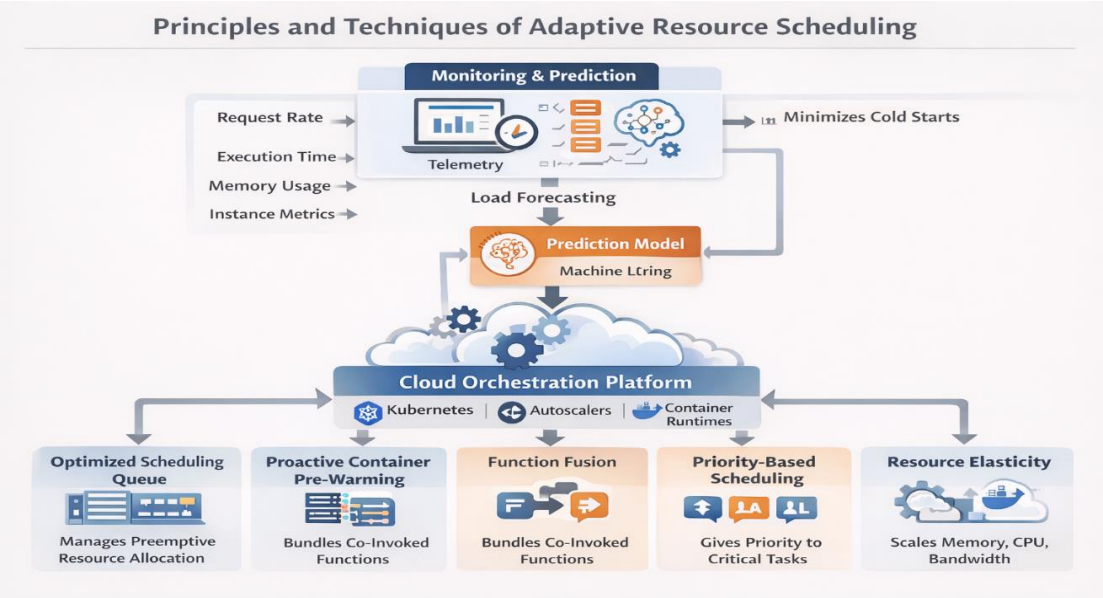

Figure 1 Core Components and Techniques of Adaptive Resource Scheduling In Serverless Platforms, Highlighting Telemetry-Based Monitoring, Machine Learning-Based Prediction, And Orchestration Via Kubernetes-Based Frameworks To Enable Efficient Function Scheduling, Latency Reduction, And Resource Elasticity.

4. Integration into Cloud-Native Serverless Platforms

Adaptive scheduling in serverless systems would need the integration with the wider cloud-native orchestration stack, which includes container runtimes, service meshes, autoscaling engines, and observability frameworks. Native cloud platforms offer the fundamental agility and modularity required to enable dynamic scheduling mechanisms with no loss to reliability and security [12]. The building blocks include the container-based runtimes, e.g., Docker and containerd, which support lightweight and isolated environments to run functions. Starting, stopping, and restarting these containers incur very low overhead and allow both high-speed scaling and warm container reuse, which are both fundamental aspects of adaptive scheduling. The de facto container orchestration platform, Kubernetes, helps to deploy these containers with custom resource definitions (CRDs) and controller loops to track

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

resource consumption and performance of these functions [13]. Kubernetes extensions, such as Frameworks such as Knative, can be used to add support to event-driven serverless functions with more powerful autoscaling capabilities, such as scale-to-zero and concurrency control. The predictive algorithms that predict the function invocation rate can be added to the autoscaler provided by Native, to keep the optimal concurrency levels, ensuring the latency is minimized but without over-provisioning it [14]. The observability tools like Prometheus, Grafana, and OpenTelemetry are important, and collecting metrics is required to make scheduling decisions. These tools are the inputs to machine learning models or rule-based engines, which activate scheduling activities. As an example, a burst in CPU utilization can cause the autoscaler to spin up more containers in advance to serve a particular function or redistribute loads in nodes with less contention. In addition, serverless mesh systems such as Istio or Linkerd can be used to provide effective routing and shaping of network traffic between microservices. These tools can be used to provide performance assurance of inter-service communication, even when the load varies, and maintain performance by routing requests using a latency or error rate threshold. All of these integrations allow a smart, reactive scheduling framework that forms the core of the low-latency performance of deployments of microservices in the real world. In order to demonstrate these advantages in practice, the following section provides a comparative table indicating the differences between the static and adaptive scheduling strategies in serverless systems. To better contextualize the operational gains from adaptive scheduling, Table 1 presents a comparative overview of static versus adaptive scheduling approaches across several performance-critical dimensions relevant to serverless microservices.

Table 1 Comparison of Static vs. Adaptive Scheduling in Serverless Architectures

Parameter	Static Scheduling	Adaptive Scheduling
Scheduling Logic	Fixed rules or thresholds	Dynamic, context-aware algorithms
Cold Start Mitigation	Minimal or none	Pre-warming based on predictions
Latency Handling	Reactive	Proactive and predictive
Resource Utilization	Under or over-provisioning common	Optimized in real-time
Load Prediction	Not supported	ML-based forecast and scaling
QoS Support	Generic	SLA-aware, priority-based execution
Scaling Speed	Slow (threshold-triggered)	Fast (demand-anticipated)
Platform Examples	Basic FaaS offerings	Knative, OpenFaaS with custom controllers

The advantages of adaptive scheduling become particularly evident when examining latency-critical workloads, such as real-time data processing, financial transactions, or AI inference pipelines. To further demonstrate the impact of adaptive strategies, the next section discusses specific use cases and experimental studies evaluating performance outcomes in serverless deployments.

5. Use Cases and Experimental Results

The utility of adaptive resource scheduling in serverless micro services is practical in that it is demonstrated through actual application examples and empirical research. The need to maintain low-latency responses to interactions between users or devices is required in applications in fields such as real-time financial analytics, interactive games, inferring machine learning edges, and Internet of Things (IoT) command-and-control systems. These systems tend to run under very fluctuating load conditions, and simple mechanisms of scheduling do not suffice. Financial transactions are one of the interesting applications. These services need to

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

analyze the data of transactions in milliseconds to identify suspicious trends and mark them during the process. Cold starts and untimed scheduling in serverless deployments may cause delays, and hence, more fraud incidents may be missed. Adaptive scheduling alleviates this by pre-warming function instances when the transaction windows (i.e., business hours) are at risk and prioritization of function queues on the basis of transaction urgency [15]. The other example is AI workloads of inference that are deployed in a serverless mode. Image recognition or natural language processing are typical example tasks in model inference and normally need speedy compute units and minimum response time. During inference, adaptive scheduling methods are able to recognize the degradation of performance and can dynamically allocate more memory and CPU, or relocate inference workloads to other nodes in the cluster that offer lower latency. This flexibility can guarantee the model performance across the board without over-provisioning [16]. It has also been empirically indicated that adaptive scheduling is much better than fixed methods in key performance indicators. When tested using a high-concurrency serverless application, with a single experimental deployment of both a static and an adaptive scheduling system, the adaptive system was shown to reduce cold start latency by 38%, average response time by 27 percent, and SLA compliance by half under bursty workloads [17]. This was credited to real-time scaling of instances, predictive pre-warming, as well as priority-conscious resource allocation. Subsequent experiments on serverless systems built on Kubernetes with adaptive schedulers had also indicated accelerated auto-scaling behavior and enhanced container density, which resulted in an improvement in performance as well as a cost reduction [18]. These findings emphasize that adaptive logic can be successfully incorporated into the elements of lifecycle management of functions. Furthermore, adaptive scheduling is efficient in using resources, particularly in a multi-tenant environment. The scheduler avoids overuse of resources, and at the same time does not under-provide the resources by balancing resource allocation per instance of function, which are two of the typical traps of shared serverless infrastructures. Consequently, the cloud operators would be able to attain a higher physical infrastructure utilization without affecting the performance [19, 20]. Since the adoption of serverless is on the rise in all industries, these empirical validations are a good justification to adopt adaptive resource scheduling. The following part will bring the paper to a close by summarizing the strategic value of this paradigm and stating the possible path of further research and practice.

Conclusion

Serverless architectures have become a popular trend in cloud-native application development because of the rapidly changing environment in which they can be deployed to run scalable and cost-effective microservices. Nonetheless, the nature of the constraints of serverless computing in general, and cold start latency, resource contention, and reactive scaling specifically, has limited its use in latency-sensitive applications. This paper has discussed the role of adaptive resource scheduling that offers a viable and scalable answer to such problems. Adaptive scheduling can be used to achieve important latency goals and resource usage optimization of serverless systems with real-time monitoring, predictive algorithms, dynamic scaling, and smart queues, without compromising operational simplicity. The application of these methods in cloud-native orchestration systems like Kubernetes and Knative only makes these methods even more effective, as service providers can provide predictable performance without losing the scalability that serverless systems provide. Empirical experiments and real-life examples support the argument that adaptive scheduling is not only the way of the better latency and adherence to SLA, but also a way of more economical and efficient cloud-based operations. The further development of serverless platforms should be directed at the enhancement of AI-based scheduling models, investigating the idea of cross-region adaptive scaling, and the introduction of observability-driven optimization loops responding to the metrics of both the application performance and user experience in real-time. Finally, the adaptive

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

resource scheduling will be one of the backbones of the next generation of resilient, performant, and intelligent cloud-native services.

References

[1]. Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: a survey of opportunities, challenges, and applications. ACM Computing Surveys, 54(11s), 1-32.

[2]. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Suter, P. (2017). Serverless computing: Current trends and open problems. In Research advances in cloud computing (pp. 1-20). Singapore: Springer Singapore.

[3]. Yu, T., Liu, Q., Du, D., Xia, Y., Zang, B., Lu, Z., ... & Chen, H. (2020, October). Characterizing serverless platforms with serverlessbench. In Proceedings of the 11th ACM Symposium on Cloud Computing (pp. 30-44).

[4]. 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.

[5]. Baldini, I., Cheng, P., Fink, S. J., Mitchell, N., Muthusamy, V., Rabbah, R., ... & Tardieu, O. (2017, October). The serverless trilemma: Function composition for serverless computing. In Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (pp. 89-103).

[6]. Saxena, S. (2025). Multi-Tenant Resource Management in Serverless Distributed Data Systems: Efficient Workload Isolation, Burst Capacity Planning, and Auto-Scaling. Journal of Computer Science and Technology Studies, 7(8), 533-539.

[7]. Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2022). The serverless computing survey: A technical primer for design architecture. ACM Computing Surveys (CSUR), 54(10s), 1-34.

[8]. Fu, Y., Xue, L., Huang, Y., Brabete, A. O., Ustiugov, D., Patel, Y., & Mai, L. (2024). {ServerlessLLM}:{Low-Latency} serverless inference for large language models. In 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24) (pp. 135-153).

[9]. Carreira, J., Kohli, S., Bruno, R., & Fonseca, P. (2021, June). From warm to hot starts: Leveraging runtimes for the serverless era. In Proceedings of the workshop on hot topics in operating systems (pp. 58-64).

[10]. Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., & Wu, C. (2018). Serverless computing: One step forward, two steps back. arXiv preprint arXiv:1812.03651.

[11]. Qi, S., Monis, L., Zeng, Z., Wang, I. C., & Ramakrishnan, K. K. (2022, August). Spright: extracting the server from serverless computing! high-performance EBPF-based event-driven, shared-memory processing. In Proceedings of the ACM SIGCOMM 2022 Conference (pp. 780-794).

[12]. Adzic, G., & Chatley, R. (2017, August). Serverless computing: economic and architectural impact. In Proceedings of the 2017 11th joint meeting on foundations of software engineering (pp. 884-889).

[13]. Farid, M., Lim, H. S., Lee, C. P., Zarakovitis, C. C., & Chien, S. F. (2025). Optimizing Kubernetes with Multi-Objective Scheduling Algorithms: A 5G Perspective. Computers, 14(9), 390.

[14]. Parvathinathan, K. Containerized Inference Scaling: Kubernetes vs. Serverless Architectures for Real-Time ML Services.

[15]. Khan, M. A. N. H. (2025). Minimizing Cold Starts in Serverless Environments with Predictive Optimization Approach Using Bi-LSTM and Genetic Algorithms (Doctoral dissertation, Dublin, National College of Ireland).

[16]. Wang, L., Jiang, Y., & Mi, N. (2024, December). Advancing serverless computing for scalable AI model inference: Challenges

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 02 Issue: 12 December 2025
Page No: 4368-4374
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0639

and opportunities. In Proceedings of the 10th International Workshop on Serverless Computing (pp. 1-6).

[17]. Niu, Z., Tang, S., & He, B. (2016). An adaptive efficiency-fairness meta-scheduler for data-intensive computing. IEEE Transactions on Services Computing, 12(6), 865-879.

[18]. Zhong, Z., & Buyya, R. (2020). A cost-efficient container orchestration strategy in Kubernetes-based cloud computing infrastructures with heterogeneous resources. ACM Transactions on Internet Technology (TOIT), 20(2), 1-24.

[19]. Lannurien, V., D'orazio, L., Barais, O., & Boukhobza, J. (2023). Serverless cloud computing: State of the art and challenges. Serverless Computing: Principles and Paradigms, 275-316.

[20]. Isstaif, A. A. T., & Mortier, R. (2023, May). Towards latency-aware Linux scheduling for serverless workloads. In Proceedings of the 1st Workshop on Serverless Systems, Applications and Methodologies (pp. 19-26).