ISSN: 3107-6513

Serverless Computing for Small-Scale Applications: A Critical Review of Cost, Performance, and Deployment Trade-offs

Param Bhosale[1], Ruchika Surve[2], Sakshi Patil[3], Meghendra Gajpal[4]

[123] *MCA Students, Department of Computer Applications, D. Y. Patil School of Humanities and Science, Navi Mumbai, India*
[4] *Professor of Practice, D. Y. Patil School of Humanities and Science, Navi Mumbai, India*

[1]parambhosale1725@gmail.com, [2]ruchikasurve29@gmail.com, [3]sakshipatil3307@gmail.com

Abstract—

Serverless computing lets developers run application code without having to manage the servers directly. Its event-driven design, automatic scaling, and usage-based billing can make it a useful option for small applications whose traffic rises and falls. However, serverless is not always the cheapest or fastest choice. Cold starts, execution limits, data-transfer fees, dependence on provider-specific services, monitoring requirements, and consistently high workloads can all affect the final outcome. This paper reviews research on serverless computing for small-scale applications, focusing on cost, performance, scalability, and operational effort. It uses a focused narrative review of selected surveys and studies on cold starts and cost, mainly published between 2021 and 2026, with earlier foundational work. Based on this literature, the paper develops a qualitative framework for comparing Function-as-a-Service (FaaS), virtual machines (VMs), and container-based services across different workload patterns. No new experiments are reported, and no single platform is presented as the best choice for every application. Instead, the review argues that developers should consider how predictable the workload is, how often functions run, how much delay users can tolerate, whether the application needs persistent state, and what the supporting services cost. The framework is an initial guide for the students and developers; before production use, it should be checked against the selected provider, current prices, and the application's actual requirements.

Keywords— Serverless computing, Function-as-a-Service, cloud deployment, cold-start latency, cost efficiency, workload patterns, scalability, virtual machines.

I. INTRODUCTION

Cloud applications can be deployed in several ways, including virtual machines (VMs), containers, managed application platforms, and serverless functions. With a traditional VM, users choose and manage a virtual server and usually pay for the capacity they keep available. In a serverless setup, the provider takes care of much of the infrastructure and runs code when a request or event occurs. Function-as-a-Service (FaaS) is a common serverless approach in which individual pieces of code run on demand. In practice, these functions often work alongside managed databases, object storage, queues, and API gateways.

One reason developers choose serverless is that it can reduce routine infrastructure work. They can spend more time on application logic while the platform adjusts capacity to match incoming demand. Usage-based billing may also suit applications that receive requests only occasionally, because resources do not always need to stay allocated between requests. However, Shafiei, Khonsari, and Mousavi explain that the financial benefit depends on the workload; in some situations, renting a VM costs less than running serverless functions [2]. Paying only for usage therefore does not automatically mean paying less overall.

Once a function's environment is ready, it may respond quickly, but a request can take longer if the platform first needs to create a new execution environment. This delay is known as cold-start latency. Factors such as package size, programming-language runtime, dependencies, platform design, and request frequency can affect the delay [3], [6]. It can be particularly noticeable in a small application that receives little traffic but still needs to respond immediately after a user returns.

Several existing studies already review serverless architecture and its challenges [1],[3]. Meanwhile other papers focus specifically on reducing cold starts [6], [7]. More recent reviews compare serverless with traditional cloud deployment in terms of cost and response time [8] and discuss performance metrics and optimization [9]. This paper does not aim to repeat a general survey. Instead, it brings the existing findings together in a practical framework for small-application developers who are deciding whether to begin with FaaS, a VM, or a managed container service.

A. Aim and Objectives

This paper examines the main trade-offs that determine when serverless computing is a suitable option for a small cloud application.

The objectives are to: (i) outline the cost and operational differences among FaaS, VMs, and containers; (ii) explain how workload patterns and response-time needs affect deployment choices; (iii) identify limitations that appear repeatedly in the literature; and (iv) develop a qualitative framework to support an initial architecture decision.

B. Research Questions

RQ1: Which workload characteristics make serverless computing more or less suitable for a small application?

RQ2: How do cost, cold-start latency, scalability, and operational complexity interact when choosing a deployment model?

RQ3: What practical checks should developers perform before selecting serverless for a real application?

II. BACKGROUND

A. Serverless and Function-as-a-Service

Serverless does not mean that an application runs without servers. Rather, the cloud provider manages the servers and much of the resource provisioning, leaving developers to focus on their code. A FaaS function may run when an HTTP request arrives, a message is received, a schedule is triggered, a file is uploaded, or another service generates an event. The platform provides an execution environment and charges according to its pricing model. Applications commonly combine several functions with managed backend services. This can reduce server administration, although it also means that costs and possible failure points are spread across multiple services.

Serverless architecture surveys describe several layers, including execution isolation, encapsulation, orchestration, and coordination

ISSN: 3107-6513

with backend services [3]. These layers matter because the speed of a function's code does not determine the entire response time. Network requests, storage access, orchestration, and calls to other services may take longer than the function itself. As a result, even a small function can become part of a costly or slow workflow when it repeatedly depends on remote services.

B. Alternative Deployment Models

A VM offers a general-purpose environment and gives developers more control over the operating system, runtime, and processes that remain active. This can make it a good fit for steady workloads, persistent services, specialized software, or applications that need greater control over their environment. That control also brings extra responsibilities, such as planning capacity, applying patches, monitoring the system, configuring scaling, and maintaining availability.

A container packages an application together with its dependencies. Managed container platforms can reduce some of the work involved in running VMs while still supporting long-running services and offering more control than short-lived functions. Costs depend on minimum capacity, scaling behavior, orchestration, and the provider's pricing. The options can also be combined: for example, an application might use serverless functions for event-driven tasks, a container for a persistent API, and managed storage for its data.

III. REVIEW METHODOLOGY

This paper follows a focused narrative critical-review approach rather than a full systematic review. The goal is to bring together relevant findings to help with a practical deployment decision, not to calculate an overall effect size or claim that every publication has been covered. For this draft, the literature search was carried out on 13 October 2026 using publisher websites and scholarly search results. Search terms included “serverless computing survey,” “serverless cost dynamics,” “cold start latency serverless,” “serverless versus virtual machines,” and “serverless performance optimization.”

Studies were considered relevant if they addressed serverless architecture, cost, performance, workload suitability, or cold starts; appeared in a scholarly journal, conference, or recognized research venue; and offered a review, empirical study, or framework related to deployment choices. The review mainly focuses on work published from 2021 to 2026, while retaining earlier studies where they provide useful context about workloads. Publisher and bibliographic pages were used, where available, to check publication titles, author details, dates, and persistent identifiers.

Because the review is narrative and focused, it cannot claim to include every relevant publication. It does not provide a PRISMA screening count, a formal study-quality score, a meta-analysis, or a replication of earlier experiments. The decision matrix in Section 5 is an interpretation of the selected literature, not the result of a benchmark conducted for this paper. A final submission should provide more detail about the search dates and databases, the screening process, and how full texts were assessed.

IV. LITERATURE REVIEW AND CRITICAL SYNTHESIS

A. Broad Surveys and Architecture

Hassan, Barakat, and Sarhan reviewed 275 research papers and brought together findings on serverless concepts, platforms, applications, benefits, and challenges [1]. Their work helps explain the breadth of the field, but a broad survey alone cannot tell a developer which deployment model best fits a particular small application. Shafiei et al. also reviewed serverless applications and their challenges, emphasizing that affordability depends on the use case [2]. This point is central to the present review: the same architecture may be cost-effective for one workload and a poor fit for another.

Li et al. examined serverless design from an architectural perspective and discussed performance factors such as the runtime and execution environment [3]. Their work highlights why it is not enough to compare the time taken by a function's code alone. A fair assessment should account for the full request path, including initialization, data access, network communication, and calls to external services. The workload and service configuration should also be stated, since results from one provider or runtime may not apply to another.

B. Workload Shape and Cost

The cost of serverless depends on how much work is performed, how frequently functions are called, how long they run, how much resource they use, and which supporting services are involved. Shafiei et al. note that affordability varies by workload and that a VM can be cheaper in some cases [2]. Hamza, Akbar, and Capilla explored how organizations assess serverless costs through interviews with 15 experts from eight companies. Their findings suggest that serverless can work well for unpredictable demand, but may not be economical for some high-scale applications [5]. Together, these studies support choosing a deployment model based on the workload rather than on the technology's popularity.

For an application that receives occasional requests, usage-based billing may avoid the expense of keeping an application server running all the time. With frequent, steady traffic, however, invocation and execution charges can add up. Developers should look beyond the headline compute rate and include API gateways, databases, logging, data transfer, retries, orchestration, and any minimum capacity required by alternative services. Prices also differ by provider, region, and configuration and may change over time, so this paper does not suggest a universal price threshold.

C. Cold Starts and Performance

A cold start happens when a platform needs to prepare a function's execution environment before it can handle a request. Ebrahimi et al. group approaches to reducing this delay into application-based, checkpoint-based, invocation-prediction-based, and cache-based methods [6]. Golec et al. later provide a systematic review and taxonomy of cold-start techniques, showing that the issue continues to receive research attention [7]. These methods may improve response times, but some require extra resources, pre-warming, prediction, or configuration tied to a particular platform.

How much a cold start matters depends on what the application does and how quickly users expect a response. A background task that processes a file asynchronously may tolerate a delay, while a login page or interactive API may not. Average response time can also hide occasional slow requests. When responsiveness is important, developers should examine p95 and p99 latency as well as throughput and error rates. These are suggested measures for future evaluation; they were not measured in this review.

D. Recent Reviews and the Need for Focus

Kumari et al. (2026) compare serverless and traditional cloud models by looking at cost and response latency, including cold starts, throughput, and evaluation methods [8]. Jarallah et al. (2026) review performance measures and optimization strategies, highlighting latency, cost, and execution time in the studies they examined [9]. Given these recent reviews, another general list of serverless benefits and drawbacks would add little on its own.

For that reason, this paper takes a deliberately practical approach: it links common workload patterns to an initial deployment choice and makes the assumptions behind that choice clear. It does not claim to introduce a universal cost model. The framework still needs to be tested through a case study or an evaluation with practitioners.

ISSN: 3107-6513

V. COMPARATIVE ANALYSIS

A. Main Trade-off Dimensions

Dimension	Serverless / FaaS	VM or Container Service	Decision implication
Cost model	Often usage-based; may avoid idle compute charges, but connected-service and invocation costs accumulate.	May involve a baseline capacity cost; can be economical under sustained use, depending on configuration.	Estimate total application cost under realistic low, typical, peak, and sustained demand.
Scaling	Platform-managed scaling can help with bursty events, subject to concurrency, quotas, and downstream limits.	Scaling may require configuration and capacity planning; managed container platforms can automate parts of it.	Check both scaling limits and whether the database or API can handle the same burst.
Latency	Cold starts and remote-service calls can add variable delay.	A continuously running service may avoid function cold starts but can still experience startup, queuing, or resource contention.	Measure end-to-end p50/p95/p99 latency on the target platform.
Operations	Less server maintenance, but platform configuration, observability, IAM, and event integration remain.	More control, with greater responsibility for patching, monitoring, deployment, and availability.	Compare total engineering effort, not only infrastructure administration.
Runtime and state	Execution and runtime constraints vary by provider; external state is commonly managed separately.	More flexibility for long-running processes and persistent runtime behavior.	Choose based on execution duration, dependencies, state, and portability needs.
Portability	Provider triggers and managed services can create lock-in.	Containers can improve packaging portability, though platform dependencies remain.	Document exit costs and keep business logic separated from provider-specific integrations where practical.

Table 1 compares the main trade-offs without treating any deployment model as the winner in every situation. The actual result will depend on the provider, region, runtime, allocated resources, application design, and billing setup. For a meaningful cost comparison, the same workload should be used for each option, and the estimate should include all services needed to provide the same user-facing features.

B. Workload-Oriented Decision Matrix

Workload pattern	Potentially suitable starting point	Main risks to check	Practical recommendation
Low or occasional traffic	FaaS or a serverless application platform	Cold-start delay, minimum charges in connected services, and unexpected background invocations.	Model monthly cost at realistic request volumes; test first-request latency.
Bursty or unpredictable traffic	FaaS or an autoscaling managed platform	Concurrency quotas, throttling, downstream bottlenecks, and sudden cost increases.	Test a burst scenario and verify rate limits, retries, and database capacity.
Steady, high utilization	VM or container service may be competitive;	Overprovisioning in VMs versus accumulated usage charges in FaaS.	Estimate total cost at average and peak utilization; do not
	compare against FaaS		assume pay-per-use is cheaper.
Latency-sensitive interactive endpoint	Depends on cold-start tolerance and runtime behavior	Tail latency, initialization overhead, and network hops to storage or APIs.	Measure p95/p99 response time; consider provisioned/warm capacity or a persistent service if needed.
Background event or scheduled task	FaaS is often a reasonable candidate	Execution-duration limits, retries, duplicate events, and observability.	Confirm task duration, idempotency, failure handling, and total workflow cost.
Long-running or stateful process	VM or container may fit more naturally	Function execution limits, external state coordination, and workflow complexity.	Prefer the simplest model that meets runtime and state requirements.

C. A Practical Selection Procedure

The review findings can be turned into the following practical steps before implementation:

1) Describe the workload. Estimate the number of requests per day or month, typical and peak concurrency, execution time, payload size, and whether demand is steady or unpredictable.

2) Set the service requirements. Specify acceptable response times, availability goals, execution limits, state requirements, and any security or data-residency constraints.

3) Map the full architecture. Include API gateways, queues, databases, storage, monitoring, logs, network transfer, and retries. Comparing compute prices alone can give a misleading result.

4) Estimate costs for three scenarios. Use low, expected, and peak or sustained demand, and check current provider pricing. Record the assumptions and revisit the estimate when usage or prices change.

5) Test the biggest uncertainty. For FaaS, this may be cold-start delay or concurrency limits; for a VM, it may be idle cost or the effort needed to scale; for containers, it may be minimum capacity or operational workload.

6) Select the simplest option that meets the requirements, then review the decision when real usage data are available. A hybrid design may be appropriate if different parts of the application have different workload patterns.

VI. DISCUSSION

The literature reviewed here suggests that serverless is best treated as one possible deployment model, not as a replacement for every VM or container. It is often a strong candidate for event-driven components with changing demand, short or bounded execution times, and little need for a process to remain active continuously. In those cases, automatic scaling and reduced server maintenance may be worth the extra design complexity of splitting work into functions.

One common source of inaccurate cost estimates is treating the price of running a function as the total cost of the application. Real systems also need data storage, networking, authentication, monitoring, and integration services. A workflow that calls several functions and repeatedly reaches a remote database can end up slower or more expensive than one long-running service. Hamza et al.'s cost study reinforces the need to consider the actual organizational workload rather than relying on the billing model alone [5].

Performance should be assessed across the entire request, not just the time spent executing function code. Cold starts, initialization, network

ISSN: 3107-6513

delays, queueing, storage access, and calls to dependencies all influence what users experience. Reviews of cold-start mitigation describe a wide range of techniques, but later reviews continue to examine the issue, suggesting that it remains relevant across platforms and workload types [6], [7].

For MCA students and small development teams, this framework offers a way to make a reasonable first choice without optimizing too early. A managed service may be a sensible starting point if it reduces setup work and meets the application's cost and response-time needs. Developers should still document their assumptions and avoid tying business logic too closely to provider-specific services, which can make migration harder. The choice should be reconsidered as real usage, performance needs, and costs become clearer.

VII. LIMITATIONS AND FUTURE WORK

There are several limitations to this paper. First, it is a focused narrative review rather than an exhaustive systematic review, so the selected studies do not cover every provider, platform, or workload. Second, the comparison is qualitative; it includes no original benchmark results, provider-billing simulations, or statistical analysis. Third, prices, quotas, runtime limits, and platform features change, so any real deployment decision must be checked against current provider documentation. Finally, small applications vary widely, meaning the workload categories in the decision matrix are illustrative rather than a classification validated using a dataset.

Future work could test the framework using a small, repeatable benchmark that runs the same application on FaaS, a VM, and a managed container service. The test should include low, bursty, and sustained traffic; calculate cost per successful request; measure p50, p95, and p99 latency; record how often cold starts occur; and include storage, API, and logging charges. Another useful step would be to ask developers whether the framework helps them choose an architecture. These evaluations would add practical evidence to the current literature review.

VIII. CONCLUSION

Serverless computing can be useful for small applications because it reduces direct server management and can scale in response to events. It is not the right fit for every workload, however. The decision depends on traffic patterns, acceptable latency, execution and state requirements, the total cost of the application, and the complexity of its supporting services. The literature suggests that usage-based execution may suit unpredictable demand, while VMs or containers may be more competitive for steady, heavy use or some latency-sensitive applications. Cold starts and the performance of connected services remain important considerations.

This paper brings selected research together in a workload-based decision matrix and a practical selection process. The framework is intended to help developers reason about architecture at an early stage, not to replace provider-specific cost estimates or performance tests. The main recommendation is to compare the full application under low, expected, and peak demand, and to measure end-to-end response times before committing to a deployment model. Further experiments are needed to establish how well the framework works in practice.

REFERENCES

[1] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, “Survey on serverless computing,” Journal of Cloud Computing, vol. 10, art. no. 39, 2021. doi: 10.1186/s13677-021-00253-7.

[2] H. Shafiei, A. Khonsari, and P. Mousavi, “Serverless computing: A survey of opportunities, challenges, and applications,” ACM Computing Surveys, vol. 54, no. 11s, art. no. 239, pp. 1–32, 2022. doi: 10.1145/3510611.

[3] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, “The serverless computing survey: A technical primer for design architecture,” ACM Computing Surveys, vol. 54, no. 10s, art. no. 220, pp. 1–34, 2022. doi: 10.1145/3508360.

[4] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Sigaev, and R. Bianchini, “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” in Proc. USENIX Annual Technical Conference (USENIX ATC), 2020, pp. 205–218. Available: https://www.usenix.org/conference/atc20/presentation/shahrad.

[5] M. Hamza, M. A. Akbar, and R. Capilla, “Understanding cost dynamics of serverless computing: An empirical study,” in International Conference on Software Business, 2024. doi: 10.1007/978-3-031-53227-6_32.

[6] A. Ebrahimi, M. Ghobaei-Arani, and H. Saboohi, “Cold start latency mitigation mechanisms in serverless computing: Taxonomy, review, and future directions,” Journal of Systems Architecture, vol. 151, art. no. 103115, 2024. doi: 10.1016/j.sysarc.2024.103115.

[7] M. Golec, G. K. Walia, M. Kumar, F. Cuadrado, S. S. Gill, and S. Uhlig, “Cold start latency in serverless computing: A systematic review, taxonomy, and future directions,” ACM Computing Surveys, vol. 57, no. 3, art. no. 65, 2025. doi: 10.1145/3700875.

[8] A. Kumari, R. K. Behera, K. S. Sahoo, B. Sahoo, and A. H. Gandomi, “Emergence of serverless computing in cloud: A state-of-the-art review of challenges and opportunities,” Machine Learning with Applications, vol. 24, art. no. 100862, 2026. doi: 10.1016/j.mlwa.2026.100862.

[9] Z. F. Jarallah, H. S. Alhussian, E. H. Sumiea, M. Eltahir, N. Aziz, A. W. Al Abri, I. S. Almuniri, Y. Saidu, and S. Adamu, “AI-driven performance optimization strategies and key metrics in serverless cloud computing: A systematic literature review,” Machine Learning with Applications, vol. 25, art. no. 100994, 2026. doi: 10.1016/j.mlwa.2026.100994.

[10] P. Zhao and M. Ghorbian, “Analyzing economic frameworks for serverless computing: Taxonomy, metrics, and future directions,” International Journal of Computational Intelligence Systems, vol. 19, art. no. 223, 2026. doi: 10.1007/s44196-026-01330-y.

.