ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 2 Issue 1 February 2022 / Page No: 112-121
Paper Id: JETA-V2I1P114 / Doi: 10.56472/25832646/JETA-V2I1P114

Original Article

The Rise of Serverless: Redefining Scalability and Flexibility in the Cloud

Joseph Franklin Santhosh Kumar
Senior Software Engineer, USA.

Received Date: 23 December 2021 Revised Date: 27 January 2022 Accepted Date: 26 February 2022

Abstract: *Serverless computing has changed the cloud computing paradigm, where developers do not have to worry about where the code is running or any access to the server. Although it is entirely different from infrastructure as a Service (IaaS) or Platform as a Service (PaaS), serverless offers flexibility through the dynamic nature of the application's usage. This implies that resources are committed, based on certain event occurrences, dynamically, and so both cost and value are optimized. Unlike the traditional models where one would need to over-provision or spend time and effort to control the idle resources as costs are billed on the basis of the actual time spent executing the functions, serverless computing is therefore fitting for solutions with unpredictable usage and demand. Cognitive reasons have influenced the transition to serverless to enhance the speed of delivery, decentralize the conveying of intricacy, and cut on the expenses, majorly in the microservices architectural style and updates to the event-driven applications. Since key cloud service providers such as AWS, Google Cloud, and Azure have introduced serverless capabilities to their services, the technology has emerged in more applications ranging from e-commerce to the Internet of Things. But as with so many things in computer science, it succeeds at the cost of certain downsides, such as server cold start and limitations on processes that can take a long time. To weigh these benefits against the risks, this paper considers prior research, use cases, and data from experiments that explore serverless computing today and in the future of the cloud.*

Keywords: *Serverless Computing, Cloud Computing, Scalability, Flexibility, Platform as a Service (PaaS), Microservices.*

I.INTRODUCTION

With the help of cloud computing, application development has become incredibly important in modern technologies and has significantly evolved the approaches to creating, deploying and growing applications. That makes it a foundation for organizations willing to build increased flexibility and efficient processes, as well as to scale computing resources up and down quickly. [1-3] Of all the models of cloud computing, serverless computing, also called Function as a Service (FaaS), has become perhaps one of the most revolutionary architectural patterns. Serverless computing reduces the hardware foundational elements of computing where the developer strictly focuses on writing and deploying code without the added responsibility of underlying servers, storage or network.

Serverless computing is, on the other hand, in contrast to the IaaS paradigm, where developers have to provide for their virtual machines and capacity planning or PaaS, where scaling is done manually. Dynamics take place as events, like API, database or messaging calls, and scripted functions are triggered to work. Thus, the extreme automation of the scaling, resource distribution, and maintenance, with no need for any input from the individual client, results in faster deployment cycles and efficient use of the resources. This makes serverless cool, especially for those developers who are designing apps that have real-time unpredictable demand, for example, microservices, IoT and data processing pipelines among others.

A. Problem Statement

A main problem of traditional cloud computing paradigms has been the insufficient control a developer has over the services' resource allocation, as well as the constant pressure of having to guess a service's demand and scalability and, in turn, how to control its costs. In IaaS and PaaS services, a developer needs to allocate physical servers and usually allocates extra resources to avoid concerns over possible bottlenecks. This may cause trouble and lead to some servers being over-provisioned, which means that the servers are very idle during periods of low traffic, which is very expensive. Furthermore, manual scaling can be quite tasking and does not allow for the anticipation of traffic variations or the need to rush through scaling in order to meet increased demand.

Serverless computing does not have these problems because it adopts an event-driven execution model where resources are only procured and charged only when events occur. The scale is managed on its own by the cloud provider, and new functions are established or closed down depending on the demand; this is cost-effective. Old methods of having specific servers pre-allocated take away the burden from the developers and leave them to manage the infrastructure, plan for capacity or the expenses of unused capacities. Instead, they can concentrate on the writing of scenarios and particular business logic, as well as fine-tuning code as much as possible. This shift in paradigms reduces the complexity surrounding application deployment, thereby enabling developers to design and implement complex applications based on a cloud model while shouldering little or no operational burden characteristic of multi-tenant, cloud-style environments.

B. Objectives

a) *Explore the Evolution and Impact of Serverless Computing:*

The first research question is: What has serverless computing transformed from conventional cloud models, and how is it being utilized in application development at present? This includes who developed serverless computing, how and why it was required to support certain classes of applications after developers identified IaaS and PaaS limitations, and how developers have used it to address application requirements, including microservices, real-time data processing and event-driven applications.

b) *Analyze Serverless Models and Compare Them with Traditional Cloud Architectures:*

The second aim is to provide further discussion on the serverless computing models and compare them with the conventional server models – IaaS and PaaS in terms of scalability, cost, flexibility and performance. This paper will also discuss some of the drawbacks of serverless, including cold start latency, throttling effects, and executing time limits, and the benefits of serverless, including pay-per-execution cost and auto-scaling.

c) *Present Experimental Results Demonstrating the Benefits of Serverless in Real-World Scenarios:*

The last goal is to present quantitative evaluation and experimental outcomes that show that serverless computing can be beneficial in practice. Over a normal cloud environment, models like serverless will be deployed in real-life workloads with different workloads in this paper to highlight performance, costing and scalability benefits with real-life valuable applications in different sectors.

II.LITERATURE SURVEY

This section introduces an understanding of the development of serverless architecture and contrasts serverless computing with IaaS and PaaS models of cloud computing. [4-7] it also discusses the main research finding in scalability and performance and discusses the richness and constraints of serverless computing.

A. Serverless Architecture Evolution

While still a fairly young approach, serverless computing traces its origin to the broader evolution of the notion of cloud computing. AWS originally introduced the general public to the concept of serverless with the launch of AWS Lambda in 2014. What AWS Lambdas enabled the developers to run the code that executes in response to an event simply elided the servers that must be provisioned and managed; this was an entirely new paradigm of the cloud-computing. This created much attention and demand, and other significant cloud players such as Microsoft Azure and Google Cloud introduced their own serverless environments, Azure Functions and Google Cloud Functions, correspondingly.

As for the theory, the first articles devoted to serverless architecture conventionally considered the novelty as a departure from infrastructure management to code execution. In serverless computing universally, the requirement to manage servers, scale them, and, in the event of failure, replace them was taken over by the provider. This architecture was planned for the use of applications that respond to events like API calls, database updates or file uploads in a very short time.

When serverless computing emerged, subsequent studies were conducted to measure its effects on issues important to the metric of concern of Breakdowns, developers, and businesses cost, scaling, and developer efficiency. Several of these works pointed out those serverless computing reduced operational costs because of the pay-as-you-go model, whereby an organization only pays for the execution of the function and the amount of resources allocated to the function. Also, a feature to create functions that automatically scale up or down is added as an advantage because it makes serverless solutions more flexible and responsive.

B. Serverless and Traditional Cloud Models

With the increase of interest in serverless computing, many subsequent works aimed at rewriting serverless computing for comparative analysis of its performance, flexibility, and cost advantage over heritage cloud offerings such as IAAs and PaaS. Classic IaaS just offers users the ability to hire virtual hosts on an as-needed basis while granting full control over the substrate infrastructure. While PaaS hides the physical hardware, the developer is still aware of the physical application servers to some extent, as he will have to be involved in some form of server configuration, especially in terms of scalability and updation.

They found that when the workload of an organization is unpredictable or highly volatile, serverless computing is much more scalable and far less expensive than IaaS and PaaS. This is because serverless computing is one whose resource is not pre-provisioned but instead provisions itself based on the demand on the server. One aspect of this process is that developers do not necessarily have to interfere with the process or tweak an auto-scaling setting, which makes it even easier concerning the operational overhead.

However the research also brought out the fact that serverless computing is not good to be used on all the workloads. However, it can be pointed out that the approach is not well suited for performing long-running tasks because of the execution time constraints most cloud vendors impose. For example, AWS Lambda commonly has a limit of 15 minutes of function duration, which means serverless is not very suitable for computation-based workflows. In addition, since serverless functions are always executed in stateless containers, if your application needs to include the management of the state, especially the state in its external manifest form, then traditional cloud models are more appropriate for you.

C. Sustainability in Serverless Computing

The first benefit of serverless architecture that few would dispute is the inherent flexibility of the system, which can grow and shrink as needed with no human intervention. Studies conducted by Liu et al. about serverless platforms show that such platforms should be able to handle almost limitless loads because they are actively scaled according to incoming traffic. This characteristic of serverless computing makes it very useful when traffic patterns of applications are variable, and unpredictive variances in demand can happen, for example, in e-commerce during promotional campaigns or in real-time big data processing.

In serverless paradigms, applications are kicked off based on events, and the provider takes care of the instantiation and de-provisioning of the infrastructure for executing the functions. It also automatically scales resources and allows organizations to pay for accurate computing power used by functions. However, in more conventional IaaS or PaaS arrangements, developers are expected to set up auto-scaling policies manually that might not scale up fast enough when traffic is high or scale up too often and cause performance issues or costs they would not have otherwise incurred.

Note one disadvantage arising from scalability cold start latency, though they admit that this is not necessarily an empirical weakness. Cold starts happen when a serverless function has been called for the first time or after a while when no one has been using it, and the container needs to be created anew. This process can cause response delays of up to several hundred milliseconds to happen in applications that are sensitive to response time. Cold start latency is even more significant in high dynamism because functions may be called after long intervals have elapsed.

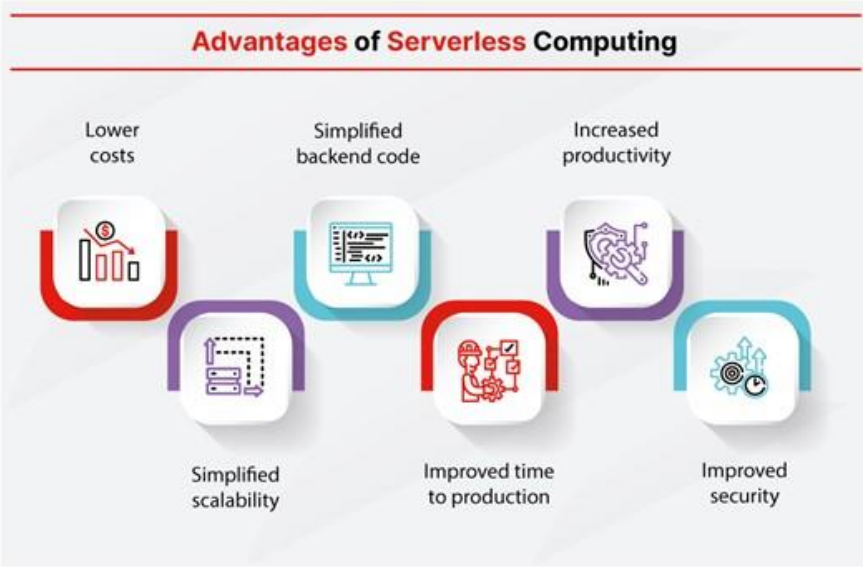

Figure 1: Advantages of Serverless Computing

D. Advantages of Serverless Computing

A focused strategy is used with 6 icons representing the major benefits of serverless architecture. Here is a detailed explanation of each:

a) Lower Costs:

Serverless computing uses a "consumption-based" model, where businesses consume compute power and pay only for the actual usage and time that is needed without having to keep provisioned and managed servers waiting for traffic spikes to occur. Such tendencies lead to considerable cost reduction, especially in applications with variable loads.

b) Simplified Backend Code:

In serverless application structures, it is possible to write the code of the application and not to think of the infrastructures or servers. They make the code behind the scenes easier to manage as the cloud providers take care of scaling, patching times and resource management. They are less consuming resources to implement because developers write them in a way that they will react or respond to a particular event/occurrence.

c) Increased Productivity:

By using cloud hosting services, developers are relieved of the underlying issues such as scaling, monitoring and server maintenance and thus can deploy applications more easily. This outsourcing of operational responsibilities consequently enhances productivity by minimizing the steps of implementation and burdensome tasks so that different groups can focus on creating additional qualities and capabilities.

d) Simplified Scalability:

A well-known advantage of serverless computing is the possibility of automatic and, what is more, unnoticeable scaling. It provides for functions to be easily resized to accommodate traffic that may surge at any given time of the day without requiring the input of human interference. The precision here makes serverless perfect for applications whose traffic is not very constant; thus, resources will be prorated in real time.

e) Improved Time to Production:

Indeed, serverless computing greatly helps shorten the time to market new application solutions or just new versions, as the selected CSP already provides the necessary IT environment. This enables the teams to deploy code to production rapidly without a lot of formality making the overall time for the production to enhance.

f) Improved Security:

Given that the cloud provider is responsible for the deployment and control of the underlying infrastructure including the OS layers, then security is boosted. Serverless architectures implement security through dedicated services like encryption and

network security that automatically accompany the infrastructure where the application is to be hosted; system security becomes harder-coded into the application, thus inherently more secure than traditional server-based architecture.

For these reasons, serverless applications are becoming increasingly popular in modern organizations to achieve the best results for development and cut expenses while maintaining scalability and security.

Table 1: Comparison of Cloud Models

Feature	IaaS	PaaS	Serverless
Infrastructure Control	Full control over VMs	Partial control (middleware)	No control over infrastructure
Scalability	Manual scaling required	Auto-scaling with limits	Automatic, near-infinite scaling
Cost Model	Pay for server provisioning	Pay for platform usage	Pay for function execution only
Deployment Time	Slow (set up virtual servers)	Moderate (pre-configured platforms)	Instantaneous (on-demand execution)

g) Infrastructure Control:

IaaS supplies the most control over the virtual machine, and users can reconfigure the operating system, network, and storage. PaaS provides even lesser control and mostly concentrates on application deployment while the firm takes care of the physical infrastructure. Serverless does not allow usage of the infrastructure because the cloud provider holds all the responsibility.

h) Scalability:

In IaaS, scaling is either manual or you need to put into place your own auto-scaling policies. PaaS platforms enable riders to scale automatically, however, often on a set of different configurations. Serverless computing, on the other hand, is automatic and theoretically unlimited and elicits resources reserved for function invocations.

i) Cost Model:

Existing applications and On-Demands charges: IaaS charges are made based on the provision of virtual servers, even if the servers are asleep. PaaS models use utility-based billing, and you pay for the usage of the platform; however, idle resources must also be paid for. On the contrary, Serverless only made charges based on the consumption of serverless functions in terms of the function duration, which makes it very cheap.

j) Deployment Time:

Implementation in IaaS can take time because the IaaS providers need to create virtual machines and then configure the infrastructure. PaaS cuts down deployment time, while serverless functions take nearly no time as they are run only when triggered and have no server setup.These comparisons show that although serverless is a scalable and inexpensive model, it poses challenges that developers must address due to constraints such as aptness in managing long-running processes or workloads that are intolerant to latency.

III.METHODOLOGY

The following Subsection explains how the research study was conducted in terms of performance, scalability and cost analysis of serverless computing against conventional cloud models. [9-13] A microservices-based architecture was used as the foundation for this study, with two deployment scenarios: In one of the use cases, it employs serverless technology, AWS Lambda, to perform its task, while in the other use case, it uses more generalized IaaS model of Amazon EC2. In both scenarios, the workloads of the two architectures were varied, with the intention of evaluating their behavior under different prototypes.

A. System Architecture

To avoid any gaps and better evaluate the results, we based the experimental setup on the microservices architecture approach. Every microservice served a particular purpose, including data intake, analysis, and data output. The architecture consists of several components that interact with out-of-band services such as API, database, and Cloud event. All these elements have been employed on the serverless platforms alongside the traditional cloud for comparison.

Serverless Environment: In the serverless scenario, there was one AWS Lambda function per microservice that ran the microservice. These functions were specifically invoked as events from various Cloud services such as Amazon API Gateway for HTTP request configuration, Amazon DynamoDB as NoSQL Database for dynamic data storage, and Amazon S3 for file storage. Event-driven architecture helped to run the Lambda function on the basis of event occurrence without occupying more resources

and charges. In addition, AWS CloudWatch was employed for measuring the system efficiency, it tracked all invocations and serverless function response time.

Traditional Cloud Environment: In the traditional scenario, we had the same composing microservices deployed on the Amazon EC2 instances. The use of auto scaling policies was applied to the EC2 instances to be able to provide for the increasing traffic loads while applying the load balancer to spread out the traffic in intervals across more than one instance. Its organization provided an uninterrupted flow to each service, although the infrastructure was reserved even if there were no requests. The permanent storage of data was also handled with the help of the Amazon DynamoDB service, just as in the case of serverlessness. In the case of both scenarios, monitoring and logging were carried out with CloudWatch, considering the solution was the same.

B. System Architecture for Serverless and Traditional Models

a) Serverless Components:

AWS Lambda represents events from services like API Gateway, DynamoDB, and S3 as comprehensive and full-pledged AWS Lambda functions.

b) Traditional Components:

Proceeding from the EC2 instance to the load balancer, using the instance and manually adjusting according to use without application load balancer, and inserting data into DynamoDB for data warehousing.

c) Event-Driven Interaction:

Explaining how API requests are invoked to call functions or hit servers in both cases.As a result, the given high-level system architecture directly correlates serverless functions with traditional instances in terms of their interaction with cloud services. The main distinction is in automatical scaling and running serverless functions compared to perpetually assigned server resources in EC2-based setups.

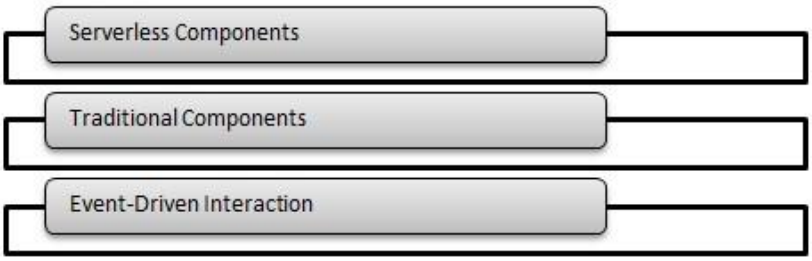

Figure 2: System Architecture for Serverless and Traditional Models

C. Experiment Design

It was decided to compare the serverless model and the traditional cloud model under similar circumstances in the experiment. The emphasis was given to what scale of traffic each architecture supports, the performance of the entire system under load, and the corresponding costs.

a) Scenario 1: Serverless Deployment

For the serverless deployment model, all micro-services were implemented using AWS Lambda functions. HTTP calls were made to the system via Amazon API Gateway, which also served as the front door, responsible for routing HTTP calls to Lambda functions. The data was stored at Amazon DynamoDB, which is a managed NoSQL database that would scale on its own based on the traffic of reads/writes. This architecture did not need resources to be provisioned in advanced Lambda functions that were brought up and run only when needed.

Lambda Functions: Business processes that began with simple event handling services and then self-extended to support higher levels of requests as the latter became more frequently requested.

- *API Gateway:* Coordinated and linked HTTP requests to Lambda functions.
- *DynamoDB:* Responsible for data storage and was able to grow based on the traffic.This setup was chosen for scaler, and doing so without the explicit involvement of a developer is cost-efficient since Lambda equates cost to the number of executions.

b) Scenario 2: Traditional Cloud Deployment

In the case of the traditional setup, the same micro-services ran on Amazon EC2 instances. These instances were specifically provisioned and scheduled to run always, irrespective of the load of the system. Auto-scaling policies were next set with the aim of increasing or decreasing the number of EC2 instances depending on the traffic level received in the system. To address the issue of handling many requests, an Elastic Load Balancer (ELB) was created to sit in front of the EC2 instances. In order to maintain coherence between both scenarios, data was dumped into Amazon DynamoDB.

- *EC2 Instances:* Automated manually scaled virtual machines that would perform each microservice.
- *Load Balancer:* Organized the distribution of HTTP requests of different EC2 instances.
- *DynamoDB:* Used for data persistence in the same way as the first case we described. In this case, the main difference, as mentioned earlier, was the flexibility of not requiring manual intervention to set up and scale policies. However, the flexibility of the system was lower than in the serverless one. Furthermore, EC2 instances span up round the clock, the cost was attributed to the uptime and not the actual workload.

D. Performance Metrics

The experiment measured several key metrics to evaluate [14-17] both deployment models under identical conditions:

- *Scalability***:** How efficient the system was in terms of its growth from small to high loads of data.
- *Cost Efficiency:* The total price analyzed during the experiment in solitude with reference to AWS cost detailed on the screenshot below, to focus on specific resources: Amount of used computing resources that were consumed by Lambda functions as opposed to the number of used EC2 instances.
- *Latency:* The time it takes each request to run, including stateful run times in a serverless scenario or the time in an EC2 instance to provision a request.
- *Throughput:* The number of completed requests per second at different working loads.

E. Workload Configuration

In both cases, the load level of the system was also varied to determine the scalability and performance of the system with various load levels for each of the scenarios. The function invocation rates ranged from 10 rps (low throughput) to 1000 rps (high throughput). These workload configurations enabled us to monitor how each architecture tackled abrupt surges in traffic and sustained high-load situations.

a) Low Traffic (10 RPS):

The computer programs used in the model are configured to simulate steady, normal operating conditions.

b) Moderate traffic (100 RPS):

Emulated a typical busy time, for instance, during a particular marketing campaign or during consolidation to launch a new product.

c) High Traffic (1,000 RPS):

Simulated a dramatic increase in traffic, for example, when using Black Friday or a ‘viral’ post.For each traffic level, parameters including the rate of response, latency and cost were recorded relating to the System Metrics. This made it possible to make a comparison of the performance of the two serverless and the traditional cloud model.

IV.RESULTS AND DISCUSSION

Therefore the experiment offers insights on the comparative performance and costs between the serverless and a standard cloud computing environment. Response times, scalability, and cost efficiency under varying loads with comparison in relation to the experimental data gathered for the work are evaluated in this portion of the paper.

A. Performance Comparison

The efficacy of the different approaches and optimizations was quantified primarily based on the throughput response time within the system from the number of requests per second. The main area of interest was to find out how serverless functions (AWS Lambda) and Standard EC2 instances work and perform under varied traffic loads.

a) Response Time Comparison

Evaluation of response times was used to establish the performance of different serverless and traditional deployments under various loads, beginning with a low load of 10 requests per second and progressing to a high load of 1000 requests per second.

Table 2: Response Time Comparison (Serverless vs. EC2)

Load (Requests/sec)	Serverless Response Time (ms)	EC2 Response Time (ms)
10	120	90
100	140	120
500	200	220
1,000	260	300

When the number of requests per second was low (10), EC2 instances had better results than the serverless model: 90 ms against 120 ms. This has led to a cold start problem in serverless computation, where a function lags at the first invocation after a period of idle time. The presence of cold start overhead leads to the high initial response time of serverless functions, especially when traffic is low.

At the 100 requests per second traffic load, both serverless and EC2 almost had similar results. Serverless functions had slightly higher latency still than called functions (140 ms to 120 ms ASL).

In higher loads, thereby executing 500 requests/sec and above, serverless functions outcompeted EC2 instances. As for response time it was below at both 500 and 1000 requests per second for the serverless category. At 500 requests/sec, the serverless architecture yielded an average response time of 200 ms while EC2 at a similar request rate took 220 ms, and at 1,000 requests/sec, serverless responded to requests within 260 ms, while at the same rate, EC2 took 300 ms.

b) Advantages of Serverless based on scalability

The metrics validate that serverless architecture outperforms the EC2-based solutions even at higher traffic and confirm that serverless computing boasts improved scalability for runtimes. The feature of auto-scaling of serverless functions enables the application of the required amount of computational resources only when it is really needed so that response times would still remain satisfactory even after increasing request rates.

However, with EC2 instances, there are already some policies set in auto-scaling that might not respond fast enough in terms of scaling in order to accommodate traffic spikes. From the experiment presented above, one can also observe that EC2's scaling process deprived the instance of some extra responsiveness under larger loads. This is so because auto-scaling of EC2 instances is characterized by time-consuming events such as the booting of new virtual machines, which forms a major hurdle when executing requests.

c) Cold Start Impact on Serverless Performance

That is, one of the insights identified from the experiment is the cold-start effect on serverless performance at low traffic conditions. However, when a function is not often used, it might be in a cold state, so the request will take some time to run after initializing the function. This latency occurred mainly under low load (10 requests/sec), and all the time, serverless functions recorded a higher response time than EC2 instances. Cold start latency remains the slow speed obstacle to latency-sensitive applications in serverless frameworks.

B. Cost Efficiency

Apart from performance, the experiment was also used to determine the cost difference between serverless and normal EC2 implementations. Price was estimated by the measure of computing resources used throughout the 24-hour test period based on the rates of invocations and resource configurations of the targets of each deployment.

a) Cost Model Comparison

i) Serverless Model:

AWS Lambda charges based on the number of invocations and the amount of time required to execute a Lambda function in milliseconds. This pay-per-execution model also helps reduce operating costs during low traffic since Lamba does not make any resource computations when its functions are not being called.

ii) Traditional Model (EC2):

EC2 has an interesting pricing model for EC2, where charges are made according to the number of hours instances are run regardless of their utilization levels. This means that organizations have to continue to pay the running cost of the virtual machines even at times when the system is not fully functional.

Table 3: Cost Comparison Over 24 Hours

Scenario	Average Requests/sec	Total Cost (USD)

Serverless	50	$5.60
Traditional (EC2)	50	$15.00
Serverless	500	$10.20
Traditional (EC2)	500	$15.00

b) Analysis of Cost Efficiency

A similar situation was observed at low traffic, specifically at 50 requests per second. Scale Computing was approximately 3 times more cost-effective. For the serverless deployment, the total cost was $5.60 over 24 hours of usage, while for the traditional EC2 setup, the total cost was $15.00. The following is a proven fact that the serverless function is wholly based on the pay-per-execution structure that does not attract any charges during non-execution. However, the EC2 instances charged regardless of the traffic being received or not, consistently throughout the execution of the experiment.

As the traffic level increased to let the server handle 500 requests per second, serverless computing offered lower costs, although the winning difference from the conventional model was even slimmer. At 500 requests/sec, Serverless cost amounted to 10.20$ whereas EC2 was a fixed 15$. This is evident that even at high loads, it is possible to save money through a serverless approach since the resources are not reserved at a fixed capacity.

c) Effect of idle time on cost

A major benefit of serverless computing is that it does not allow for costs when the server is not in use. Said simple fact still leaves no room for infrastructure cost savings even when its EC2 deployments are inactive since the instances remain functional. This can lead to unnecessary overhead for those applications that are not regularly used or require a short high volume of traffic.

C. Discussion

a) Scalability and Performance Trade-offs

The outcomes have shown that serverless computing outperforms itself in terms of scalability and is indeed more apt at allowing workloads with unpredictable or fluctuating patterns. Such functionality allows for managing functions, changing them based on the need for demand, which in turn helps to avoid excessive resource consumption, as well as minimize latency and cost. Yet, cold-start latency, which can be observed when applications are launched in a serverless environment, can be considered a weakness, especially for applications with high demands for latency. Actors who are managing 'heavyweight' services probably should look at warm-up strategies or provisioned concurrency to keep cold-start issues at bay.

b) Cost Efficiency for Different Workloads

In the economic consideration, serverless satisfies the irregular and idle workloads better than a server-based approach. In some regular traffic applications, perhaps the cost advantage may be slightly reduced but the serverless model is still cheaper compared to conventional practices. While more customizable and providing infrastructure control compared to EC2, the structures employed prior to using ASG introduce higher costs during periods of lower traffic and failure to function without extra effort at scale.

V.CONCLUSION

In this paper, it has been demonstrated that serverless computing accrues tremendous benefits over conventional cloud models, especially concerning the aspects of elasticity, versatility, and affordability. This is because serverless platforms allow for workload scaling needs with ease, depending on the intensity of the workload. Dynamic scaling as a feature ensures that the resources are appropriately chased since, during low use, the cost of resources is also low. However, traditional cloud types like EC2 demand constant configuration of the resources, which leads to constant costs in comparison with idle ones. The results of the experiments proved that serverless performed significantly better than EC2 under high request rates in terms of response time and operation cost. On the other hand, the recorded cold start latency at low levels of site traffic illustrates a problemRow. At which developers must achieve very low latencies, specifically for latency-intensive applications.

While serverless computing may feel somewhat immature today, it is set to become even more tightly interwoven into conventional contemporary cloud-native designs. Along with the evolution of technology, it is expected that problems such as cold start delays, poor management of long-lasting tasks, and overall support of complicated tasks are solved. Along with the increasing usage of event-based and microservices architecture, serverless makes for a valuable innovation driver. Looking to the

future, it is clear that serverless platforms will continue to be the key drivers of ultra-scalable, failure-proof and cost-efficient applications, extending the evolution of cloud infrastructure to the next level.

VI.REFERENCES

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

[2] 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).

[3] Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., & Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In 2020 USENIX annual technical conference (USENIX ATC 20) (pp. 205-218).

[4] 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.

[5] Rani, D., & Ranjan, R. K. (2014). A comparative study of SaaS, PaaS and IaaS in cloud computing. International Journal of Advanced Research in Computer Science and Software Engineering, 4(6).

[6] Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey on serverless computing. Journal of Cloud Computing, 10, 1-29.

[7] McGrath, G., & Brenner, P. R. (2017, June). Serverless computing: Design, implementation, and performance. In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) (pp. 405-410). IEEE.

[8] Rajan, R. A. P. (2018, December). Serverless architecture-a revolution in cloud computing. In 2018 Tenth International Conference on Advanced Computing (ICoAC) (pp. 88-93). IEEE.

[9] Rajan, A. P. (2020). A review on serverless architectures-function as a service (FaaS) in cloud computing. TELKOMNIKA (Telecommunication Computing Electronics and Control), 18(1), 530-537.

[10] Mahmoudi, N., & Khazaei, H. (2020). Performance modeling of serverless computing platforms. IEEE Transactions on Cloud Computing, 10(4), 2834-2847.

[11] Kritikos, K., & Skrzypek, P. (2018, December). A review of serverless frameworks. In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion) (pp. 161-168). IEEE.

[12] Gorelik, E. (2013). Cloud computing models (Doctoral dissertation, Massachusetts Institute of Technology).

[13] Van Eyk, E., Toader, L., Talluri, S., Versluis, L., Uță, A., & Iosup, A. (2018). Serverless is more: From paas to present cloud computing. IEEE Internet Computing, 22(5), 8-17.

[14] Patros, P., Spillner, J., Papadopoulos, A. V., Varghese, B., Rana, O., & Dustdar, S. (2021). Toward sustainable serverless computing. IEEE Internet Computing, 25(6), 42-50.

[15] Bakshi, K. (2017, March). Microservices-based software architecture and approaches. In 2017 IEEE aerospace conference (pp. 1-8). IEEE.

[16] Ishakian, V., Muthusamy, V., & Slominski, A. (2018, April). Serving deep learning models in a serverless platform. In 2018 IEEE International Conference On Cloud Engineering (IC2E) (pp. 257-262). IEEE.

[17] Vahidinia, P., Farahani, B., & Aliee, F. S. (2020, August). Cold start in serverless computing: Current trends and mitigation strategies. In 2020 International Conference on Omni-layer Intelligent Systems (COINS) (pp. 1-7). IEEE.

[18] Ghosh, R., Naik, V. K., & Trivedi, K. S. (2011, June). Power-performance trade-offs in IaaS cloud: A scalable analytic approach. In 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W) (pp. 152-157). IEEE.

[19] Aslanpour, M. S., Toosi, A. N., Cicconetti, C., Javadi, B., Sbarski, P., Taibi, D., ... & Dustdar, S. (2021, February). Serverless edge computing: vision and challenges. In Proceedings of the 2021 Australasian Computer Science Week multiconference (pp. 1-10).