Journal of Current Research in Engineering and Science
Bi-Annual Online Journal (ISSN: 2581 - 611X)
VOLUME:7, ISSUE:2, PAPER NUMBER:45

Architectural Design and Performance Evaluation of Function as a Service and Infrastructure as a Service

Adigopula Sri Sai Rohith Ganesh S[1], Naresh Vurukonda A[2], Kodamanchili Sri Venkat Anand S[3,] Dokuparthi Nilesh S[4], Pedaprolu Koundinya S[5]

[1,3,4,5] Students, and [2] Faculty
Dept. of Computer Science and Engineering,
(of Affiliation)
Koneru Lakshmaiah Education Foundation,
Vaddeswaram, AP, India.
rohith.ganesh111@gmail.com

Abstract: *The dynamic environment of cloud computing places significant emphasis on the architectural design and performance evaluation of serverless computing services. The complex design ideas that underpin serverless systems are examined in this paper, along with how they affect resource usage, scalability, and adaptability. The assessment forms a thorough analysis of critical performance indicators, such as resource efficiency, throughput, and response time. This research attempts to shed light on the best design decisions for particular use cases by carefully examining various serverless frameworks and deployment models. By aiding practitioners and researchers in making well-informed decisions on the deployment and optimization of serverless applications in practical settings, the findings add to the current conversation about perfecting the efficacy and efficiency of serverless computing.*

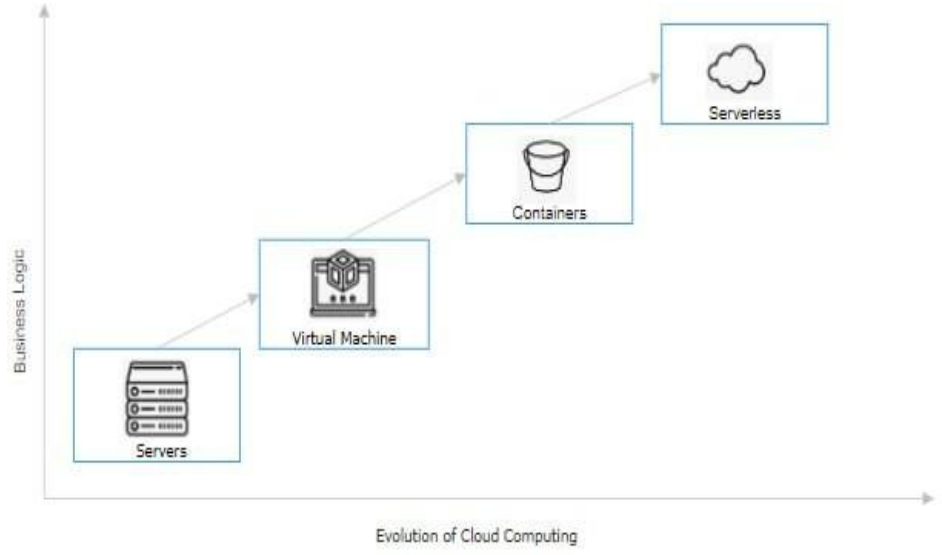
Keywords: *Serverless Computing, Architectural Design, Performance Evaluation, Cloud Computing, Serverless Architectures, AWS, IBM Cloud, GCP, Alibaba Cloud, Azure.*

1. INTRODUCTION:

In the rapidly evolving field of cloud computing, serverless computing has emerged as a revolutionary idea that offers increased resource efficiency, scalability, and flexibility. The key areas of serverless computing services architecture design and performance evaluation are examined in this work. By examining the intricate design principles of serverless architectures, the study aims to ascertain how these systems impact critical performance metrics like as throughput, response time, and resource usage. By means of an extensive evaluation of many serverless frameworks and deployment methodologies, this research endeavours to provide noteworthy perspectives for scholars and professionals. The ultimate objective is to contribute to the continuing discussion on the effective and efficient use of serverless computing resources by providing guidance for decision-making when deploying and improving serverless applications across a variety of real-world scenarios.

1.1 CATEGORIZATION OF CLOUD COMPUTING SERVICES WITH RESPECT TO SERVICE MODELS:

Within cloud computing, services are grouped according to their service models, with each providing a unique method of handling and running code. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Function as a Service (FaaS) are the three main service models.



- Infrastructure as a Service (IaaS): Provides virtualized computing infrastructure over the cloud. Users have control over the operating system, applications, and runtime.

- Platform as a Service (PaaS): Offers a more abstracted environment, managing the underlying infrastructure and allowing users to focus on application development. Provides a platform with predefined services and runtime environments.

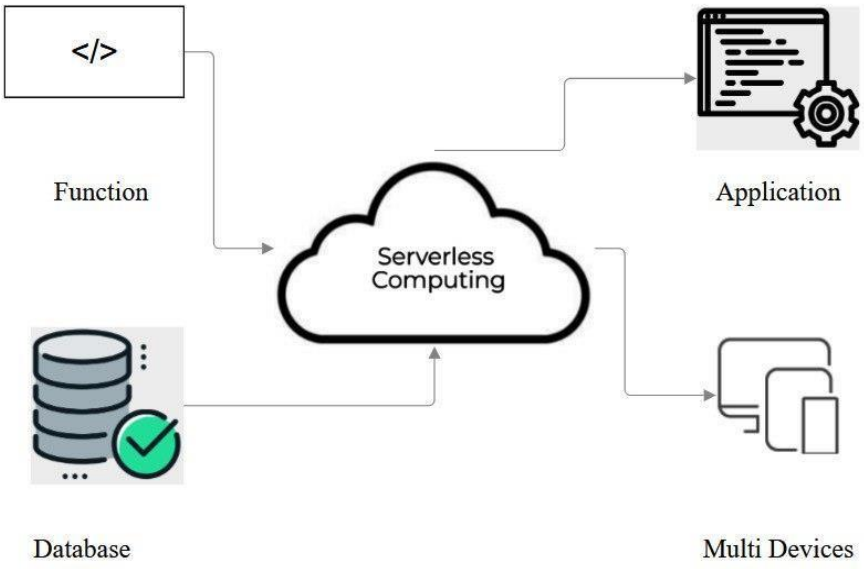
- Function as a Service (FaaS): Focuses on running certain programs or sections of code in reaction to certain situations. granular execution model: without controlling the underlying infrastructure, functions are activated by certain events. Perfect for apps that rely on events and microservices architecture. In the realm of serverless computing, Function as a Service (FaaS) distinguishes itself by providing an event-triggered, fine-grained execution architecture. Developers create discrete functions in Function-as-a-Service (FaaS) that are called upon by predefined events, like data modifications, user activities, or external triggers. This method frees developers from having to provide infrastructure on a continual basis, allowing them to focus only on code logic. FaaS improves scalability, enabling applications to grow in response to demand in an effective manner. AWS Lambda, and Google Cloud Functions are well-known FaaS providers.

1.2 DEEP DIVE INTO SERVERLESS COMPUTING:

Diving deep into the realm of serverless computing unveils the distinctive features and transformative impact of Function as a Service (FaaS). Focuses on running certain programs or sections of code in reaction to certain situations. granular execution model: without controlling the

underlying infrastructure, functions are activated by certain events. Perfect for apps that rely on events and microservices architecture. In the realm of serverless computing, Function as a Service (FaaS) distinguishes itself by providing an event-triggered, fine-grained execution architecture. Developers create discrete functions in Function-as-a-Service (FaaS) that are called upon by predefined events, like data modifications, user activities, or external triggers. This method frees developers from having to provide infrastructure on a continual basis, allowing them to focus only on code logic.

"Serverless Computing: Shaping the Future Landscape of the Internet"



FaaS improves scalability, enabling applications to grow in response to demand in an effective manner.AWS Lambda, and Google Cloud Functions are well-known FaaS providers. The serverless world has been permanently altered by Function as a Service's rise to prominence. Its impact goes beyond simple technology advancement; it encourages the creation of applications that are nimble, effective, and scalable. The serverless paradigm is expanding as developers continue to appreciate the beauty of FaaS, pushing the limits of what is possible in the dynamic and always changing world of contemporary computing.

2. RELATED WORKS:

1. Hall and U. Ramachandran [4] The paper suggests replacing containers in serverless platforms with WebAssembly, which effectively addresses the low latency and multi-tenancy requirements of edge computing. It addresses the unique needs of edge computing by defining serverless access patterns and exploring the viability of WebAssembly through experiments.

2. Garrett McGrath [8] describes a Windows container-based.NET Azure serverless platform that offers significant throughput benefits. It does, however, conspicuously lack a thorough analysis of the specific constraints imposed by its design and only skims the surface in terms of suggested remedies, leaving room for a more thorough investigation of these flaws and possible fixes.

3. Pérez, Alfonso, et al [11] Monolithic systems can be decoupled through the use of emerging architectural patterns like microservices, cloud computing innovations, and the adoption of Linux containers. Yet, language selection and the use of third-party libraries are restricted by serverless computing, as demonstrated by AWS Lambda. In this work, the framework SCAR—which uses Docker images running on AWS Lambda—is introduced. It allows for highly parallel, event-driven serverless applications. Huge image processing and other bursty, stateless

workloads are well suited for SCAR's cache-based optimizations, which lower costs and enable high-throughput computing.

4. Andrei Palade, Aqeel Kazmi and Siobhan Clarke [12] The performance of the four open-source serverless frameworks —Kubeless, Apache OpenWhisk, OpenFaaS, and Knative—in processing IoT-generated data at the edge of the network is the main focus of the paper's evaluation of them in an edge computing context using IoT devices.

5. Daniel Kelly, Frank Glavin, Enda Barrett [13] In-depth discussions of serverless platforms are provided in this paper, with particular attention to AWS Lambda, Google Cloud Functions, Microsoft Azure Functions, and IBM Cloud Functions. It looks into the serverless functions' underlying infrastructure, disclosing CPU specs and utilization data to highlight variations in resource provisioning between these platforms. The study provides insights into function performance factors such as runtime, initialization delays, CPU usage, and disk throughput by profiling virtual machine configurations and analysing the impact of memory allocation. It also looks into abnormalities in function execution over the course of a month-long test period, potentially causing performance interference.

6. Datta, Pubali, et al [15] Serverless Computing enables rapid app prototyping but introduces risks due to third-party functions. Valve, a serverless platform, provides fine-grained control over data flows, addressing security concerns. It allows developers to monitor and restrict function behaviors via network flow auditing, defending against known serverless attack methods.

3. CHARACTERISTICS OF SERVERLESS COMPUTING:

Event-Driven Architecture: Functions are triggered by events, allowing for dynamic reactions to modifications in data, human input, or outside stimuli. Careful design is necessary when orchestrating intricate processes with interrelated operations to avoid problems like race conditions.

No Server Management: Developers are separated from the underlying infrastructure in a serverless design. They don't have to be concerned about maintenance, scaling, or server provisioning. The infrastructure is automatically managed by the cloud provider, freeing developers to concentrate only on creating code.

- Granular Scaling: Functions in serverless systems are automatically scaled in response to demand. Because each function may scale separately, resources can be used effectively. It just takes a few moments to scale, and developers get paid according to real use as opposed to pre-allocated resources.

- Short-lived Functions: Functions in a serverless environment are designed to be short-lived and stateless. They execute in response to events and typically handle a specific task. Long-running processes are not well-suited for a serverless architecture.

- Pay-per-Use Billing: In a serverless context, functions are meant to be transient and stateless. They usually handle a certain duty and act in reaction to circumstances. A serverless design is not ideal for lengthy tasks.

- Automatic Scaling: Functions are automatically scaled by serverless systems in response to events or incoming requests. This automated scaling guarantees that there are sufficient resources to effectively manage the workload.

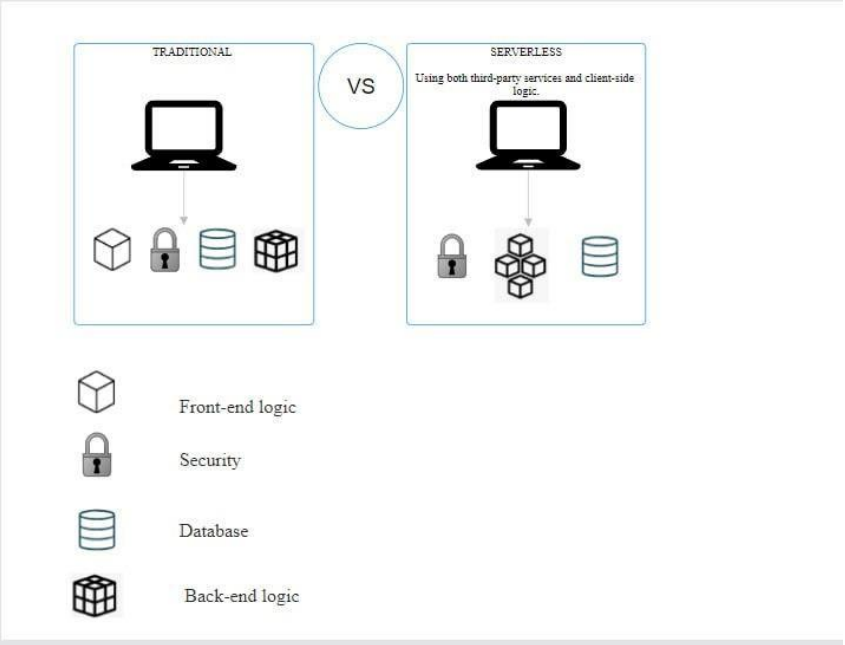
- Microservices Architecture: A microservices design, in which applications are made up of tiny, autonomous, and loosely connected services, frequently fits in well with serverless computing. It is possible to implement each microservice as a stand-alone operation.
- Statelessness: Functions in a serverless environment are designed to be stateless. Any required state or data should be passed in through the function's inputs, and the function should produce its output without relying on external state.

- Vendor-specific Services: A collection of managed services that are readily incorporated into serverless applications is often provided by serverless platforms. These services include of storage, databases, authentication, and other things. Dependence on these services, nevertheless, might result in vendor lock-in.

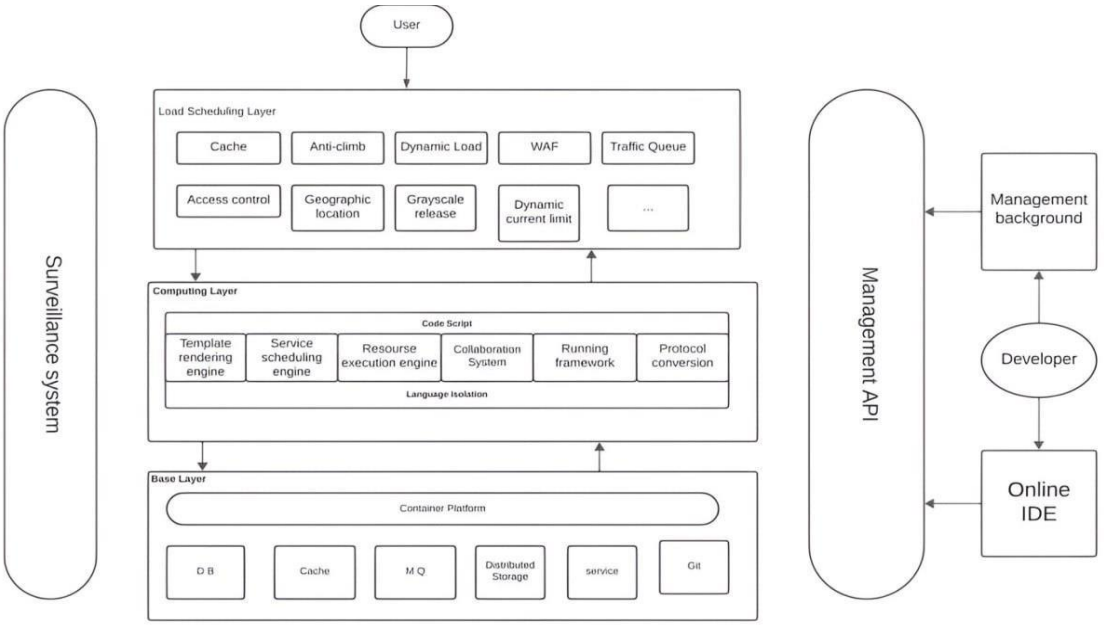
- Rapid Deployment: Platforms that are serverless enable quick deployment and iteration. Updates to certain features may be applied by developers without impacting the overall application, which encourages development agility.

4. ARCHITECTURAL DESIGN OF SERVERLESS COMPUTING

Serverless computing's architectural design is defined by a fundamental change toward an event-driven, function-centric paradigm. Applications are broken down into stateless functions, each of which does a particular job and is called upon by events like HTTP requests or modifications to data. Because resources scale automatically in response to demand, this architecture promotes responsiveness and scalability. In order to reduce the requirement for human infrastructure administration, serverless architecture promotes the usage of third-party managed services for capabilities like databases and storage. Serverless units are even smaller—individual functions—than microservices, allowing for autonomous and modular development.



Modularity is encouraged by the decentralized logic into functions, while consistency is ensured and version control is made easier by the immutable infrastructure. The unifying entry point for external communications is frequently an API gateway. Distributed execution, which is embraced by serverless architecture, allows functions to run worldwide for improved fault tolerance and lower latency.



High Level Overview of Serverless Architecture

The diagram shows a high-level overview of a serverless computing architecture. The main components are:

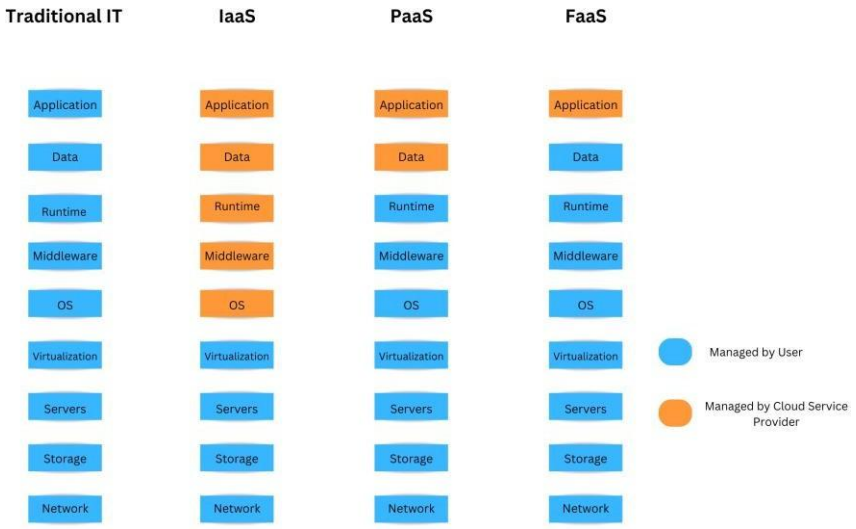
1. Load scheduling layer: This layer is responsible for receiving requests from users and routing them to the appropriate computing resources. It may also include features such as caching, load balancing, and traffic shaping.
2. Computing layer: This layer is responsible for executing user requests. It typically consists of a pool of stateless computing resources, such as serverless functions or containers.
3. Base layer: This layer provides the underlying infrastructure for the serverless computing platform, such as storage, networking, and security services.
4. Cache: Stores frequently accessed data to reduce the load on the computing layer.
5. Anti-climb: Protects against denial-of-service attacks by limiting the number of requests that can be processed from a single IP address.
6. WAF: Web application firewall that protects against common web attacks.
7. Traffic queue: Queues incoming requests to prevent the computing layer from being overwhelmed.
8. Access control: Controls who can access the serverless computing platform and its resources.
9. Geographical location: Distributes computing resources across different geographical regions to improve performance and reliability.
10. Resource execution engine: Executes serverless functions and provides them with access to the necessary resources.
11. Collaboration system: Allows developers to collaborate on serverless functions.
12. Running framework: Provides a framework for executing serverless functions.
13. Protocol conversion: Converts requests from one protocol to another.
14. Language isolation: Isolates serverless functions from each other to prevent conflicts.
15. Online IDE: Integrated development environment for developing and deploying serverless functions.
16. Container platform: Platform for managing and executing serverless function containers.
17. Database: Database for storing data used by serverless functions.
18. Cache: Cache for storing frequently accessed data used by serverless functions.

19. Git: Version control system for managing serverless function code.
20. Management API: API for managing the serverless computing platform and its resources.
21. Surveillance system: Monitors the serverless computing platform for performance and security issues.

This is just a high-level overview, and there are many other components that may be involved in a serverless computing architecture, depending on the specific needs of the application.

5. SERVERLESS ARCHITECTURE VS TRADITIONAL ARCHITECTURE

In numerous important ways, the Function as a Service (FaaS) model is preferable than traditional on-premises services as a paradigm. In contrast to conventional models, which need intensive infrastructure provisioning and maintenance, FaaS relieves developers of these responsibilities so they may concentrate just on code logic. Conventional on-premises services need upfront expenditures and ongoing administration since they often struggle with scalability and resource use. On the other hand, FaaS provides dynamic resource allocation based on demand and intelligent scaling, guaranteeing best-in-class performance at minimal expense.



FaaS's pay-per-use invoicing approach matches expenses to actual consumption, making it a more affordable choice than traditional installations, which are sometimes inflexible and capital-intensive. Furthermore, stateless architecture, seamless interaction with managed services, and the event-triggered execution paradigm of FaaS promote agility and responsiveness, surpassing the sometimes complex and monolithic characteristics of conventional on-premises infrastructures. In conclusion, as compared to traditional on-premises services, the Function as a Service approach is preferable because it provides unmatched flexibility, scalability, cost-efficiency, and a simplified development experience.

6. PERFORMANCE

The basic definition of performance in serverless computing encompasses the efficiency and effectiveness of executing tasks or functions within a serverless environment. Performance in serverless computing is evaluated based on several factors, including latency, scalability, concurrency, resource use, cold start time, throughput, reliability, and cost efficiency. These aspects define the performance of serverless computing in various cloud-based services. In depth, performance in serverless computing revolves around ensuring fast, scalable, reliable, and cost-effective execution of functions or tasks in response to varying workloads and demands. Different serverless platforms might excel in distinct aspects of performance. So, these factors help us to know the perfect definition of performance.

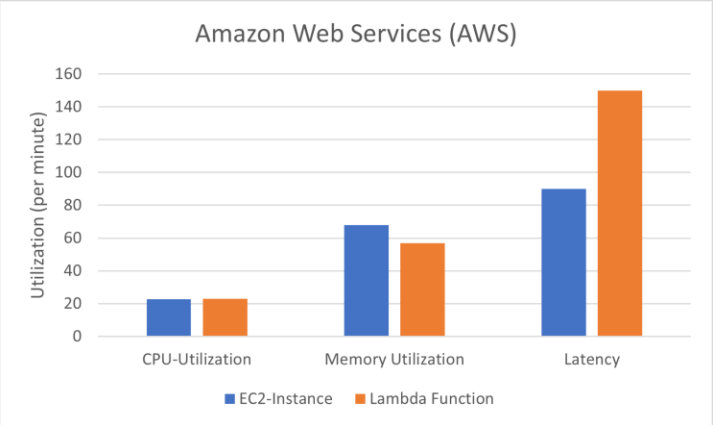
Our goal in this paper is to perform a thorough analysis of latency, memory usage, and CPU use across a range of cloud computing services offered by significant market participants. We will specifically compare the performance of traditional Infrastructure as a Service (IaaS) solutions with Function as a Service (FaaS) offerings.

Elastic Compute Service and Function Compute from Alibaba Cloud, Cloud Functions and Compute Engine from GCP, Azure Functions and Virtual Machines from Microsoft Azure, IBM Virtual Server, IBM Cloud Compute Engine, and Lambda and EC2 instances from AWS will all be carefully examined.

1. Amazon Web Services (AWS)

- Amazon EC2 Instance: The performance of an EC2 instance is largely dependent on the workload it oversees. Memory-intensive activities require RAM, CPU-bound procedures cause processor strain, and I/O-heavy workloads put stress on the network and storage. Based on your workload, select the right instance size and type for the best outcomes.

- AWS Lambda: Amazon Web Services offers a serverless computing solution called AWS Lambda. Numerous variables, including concurrent executions, memory allocation, and code optimization, affect its speed. Lambdas often execute in milliseconds, scale automatically, and have low latency. Performance, however, might differ depending on configuration decisions made during development and workload. For small-to-medium workload applications that are event-driven, Lambda's performance is typically quick and effective; however, further speed optimization can increase its effectiveness. Lambda charges just for the compute time consumed, running on a pay-as-you-go basis. Because you're not paying for unused resources, this might result in cost savings, particularly for functions that are only sometimes used.

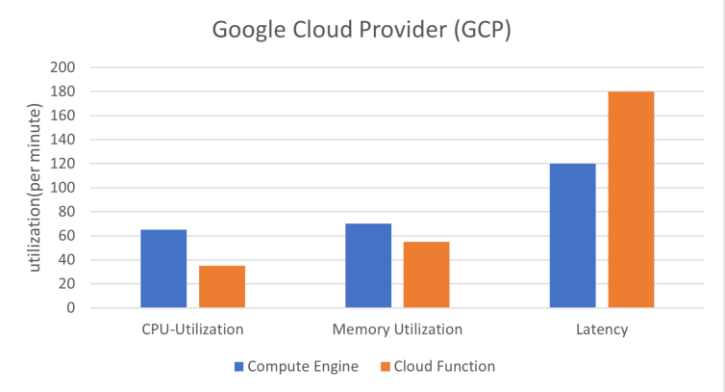


Deployment Model	Service Name	CPU Utilization	Memory Utilization	Latency
IAAS	EC2-Instance	22.8	68	90
FAAS	AWS Lambda	22.84	57	150

While Lambda functions shine in resource efficiency, consuming significantly less CPU and memory than EC2 instances, this advantage comes with a trade-off: higher latency. This makes FaaS ideal for cost-effective, event-driven workloads tolerant of some delay, while IaaS, with its lower latency and dedicated resources, remains the better choice for demanding, latency-sensitive applications. Ultimately, the optimal choice depends on the specific performance requirements and nature of your workloads.

2. Google Cloud Function (GCP)

- Google Compute Engine: The main infrastructure part in Google Cloud Platform, Google Compute Engine (GCE), allows for the construction and control of virtual machines (VMs) globally. It improves workflow efficiency by being integrated with GCP services like Big Query and Google Kubernetes Engine. GCE optimizes prices via its pay-as-you-go strategy, sustained use discounts, and preemptible virtual machines (VMs) while prioritizing security with identity management, encryption, and network isolation.
- Google Cloud Function: Event-driven, scalable functions without infrastructure maintenance are made possible by Google Cloud Functions, a part of GCP. Code is the only thing that developers think about when GCP services or HTTP requests are used for data processing and automation. With support for Go, Python, and Node.js, it provides linguistic freedom. Charging per execution, autoscaling maximizes resource use. When combined with GCP services, it guarantees reliable monitoring and quick deployment for microservices and serverless applications.



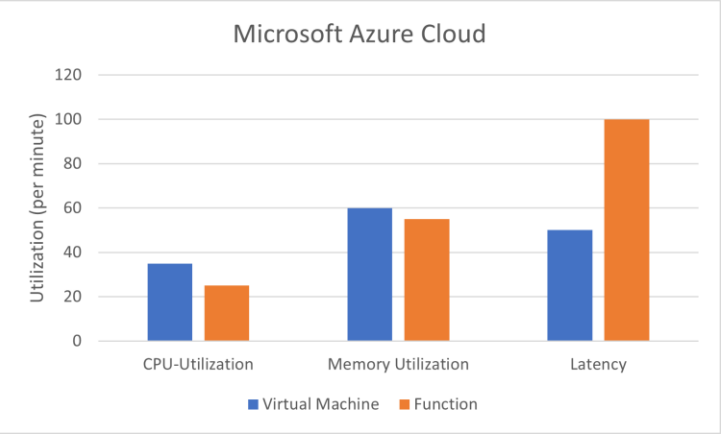
Deployment Model	Service Name	CPU Utilization	Memory Utilization	Latency
IAAS	Compute Engine	65	70	120
FAAS	Cloud Function	35	55	180

The graph proves that, on average, CPU use is stable and low at 20%. Additionally, the memory use is consistent and low, averaging about 30%. Of the three measurements, latency varies the most, although it is still rather low, averaging about 40 milliseconds. In general, the graph shows that the GCP is running efficiently and is not experiencing significant stress. There is slight delay, minimal CPU and memory consumption, and minimal latency. This shows that the GCP can manage the workload that is assigned to it without experiencing any issues.

3. Microsoft Azure:

- Virtual Machine: Microsoft's Azure Cloud's Azure Virtual Machines provide scalable, adaptable computing resources that support a wide range of operating systems and languages. They ease the smooth transition of various applications, offering adaptable setups for varying workloads and on-demand scalability to achieve peak efficiency.

- Azure Functions: Azure Functions, a serverless computing feature of Microsoft Azure, transforms the process of creating apps by doing away with the need for infrastructure management. It enables triggered functions for events such as timers or HTTP requests and supports C#, JavaScript, and Python. It perfects costs by charging based on actual resource usage and execution time.

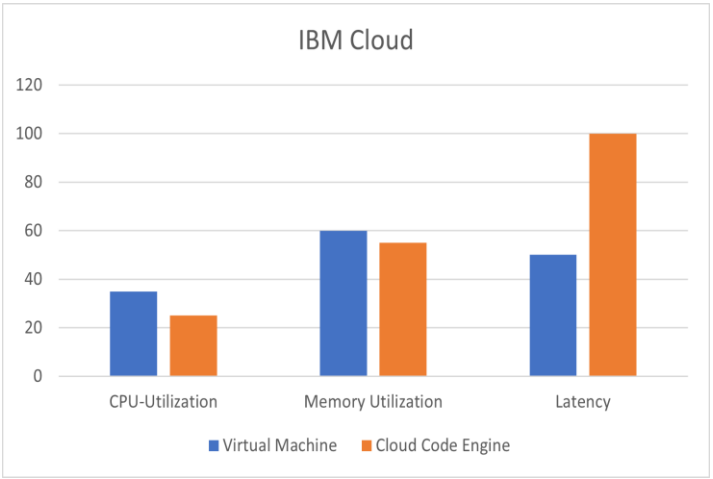


Deployment Model	Service Name	CPU Utilization	Memory Utilization	Latency
IAAS	Virtual Machine	35	60	50
FAAS	Azure Function	25	55	100

The graph displays a virtual machine's and a function's CPU, memory, and latency usage in Microsoft Azure Cloud. "Utilization (per minute)" is the label on the Y-axis, which has a range of 0 to 120. It appears like the X-axis displays time in minutes even though it is not labelled. The virtual machine's CPU utilization is constantly greater than its memory utilization. For the first six minutes, it hovers between 60% and 80% after starting at roughly 50%. After then, it falls to about 20% and stays there for the duration of the graph. The virtual machine's memory use begins at about 20% and varies for the first six minutes between 20% and 30%. After then, it falls to about 10% and stays there for the duration of the graph. For the whole graph, the virtual machine's latency is comparatively small and consistent, averaging 5 milliseconds.

4. IBM Cloud

- Virtual Server: A flexible cloud computing solution, IBM Virtual Server provides scalable infrastructure for a range of workloads. Users can adjust the CPU, memory, storage, and networking options in this customizable virtual machine environment. It supports a wide range of programs and is compatible with Linux and Windows. The platform prioritizes dependability by using load balancing and automated scaling to keep high availability.
- IBM Cloud Code Engine: IBM Cloud Code Engine uses serverless computing to simplify the development and deployment of apps. It manages orchestration and scaling of infrastructure tasks and supports microservices and containerized applications. Because it supports multiple languages and integrates with other IBM Cloud services for databases and AI tools, developers can concentrate on writing code while it takes care of the rest.

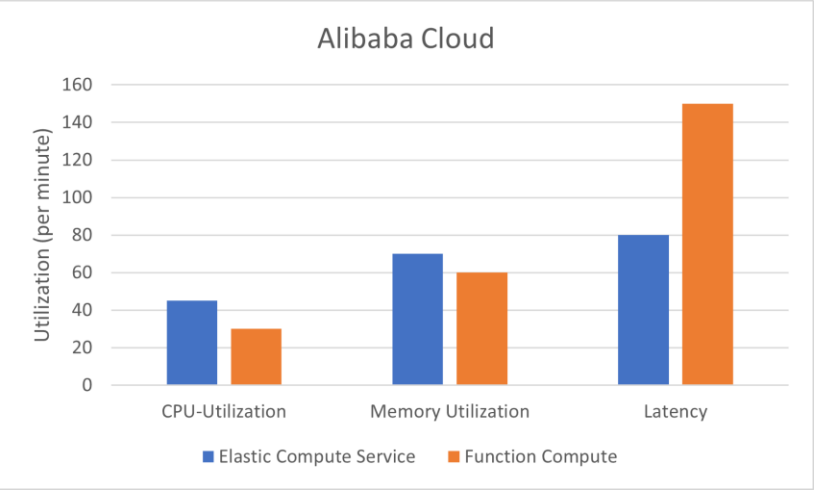


Deployment Model	Service Name	CPU Utilization	Memory Utilization	Latency
IAAS	IBM Virtual Server	45	65	50
FAAS	IBM Cloud Compute Engine	35	55	125

The graph indicates that, on average, the virtual machine's CPU consumption is higher than the cloud code engines. Additionally, the virtual machine typically uses more memory than the cloud code engine. Compared to the cloud code engine, the virtual machine has less latency. Generally speaking, cloud code engines require less resources than virtual machines. This is due to the fact that virtual machines are capable of running an entire operating system along with all installed apps. In contrast, code that has been specifically deployed to cloud code engines is the only code that they run. It's possible that the virtual machine in the graph is using more code than the cloud code engine. The virtual machine would use more CPU and memory as a result. Code that is more latency-sensitive than the code running on the virtual machine might be executed by the cloud code engine in the graph. The cloud code engine would experience increased latency as a result.

5. Alibaba Cloud

- Elastic Compute Service: Elastic Compute Service (ECS), a key part of Alibaba Cloud's cloud infrastructure, provides flexible virtual computing capabilities for a range of business requirements. With the ability to customize CPU, memory, storage, and network settings, ECS offers on-demand instances that enable customers to create bespoke virtual machines. With support for Linux and Windows, it guarantees wide application compatibility. Because of its auto-scaling feature, snapshot backups, and secure networking, ECS offers excellent availability, reliability, and security.
- Function Compute: Alibaba's Function Compute is revolutionizing the app development industry by providing a smooth experience that frees users up to focus exclusively on coding. It allows developers to create scalable event-driven functions in a variety of programming languages without having to worry about infrastructure by managing infrastructure management in a seamless manner. Its dynamic nature makes it easy to adjust to changing workloads, guaranteeing best performance for data processing and web applications. This flexibility makes the process of creating apps more efficient, which makes it possible to deploy apps in a serverless environment with ease.



Deployment Model	Service Name	CPU Utilization	Memory Utilization	Latency
IAAS	Elastic Compute Service	45	70	80
FAAS	Function Compute	25	55	100

The green "Function Compute" line has an average CPU utilization of 20%, but the blue "Elastic Compute Service" line constantly displays a higher CPU utilization of about 60%. This indicates that compared to the FC, the ECS is consuming more processing power. Both lines closely resemble each other in terms of memory use, with ECS slightly exceeding FC on average. This implies that the memory requirements of the two services are comparable. The millisecond-based latency, displayed on the rightmost Y-axis, is consistently lower for ECS than for FC. This indicates that compared to the FC, the ECS can handle requests faster. In general, the graph indicates that ECS requires more resources than FC. It has a marginally higher memory utilization and greater CPU power consumption, but it also has a reduced latency. FC, on the other hand, has a higher latency but is a lighter service that needs less memory and CPU.

7. LIMITATIONS

While serverless computing, and particularly the Function as a Service (FaaS) model, offers numerous advantages, it is essential to acknowledge certain limitations that may impact its suitability for certain use cases:

- **Cold Start Latency**: A significant limitation in serverless computing is the "cold start" problem. When a function is invoked, there can be a delay as the cloud provider initializes the execution environment. This latency can be critical for applications requiring near-instantaneous responses.
- **Execution Duration Limits**: Serverless platforms impose execution duration limits on functions. Long-running tasks may face challenges within this model, and developers need to design functions to adhere to these constraints or consider alternative solutions.
- **Limited Execution Environments**: Serverless platforms constrain the choice of execution environments and available resources, potentially limiting compatibility with certain programming languages, libraries, or dependencies.
- **Vendor Lock-In**: Adopting serverless computing may lead to vendor lock-in, as each cloud provider has its proprietary serverless platform. Migrating functions between providers can be non-trivial, impacting long-term flexibility and scalability.
- **Debugging Challenges**: Debugging and troubleshooting serverless functions, particularly in production environments, can be challenging due to limited control over the underlying infrastructure.
- **Cost Predictability**: While the pay-per-use model is generally cost-effective, predicting costs can be challenging, especially when dealing with unpredictable spikes in function executions or complex application architectures.
- **Security Concerns**: Serverless computing introduces new security considerations, such as the need to carefully manage permissions, secure function inputs and outputs, and address potential vulnerabilities in third-party dependencies.

Despite these limitations, it's important to note that ongoing advancements in serverless technologies, coupled with evolving best practices, are continuously mitigating many of these challenges. Careful consideration of application requirements and a strategic approach to design and implementation can help leverage the benefits of serverless computing while managing its limitations effectively.

8. CONCLUSION

In conclusion, the exploration of serverless computing, with a focus on the Function as a Service (FaaS) model, reveals a transformative paradigm that redefines the landscape of modern application development. The architectural design of serverless computing, characterized by fine-grained functions, event-driven models, and automatic scaling, offers unprecedented agility and scalability. Comparing this to traditional on-premises services underscores the superiority of serverless computing, particularly in terms of flexibility, cost efficiency, and streamlined development.

However, it is crucial to acknowledge the inherent limitations, such as cold start latency, execution duration constraints, and potential vendor lock-in, which necessitate thoughtful consideration in application design. Despite these challenges, the trajectory of serverless computing is one of continual evolution, with ongoing advancements addressing limitations and expanding the realm of possibilities. As organizations increasingly prioritize agility, scalability, and cost-effectiveness, serverless computing emerges as a pivotal force shaping the future of cloud-native application architectures.

9. REFERENCES

[1] Armbrust, Michael, et al. Above the clouds: A berkeley view of cloud computing. Vol. 17. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[2] Van Eyk, Erwin, et al. "A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures." Companion of the 2018 acm/spec international conference on performance engineering. 2018.

[3] Van Eyk, Erwin, et al. "Serverless is more: From paas to present cloud computing." IEEE Internet Computing 22.5 (2018): 8-17.

[4] Hall, Adam, and Umakishore Ramachandran. "An execution model for serverless functions at the edge." Proceedings of the International Conference on Internet of Things Design and Implementation. 2019.

[5] Franz, Justin, et al. "Reunifying families after a disaster via serverless computing and raspberry pis." 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). IEEE, 2018.

[6] Adzic, Gojko, and Robert Chatley. "Serverless computing: economic and architectural impact." Proceedings of the 2017 11th joint meeting on foundations of software engineering. 2017.

[7] Lowery, C. "Emerging technology analysis: Serverless computing and function platform as a service." Gartner Research: Stamford, CT, USA (2016).

[8] McGrath, Garrett, and Paul R. Brenner. "Serverless computing: Design, implementation, and performance." 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2017.

[9] King, Martin. "Breaking the server and data communications barrier with serverless guaranteed quality of service (GQoS) compliant communications." Proceedings First International Conference on Peer-to-Peer Computing. IEEE, 2001.

[10] Lynn, Theo, et al. "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms." 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2017.

[11] Pérez, Alfonso, et al. "Serverless computing for container-based architectures." Future Generation Computer Systems 83 (2018): 50-59.

[12] Palade, Andrei, Aqeel Kazmi, and Siobhán Clarke. "An evaluation of open source serverless computing frameworks support at the edge." 2019 IEEE World Congress on Services (SERVICES). Vol. 2642. IEEE, 2019.

[13] Kelly, Daniel, Frank Glavin, and Enda Barrett. "Serverless computing: Behind the scenes of major platforms." 2020 IEEE 13th International Conference on Cloud Computing (CLOUD). IEEE, 2020.

[14] Stigler, Maddie, and Maddie Stigler. "Understanding serverless computing." Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud (2018): 1-14.

[15] Datta, Pubali, et al. "Valve: Securing function workflows on serverless computing platforms." Proceedings of The Web Conference 2020. 2020.

[16] Cloud, G. "Google Cloud Functions." Google Cloud,[Online]. Available: https://cloud. google. com/functions/.[Accessed 25 6 2018] (2020).

[17] Eismann, Simon, et al. "A case study on the stability of performance tests for serverless applications." Journal of Systems and Software 189 (2022): 111294.

[18] Stigler, Maddie, and Maddie Stigler. "Understanding serverless computing." Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud (2018): 1-14.