National Conference on Information Technology & Digital Application (NCITSA-2025)
International Journal of Advanced Engineering, Management and Science (IJAEMS)
Special Issue
https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

Serverless Computing Performance Evaluation

Ms. Preeti Sharma[1] , Mihir Kumar[2], Kalpit Chouthwani[3]

[1]Assistant Professor Department of Information Technology,Jaipur Engineering College & Research Centre, Jaipur
[2]Student Department of Information Technology, Jaipur Engineering College and Research Centre, Jaipur
Email: mihirkumar.it25@jecrc.ac.in
[2]Student Department of Information Technology, Jaipur Engineering College and Research Centre, Jaipur
Email :kalpitchouthwani.it25@jecrc.ac.in

Abstract—Serverless computing, characterized by its Function-as-a-Service (FaaS) model, offers scalable and cost-effective solutions for deploying applications. However, its performance evaluation reveals a critical challenge: the cold start problem. This issue arises when serverless platforms initialize new function instances to handle requests, leading to noticeable delays, especially in latency-sensitive applications. Cold starts are influenced by factors such as runtime environment, deployment package size, memory allocation, and inactivity periods. They degrade user experience, reduce throughput, and hinder real-time application performance.

Mitigating the cold start problem involves optimizing runtime configurations, reducing deployment package size, and adopting lightweight runtimes like Node.js or Python. Strategies such as provisioning concurrency, warming up functions through periodic invocations, and using edge computing platforms further minimize latency. Additionally, lazy initialization and dependency optimization help streamline function startup.

The evaluation process also includes monitoring metrics like execution time, scalability, and resource utilization to ensure efficiency. Combining serverless with alternative architectures, such as microservices or container-based solutions, can address cold starts in specific scenarios. Addressing these challenges will enhance the reliability and responsiveness of serverless applications, ensuring their viability for diverse use cases.

1. INTRODUCTION

1.1 Cloud Computing

Cloud computing has revolutionized the way businesses and individuals' access, store, and process data by providing on-demand availability of computing resources over the internet. Unlike traditional on-premises computing, where organizations had to invest heavily in infrastructure, cloud computing offers a scalable and cost-efficient alternative. It allows users to leverage shared resources, such as servers, storage, databases, and software, without the need to own or maintain the underlying hardware.

The evolution of cloud computing can be traced back to the late 1990s, but it gained significant momentum in the mid-2000s with the introduction of major platforms like Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. These platforms offer a wide range of services categorized as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). This layered approach enables users to select the level of control and customization that best suits their needs, from basic infrastructure to fully managed applications.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

Cloud computing provides numerous benefits, including enhanced scalability, flexibility, and cost optimization. Organizations can quickly scale resources up or down based on demand, pay only for what they use, and reduce capital expenditures. Additionally, it enables global access to data and applications, promoting collaboration and innovation. Cloud-native technologies, such as containers, microservices, and serverless computing, further enhance its utility by enabling rapid application development and deployment. Despite its advantages, cloud computing comes with challenges that must be addressed to maximize its potential. Concerns such as data security, privacy, compliance with regulatory requirements, and potential vendor lock-in are critical considerations. Performance evaluation, including metrics like uptime, latency, and resource efficiency, is essential to ensure the reliability and effectiveness of cloud services.

Emerging trends, such as hybrid and multi-cloud strategies, edge computing, and artificial intelligence integration, are shaping the future of cloud computing. These innovations aim to address

existing limitations while expanding the scope of applications. As cloud computing continues to evolve, it remains a cornerstone of digital transformation, empowering organizations to innovate and adapt in an increasingly dynamic technological landscape. Understanding and addressing its inherent challenges will ensure its sustained growth and adoption across industries.

1.2 Serverless Computing and Function as a Service (FaaS)

Serverless computing is a cloud computing model that abstracts infrastructure management from developers, enabling them to focus solely on writing and deploying code. Unlike traditional architectures, where servers must be provisioned, scaled, and maintained, serverless computing dynamically allocates resources and scales applications automatically in response to demand. This model eliminates the need for server management and operates on a pay-as-you-go basis, where users are charged only for the execution time of their code, making it highly cost-efficient.

At the core of serverless computing lies **Function as a Service (FaaS)**, a paradigm that allows developers to run individual functions in response to specific events. Functions in FaaS are stateless, event-driven pieces of code that execute in isolated containers. When an event, such as an HTTP request or a database update, triggers a function, the cloud provider spins up a container, executes the function, and then deallocates the resources once the task is complete. This model ensures rapid scalability, efficient resource utilization, and minimal overhead.

Key Features of Serverless Computing and FaaS:

1. **Event-Driven Architecture**: Functions are triggered by predefined events, such as API calls, file uploads, or database changes.
2. **Automatic Scaling**: Functions scale seamlessly to handle varying workloads without manual intervention.
3. **Cost Efficiency**: Charges are based only on execution time and resource consumption, with no cost for idle time.
4. **Language Agnosticism**: Functions can be written in various programming languages supported by the

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

provider.

5. **Reduced Operational Complexity**: Developers do not need to manage or patch servers, focusing entirely on application logic.

Applications of Serverless and FaaS:

- **Microservices**: Breaking down applications into smaller, independently deployable components.
- **Real-Time Data Processing**: Handling streams of data, such as IoT sensor data or log analysis.
- **Event-Driven Workflows**: Automating tasks like image processing, notifications.
- **API Backends**: Implementing lightweight, scalable RESTful APIs.

Despite its benefits, serverless computing and FaaS face challenges, such as the **cold start problem**, where initializing a function instance introduces latency. Additionally, execution time limits, stateless constraints, and potential vendor lock-in require careful planning.

By addressing these limitations and leveraging its strengths, serverless computing and FaaS continue to redefine the future of cloud-native application development, offering unparalleled scalability, flexibility, and efficiency.

1.3 Performance Evaluation

Performance evaluation, often referred to as performance benchmarking or performance testing, is the systematic process of assessing the performance characteristics of computing systems and applications. This involves measuring key features, such as latency, throughput, scalability, and resource utilization, under controlled and real-world scenarios. Performance evaluation is critical in understanding how systems respond to varying workloads and identifying bottlenecks that could impact efficiency or user experience.

At its core, performance evaluation involves several essential components:

1. **System Under Test (SUT)**: The environment or component being evaluated. For instance, in serverless computing, the SUT could be a Function-as-a-Service (FaaS) platform, such as AWS Lambda or Google Cloud Functions. Metrics like response time and cold start latency are commonly analyzed.
2. **Workload**: The stimuli applied to the SUT to gauge performance. Workloads can be:
    - **Synthetic**: Designed for micro-benchmarks to evaluate specific features, such as CPU or memory usage.
    - **Realistic**: Designed for application benchmarks to mimic real-world scenarios, such as processing API requests or running business logic.
3. **Benchmark**: A controlled experiment that measures performance by applying a workload to the SUT. It evaluates predefined metrics, such as execution time, resource consumption, or scalability.
4. **Benchmark Suite**: A collection of related benchmarks executed under a unified methodology to provide

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

comprehensive insights into system performance.

Key performance features include latency (time taken to respond to a request), throughput (number of requests processed per unit time), and resource efficiency (optimal utilization of CPU, memory, and storage). Evaluation methods range from experimental setups using simulated traffic to real-world deployments for observing performance under actual usage conditions.

In serverless computing, performance evaluation is particularly important due to unique challenges like the cold start problem, which impacts latency, and scaling dynamics that affect throughput. These evaluations help identify optimization strategies, such as reducing function initialization times or enhancing concurrency handling.

By systematically analyzing performance, organizations can ensure that their systems meet reliability, scalability, and efficiency requirements. This process not only validates the design of a system but also provides actionable insights for improving future deployments and applications.

1.4 Micro- and Application-Benchmarks in Performance Evaluation

Performance evaluation utilizes benchmarks to assess the efficiency and responsiveness of computing systems. These benchmarks fall into two primary categories: **micro-benchmarks** and **application-benchmarks**. Each serves a distinct purpose and focuses on different aspects of system performance.

Micro-Benchmarks

Micro-benchmarks are designed to evaluate specific performance attributes of a system, such as CPU speed, memory access latency, or floating-point computational efficiency. These tests employ synthetic workloads to isolate and measure individual components or resources within an execution environment. Because they focus narrowly on particular features, micro-benchmarks are not domain-specific and can often provide transferable insights. For instance, the floating-point performance of a virtual machine instance type can inform decisions about suitable workloads across multiple applications. Micro-benchmarks are typically lightweight, easy to configure, and well-suited for identifying bottlenecks in isolated scenarios.

Application-Benchmarks

In contrast, application-benchmarks, also referred to as macro-benchmarks, aim to evaluate the overall performance of real-world application scenarios. These benchmarks simulate complex, dynamic workloads, reflecting actual use cases such as web serving, data processing, or scientific computing. Metrics such as end-to-end response time and throughput are key indicators of performance in application-benchmarks. However, their broad scope often leads to increased complexity, longer execution times, and intricate configurations. Unlike micro-benchmarks, application-benchmarks are tightly coupled with the domain or application being tested, offering results specific to the workload and operational context.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

For **serverless applications**, the asynchronous nature of workflows adds complexity to performance evaluation. While metrics like client-side response time are straightforward for synchronous applications, serverless applications often require distributed tracing to account for latency across external services and asynchronous boundaries. Distributed tracing provides full observability by capturing end-to-end latency, enabling a comprehensive view of the interactions among system components.

Both micro- and application-benchmarks play critical roles in performance evaluation. Micro-benchmarks excel at diagnosing specific resource constraints, while application-benchmarks provide insights into system behavior under realistic workloads. Together, they form a comprehensive framework for understanding and optimizing the performance of modern computing systems, including cloud and serverless architectures.

1.5 Distributed Tracing

Distributed tracing is a powerful technique for achieving **end-to-end observability** of requests as they flow through distributed systems, allowing organizations to monitor and diagnose the performance of applications composed of multiple interconnected services. The concept of distributed tracing was first introduced in 1994 by Schwarz and Mattern, who developed models to detect causal relationships in distributed systems. The technique became more widely adopted in the early 2000s, with solutions such as **Magpie** and **X-Trace** emerging. However, it was Google's **Dapper** that significantly popularized distributed tracing, and today, the practice is commonly used by many organizations across industries to enhance system reliability and performance monitoring.

The core idea of distributed tracing is to track a request as it travels through multiple services or functions, providing insights into each component's performance and how they contribute to the overall request lifecycle. A typical trace involves several components, starting from the service that receives the incoming request (e.g., Service1), passing through intermediary functions or services (e.g., Function1), and ultimately ending in the final service (e.g., Service2).

To implement distributed tracing, when Service1 receives a request, it generates a **unique tracing token** for the request. This token is passed along with the request to each downstream service or function involved in processing the request. Each service or function then records timestamps at important trace points, marking the start and end of specific operations. These timestamps are grouped into **trace spans**, which represent individual operations within the same service or component. A **centralized tracing service** collects these trace spans from various components and correlates them using the tracing token, ultimately creating a trace graph that visualizes the causal relationships between operations across the distributed system.

For example, a simplified causal-time trace diagram for a synchronous invocation might look like the following:

1. **Service1** generates the tracing token.
2. The tracing token is passed to **Function1**.
3. **Function1** processes the request and passes it to **Service2**.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

4. Each service and function record the necessary timestamps at trace points, and the centralized tracing service correlates these into a full trace, showing the **causal relationships** and how long each operation took.

The trace provides valuable performance data such as latency, throughput, and resource utilization, helping engineers pinpoint bottlenecks, errors, and areas for optimization in the system.

Distributed tracing has become essential in modern distributed applications, especially those using microservices or serverless architectures, as it allows teams to monitor system behavior at a granular level and ensure optimal performance and reliability.

1) *1.6 Reproducibility in Cloud Experimentation*

Reproducibility is a fundamental principle in scientific research, emphasizing the need to achieve

consistent results under different conditions. In the context of computer science, the terms **repeatability** and **reproducibility** are central to validating experimental findings. **Repeatability** refers to the ability to obtain the same results when conducting the same experiment using the same method and conditions. However, research has shown that repeatability is often compromised, particularly in fields like computer science, where issues such as code availability and build failures hinder the ability to replicate studies. A study found that over half of the papers from top-rated ACM systems conferences lacked functional code, making repeatability impossible even after extensive efforts to fix build failures.

In contrast, **reproducibility** involves achieving the same results using the same method but under different conditions. This is often seen as more achievable than repeatability, especially in public cloud environments, where researchers lack control over the underlying infrastructure due to the multi-tenant nature of cloud services. Reproducibility, however, still presents significant challenges in cloud experimentation. The rapid evolution of cloud environments, along with the inherent complexity and variability of cloud-based infrastructure, makes it difficult to guarantee consistent results over time.

Technical reproducibility in cloud experimentation requires a comprehensive approach, which includes publishing the necessary artifacts such as source code, input data, and detailed technical descriptions to ensure that experiments can be replicated. However, space limitations in research papers often make it difficult to include all necessary information, which is why supplementary materials—such as online appendices—are crucial. While public clouds offer broad access to resources, the evolution of these platforms and the high associated costs can obstruct long-term reproducibility. Lin and Zhang argue that **reproducibility** should be seen not as a one-time achievement but as an ongoing process that requires continual adaptation to new technologies and methodologies in a fast-paced computational environment.

In conclusion, ensuring reproducibility in cloud-based research is a complex and dynamic challenge, requiring collaboration between researchers, transparent publication of technical details, and a recognition of the evolving nature of cloud infrastructure.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

1.7 Distributed Trace Analysis in Serverless Computing

Distributed tracing has become an essential tool in microservice architectures, providing deep insights into system performance by tracking requests across multiple services. However, analyzing and managing distributed traces presents significant challenges, even in traditional software engineering. A survey of 106 practitioners revealed that distributed tracing is one of the top observability challenges in microservices, with 45% of respondents highlighting it as a key concern. In response to these challenges, studies have pointed to intelligent trace analysis techniques as a new big data problem in software engineering.

Existing tools, such as Facebook's **Canopy**, are mainly used for ad hoc, manual analysis of distributed traces. However, there have been advancements in automated trace analysis. For instance, **Pivot tracing** introduces an efficient technique for event correlation across components, while algorithms for **critical path analysis** and **trace graph comparison** have been proposed to streamline the analysis process. Tools like **FIRM** combine machine learning with critical path analysis to help identify and mitigate service-level objective (SLO) violations, and **Luo et al.** leverage graph clustering to improve the characterization of performance and dependencies in microservices.

Despite these advancements, distributed trace analysis in serverless computing presents unique challenges. While there are efforts like **GammaRay**, which augments AWS X-Ray to track causal relationships, and **Lowgo**, a tool for multi-cloud serverless applications, research in this area remains limited. A study comparing serverless tracing tools revealed that these tools vary in their ability to detect different types of faults, indicating that serverless tracing is still in its nascent stages.

One of the critical aspects of trace analysis was dependency mapping, which revealed the relationships between services and their respective latencies. This mapping highlighted services relying heavily on external APIs or databases as primary sources of delay. By optimizing these interactions through caching and parallel execution, the system achieved a significant reduction in latency. Additionally, trace data proved instrumental in root cause analysis by pinpointing issues such as misconfigured timeout settings, inefficient queries, or inadequate scaling thresholds.

Cold starts, a common challenge in serverless environments, were quantified and assessed using distributed trace analysis. Functions with high cold start latency were prioritized for optimization through pre-warming strategies and container reuse, leading to more predictable performance. Tracing tools also supported real-time debugging, providing alerts for errors or performance degradation. This proactive monitoring approach minimized disruptions and ensured rapid resolution of anomalies.

One of the primary limitations in serverless trace analysis is the lack of access to fine-grained instrumentation due to provider-managed infrastructure. This restricts the ability of developers to directly instrument their applications and forces them to rely on the distributed tracing services provided by cloud providers. This often leads to observability gaps, especially when dealing with downstream services that do not support tracing. Moreover, the event-driven nature of serverless architectures necessitates adaptations to traditional tracing techniques, such as **critical path analysis** used in tools like **FIRM**, to account for the asynchronous, stateless

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

nature of serverless invocations.

In summary, while distributed tracing has gained traction in microservices, its application and analysis in serverless environments face distinct challenges. As serverless computing continues to grow, further advancements in trace analysis techniques and tooling are needed to address.

2. BACKGROUND STUDY

B. *2.1 Sample Study in Serverless Application Analysis*

Sample studies are an essential research methodology for systematically analyzing artifacts like documentation or source code. In the context of serverless applications, such studies provide observational insights without interactive data collection, thereby ensuring generalizability. For example, a study described in this text focused on characterizing serverless applications across diverse sources, using a structured review process to capture 22 pre-defined characteristics.

Process Overview

The study analyzed 89 serverless applications sourced from open-source projects, academic and industrial literature, and scientific organizations. The applications were characterized collaboratively by reviewers using predefined criteria. Key steps included:

1. **Data Collection**: Applications were gathered using purposive sampling for publicly available projects and convenience sampling for internal applications. Heterogeneity sampling ensured a balanced mix of sources.
2. **Review Process**: Two reviewers independently characterized each application, resolving discrepancies through discussion and group consultations. Internal applications were characterized by domain experts.
3. **Sampling Strategy**: Techniques like search-based sampling (e.g., GitHub mirrors, keyword searches) and referral-chain sampling (e.g., case studies, blog posts) were used to ensure diverse and representative samples.

Challenges and Insights

The non-interactive nature of data collection meant that certain characteristics were excluded due to insufficient documentation. Nevertheless, the study achieved high generalizability by including a wide range of applications and refining characteristics iteratively during the review process.

Engineering Research and Benchmarking

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

Engineering research combines problem identification with solution evaluation through technical artifacts. This approach is particularly relevant in benchmarking studies, which evaluate proposed solutions to practical problems in serverless computing.

Benchmarking Methodology

The benchmarking process follows four key phases:

1. **Benchmark Design**: Involves creating workloads, measurement tools, and benchmark suites guided by literature and existing practices. This step emphasizes fairness, configurability, and rigorous testing of key implementations.
2. **Benchmark Execution**: Large-scale experiments are conducted in public cloud environments, generating extensive performance data. Experiment plans include scheduling and monitoring multi-week trials.
3. **Data Pre-processing**: Raw data is filtered, validated, reshaped, and refined for analysis. This ensures that only relevant, error-free data is considered.
4. **Data Analysis**: Statistical models and visualizations are applied to summarize performance distributions and answer specific research questions.

Outcomes

The artifacts and results of these studies are made available as open-source software with extensive documentation, promoting transparency and reproducibility. This methodology supports advancing the understanding and optimization of serverless architectures while addressing real-world challenges

2.2Field Experiment in Cloud Benchmarking

Field experiments are a cornerstone of research in cloud computing, offering a natural setting for studying the performance of software systems deployed in massive-scale, multi-tenant public cloud environments. Unlike laboratory experiments, which operate in controlled settings, field experiments leverage the realism of public clouds to evaluate the emergent performance properties of applications.

Key Characteristics

- **Natural Setting:** Conducted directly in real cloud environments, such as public clouds, to achieve maximum realism.
- **Variable Manipulation:** Researchers manipulate specific variables (e.g., instance type, benchmark configurations) to study their effects on performance metrics.
- **Limited Control:** The inherent variability of public cloud environments (e.g., resource contention, network latency) challenges reproducibility. This limitation applies to both IaaS (Infrastructure-as-a-Service) and FaaS (Function-as-a-Service) clouds.

Field experiments in cloud benchmarking follow a structured approach to balance realism, reproducibility, and automation. These experiments often combine benchmarking research with engineering methodologies.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

1. **Automated Deployment**

The application is deployed into the cloud using automated deployment scripts, often leveraging tools like infrastructure-as-code and configuration management systems. This ensures consistency and repeatability in the setup.

2. **Instrumentation and Tracing**

Applications are instrumented with detailed trace points to monitor performance. Trace spans are forwarded to provider-specific tracing services for further analysis.

3. **Workload Invocation**

A workload profile is applied to invoke the application, simulating real-world usage scenarios.

4. **Data Collection and Pre-Processing**

Raw performance data and traces are retrieved from the cloud provider's tracing service. This data is then pre-processed, which involves filtering irrelevant data, validating results, and formatting the data for analysis.

5. **Analysis and Visualization**

The processed data is analyzed to measure performance metrics, such as latency, throughput, and resource utilization. Results are visualized to provide insights into the system's behavior under varying conditions.

Challenges and Mitigation

1. **Reproducibility Challenges:**
   - Due to the variability of cloud environments, identical results cannot always be guaranteed.
   - Mitigation: Automation through programmable experiments, containerization, and detailed replication packages that include datasets, analysis scripts, and executable experiment plans.
2. **Generalizability:**
   - Field experiments are inherently less generalizable compared to less intrusive methods like sample studies.
   - Mitigation: Focus on diverse application scenarios and rigorous experimental designs to improve the breadth of findings.
3. **Performance Variability:**
   - Resource contention and unpredictable workload patterns in shared environments introduce noise.
   - Mitigation: Repeated measurements and statistical analysis to filter out noise.

2.3 Process Overview

The high-level architecture of a benchmarking field experiment includes five key steps, as visualized in Figure 1.10:

1. **Deploy**: Application package and deployment scripts are used to deploy the application in the cloud.
2. **Invoke**: Workload profiles trigger application invocations.
3. **Retrieve**: The benchmark orchestrator collects raw traces or performance data.
4. **Analyze**: Data is processed to generate meaningful metrics.
5. **Visualize**: Results are visualized to interpret the application’s performance characteristics.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

Relevance

Field experiments provide valuable insights into the real-world performance of cloud-based applications. They are particularly critical for understanding serverless architectures and distributed systems where performance variability and limited access to low-level instrumentation are key challenges. By combining rigorous experimental processes with the realism of public cloud environments, field experiments address practical research questions and drive innovations in cloud computing.

3. METHODOLOGY

3.1 Serverless Application Benchmark: ServiTrace

Traditional serverless performance studies primarily focus on single-purpose microbenchmarks, which fail to reflect the complexities of real-world applications. Existing benchmarks are constrained to single-function applications that interact with at most one type of external service. Additionally, these benchmarks often overlook the asynchronous coordination that defines event-driven architectures, a core characteristic of serverless computing.

Contribution

ServiTrace addresses these limitations by introducing a comprehensive benchmarking suite designed to evaluate heterogeneous serverless applications. The contributions of ServiTrace include:

1. **Heterogeneous Application Benchmarking**:
    - A suite of 10 diverse applications reflecting real-world invocation patterns derived from actual logs.
    - Support for asynchronously coordinated applications and a variety of external services.
2. **Latency Breakdown Analysis**:
    - A novel algorithm and heuristic for detailed latency breakdown analysis, even in asynchronous settings.
    - Insight into key performance metrics such as median latency, cold starts, and tail latency across different application types.
3. **Empirical Study**:
    - Conducted a large-scale study in AWS, the leading serverless environment, gathering over 7.5 million traces.
    - Provided fine-grained performance insights for a broad range of serverless application scenarios.
4. **Open-Source Availability**:
    - Released under **FAIR principles** (Findable, Accessible, Interoperable, Reusable) with comprehensive software, data, results, and documentation.
    - Designed to be extensible, encouraging community-driven enhancements and application to

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

various research problems.

ServiTrace follows rigorous engineering and benchmarking methodologies:

1. **Design and Implementation**:
    - Built using insights from real-world application studies and guided by goals identified in prior literature reviews.
    - Includes engineering principles for designing benchmarks based on realistic invocation and coordination patterns.
2. **Field Experimentation**:
    - Follows benchmarking guidelines and reproducibility standards for cloud experimentation.
    - Automates data collection and performance analysis to mitigate variability inherent to public cloud environments.
3. **Algorithmic Innovation**:
    - Introduced algorithms for asynchronously coordinated application performance analysis, addressing challenges like disconnected traces and event-driven triggers.

Key Insights

ServiTrace enabled detailed performance analysis, yielding several significant findings:

1. **Latency Distribution**:
    - Highlighted variations in median latency, cold start durations, and tail latencies across different application types and invocation patterns.
2. **Trigger Latency**:
    - Demonstrated that slow trigger latency from external services contributes significantly to overall delay, underscoring the importance of optimizing trigger mechanisms.

ServiTrace has set a foundation for further research:

- **CrossFit (Paper δ)**: Builds on ServiTrace by enabling cross-cloud provider benchmarking and refining disconnected trace correlation for specific scenarios.
- **TriggerBench (Paper ε)**: Investigates the latency of serverless function triggers in detail, motivated by ServiTrace findings that identified trigger delays as a bottleneck.

3.2 Cross-Provider Application Benchmarking: CrossFit

Fairly comparing serverless applications across cloud providers is inherently difficult due to:

1. **Heterogeneous Ecosystems**:

   FaaS platforms are deeply integrated with provider-specific services and lack standardization, unlike VMs in IaaS systems.

2. **Limited Observability**: Existing comparisons focus solely on overall response times, offering little insight into the underlying reasons for performance differences.

3. **Provider-Specific Implementations**: Integrations with event sources and external services vary

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

widely, complicating portability and benchmarking.

CrossFit addresses these challenges by introducing methodologies and tools for fair and insightful cross-provider benchmarking of serverless applications:

1. **Provider-Independent Tracing Model**:
    - Developed a model to identify equivalent trace points across multiple providers.
    - Provides observability into specific performance bottlenecks in a provider-agnostic manner.
2. **Fairness Guidelines**:
    - Defined 12 critical aspects for fair performance comparison, including workload parity, platform configuration alignment, and equivalent service integrations.
3. **Drill-Down Analysis**:
    - Demonstrated the ability to pinpoint performance challenges across cloud providers by analyzing a realistic application scenario under varying workloads (e.g., constant and bursty).
    - Explored how provider-specific optimizations and limitations impact application latency and scalability.

Methodologies involved:

1. **Engineering Research**:
    - Refined an application from **ServiTrace** (Paper γ) to adapt it for another cloud provider.
    - Inspired by serverless application migration strategies, the application was ported while ensuring functional equivalence.7
2. **Field Experimentation**:
    - Conducted benchmark experiments using workloads designed for reproducibility and realism.
    - Compared performance under controlled and bursty conditions to evaluate cross-provider behaviors.

3. Key Insights

CrossFit revealed important performance variations and their causes:

1. **Performance Differences**:
    - Detailed breakdown of why certain providers performs better or worse under specific workloads.
    - Highlighted inefficiencies in event trigger latency and execution scaling.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

2. **Guideline Validation**:
    - Fairness guidelines proved effective in ensuring unbiased comparisons, reducing confounding variables.
3. **Workload Impact**:
    - Demonstrated that provider-specific architectures handle bursty workloads differently, affecting scalability and tail latency.

CrossFit directly addresses research gaps identified in the literature review of Paper α:

- Tackles the challenge of **cross-provider benchmarking** by providing tools and methods to align configurations and workloads.
- Builds on **ServiTrace** (Paper γ) to overcome reproducibility challenges, demonstrating the extensibility of the benchmark suite.

3.3 Serverless Function Trigger Benchmark: TriggerBench

Function triggers are fundamental to serverless architectures as they initiate function execution. Despite their critical role:

1. **Underexplored Domain**: Paper α highlights a lack of detailed studies on trigger mechanisms, a gap corroborated by Paper β's findings on the prevalence of event-based architectures.
2. **Measurement Complexity**: Trigger performance is challenging to evaluate due to the distributed, ephemeral, and asynchronous nature of serverless systems.

TriggerBench addresses these challenges by introducing:

1. **Cross-Provider Trigger Benchmarking**:
    - Supports evaluation of serverless function triggers across AWS and Microsoft Azure.
    - Implements three of the most popular trigger types identified in Paper β: HTTP, storage, and queue.
    - Expands Azure-specific support to include additional trigger types such as database, event, stream, message, and timer triggers.
2. **Distributed Tracing Methodology**:
    - Developed a methodology for measuring synchronous and asynchronous triggers.
    - Supports trace correlation of disconnected partial traces, overcoming challenges posed by the distributed nature of serverless systems.
3. **Benchmark Experimentation**:
    - Evaluated 11 trigger types across two major cloud providers, offering insights into cross-provider trigger performance.

Methodologies involved:

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

1. **Engineering Research**:
    - Built on the ServiTrace framework (Paper γ) for tracing and measurement tools.
    - Generalized trace correlation techniques demonstrated in CrossFit (Paper δ) to handle disconnected traces in asynchronous workflows.
2. **Field Experimentation**:
    - Conducted benchmarking experiments for 11 different trigger types across AWS and Azure.
    - Evaluated performance metrics such as trigger latency, scalability, and cold-start impact for synchronous and asynchronous scenarios.

TriggerBench revealed critical performance insights:

1. **Trigger Latency**:
    - Identified significant latency variations across trigger types and cloud providers, particularly for asynchronous triggers.
    - Highlighted slower performance for some provider-specific triggers, such as message queues under high workloads.
2. **Scalability**:
    - Evaluated the scalability of trigger mechanisms under bursty and sustained workloads.
    - Showed differences in how providers handle high-concurrency workloads, impacting tail latency.
3. **Cold-Start Impact**:
    - Measured the influence of cold starts on trigger latency, especially for infrequently used triggers.

Relationship to Thesis

TriggerBench addresses the research gap highlighted in Paper α by providing a comprehensive study of function triggers:

1. Extends the **ServiTrace** framework from Paper γ, leveraging its distributed tracing capabilities.
2. Builds on cross-provider methodologies introduced in Paper δ to generalize trace correlation.
3. Motivated by findings in Papers γ and δ regarding poor trigger performance, it explores the root causes and variations across providers.

3.4 Integrating Micro- and Application-Level Benchmarks

Traditional benchmarking of IaaS clouds has focused primarily on isolated performance metrics using either micro- or application-level benchmarks. However, combining these two approaches for holistic performance evaluation has not been systematically explored, and reproducibility challenges remain significant.

1. **Execution Methodology**:

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

- Combines micro- and application-level benchmarks into a unified suite.
- Automates the execution within a cloud benchmarking framework.

2. **Benchmarking Framework**:
    - Integrated into Cloud WorkBench (CWB), automating the benchmarking process.
    - Instantiated in the AWS EC2 cloud to evaluate metrics such as cost-performance efficiency, network bandwidth, and disk utilization.

Methodologies involved:

- Benchmarks were selected and integrated based on established cloud benchmarking guidelines.
- Automated execution followed the RMIT methodology proposed by Abedi and Brecht, ensuring repeatability.

Relationship to Thesis

This work provides foundational tools for systematic cloud benchmarking. It connects IaaS benchmarking insights with serverless performance evaluations (e.g., from Papers α to ε) by addressing reproducibility and providing a base for follow-up studies.

Application Performance Estimation Using Micro-Benchmarks

The wide variety of cloud services has led to an increased need for reliable service selection methods. While micro-benchmarks are widely used for this purpose, their relevance in predicting real-world application performance remains uncertain.

Contribution

1. **Estimation Methodology**:
    - Proposed a methodology for predicting application-level performance based on micro-benchmark profiling.
    - Trained a linear regression model using 38 metrics from 23 micro-benchmarks across 11 VM instance types.
2. **Evaluation**:
    - Conducted field experiments in AWS EC2 to quantify performance variability.
    - Compared the regression model against three baselines using forward feature selection.
    - Released a dataset of over 60,000 measurements spanning 240 VMs and 11 VM types.

Key Insights

- Demonstrated that micro-benchmarks can effectively predict application performance within acceptable error margins.
- Highlighted variability between VM types, emphasizing the importance of tailored service selection for

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

specific application needs.

3.5 Software Micro benchmarking in Public Clouds

Public cloud environments, being inherently multi-tenant, are susceptible to stochastic variability from factors like noisy neighbors, hardware changes, and network instabilities. This raises questions about the suitability of clouds for software micro benchmarking and the reliability of detecting performance regressions.

Contribution

1. Quantifying Variability:
    - Measured variability in performance across 19 benchmarks, 9 cloud environments, and 4.5 million results.
    - Analyzed sources of variability (e.g., benchmark-level, trial-level, or total).
2. Slowdown Detection:
    - Compared two statistical tests (Wilcoxon Rank-Sum and Overlapping Confidence Intervals) for their false positive rate and ability to detect minimal slowdowns.
    - Simulated performance slowdowns to determine the smallest detectable changes under a 5% false-positive threshold.

Key Insights

- Found that despite inherent variability, well-designed statistical methods can reliably detect small performance regressions.
- Variability at the trial and benchmark levels was manageable, providing confidence in cloud-based performance testing.

3.6 Impact of Autoscaling on Serverless Frameworks

Autoscaling is a vital feature of serverless computing platforms, enabling dynamic resource allocation based on workload demands. This evaluation focuses on the effect of autoscaling on response times across various serverless frameworks using machine learning (ML) models such as CNN and LSTM.

The experiment scaled functions based on CPU utilization, with a threshold of 50% to trigger pod creation. Initial setups had one pod per framework, expanding dynamically with autoscaling. A parallel experiment used a fixed number of pods without autoscaling. Popular tools like Kubernetes Horizontal Pod Autoscaler (except for funcX) managed scaling. The JMeter tool was used to simulate 1000 requests with 1 and 5 concurrent users, and each test was repeated three times.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

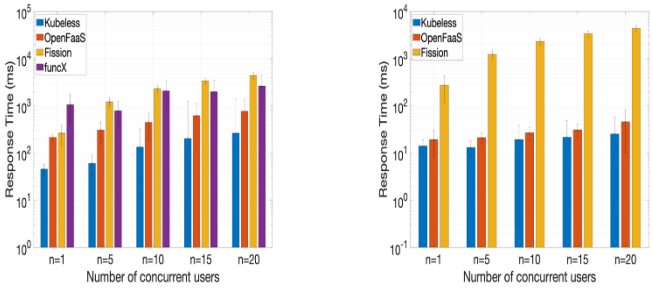

Fig. 3.1 - Average response time for Helloworld function for different frameworks in Python and Node.js

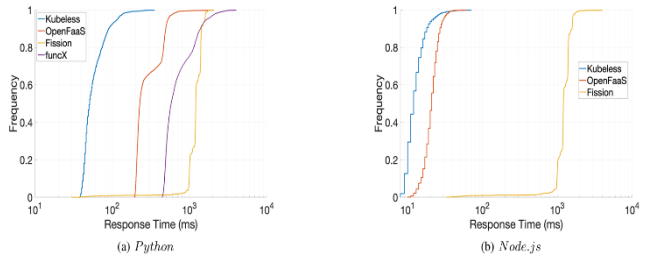

Fig. 3.2 - CDF of response time for Helloworld function (n = 5) for different frameworks in Python and Node.js

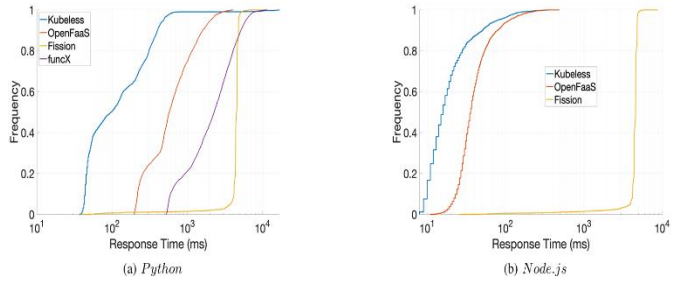

Fig. 3.3 - CDF of response time for Helloworld function (n = 20) for different frameworks in Python and Node.js

Results and Observations

1. **Response Times**
   - **CNN Functions**: Kubeless consistently achieved the best response times, with and without autoscaling. Fission showed significant improvements with autoscaling, reducing response times by 27% and 100% for 1 and 5 users, respectively.
   - **LSTM Functions**: Fission and funcX delivered better response times. Fission's autoscaling improved performance by 5% for 1 user and 30% for 5 users.
2. **Pod Scaling Behavior**
   - Both Fission and Kubeless scaled more aggressively by adding pods in response to increased CPU utilization. OpenFaaS took a more conservative approach, scaling fewer pods and leading to longer response times due to serialized request processing.
   - funcX, which does not support autoscaling with CPU utilization, exhibited minimal pod

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

scaling.

3. **Success Rates**
    - At lower concurrency levels (1 user), Kubeless and Fission maintained a 100% success rate, while OpenFaaS and funcX encountered a 1–2% error rate. At higher concurrency levels, funcX showed reduced success rates with up to a 3% error rate.

Edge Model Training

The study also evaluated training performance for CNN and LSTM models on edge devices with ARM processors. CNN models exhibited higher classification accuracy but required longer training times compared to LSTM. Notably, CNN classification was faster than LSTM, attributable to its architecture being more amenable to parallel processing.

- **Training Time Comparison**: CNN models showed a higher ARM-to-x86 training time ratio (15.58) compared to LSTM models (7.38), emphasizing the resource-intensive nature of CNNs during training. However, LSTMs' complexity in sequential execution made them less efficient for classification tasks.

4. TOOLS

Benchmarking tools used in serverless computing evaluation focus on assessing the performance, scalability, and efficiency of serverless platforms. Here are some widely recognized tools:

1. **Serverless Framework Benchmarking Plugins**
    - Tools or plugins within the Serverless Framework for evaluating deployment and execution times.
2. **AWS Lambda Power Tuning**
    - Used to optimize and benchmark AWS Lambda function performance by exploring different memory configurations.
3. **Artillery**
    - A modern, powerful, and easy-to-use load-testing toolkit for serverless applications.
4. **Wrk and Wrk2**
    - High-performance HTTP benchmarking tools to test API response times and throughput.
5. **Apache JMeter**
    - Used to load test and measure the performance of serverless applications and APIs.
6. **Hey**
    - A tiny load-generator tool that helps benchmark serverless APIs with simple HTTP requests.
7. **Phoronix Test Suite**
    - A general-purpose benchmarking tool that can also be adapted to serverless workloads.
8. **Serverless Benchmark**
    - A tool designed to compare serverless platforms like AWS Lambda, Google Cloud Functions,

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

and Azure Functions.

9. **CloudState**
    - A benchmarking and evaluation tool for stateful serverless computing.
10. **FaaSProfiler**
    - A specialized tool for performance and cost profiling of Function-as-a-Service (FaaS) platforms.
11. **OpenFaaS Benchmark Suite**
    - Focused on benchmarking functions deployed on OpenFaaS.
12. **Benchmarking Harness**
    - Custom solutions often developed for specific research, such as benchmark suites provided in academic papers like "The Serverless Application Benchmark (SAB)."

These tools help evaluate serverless platforms' latency, cold starts, scalability, cost efficiency, and reliability.

RESULTS

The current state of serverless applications and their performance evaluation highlights key gaps and opportunities for improvement. One of the most efficient yet often overlooked approaches to advancing serverless computing performance is a hift from isolated micro-benchmark evaluations to a more integrated application-level benchmarking framework that addresses real-world complexities. This paradigm shift would not only make performance evaluations more reflective of real-world scenarios but also enhance the efficiency of serverless systems by focusing on the interplay between external service calls, function triggers, and runtime environments.

Such an approach would accelerate execution times, reduce latency, and improve the overall scalability of serverless platforms. By focusing on heterogeneous and extensible benchmarks that incorporate diverse function triggers, external service integrations, and asynchronous architectures, the benchmarking methodology would better support cross-cloud provider evaluations. This enhanced benchmarking framework could further incentivize widespread adoption by improving interoperability, reducing operational overhead, and supporting real-time performance optimization.

An improved understanding of serverless performance bottlenecks—such as delays caused by external services, trigger-based coordination, and cold start overheads—would not only attract more users but also unlock greater potential for innovation in serverless computing. By systematically addressing these challenges, serverless platforms could achieve higher total value locked in their systems and encourage developers to adopt a more streamlined, scalable, and efficient development approach.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

DISCUSSION

This thesis emphasizes the importance of detailed tracing and trace analysis for improving serverless application performance. Without tracing, issues in asynchronous event-driven processing often go unnoticed. Detailed observability provides insights into the end-to-end request lifecycle, helping identify performance bottlenecks. Tracing challenges include data quality, missing annotations, and disconnected traces, which hinder robust analysis. To address these issues, enhanced trace annotations and solutions for merging disconnected traces are proposed. Despite the overhead of tracing, which can affect cold start performance, the impact on runtime is minimal when implemented asynchronously. However, dedicated tracing services may add scalability hallenges.

For interactive applications, the thesis finds that serverless can meet latency requirements, provided the right service selection is made. HTTP triggers and low cold start overhead are critical for sub-0.1s latency, though external data services can introduce delays. Tracing and optimizing slow segments are necessary for improving performance. Reproducibility in cloud performance evaluation is also addressed, highlighting challenges like incomplete experimental setups and the need for automated experiment orchestration. Statistical methods and probabilistic result descriptions are recommended for evaluating performance variability and minimizing human error in experiments. Open access artifacts and replication packages are provided for transparency and reproducibility.

Cross-provider portability remains a major challenge in serverless applications, requiring trade-offs. Unlike IaaS, where the standardized VM abstraction allows full code reuse, serverless APIs are highly provider-specific for both source code and deployment options. Prior research highlights vendor lock-in as a significant issue, as confirmed in studies and surveys. Existing solutions focus on specific domains like data analytics or simple functions. Vendor lock-in is also a barrier to multi-cloud approaches. Due to the lack of a common interface, implementing a single provider-agnostic benchmark is not possible, and cross-provider support requires careful application migration.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

CHALLENGES

Evaluating the performance of serverless computing involves several challenges due to the dynamic and abstract nature of this architecture. Key challenges include:

1. Cold Start Latency

- Serverless functions often face delays during their initial invocation due to container initialization. Measuring and mitigating these cold starts is crucial for applications requiring low-latency responses.

2. Resource Allocation Variability

- Cloud providers dynamically allocate resources, which can lead to inconsistent performance across identical workloads. Understanding and quantifying this variability is challenging.

3. Concurrency and Scalability

- Performance can degrade when high concurrency levels strain the system. Evaluating how platforms handle concurrent invocations and scale under different workloads is complex.**4. Lack of Standardized Metrics**
- There is no universally accepted framework for evaluating serverless performance, making it difficult to compare metrics such as latency, throughput, and cost-effectiveness across platforms.

5. Platform-Specific Optimizations

- Each cloud provider implements unique optimizations (e.g., AWS Lambda's Provisioned Concurrency), complicating cross-platform performance comparisons.

6. Hidden Infrastructure Details

- Providers abstract the underlying infrastructure, limiting visibility into factors like network latency, hardware configurations, and VM-level performance.

7. Dynamic Pricing Models

- Costs are tightly coupled to execution duration and resource consumption. Analyzing the cost-performance trade-offs requires careful modeling.

8. Diverse Workload Requirements

- Workloads vary in characteristics such as memory needs and execution time, making it difficult to create representative benchmarks for general performance evaluation.

9. Testing Under Real-World Conditions

- Simulating real-world usage patterns, including bursty traffic and variable workloads, is challenging but necessary for accurate performance insights.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

FUTUREWORK

Recent studies reveal gaps in serverless performance evaluation. There is a significant underrepresentation of certain serverless characteristics in academic research, particularly in terms of external services and event-based triggers. While cloud events are common in real-world serverless applications (41%), performance studies rarely use event-based triggers (<16%). Similarly, databases, which are used in 48% of serverless applications, are underrepresented in studies, appearing in only 10-15% of academic and industrial research. API gateways and cloud storage are the exceptions, receiving more attention. More studies are needed on external services like publish/subscribe, streaming, and queues to address these gaps. Recent trends indicate an increase in the number of studies, including those focusing on new platforms such as Alibaba, Native, OpenWhisk, and Firecracker. However, most studies still concentrate on micro-benchmarks and neglect broader external services, with some exceptions in specific domains like data processing and workflows. Additionally, performance evaluation for edge computing platforms is becoming an merging research area.

Traditional distributed tracing methods are not well-suited for serverless architectures, which involve asynchronous invocations. This introduces new challenges related to collection, analysis, and visualization of traces. Existing methods suffer from issues such as bad trace quality, which is exacerbated in serverless due to limited control over infrastructure. Manual instrumentation and custom trace correlation are often required to fix disconnected traces, and sampling is necessary to handle high invocation rates. Future research should focus on improving trace quality management, particularly for serverless architectures, by exploring better techniques for trace collection and analysis. Additionally, the Open Telemetry standard and its implementation in serverless environments should be further studied, including suggestions for trace annotations and handling batch invocations, which violate traditional assumptions of synchronous trace chains.

While this thesis focuses on performance evaluation, a promising direction for future work is to automate performance optimization based on the findings of performance assessments. The goal is to integrate performance considerations into the development lifecycle, making performance insights more actionable.

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

CONCLUSION

In conclusion, this report highlights critical challenges and future directions in serverless performance evaluation, trace analysis, and automated performance optimization. While serverless computing offers scalability and cost-effectiveness, it introduces unique complexities that require more in-depth research to fully understand and optimize performance.ey gaps in serverless performance evaluation were identified, particularly the nderrepresentation of critical components like event-based triggers and external services, such as databases, in academic studies. Despite their frequent use in real-world applications, these elements are often overlooked in research. Expanding performance studies to include a broader range of triggers and external services, such as publish/subscribe systems and streaming, is necessary to more accurately reflect the performance characteristics of serverless environments.

Trace analysis in serverless systems also presents challenges due to the asynchronous nature of invocations. Traditional tracing methods, designed for synchronous systems, struggle to capture complete and accurate trace data. Issues like bad trace quality, missing segments, and disconnected traces emphasize the need for more robust methods in trace collection and analysis. The development of enhanced standards, like Open Telemetry, and novel tracing concepts for batch invocations are essential for improving trace quality and completeness.

Automated performance optimization is a promising area for future research, aiming to integrate performance insights directly into the development lifecycle. Techniques like dynamic transpiration, function fusion, and workload-specific tuning can optimize resource allocation and reduce latency. Expanding these optimization approaches to consider external services, such as trigger types, would further enhance overall performance.

REFERENCES

1. Baldini, I., et al. (2017).
   "Serverless Computing: Current Trends and Open Problems." 2017 IEEE International Conference on Cloud Engineering (IC2E), 1-4.
2. Jiang, H., et al. (2019).
   "The Impact of Cold Start on Serverless Computing Performance." 2019 IEEE International Conference on Cloud Computing (CLOUD), 72-79.
3. Finkel, H., et al. (2020).
   "A Benchmarking Framework for Serverless Computing." ACM Transactions on Internet Technology, 20(4), 1-24.
4. Hassan, S., et al. (2020).
   "Resource Allocation in Serverless Computing: A Survey." ACM Computing Surveys, 53(4), 1-35.
5. Jinfeng Wen, Zhenpeng Chen, Federica Sarro, Xuanzhe Liu (2023).
   “Revisiting the Performance of Serverless Computing: An Analysis of Variance”

https://dx.doi.org/10.22161/ijaems.ncitsa.2025.53 ISSN: 2454-1311

[1] Philip, E., & Philip, A. (2022). The influence of positive self-affirmation towards Malaysian ESL students at tertiary level of Education. *Journal of Humanities and Education Development,4*(4), 09-17. doi:10.22161/jhed.4.4.2

[2] Ifeanyi, I. C. (2022). Micronutrient concentrations of cassava continuously cultivated soils in EzinihitteMbaise LGA Imo State, Nigeria. *International Journal of Environment, Agriculture and Biotechnology,7*(5), 001-007. doi:10.22161/ijeab.75.1