International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/IJSRCSEIT

Leveraging Java for Optimizing Serverless Cloud Computing

Chinmay Mukeshbhai Gangani
Independent Researcher, USA

Article Info
Volume 7, Issue 5
Page Number: 155-165

Publication Issue :
September-October-2021

Article History
Accepted : 07 Oct 2021
Published : 30 Oct 2021

ABSTRACT

By separating the display layer or front-end from the content repository or back-end, headless content management systems (CMS) provide a great deal of freedom. Serverless computing architectures, on the other hand, provide many benefits, such as improved performance, automated scalability, high availability, cost effectiveness, and better resource and execution time management. This data is essential for assessing and forecasting a number of occurrences. Traffic flow, forest cover, disease surveillance, and other application areas are among those that often call for a real-time study. As a result, the majority of current systems exhibit certain limits at different processing and implementation levels. Lack of scalability, excessive processing expenses, and unreliability are some of the most often noted issues. However, in order to assess performance and determine which platform is ideal, organisations need to be aware of a number of principles. These guidelines place a strong emphasis on choosing platforms according to their scalability, price structures, security features, programming language compatibility, and usefulness. The main goals of performance metrics are to reduce latency, maximise resource use, and provide customised insights. Java is perfect for creating cloud-native apps that can easily grow with business needs because of its stability and platform freedom. It explores the performance snags that come with serverless architectures, paying special attention to network latency, function execution time, resource use, and cold start delay. Through fault-tolerant designs and distributed computing, enterprises may increase dependability by utilising the flexibility of cloud computing. The creation of microservices and containerised applications is further facilitated by Java's vast ecosystem of frameworks and libraries, which guarantees deployment agility and efficiency. Cloud computing and Java work together to enable businesses to innovate quickly, use resources efficiently, and provide robust solutions that satisfy the changing needs of contemporary corporate settings.

Keywords :- Content Management Systems (CMS), Network Latency, Business Environments, Reliability, Scalability, Security Features, Automatic Scalability, Optimized Management, Java's Extensive, Java's Robustness, Enterprise Demands, Cloud-Native Applications, Programming Language.

INTRODUCTION

With major benefits including pay-as-you-go pricing, automated scalability, and less operational complexity, serverless computing has become a game-changer in the cloud computing space [1]. These characteristics have made serverless platforms very appealing to developers and businesses alike, and as a result, the industry is expected to expand at a compound annual growth rate (CAGR) of 22.7% from $7.6 billion in 2020 to $21.1 billion by 2025 [1, 2]. With companies claiming up to 4x quicker time-to-market and 20-30% lower operating costs when compared to conventional cloud models, the promise of increased productivity and cost effectiveness is what is driving this rapid adoption [3, 4]. But since serverless systems abstract away underpinning infrastructure, they present special difficulties for performance optimisation.

Serverless systems are completely controlled by cloud providers, in contrast to typical cloud computing models where customers have control over infrastructure resources. This results in issues like the cost of stateless execution models, uncertain performance because of multi-tenancy, and cold start delay [5, 6]. Depending on the runtime and configuration, cold start latency, one of the biggest problems, may cause delays of anywhere from 100 ms to several seconds [6, 7].

Cold start latencies in AWS Lambda may range from 100 ms for Node.js functions to up to 2 seconds for Java functions, according to research by Wang et al. [7, 8]. The user experience may be greatly impacted by this unpredictability, particularly for applications that are sensitive to latency. Performance unpredictability may result from multi-tenancy in serverless systems; research indicates that resource contention can cause function execution durations to fluctuate by up to 20%. Furthermore, applications that need shared state or persistent connections incur expense due to the stateless nature of serverless functions [9, 10], which might result in a 15–30% increase in execution time for certain workloads.

Considering how quickly serverless computing is being adopted by different sectors, it is becoming more and more important to optimise serverless application performance. In serverless computing, performance optimisation entails more than simply cutting down on latency or speeding up execution times; it also entails lowering expenses, enhancing resource use, and guaranteeing the scalability of programs [10]. According to research, when compared to poorly optimised serverless apps, optimised ones may save up to 90% on costs and increase response times by 60%.

Incorporate stateless functions, pay-as-you-go pricing mechanisms, automated scalability, and event-driven execution. While automated scaling lowers operational overhead, event-driven execution minimises resource utilisation during idle times [10]. Furthermore, the pay-as-you-go pricing model encourages cost efficiency by charging consumers according to real resource utilisation. Finally, serverless operations' statelessness improves dependability and fault tolerance.

- **Cloud computing and the evolution of serverless architectures**

Infrastructure as a Service (IaaS), in which users control virtualised resources, Platform as a Service (PaaS), which abstracts further infrastructure components, and Function as a Service (FaaS), a fundamental component of serverless computing, are the evolution of cloud computing. This progression lowers operational complexity, improves resource utilisation, and streamlines application development [11, 12]. The most recent advancement in this process is represented by serverless architectures, which are transforming the creation, deployment, and operation of applications. Specifically, the intricacy of cloud infrastructure management and the need for scalable and affordable solutions drove the move to serverless architectures. Major cloud service providers

including AWS Lambda, Azure Functions, [10], Google Cloud Functions, and open-source substitutes like Open-FaaS and K-native have now introduced serverless computing platforms as a result of this shift.

- **The importance of performance evaluation for optimal platform selection**

The speed, affordability, and scalability of an application are all significantly impacted by the serverless computing platform choice used in a cloud setting. It might be difficult to choose the best serverless platform for a given use case due to the wide variety of options available. Performance review is thus crucial to this decision-making process. Metrics including response time, throughput, resource use, [11], and scalability are analysed. Additionally, it enables developers to evaluate how well a platform fits the needs of their applications, which promotes optimisation and well-informed choices. Without careful consideration, developers run the danger of selecting a platform that might result in inefficiencies, higher expenses, and restricted capabilities for their applications. Additionally, benchmarking and performance target setting are ensured by performance assessment [10,12], ensuring that apps fulfil user expectations and provide a flawless experience.

- **Selection criteria for serverless platforms**

The cost-effectiveness and performance of applications are greatly impacted by the choice of serverless platform [10, 13]. Thus, take into account these important factors to make an educated decision:

1. **Functionality and Services Offered:** Examine the serverless platform's array of features and services. Databases, event sources, tools, and connectors vary across systems. Select a platform that meets the unique needs of your application [10]. Take into account elements such as third-party service compatibility, authorisation procedures, authentication, and database availability.
2. **Programming Languages and Runtimes:** Make sure the serverless platform is compatible with the runtimes and programming languages you want to employ for your application. While some systems have a large assortment, some have less options [10]. For smooth development and maintenance, compatibility with your development stack is essential [11].
3. **Scalability and Concurrency:** Examine the platform's concurrency and scalability. The maximum number of concurrent execution and available resources varies throughout systems [10, 11]. Select a platform that can manage your application's anticipated load and traffic patterns without experiencing scaling issues.
4. **Pricing and Cost Structure:** Understand the pricing model of the serverless platform. Evaluate how pricing is structured, including the cost of function executions, memory usage, and any additional services you plan to use [12]. Consider how your application's expected usage patterns will impact the overall cost and budget accordingly.
5. **Performance and Latency:** Analyse the platform's performance metrics, such as throughput, latency, and response times. Make sure the platform satisfies your performance needs by benchmarking it [14], particularly for interactive or real-time applications where low latency is essential.
6. **Ecosystem and Community Support:** Think about how big and active the platform's development community is [10]. More resources, libraries, and assistance are often accessible in a vibrant community. Additionally, it may serve as a predictor of the platform's long-term sustainability and potential improvements.

The cloud computing paradigm has significantly decreased the prices of compute and storage while serving a number of intricate applications. The idea of serverless computing emerged as a way to handle data collecting

that was growing in size. The serverless techniques' technical advancements have increased their adaptability in managing latency problems and providing consumers with continuous services [1, 2]. As a result, managing the system's many needs may be done effectively and consistently. The serverless platform offers outsourcing of infrastructures necessary for carrying out a number of intricate calculations and significantly expands the capabilities of cloud-based computing. Two basic classifications may be constructed based on the evolutionary notions introduced by the serverless platform [2, 3], namely.

- **Back-end-as-a-Service (BaaS):** In terms of principle, it is comparable to Software-as-a-Service (SaaS). By increasing the system's modularity, off-the-shelf solutions are positioned at the server side [3, 4]. Several user-defined external applications may include the domain generic components represented by the BaaS services. One of Google's web-based analytics tools, Firebase, uses the BaaS architecture to manage user data components [3, 4].
- **Functions-as-a-service (FaaS):** It offers a wide range of resources for creating and deploying server-side applications [3, 5]. FaaS places more focus on the processes that go into creating an application than on the host instances and the application itself. As a result, FaaS is an event-based framework for sophisticated data analysis [5]. Amazon Web Services (AWS) Lambda is a well-known illustration of this technology, with the ability to handle billions of events in shorter amounts of time.

The trade-offs between the conventional and FaaS-based approaches to software deployment are shown in Figure 1 [5, 6]. FaaS differs from typical server-side deployment in that it takes into account various procedures to create applications without requiring host instances.

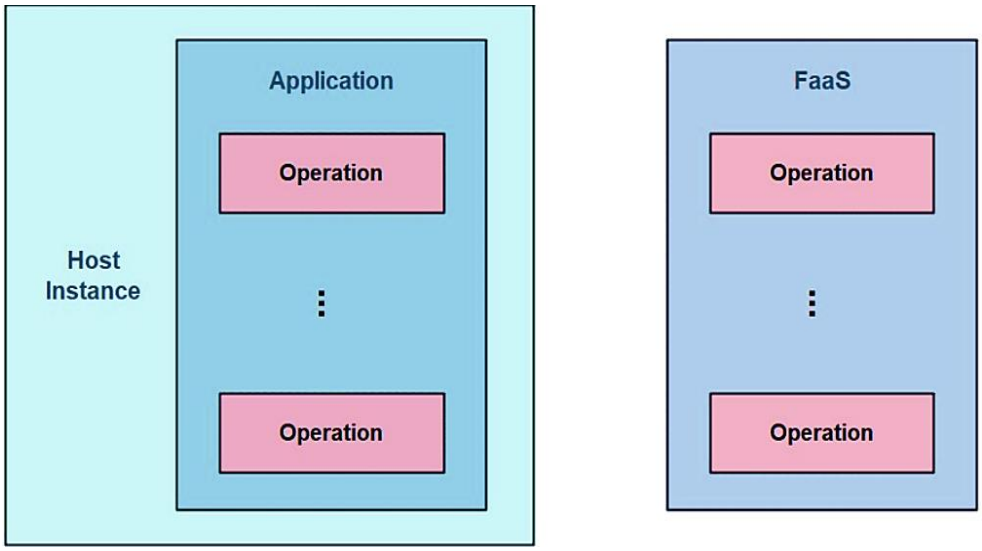

Fig. 1 (A) illustrates the conventional method for server-side software deployment; (B) shows the Functions-as-a-service (FaaS)-based method for software distribution related to various processes. [6, 5]

CMS web applications are among the many use cases that serverless computing is ideal for. All things considered, serverless computing provides web developers with a strong and effective method for creating and implementing CMS web applications, allowing for improved performance [6, 7], auto-scalability, high reliability, cost-effectiveness, and excellent resource and execution time management. Additionally, headless CMS systems are a desirable alternative for companies looking to provide dynamic, customised content experiences due to

their flexibility and scalability. A distinctive and beneficial addition to online application development is the combination of Headless Content administration Systems (CMS) with serverless computing architectures, which solves a number of enduring issues with infrastructure administration, scalability, and flexibility [6, 8]. This hybrid design, in contrast to conventional web development frameworks, enables developers to use serverless services to maximise efficiency while separating content management from the display layer [5, 6].

This dual strategy closes a significant gap in the literature, since most solutions concentrate on either scalable computing or flexible content management without completely combining the two. In order to reduce operational complexity and enable smooth adaptability to changing user expectations, this research presents a complete approach [6, 8]. By promoting this strategy, this article hopes to assist organisations and developers in producing scalable, effective digital experiences with less backend upkeep [9].

Cloud computing has completely changed how businesses install and manage IT resources by providing a flexible and affordable substitute for conventional on-premises infrastructure. Cloud services allow organisations to expand resources dynamically according to demand by providing them with internet-based access to computing power, storage, and apps [9]. By enabling quick innovation and service rollout, this change not only lowers the need for a sizable upfront capital expenditure but also improves operating efficiency. Scalability and dependability are crucial in the context of corporate systems [9, 10]. Scalability is the ability of a system to accommodate growing loads without seeing a decline in performance. Because cloud platforms provide capabilities like load balancing and auto-scaling that dynamically adapt resources to meet changing needs, they provide scalable solutions.

Even during times of high demand, this functionality guarantees that apps stay accessible and responsive. Contrarily, reliability refers to the systems' constant availability and performance. Cloud architectures are used by businesses to provide disaster recovery and high availability solutions that reduce downtime and data loss [9, 10]. Strong service level agreements (SLAs) that ensure availability and performance measurements are often provided by cloud providers. Distributing workloads across many geographical areas improves dependability even more [2, 3], guaranteeing continued operations in the event of natural catastrophes or hardware malfunctions [6, 8]. Java's extensive ecosystem, scalability, and platform independence make it an essential component of cloud architecture [9, 10].

THE ROLE OF JAVA IN CLOUD ARCHITECTURE

Java is a strong and flexible programming language that is renowned for its security, ease of use, and resilience. Its object-oriented architecture, which encourages code reusability and modularity and makes it perfect for massive corporate applications, is one of its main advantages [9, 10]. Java's automated garbage collection memory management improves program stability by preventing memory leaks. Additionally, Java's vast standard libraries streamline development and lessen the need for external dependencies by giving developers capabilities for everything from networking to data processing [9, 10].

The Java Virtual Machine (JVM) makes this possible by enabling Java apps to run on any hardware or operating system that has JVM support. For businesses wishing to implement apps in a variety of settings without requiring platform-specific changes, this portability is essential. Java programs can execute reliably on any operating system because to the JVM's abstraction of the underlying hardware. Because code doesn't need to be rewritten or modified for various platforms, this not only makes deployment easier but also lowers maintenance expenses. In addition, Java's bytecode structure guarantees compatibility across versions, enabling companies to

update systems with little interruption [11]. By offering ready-to-use components for routine processes, these solutions save developers from having to start from scratch and instead enable them to concentrate on addressing business-specific issues. With a multitude of tools, forums, and open-source projects, the Java community is among the biggest in the programming world [10, 11].

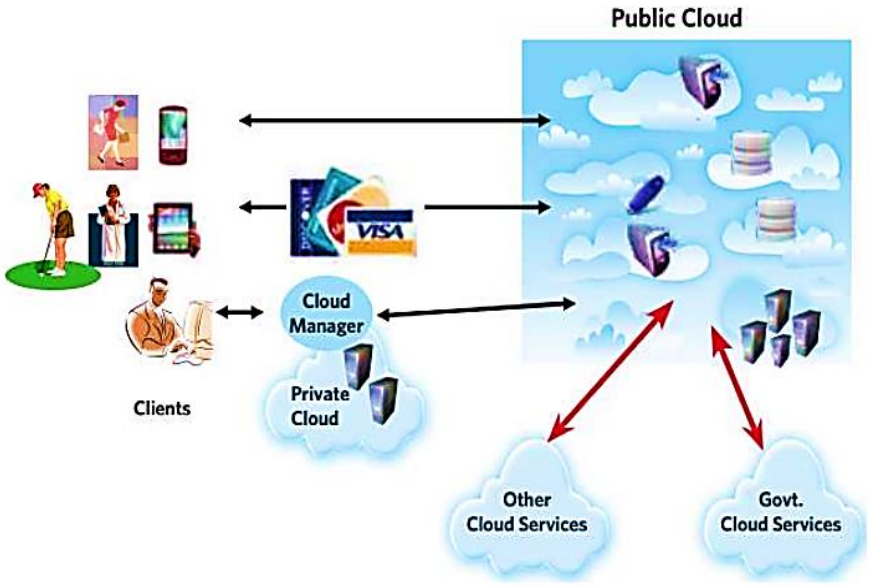

Fig. 2 A bird's eye view of Cloud computing. [12]

Building Scalable Architectures with Java

Applications are constructed using the microservices architecture technique as a group of discrete, autonomous services that interact with one another over a network [13]. Unlike monolithic designs, each service is focused on a particular business function, allowing for more flexibility and scalability. Development agility is facilitated by this modular architecture, which allows teams to work on many services at once and roll out improvements without impacting the system as a whole. Fault isolation is improved by microservices; if one service fails, the application as a whole may not be affected. Organisations may gradually implement new frameworks and technologies because to microservices' modular design [14]. Different databases or programming languages may be used to create services, giving teams the freedom to choose the most appropriate tools for each given job. Because each service can be expanded individually to meet demand, this flexibility not only speeds up innovation but also makes scaling simpler [15].

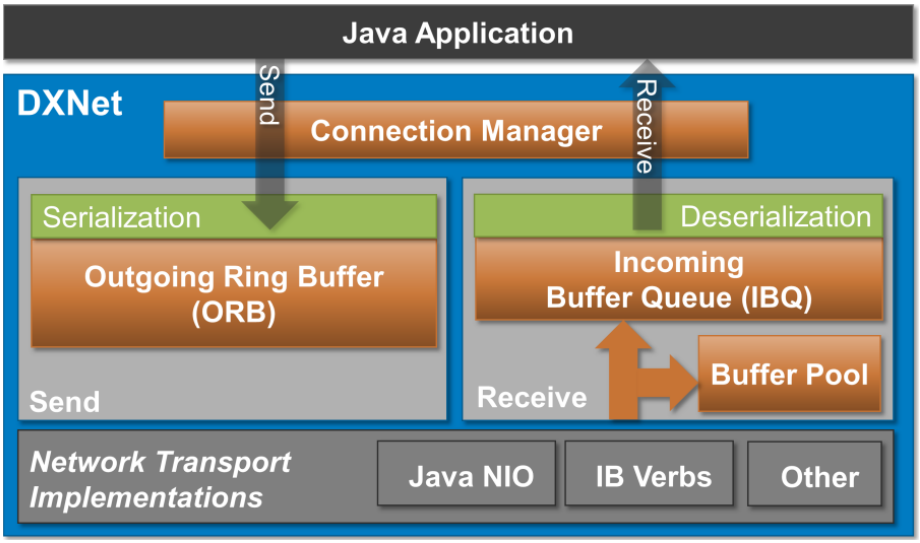

Fig. 3 Simplified DXNet Architecture. [13]

Automatic connection management

DXNet uses node IDs to abstract physical network addresses, such as IP/Port for Ethernet or GUID for InfiniBand. The node address mappings indicated above are changeable for server upscaling and downscaling and are registered in the library [14]. When a message has to be delivered to a server that hasn't been connected yet, a new connection is instantly established. The library will throw exceptions for the application to manage in the event of an error. According to a newly implemented technique, connections are terminated if the configurable connection limit is reached [14], or if there are network issues that may be identified by timeouts or provided by the transport layer, such as missing answers. sending communications. In addition to sending messages asynchronously to one or more recipients, DXNet also sends blocking requests to one recipient, which return when the relevant answer is received (DXNet transparently manages the relationship between requests and responses) [16]. Java objects make up the messages, which are serialised using DXNet's quick and concurrent serialisation (which offers default implementations for the most popular objects).[19].

The Outgoing Ring Buffer (ORB), which is created outside of the Java heap and combines messages for high throughput, receives direct writing from the serialisation. A decoupled transport thread that relies on event signalling sends data [18]. Additionally, DXNet has a flow control system, which is not covered in detail here. obtaining messages. When the network transport detects incoming data, it transfers the data into the buffer Fig. 3 and requests a pooled native memory buffer to prevent overloading the Java garbage collector. The incoming data buffer is then put to the Incoming Buffer Queue (IBQ), a ring buffer that holds references to buffers that are prepared for deserialization. Every connection shares the buffer pool and the IBQ. Dedicated threads retrieve and process the IBQ's buffers asynchronously. Parsing message headers, constructing message objects, and desterilising payload data are all examples of message processing [22]. Lastly, a pre-registered call-back function is used to return the received message to the program (as a Java object).

CHALLENGES AND FUTURE TRENDS

Businesses that use cloud computing more often run into a number of problems that might prevent them from succeeding. Managing cloud expenses is one frequent difficulty [10]. Ineffective resource use or unforeseen demand surges may cause costs to skyrocket without adequate monitoring and management. This risk may be

reduced by putting cost management procedures and tools into place, such as creating budget alerts and allocating resources as efficiently as possible. Assuring security and compliance is another pitfalls [22, 23]. Strong security measures must be put in place since cloud environments are susceptible to data breaches and cyberattacks. Encryption, access restrictions, and frequent security assessments are all essential components of an organization's overall security plan [22, 23]. Maintaining adherence to industry rules and standards also calls for constant observation and revisions to procedures and policies [23, 24]. Problems with integration and interoperability can arise, particularly when combining cloud services with already-existing on-premises systems. Data silos and operational inefficiencies may result from platform or service incompatibilities. Smoother interconnections and flawless system operation may be achieved by using middleware solutions and standardised APIs.

- **Emerging trends in serverless architectures**

Serverless architectures are always changing, and a number of new developments are anticipated to influence this technology going forward. The increasing interest in serverless containerisation systems is one noteworthy trend [23, 24]. Conventional serverless services are intended to be transitory and stateless. However, increasingly complicated workloads, such as dependencies, libraries, and even stateful components, are being packaged by containerisation technologies.

- **Potential solutions for overcoming limitations**

The "cold start" issue, which occurs when a serverless function is called for the first time or after a period of inactivity, is one of the biggest drawbacks of serverless computing [25]. It is anticipated that future systems would use sophisticated predictive scaling algorithms to mitigate this problem.

- **Serverless computing's role in edge and IoT applications**

In the rapidly growing fields of edge computing and Internet of Things applications, serverless computing is expected to be crucial [26, 27]. The combination of serverless computing with edge-native platforms is a major avenue for these improvements in the future. By taking use of computer resources' close proximity to edge devices, edge computing lowers latency and improves real-time data processing [29]. Serverless architectures are thus well adapted to this setting, enabling developers to implement features at the edge, nearer to data sources and end users.

Cloud computing is also seeing notable advancements in machine learning (ML) and artificial intelligence (AI). Advanced statistical analysis, predictive modelling, and the automation of repetitive processes are made possible by these technologies [28]. In order to provide tools for improved data analysis, security threat detection, and operational efficiency, cloud providers are incorporating AI and ML capabilities into their platforms. The development of edge computing is another noteworthy advance.

It is anticipated that this change would enhance conventional cloud models by generating hybrid solutions that capitalise on the advantages of cloud and edge computing. Java's function in cloud computing is changing as the language adjusts to new possibilities and obstacles. Java is well suited for creating cloud-native apps due to its robust ecosystem and platform neutrality. Because of its continuous development and strong community support, Java is anticipated to continue to play a significant role in the advancement of cloud technologies [29, 30]. Deeper integration with orchestration and containerisation technologies is one of Java's future cloud computing themes. It is anticipated that Java's integration with Docker and Kubernetes will improve, making it easier to create and implement microservices-based systems. Furthermore, Java's speed and efficiency

improvements will help meet the expanding needs of cloud systems. Java's emphasis on microservices and modularity fits in well with cloud computing's future trajectory [30], where apps are increasingly being created as collections of loosely linked services. The Java platform's ongoing development, which includes language and runtime changes, will guarantee that it stays current and able to handle the complexity of contemporary cloud infrastructures.

CONCLUSION

Although serverless computing has a lot of promise for deploying applications in a scalable and economical manner, a number of performance issues must be resolved before its full potential can be realised. Developers can greatly improve the performance of their serverless applications by putting the optimisation strategies covered in this article into practice, including optimising execution environments, reducing cold starts through provisioned concurrency and function warming, strategically caching, and network optimisation.

Platforms for serverless computing provide a wide range of advantages, from cost effectiveness to automated scaling. However, a thorough assessment of their performance in cloud settings is necessary to realise their full potential. Performance measurements act as the compass in this environment, pointing organisations and developers in the direction of producing smooth, effective, and very responsive products. A fundamental paradigm change in the conception, development, and implementation of programs is represented by serverless computing. Its fundamental ideas, which include cost-effectiveness and autonomous scalability, have not only altered but also completely reshaped the conventional cloud computing environment. The success of an application in terms of performance, the cost-effectiveness and scalability is strongly impacted by the choice of serverless platform.

In conclusion, using Java to use cloud computing for corporate applications provides a strong method for creating dependable and scalable systems. Java's strong ecosystem and platform freedom make it possible to develop cloud-native apps that effectively satisfy the needs of contemporary businesses. Businesses may increase their flexibility, robustness, and efficiency by using microservices and distributed computing. In addition to maximising resource use, this integration encourages creativity and adaptability, giving businesses the ability to maintain their competitiveness in the ever-changing digital market. Java continues to be an essential tool for creating dynamic and reliable solutions as businesses continue to use cloud technology.

REFERENCES

[1] M. Shahrad et al., "Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," in Proceedings of the 2020 USENIX Annual Technical Conference, 2020, pp. 205-218.

[2] W. Lloyd et al., "Serverless Computing: An Investigation of Factors Influencing Microservice Performance," in 2018 IEEE International Conference on Cloud Engineering (IC2E), 2018, pp. 159-169.

[3] K. Figiela et al., "Performance evaluation of heterogeneous cloud functions," Concurrency and Computation: Practice and Experience, vol. 30, no. 23, p. e4792, 2018.

[4] Baldini et al., "Serverless Computing: Current Trends and Open Problems," in Research Advances in Cloud Computing, S. Chaudhary, G. Somani, and R. Buyya, Eds. Singapore: Springer, 2017, pp. 1-20.

[5] Palade, A. Kazmi, and S. Clarke, "An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge," in 2019 IEEE World Congress on Services (SERVICES), Jul. 2019. Accessed: Nov. 04, 2023.

[6] L. Baresi and D. Filgueira Mendonca, "Towards a Serverless Platform for Edge Computing," in 2019 IEEE International Conference on Fog Computing (ICFC), Jun. 2019. Accessed: Nov. 04, 2023.

[7] Ivan, Vasile, and Dadarlat, "Serverless Computing: An Investigation of Deployment Environments for Web APIs," Computers, vol. 8, no. 2, p. 50, Jun. 2019.

[8] Taibi, D.; El Ioini, N.; Pahl, C.; Niederkofler, J.R.S. Patterns for Serverless Functions (Function-as-a-Service): A Multivocal Literature Review. In Proceedings of the 10th International Conference on Cloud Computing and Services Science, CLOSER 2020, Prague, Czech Republic, 7–9 May 2020.

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

[10] Shekhar, S.; Gunturi, V.; Evans, M.R.; Yang, K. Spatial big-data challenges intersecting mobility and cloud computing. In Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access, Scottsdale, AZ, USA, 20 May 2012; pp. 1–6.

[11] Crespo-Cepeda, R.; Agapito, G.; Vazquez-Poletti, J.L.; Cannataro, M. Challenges and Opportunities of Amazon Serverless Lambda Services in Bioinformatics. In Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Niagara Falls, NY, USA, 7–10 September 2019; pp. 663–668.

[12] Niu, X.; Kumanov, D.; Hung, L.H.; Lloyd, W.; Yeung, K.Y. Leveraging serverless computing to improve performance for sequence comparison. In Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, Niagara Falls, NY, USA, 7–10 September 2019; pp. 683–687.

[13] Kim, Y.; Lin, J. Serverless data analytics with flint. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 451–455.

[14] Ishakian, V.; Muthusamy, V.; Slominski, A. Serving deep learning models in a serverless platform. In Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, USA, 17–20 April 2018; pp. 257–262.

[15] Anand, S.; Johnson, A.; Mathikshara, P.; Karthik, R. Real-time GPS tracking using serverless architecture and ARM processor. In Proceedings of the 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Bangalore, India, 7–11 January 2019; pp. 541–543.

[16] Malawski, M.; Gajek, A.; Zima, A.; Balis, B.; Figiela, K. Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and Google cloud functions. Future Gener. Comput. Syst. 2017.

[17] Varghese, B.; Buyya, R. Next generation cloud computing: New trends and research directions. Future Gener. Comput. Syst. 2018, 79, 849–861.

[18] Lee, H.; Satyam, K.; Fox, G. Evaluation of production serverless computing environments. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 442–450.

[19] Goodchild, M.F.; Fu, P.; Rich, P. Sharing geographic information: an assessment of the Geospatial One-Stop. Ann. Assoc. Am. Geogr. 2007, 97, 250–266.

[20] Shashi, S. Spatial Databases; Pearson Education: Bengaluru, India, 2007.

[21] Yang, C.; Li, W.; Xie, J.; Zhou, B. Distributed geospatial information processing: Sharing distributed geospatial resources to support Digital Earth. Int. J. Digit. Earth 2008, 1, 259–278.

[22] Escamilla-Ambrosio, P.; Rodríguez-Mota, A.; Aguirre-Anaya, E.; Acosta-Bermejo, R.; Salinas-Rosales, M. Distributing Computing in the internet of things: Cloud, fog and edge computing overview. In NEO 2016; Springer: Berlin/Heidelberg, Germany, 2018; pp. 87–115.

[23] Y. Wadia, R. Udell, L. Chan, and U. Gupta, Implementing AWS: Design, Build, and Manage your Infrastructure: Leverage AWS features to build highly secure, fault-tolerant, and scalable cloud environments. Packt Publishing Ltd, 2019. [6] S. R. Goniwada, "Cloud Native Architecture and Design."

[24] S. Namasudra, D. Devi, S. Kadry, R. Sundarasekar, and A. Shanthini, "Towards DNAbased data security in the cloud computing environment," Computer Communications, vol. 151, pp. 539-547, 2020.

[25] R. Rybaric, Microsoft Power Platform Enterprise Architecture: Design tailor-made solutions for architects and decision-makers to meet complex business requirements. Packt Publishing Ltd, 2020..

[26] M. Gamallo Gascón, "Design of a container-based microservices architecture for scalability and continuous integration in a solution crowdsourcing platform," Telecommunications, 2019.

[27] [14] I. E. Akkus et al., "SAND: Towards High-Performance serverless computing," in Proc. 2018 Usenix Annual Technical Conference (USENIX ATC 18), Boston, USA, Jul. 2018, pp. 173-185.

[28] V. Goar and N. S. Yadav, "Exploring the World of Serverless Computing: Concepts, Benefits, and Challenges," in Serverless Computing Concepts, Technology and Architecture, IGI Global, pp. 51-73, 2020.

[29] G. A. S. Cassel et al., "Serverless computing for Internet of Things: A systematic literature review," Future Generation Computer Systems, vol. 128, pp. 299-316, 2020.

[30] S. O. Olabanji, "Advancing cloud technology security: Leveraging high-level coding languages like Python and SQL for strengthening security systems and automating top control processes," Journal of Scientific Research and Reports, vol. 29, no. 9, pp. 42-54, 2020