Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 2 (2023)

A Study of Triggering Serverless Functions in Serverless Environment

Shashank Srivastava[1],Bineet Kumar Gupta[2]

[1,2]IOT,DCSIS,Shri Ramswaroop Memorial University, Lucknow, Uttar Pradesh, India,225003

Abstract:

Serverless computing has gained significant popularity due to its ability to provide scalable and cost-efficient cloud services. However, the adoption of serverless functions comes with its own set of challenges. One critical aspect is the triggering mechanism that initiates the execution of serverless functions in a serverless environment. This research paper explores the various triggering-related issues that can arise when deploying serverless functions and proposes potential solutions to mitigate these problems. The paper also highlights the importance of trigger design and its impact on the overall performance and reliability of serverless applications.

Keywords: Serverless Computing, AWS, AWS Labmda, Cloud Computing,Serverless Function

1. Introduction

Serverless computing **[8][9],** also known as Function as a Service (FaaS), is a cloud computing model that allows developers to run applications without managing the underlying infrastructure. In a serverless environment, developers can focus solely on writing and deploying code in the form of small, event-driven functions, without concerning themselves with server provisioning, scaling, or maintenance. These functions are executed in response to specific events or triggers, providing a pay-as-you-go approach, where users are billed only for the actual computing resources consumed during function execution.

Serverless computing has gained popularity for its ability to simplify application development, reduce operational overhead, and offer cost-efficiency by automatically scaling the resources based on demand. However, despite its advantages, serverless computing introduces unique challenges, particularly when it comes to triggering serverless functions.

In a serverless environment, triggers act as the catalysts that initiate the execution of serverless functions. These triggers can be various types of events, such as HTTP requests, database changes, file uploads, message queues, or timers. When an event occurs, the corresponding serverless function is automatically invoked, processing the event and generating the desired output.

Figure 1: Serverless Computing Importance

2. Triggering-Related Issues in Serverless Functions:

2.1 Cold Start Problem: The cold start problem **[4][5][6]** is a significant challenge in serverless computing. When a serverless function is triggered for the first time or after a period of inactivity, the cloud provider needs to initialize a new container or compute instance to run the function. This initialization process introduces latency and can result in a delay before the function starts executing. The delay caused by cold starts can impact application responsiveness and user experience, especially in real-time or interactive applications.

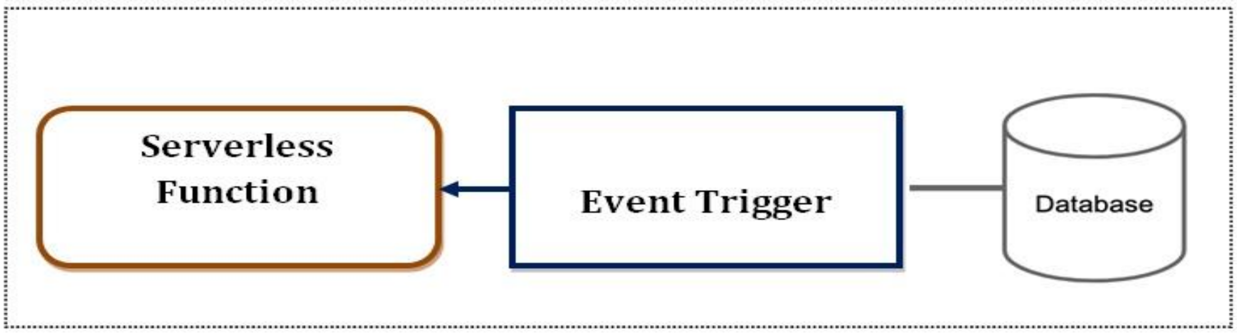

Figure 2: Triggering of Serverless Function

Mitigation Strategies:

- **Provisioned Concurrency**: Some serverless platforms offer provisioned concurrency, allowing developers to pre-warm function instances to reduce cold start delays.
- **Warm-Up Triggers:** Implementing warm-up triggers or scheduled executions can keep the function instances warm and reduce the impact of cold starts during actual usage.
- **Optimized Code and Dependencies:** Reducing the size and complexity of function code and dependencies can speed up container initialization and mitigate cold start delays.

2.2 Event Overloading: Event overloading**[4][5][6]** occurs when the volume of incoming events exceeds the capacity of the serverless platform to handle them efficiently. This can lead to function invocations being delayed, dropped, or processed with increased latency. Event overloading can be a result of unexpected spikes in incoming events or inefficient triggering configurations.

Mitigation Strategies:

- **Throttling and Load Balancing:** Implement throttling mechanisms or load balancing strategies to distribute events evenly across multiple instances of the serverless function.
- **Auto-scaling:** Configure the serverless platform to automatically scale resources based on incoming event rates to handle sudden spikes in traffic effectively.
- **Queuing:** Use message queues to buffer incoming events, allowing the serverless functions to process them at a controlled rate.

2.3 Trigger Misconfiguration: Improperly configured triggers can lead to unintended function invocations or failures to trigger the functions when needed. Trigger misconfiguration may result from incorrect event source configurations, wrong event formats, or misaligned permissions.

Mitigation Strategies:

- **Thorough Testing:** Rigorous testing of trigger configurations during development and deployment phases can help identify misconfigurations early on.
- **Monitoring and Alerting:** Implement monitoring and alerting systems to detect misfires or incorrect triggers and promptly address them.
- **Continuous Integration and Deployment (CI/CD):** Utilize CI/CD pipelines to automate trigger configuration checks during the deployment process.

2.4 Delayed Triggers: Delayed triggers occur when the serverless platform experiences internal delays in processing events and subsequently invoking the corresponding functions. Delays can be caused by platform resource contention, network latency, or other operational factors.

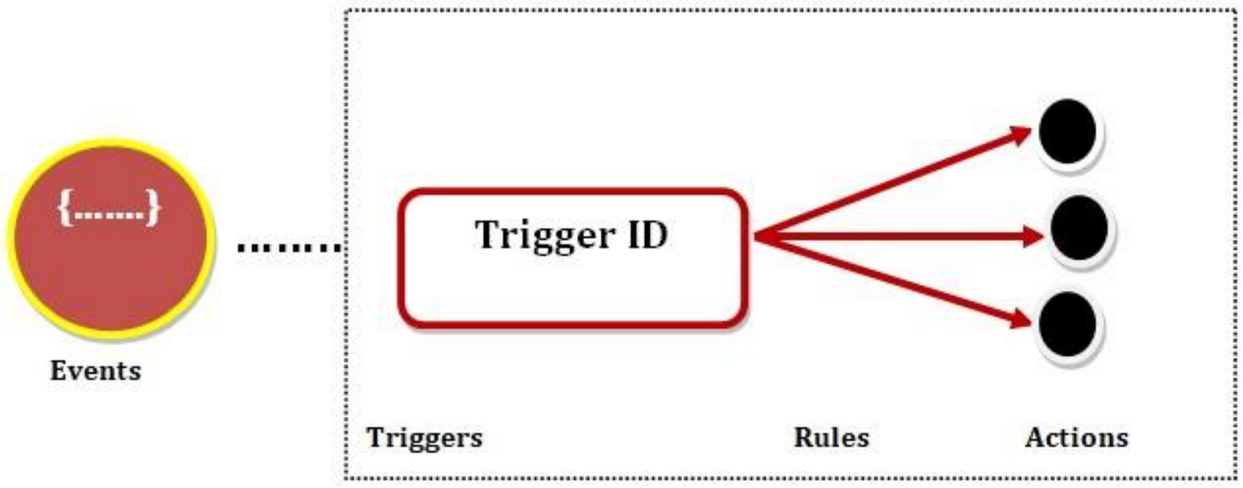

Figure 3: Triggering Mechanism in Serverless Computing

Mitigation Strategies:

- **Performance Monitoring:** Regularly monitor the serverless platform's performance to identify potential bottlenecks or areas of improvement that can reduce delays.
- **Asynchronous Processing:** Design functions to handle events asynchronously, allowing the serverless platform to process events more efficiently.

2.5 Resource Exhaustion: Serverless functions operate within resource limitations defined by the cloud provider. Resource exhaustion can occur when functions consume excessive memory, CPU, or other resources, leading to throttling or termination of the function.

Mitigation Strategies:

- Optimized Resource Allocation: Analyze function performance and resource usage patterns to allocate appropriate resources to each function.
- Fine-Tuning: Adjust function configurations, such as memory settings, to optimize resource usage and avoid unnecessary exhaustion.

By addressing these triggering-related issues in serverless functions, developers can create more resilient, responsive, and scalable serverless applications that meet the demands of modern cloud-based computing.

3. Impact on Performance and Scalability:

3.1 Latency and Response Time: Latency and response time are critical performance metrics for serverless functions. The time it takes for a function to start executing (cold start latency) and the time it takes to complete its processing (response time) directly impact application responsiveness and user experience. High latency can lead to delayed responses, affecting real-time applications, interactive user interfaces, and API endpoints.

3.1.1 Impact on Performance:

- **Cold Start Latency**: The cold start problem can introduce latency during the initialization of function instances, particularly after periods of inactivity or during sudden spikes in traffic. Mitigation strategies, such as provisioned concurrency and warm-up triggers, can help reduce the impact of cold starts on latency.

- **Execution Time:** The efficiency of function code and resource allocation directly affect the execution time. Optimized code and appropriate resource allocation can help reduce response time.

3.2 Scalability Concerns: Scalability is one of the key benefits of serverless computing. The ability to automatically scale resources based on demand allows applications to handle varying workloads effectively. However, certain factors can impact the scalability of serverless functions.

3.2.2 Impact on Scalability:

Throttling: When an application experiences a surge in incoming events, the serverless platform may throttle function invocations to prevent resource overload. This can lead to event backlogs and reduced application performance during high traffic.

Provisioned Concurrency: The number of provisioned function instances can limit scalability during peak times if not set appropriately.

Resource Utilization: Serverless functions share resources within the cloud provider's infrastructure. Efficient resource utilization is essential to achieve cost-effectiveness and maximize performance.

3.3 Impact on Resource Utilization:

Over-Allocation: Functions that consume excessive resources, either due to poorly optimized code or misconfiguration, can lead to unnecessary costs and reduced platform capacity.

Under-Allocation: Insufficient resource allocation can cause performance degradation, as functions may experience throttling or increased latency during peak periods.

3.2.3 Mitigation Strategies:

- **Auto-Scaling:** Configure the serverless platform to automatically scale resources based on demand, ensuring that functions can handle increased workloads efficiently.
- **Load Testing:** Conduct load testing to identify performance bottlenecks and determine appropriate resource allocation for different usage scenarios.
- **Performance M**onitoring: Implement monitoring and analytics to track latency, response time, and resource utilization in real-time, allowing for proactive optimization and issue resolution.

By carefully managing latency, addressing scalability concerns, and optimizing resource utilization, developers can ensure that serverless functions deliver optimal performance and scale seamlessly to meet the demands of the applications they support. A well-designed serverless architecture with proper consideration for these factors can lead to highly responsive and efficient cloud-based applications.

4. Case Studies:

4.1 Amazon Web Services (AWS) Lambda:

AWS Lambda **[1]** is a widely adopted serverless computing platform provided by Amazon Web Services. It enables developers to run code in response to various events without managing servers. Let's explore a case study of a real-world application deployed on AWS Lambda:

4.1.1 Case Study: Real-time Image Processing Service

Scenario: A photo app allows users to upload images and apply real-time filters and transformations. The app needs to handle a large number of concurrent users and process images instantly.

Solution with AWS Lambda:

- **Trigger:** S3 Object Upload

Whenever a user uploads an image to an S3 bucket, AWS Lambda is triggered by the S3 object upload event.

- **Image Processing Lambda Function:**

- A Lambda function is implemented to perform image processing tasks, such as applying filters and resizing images.
- The function uses popular image processing libraries like Pillow or ImageMagick.
- The function is optimized to minimize cold start latency by using provisioned concurrency during peak times.

- **Load Balancing and Auto-Scaling:**

AWS Lambda automatically scales the number of function instances based on the incoming event rate.

The function instances are distributed across multiple availability zones using Elastic Load Balancing for high availability and fault tolerance.

- **Monitoring and Optimization:**

AWS Cloud Watch is used to monitor function performance, latency, and error rates.

Performance insights help identify potential bottlenecks and optimize resource allocation.

Response: The photography app successfully handles a high number of concurrent uploads and delivers real-time image processing. AWS Lambda's auto-scaling capabilities ensure that the application remains responsive even during peak usage. With proper monitoring and optimization, the app achieves cost-efficient resource utilization.

4.2 Microsoft Azure Functions:

Azure Functions **[2]** is Microsoft's serverless computing offering, allowing developers to build event-driven applications without worrying about infrastructure management. Let's explore a case study showcasing the usage of Azure Functions:

4.2.1 Case Study: IoT Data Processing and Analytics

Scenario: A company deploys a large-scale Internet of Things (IoT) solution with sensors generating a vast amount of data. The data needs real-time processing and analysis to derive valuable insights.

Solution with Azure Functions:

Trigger: IoT Hub Message

The IoT Hub is configured as the event source, receiving data from sensors.

- **Data Processing and Analytics Functions:**

Multiple Azure Functions are used for different data processing tasks, such as data filtering, aggregation, and anomaly detection.

Functions are written in various languages, including C#, JavaScript, and Python, based on the specific requirements.

- **Event Hubs and Stream Analytics:**
- Processed data is sent to Azure Event Hubs for further processing and storage.
- Azure Stream Analytics is utilized to perform real-time analytics on the data streams.

- **Auto-Scaling and Resource Management:**
- Azure Functions automatically scales based on the incoming message rate from IoT sensors.
- Resource allocation is optimized to handle peak data processing loads efficiently.

Response: The IoT solution efficiently processes and analyzes data from numerous sensors in real-time. Azure Functions' auto-scaling ensures that the system adapts to varying data rates, providing near-instantaneous insights. The integration with other Azure services like Event Hubs and Stream Analytics enables seamless data flow and enables advanced analytics capabilities.

4.3 Google Cloud Functions:

Google Cloud Functions**[3]** is a serverless computing platform provided by Google Cloud Platform (GCP). It allows developers to write and deploy event-driven functions effortlessly. Let's explore a case study that showcases the power of Google Cloud Functions:

4.3.1 Case Study: Real-time Chat Application

Scenario: A real-time chat application requires handling real-time messaging between users, delivering instant notifications, and storing chat history for future retrieval.

Solution with Google Cloud Functions:

- **Trigger:** Firebase Realtime Database Changes

Firebase Realtime Database serves as the backend for the chat application, and Google Cloud Functions are triggered when new chat messages are added or updated.

- **Message Routing and Notifications:**
- Google Cloud Functions route incoming messages to the appropriate recipients, ensuring efficient message delivery.
- Firebase Cloud Messaging (FCM) is utilized to send instant notifications to users for new messages.
- Chat History Storage:

Processed messages are stored in Google Cloud Storage or Firestore, enabling easy retrieval and history access.

- **Auto-Scaling and Performance Monitoring:**

Google Cloud Functions automatically scales based on the incoming message rate, ensuring smooth and efficient chat message processing.

Google Cloud Monitoring provides real-time insights into function performance and usage.

Response: The real-time chat application handles message delivery and notifications seamlessly. Google Cloud Functions' auto-scaling capabilities ensure that the system efficiently processes and delivers messages to users, even during peak activity. The integration with Firebase services and Google Cloud Storage allows for a scalable and robust chat application.

These case studies demonstrate how serverless computing platforms like AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions can be leveraged to build scalable, responsive, and cost-efficient applications across diverse use cases. By utilizing the event-driven nature of serverless functions, these platforms empower developers to focus on application logic and functionality without worrying about infrastructure management.

5. Best Practices for Trigger Design

5.1 Trigger Selection Methodology: Selecting the appropriate trigger for a serverless function is crucial for the efficient and reliable operation of the application. Consider the following guidelines when choosing triggers:

- **Event Source Compatibility:** Ensure that the selected trigger is compatible with the event source generating the events. Different cloud providers offer various triggers, such as HTTP requests, database changes, message queues, timers, or object storage events.
- **Real-Time vs. Batch:** Choose triggers based on the nature of the application's workload. Real-time triggers, like HTTP requests or message queue events, are suitable for applications requiring immediate

processing. Batch triggers, such as scheduled timers or database changes, are more suitable for non-urgent, periodic, or large-scale data processing.

- **Scalability:** Consider triggers that can efficiently scale with the application's workload. Auto-scaling triggers are beneficial for handling varying event rates, preventing resource contention, and ensuring responsiveness.
- **Throttling and Rate Limiting:** Be mindful of the event rate that triggers generate and consider implementing throttling mechanisms or rate limiting to control the rate of incoming events and prevent overload.
- **Fault Tolerance**: Evaluate the fault tolerance capabilities of the chosen trigger. Ensure that the system can handle temporary failures, retries, and backoffs to avoid data loss or processing errors.

5.2 Real-Time vs. Batch Triggers: Choosing between real-time and batch triggers depends on the specific requirements of the application. Consider the following factors when making the decision:

5.2.1 Real-Time Triggers:

- Suitable for applications requiring immediate or near-real-time processing of events, such as real-time analytics, chat applications, or IoT data processing.
- Often triggered by user interactions or external systems generating events that need immediate response.
- Typically have lower latency requirements and should be optimized to minimize cold start delays.

5.2.2 Batch Triggers:

- Ideal for applications that can tolerate some delay in processing and can benefit from aggregating data for periodic or bulk processing.
- Suited for scenarios such as data batch processing, ETL (Extract, Transform, Load) jobs, or periodic report generation.
- May involve handling larger data volumes, so resource allocation and optimization are essential.

In some cases, a combination of both real-time and batch triggers may be appropriate, depending on the application's specific use cases.

5.3 Monitoring and Alerting: Monitoring and alerting are essential for proactively identifying issues and maintaining the health of the serverless application. Consider the following practices:

- **Real-Time Metrics:** Implement real-time monitoring of key performance metrics such as latency, response time, error rate, and resource utilization. Use cloud provider monitoring services or third-party monitoring tools to track these metrics.
- **Alerting Thresholds:** Set up appropriate alerting thresholds for performance metrics to trigger notifications when certain thresholds are breached. This enables timely response to performance issues.
- **Distributed Tracing:** Implement distributed tracing to gain visibility into the end-to-end flow of events and function invocations. This helps identify bottlenecks and performance hotspots.
- **Log Aggregation:** Aggregate logs from serverless functions and other components to provide a comprehensive view of the application's behavior. Analyzing logs can aid in debugging and troubleshooting issues.
- **Anomaly Detection:** Use anomaly detection techniques to automatically identify abnormal behavior or performance deviations and initiate appropriate responses.
- **Resource Monit**oring: Monitor resource usage, such as memory, CPU, and network bandwidth, to optimize resource allocation and avoid resource exhaustion.

6. Future Scope

The future of serverless computing **[8][9]** is promising, with vast scope for advancements. Integration with edge computing, application in IoT and AI/ML domains, and enhancements in security and developer tooling are some of the exciting areas of future exploration. As serverless computing continues to evolve, it will undoubtedly shape the way applications are developed and deployed, making cloud computing more accessible, efficient, and cost-effective.

7. Conclusion

serverless computing represents a transformative shift in cloud computing, empowering developers to build applications without worrying about infrastructure management. By understanding and addressing triggering-related issues, optimizing performance, and exploring future research directions, the serverless computing ecosystem will continue to evolve, driving innovation and efficiency in the cloud computing landscape. As more businesses and developers embrace serverless computing, it will undoubtedly shape the future of cloud-native applications, unlocking new possibilities and revolutionizing the way we interact with technology.

Acknowledgement

We would like to express our sincere gratitude to oursupervisor, Dr. Bineet Kumar Gupta, Associate Professor,DCSIS, IOT, SRMU Lucknow for his valuable guidance andsupport throughout the research process. His expertise andinsights were invaluable in shaping our research and helping usto overcome challenges. We also want to thank our SRMU University,Lucknow for their helpful feedback and support.

Reference:

[1] WS Lambda Documentation. (n.d.). Retrieved from https://aws.amazon.com/lambda/
[2] Azure Functions Documentation. (n.d.). Retrieved from https://docs.microsoft.com/en-us/azure/azure-functions/
[3] Google Cloud Functions Documentation. (n.d.). Retrieved from https://cloud.google.com/functions/
[4] Roberts, S., & McNutt, J. (2018). Serverless Architectures. O'Reilly Media.
[5] Wang, L., Von Laszewski, G., Kunze, M., & Tao, J. (2018). Serverless Computing: Current Trends and Open Problems. Journal of Cloud Computing: Advances, Systems and Applications, 7(1), 1-17.
[6] .Li, Q., & Shi, S. (2020). A Survey on Serverless Computing: Architectures, Deployment and Monitoring. IEEE Access, 8, 40582-40597.
[7] Mannersalo, P., & Hämäläinen, T. D. (2020). Survey of Serverless Computing: Architecture, Benchmark, and Challenges. In 2020 IEEE World Congress on Services (SERVICES).
[8] Rodero-Merino, L., Vaquero, L. M., Gil, V., & Galán, F. (2019). From Containers to Serverless: A Performance Study. In 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom).
[9] .Shahrad, M., Chen, W., Matin, S., Shah, M., & Buyya, R. (2020). A Comprehensive Review on Serverless Computing: From Concept to Implementation. Journal of Network and Computer Applications, 166, 102729.
[10] Botta, A., De Donato, W., Persico, V., & Pescapé, A. (2016). Integration of Cloud Computing and Internet of Things: A Survey. Future Generation Computer Systems, 56, 684-700.