Distributed Data Aggregation in Serverless Architectures

Surya Ravikumar

suryark@gmail.com

Abstract

The rapid growth of cloud computing has paved the way for the adoption of serverless architectures, which eliminate the need for developers to manage servers, thus improving scalability, cost-efficiency, and operational agility. One of the critical challenges in serverless computing is efficiently aggregating data across multiple distributed nodes and functions, especially in real-time data processing applications such as IoT, streaming analytics, and large-scale distributed systems. This paper explores the techniques and challenges related to distributed data aggregation in serverless architectures. In addition it examines the fundamentals of serverless computing, reviews existing data aggregation methods, and analyzes their applicability to serverless environments.

Keywords: Serverless Computing, Distributed Data Aggregation, Cloud Computing, Data Processing, Scalability, Cost Efficiency, Real-time Analytics

1. Introduction

Serverless architectures have revolutionized cloud computing by enabling developers to focus more on writing code and less on managing infrastructure. In this model, cloud providers automatically handle the allocation, scaling, and management of computing resources. Serverless functions, typically executed as part of a Function-as-a-Service (FaaS) model, are invoked on demand and automatically scale based on the workload. Popular cloud platforms like AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions have made serverless computing increasingly popular for event-driven applications, microservices, and data processing tasks.

However, the distributed and short lived nature of serverless functions presents unique challenges in data aggregation. Aggregating large amounts of data across multiple functions and services is essential for a variety of use cases, including real-time data analytics, Internet of Things (IoT) applications, and machine learning workflows. Traditional approaches to data aggregation, such as MapReduce or batch processing frameworks, are not well-suited for serverless environments due to the need for low-latency processing, dynamic scaling, and handling the statelessness of serverless functions.

This paper investigates the techniques and strategies for effective data aggregation in serverless architectures, focusing on challenges such as managing function state, optimizing data transfer, and achieving low-latency performance in distributed systems

2. Overview of Serverless Architectures

Serverless computing revolutionizes the way applications are built and deployed by removing the burden of infrastructure management from developers. In a serverless architecture, developers write code in the form of small, discrete functions that perform specific tasks. These functions are stateless, meaning they don’t retain any data between invocations, which allows them to be lightweight and fast. Serverless functions are typically triggered by events, such as an HTTP request, a message from a queue, or a file upload. This

event-driven nature allows developers to create highly modular and responsive applications without having to provision or manage servers.

One of the major advantages of serverless computing is its inherent scalability. Serverless platforms such as AWS Lambda, Azure Functions, and Google Cloud Functions automatically scale the number of function instances up or down based on incoming traffic or events. If a function is triggered thousands of times per second, the platform can handle the load without any manual intervention. Conversely, when demand drops, resources are automatically deallocated, which leads to cost savings, as users are only billed for the compute time they actually use. This elasticity makes serverless a great fit for applications with unpredictable or spiky workloads, such as real-time data processing, image transformation, or backend services for mobile and web applications.

Moreover, serverless computing pairs very well with cloud services, making it perfect for workflows that need to react to events. For example, if you upload a new image to Amazon S3, it can trigger a function that performs several actions: resizing the image, changing its format, and updating the database. Likewise, serverless functions can also handle events from message queues like Amazon SQS or Google Pub/Sub. This setup allows asynchronous processing of tasks like order processing or sending notifications to happen at different times without delay. Because of this strong connection with cloud systems, developers can create strong, scalable, and easy-to-maintain applications more quickly. They can concentrate on building business solutions without the hassle of managing the infrastructure.

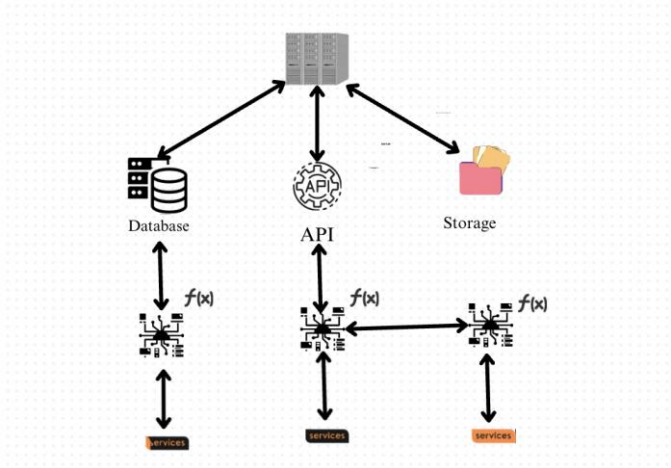

Figure 1: Serverless Architecture

While the serverless model offers numerous benefits such as reduced operational overhead, cost savings, and ease of scaling, it introduces challenges related to managing distributed tasks, maintaining state across invocations, and achieving high performance in data-heavy applications.

3. Distributed Data Aggregation: The Need in Serverless Systems

In many serverless applications, especially those that require processing large datasets or analyzing continuous data streams, data aggregation plays a critical role. Data aggregation involves collecting and combining data from various sources into a single, unified dataset to facilitate decision-making or analysis.
Serverless systems often require distributed data aggregation techniques to combine data from multiple sources, which might be distributed across different cloud functions, microservices, or even edge devices. Aggregation could include calculating sums, averages, applying filters, or executing more complex operations such as joins, group-by operations, and advanced analytics.

However, in a serverless environment, performing data aggregations poses significant challenges due to the architectural characteristics of serverless computing. These challenges stem from several core factors:

Statelessness: Serverless functions are inherently stateless, meaning that they are designed to execute in isolation without maintaining any memory or data across invocations. Each function starts with a clean slate and terminates shortly after execution. As a result, performing aggregations that require tracking ongoing state such as counting events over time, maintaining rolling averages, or consolidating session data becomes non-trivial. Developers must rely on external systems such as distributed databases (e.g; DynamoDB), in-memory data stores (e.g; Redis), or object storage (e.g; Amazon S3) to persist intermediate data, introducing additional complexity and potential points of failure.

Scalability: One of the primary benefits of serverless architectures is their ability to scale automatically in response to varying workloads. As demand increases, additional instances of functions are created to handle the load. However, this dynamic and decentralized scaling introduces challenges for aggregation. Coordinating between potentially hundreds or thousands of concurrently executing and short-lived function instances becomes a distributed systems problem. Ensuring consistency and avoiding duplication or data loss during aggregation requires careful design, often involving message queues (e.g; Amazon SQS), data streams (e.g; Kinesis, Kafka), or other forms of event-driven architecture to collect and process data in a structured and fault-tolerant manner.

Latency: Aggregating data in a distributed, serverless system often requires transferring information between services and regions over the network. This data movement can introduce significant latency, especially when dealing with real-time or near-real-time aggregation needs. The stateless and ephemeral nature of functions means they must fetch any required context or state from remote storage, further adding to execution time. Additionally, geographical dispersion of cloud functions can compound latency issues, particularly when functions are triggered by globally distributed sources such as IoT devices or edge services.

Overall, while serverless architectures offer powerful benefits in terms of scalability, cost-efficiency, and simplicity of deployment, they require careful architectural planning and external tooling to effectively handle data aggregation tasks in a reliable and performant manner.

4. Challenges in Distributed Data Aggregation for Serverless Architectures

Aggregating data in serverless environments introduces a unique set of challenges due to the distributed, stateless, and ephemeral nature of serverless functions. Unlike traditional architectures where persistent resources and centralized control are available, serverless architectures require new paradigms for managing data aggregation at scale. This section outlines key challenges encountered when implementing distributed data aggregation in serverless systems.

4.1 State Management

Serverless functions are designed to be stateless, meaning they do not retain any context or memory between invocations. This architectural trait, while beneficial for scalability and fault tolerance, poses a significant hurdle for data aggregation, which inherently requires tracking and maintaining state across time. For example, calculating a running total, session-based metrics, or temporal trends requires the function to remember previous data points.

To address this, developers must rely on external stateful services, such as distributed caches (e.g; Redis), NoSQL databases (e.g; Amazon DynamoDB), or object storage systems (e.g; S3). While effective, this approach introduces additional latency, complexity, and cost. Frequent reads and writes to these external systems can become bottlenecks, especially under high throughput conditions. Moreover, ensuring the consistency and reliability of external state across concurrent function executions is non-trivial and may require additional synchronization mechanisms.

4.2 Event Ordering and Consistency

In distributed, event-driven systems, event order is not guaranteed. Events may arrive out of order, be delayed, or even duplicated due to retries. For aggregation tasks that depend on the sequence of data such as time-series analytics or cumulative metrics processing events in the correct order is critical to ensuring accuracy and consistency.

Maintaining a consistent view of data across multiple concurrent functions becomes increasingly difficult as the number of invocations grows. Developers must often implement custom ordering logic, deduplication strategies, or leverage timestamp-based sorting mechanisms. In scenarios where eventual consistency is not acceptable such as financial transactions or critical monitoring systems these issues can compromise the integrity of the aggregation results.

4.3 Network Latency and Communication Overhead

Serverless functions typically run in isolated, ephemeral environments, and often execute independently of one another. Aggregating data across multiple functions requires significant inter-function communication, which introduces both network latency and overhead. This is particularly pronounced in systems deployed across multiple regions or availability zones, where cross-region data transfer can further degrade performance.

Additionally, many serverless applications are event-driven, with functions triggered asynchronously by queues, streams, or API gateways. While this architecture supports high throughput, it also results in increased communication complexity, as data must be routed, serialized, and deserialized across services. This not only affects latency but also contributes to resource consumption and increased billing costs.

4.4 Cost and Performance Trade offs

The pay per use model of serverless computing is cost-effective for sporadic or lightweight workloads, but can become expensive for data-intensive operations such as large-scale aggregation. Costs accumulate based on execution time, memory allocation, and the number of function invocations. Aggregating vast datasets or running complex calculations can result in long-running functions that consume significant memory and CPU resources.

Moreover, external data access such as querying a database or fetching files from object storage adds further costs and may also extend execution times. Developers must carefully balance performance optimization with cost efficiency, often requiring fine-tuning of memory settings, batching of data, or rearchitecting functions to limit the frequency and duration of invocations.

4.5 Data Sharding and Partitioning

To scale data aggregation across large datasets, it is common to divide data into smaller chunks through techniques like sharding or partitioning. In a serverless context, this allows individual functions to process a portion of the data in parallel. However, implementing efficient sharding strategies in serverless environments is challenging.

Data must be partitioned in a way that evenly distributes workload across function instances while minimizing cross-shard dependencies. Skewed partitions where some shards contain significantly more data than others can lead to resource contention or stragglers, slowing down the overall aggregation process. Ensuring optimal partitioning often requires metadata tracking, pre-processing steps, or dynamic workload balancing all of which introduce added complexity to the system design.

5. Existing Techniques for Data Aggregation in Serverless Architectures

To address the challenges inherent in serverless computing, several techniques have emerged that enable efficient and scalable data aggregation. These techniques often involve combining serverless functions with complementary technologies to overcome limitations such as statelessness, network latency, and inconsistent event ordering. Below are some commonly used approaches.

5.1 Function Chaining

Function chaining refers to the process of linking multiple serverless functions together in a sequential workflow, where the output of one function becomes the input of the next. Each function is responsible for a distinct stage in the aggregation pipeline for example:The first function might filter or clean raw data, the second performs transformation or mapping,the final function executes the reduction or aggregation logic. This modularity allows for separation of concerns, making each function simpler and easier to manage or scale independently. However, function chaining can be inefficient due to several factors:

Invocation overhead: Each step in the chain involves invoking a separate function, which introduces additional latency and cost.

Data serialization: Intermediate results must be serialized and passed between functions, often via temporary storage or message queues.

Error propagation: Failures in any step may require re-execution of the entire chain or careful orchestration to ensure idempotency. Tools like AWS Step Functions, Azure Durable Functions, or Google Cloud Workflows can help orchestrate chained workflows while managing state, retries, and execution history.

5.2 External State Management with Distributed Storage

To compensate for the statelessness of serverless functions, developers frequently rely on external state management using distributed storage systems. Common options include:Amazon S3 for large, persistent object storage, Amazon DynamoDB for low-latency, key-value access to intermediate results, Redis for high-speed, in-memory caching of frequently accessed data.

By externalizing state, functions can read the current aggregation state, update it, and write back the results without maintaining internal memory. This model is especially useful for cumulative aggregations, such as:Counting page views, tracking user activity across sessions,building time-series rollups.

However, this approach introduces I/O latency and requires careful design to avoid race conditions, particularly when multiple functions attempt to update the same state concurrently. Techniques such as conditional writes, versioning, and locking are often employed to ensure data consistency.

5.3 Event-Driven Aggregation with Pub/Sub Systems

In systems where data is produced continuously such as IoT telemetry, clickstreams, or logs, event-driven architectures provide a powerful mechanism for real-time aggregation. Here, publish-subscribe (Pub/Sub) systems act as intermediaries, decoupling data producers and consumers:Data sources publish events to a topic or stream, serverless functions are automatically triggered when new data arrives. Technologies such as AWS SNS/SQS, Amazon Kinesis, Apache Kafka, and Google Cloud Pub/Sub support high-throughput, fault-tolerant event delivery. This enables parallel processing and incremental aggregation as data flows through the system.

For example, a set of Lambda functions can: Consume events from a Kinesis stream,compute partial aggregates (e.g; per-minute statistics), store results in DynamoDB or S3 for downstream consumption. This pattern supports low-latency, scalable, and highly available data pipelines, especially when paired with windowing or buffering strategies to reduce the frequency of writes.

5.4 Edge Computing for Localized Aggregation

In use cases involving distributed data sources, such as IoT devices, connected vehicles, or mobile applications, performing aggregation at the edge where data is generated can significantly improve performance and reduce network overhead.

Edge computing enables localized aggregation before transmitting summarized data to the cloud. For example: A fleet of IoT sensors can locally calculate temperature averages or event counts,edge nodes can preprocess video feeds to detect anomalies, mobile apps can group and compress user interactions before syncing.
This approach reduces the volume of data sent over the network, minimizes cloud storage and compute costs, and improves responsiveness for real-time decision-making. Services like AWS Lambda@Edge, Azure IoT Edge, and Cloudflare Workers support these scenarios.

Edge aggregation is particularly useful when:

- Devices operate in intermittent connectivity environments.

  In many edge scenarios such as rural IoT deployments, mobile sensor networks, or industrial machinery devices frequently lose connection to the cloud due to limited infrastructure or environmental constraints. Edge aggregation allows data to be locally processed and summarized even when there's no active cloud connection. This reduces the risk of data loss and enables buffered uploads of aggregated results once connectivity is restored.

- Real-time feedback is required at the data source.

  In use cases like autonomous vehicles, financial systems, robotics, industrial automation, or health monitoring, decisions must be made in real time based on incoming data. Transmitting raw data to the cloud introduces latency that can be unacceptable for time-critical decisions. Edge aggregation

enables immediate preprocessing, threshold detection, or local inference, triggering alerts or actions directly at the source.

- Bandwidth and latency constraints are critical.

  Transmitting raw data streams (e.g; video, high-frequency sensor logs) to the cloud can be expensive and slow, especially over cellular or satellite networks. Aggregating data at the edge reduces the volume of transmitted data, sending only relevant insights, summaries, or anomalies. This approach saves network bandwidth, lowers cloud processing costs, and reduces transmission delays.

These techniques function chaining, external state management, event-driven architectures, and edge computing offer flexible strategies to overcome the limitations of serverless platforms. Each has strengths suited to different types of aggregation tasks, and they are often used in combination to build robust, scalable, and cost-efficient data aggregation solutions.

6. Conclusion

Serverless architectures have rapidly gained traction in modern cloud computing due to their inherent benefits such as automatic scalability, fine-grained billing, and minimal infrastructure management. These characteristics make them highly attractive for building flexible, event-driven applications. However, despite their advantages, implementing distributed data aggregation in serverless environments remains a complex and ongoing challenge.

The primary obstacles stem from the stateless execution model of serverless functions, the latency introduced by distributed communication, and the difficulty of managing shared state across multiple function invocations. Traditional aggregation methods such as function chaining, external state storage, and event-driven workflows offer partial solutions but often involve trade-offs in performance, cost, and architectural complexity.

Looking forward, future research should explore: Adaptive orchestration techniques that dynamically adjust function workflows based on runtime metrics and data distribution patterns, Machine learning-driven optimizations for predicting workload hotspots and tuning sharding strategies in real time, Improved consistency models and streaming frameworks that are tightly integrated with serverless platforms to support high-throughput, low-latency data aggregation.

Continued exploration and refinement in these areas will be essential to fully unlocking the potential of serverless computing for data-intensive, real-time applications.

7. References

[1] Zeldovich, N; Kaashoek, M. F; & Morris, R. (2009). Serverless Computing: What You Need to Know. ACM Computing Surveys

[2] Huang, T; & Shah, M. (2018). Efficient Distributed Data Aggregation in Serverless Computing. IEEE Cloud Computing.

[3] Reinders, J; & Laroussi, S. (2019). Data Aggregation in Distributed Systems. Springer.

[4] Manoj Kumar, “Serverless Architectures Review, Future Trend and the Solutions to Open Problems.” American Journal of Software Engineering, vol. 6, no. 1 (2019): 1-10. doi: 10.12691/ajse-6-1-1

[5] Ritchie, S; & Lambert, A. (2020). Edge Computing and Distributed Data Aggregation. IEEE Transactions on Cloud Computing.

[6] Schleier-Smith, J; Sreekanti, V; Khandelwal,A; Carreira, J; Yadwadkar, N. J; Popa, R. A; Gonzalez, J. E; Stoica, I; & Patterson, D. A. (2021). What serverless computing is and should become. Communications of the ACM, 64(5), 76–84. https://doi.org/10.1145/3406011