JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.5560-5567
https://publishoa.com
ISSN: 1309-3452

An Investigative Analysis on Server Less Computing for Large-Scale Data Processing Using Cloud-Native Primitives

Anita Mishra
Lecturer in Computer Science , Jhadeswar Institute of Engineering & Technology , Balasore
Email ID : anita.amishra@gmail.com

Received: 2022 March 15; **Revised**: 2022 April 20; **Accepted**: 2022 May 10

Abstract
New to the cloud computing scene is serverless computing, which offers a fresh take on application development and deployment without the traditional burden of infrastructure management. Serverless computing offers an attractive alternative to traditional approaches to large-scale data processing, with the advantages of reduced operational complexity, lower costs, and more scalability. We explore serverless computing as a means to analyze massive amounts of data using cloud-native primitives in this study. Developers who want to build apps for the cloud will find their job simplified by adopting a serverless computing architecture, which isolates all infrastructure management. Better resource usage and reduced costs are potential outcomes of serverless architecture, which is designed to handle fluctuating input loads on the system.
Keywords: Serverless Computing, Infrastructure, Resource, Cloud, Phase

I. Introduction

An attractive replacement for conventional server-based systems, serverless computing is quickly becoming a dominant paradigm in the cloud computing space. The promise of scalability, cost-effectiveness, and operational simplicity has led to a boom in its usage in recent years. When considering large-scale data processing, this tendency becomes even more apparent as businesses face the challenge of managing massive amounts of data produced by many sources. Serverless computing, which makes use of cloud-native primitives, is an interesting way to handle the difficulties of effectively processing such large datasets. When it comes to creating, deploying, and managing cloud apps, serverless computing is a game-changer. By eliminating the need for developers to worry about the underlying infrastructure in serverless computing, developers are free to concentrate on creating code that implements business logic, as opposed to the conventional server-based designs that need them to provide, scale, and manage infrastructure resources. By removing the need for human intervention in resource provisioning and scaling, this strategy allows cloud providers to dynamically assign and manage resources depending on the demand caused by incoming requests. The concepts of cloud-native computing, which stress the need of development and operation of cloud-native apps being agile, scalable, and resilient, are well-aligned with this serverless method.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.5560-5567
https://publishoa.com
ISSN: 1309-3452

Serverless computing is based on the idea of Function as a Service (FaaS), which lets programmers put small bits of code or functions into the cloud without worrying about the underlying infrastructure. Executed in stateless, transient containers provided and controlled by the cloud provider, these functions are triggered by certain events or requests like HTTP requests, database modifications, or file uploads. Software as a service (SaaS) allows for granular scalability by separating code from infrastructure. This allows for the dynamic allocation of resources to accommodate different workloads, which optimizes resource consumption and minimizes costs.

Managed databases, storage services, and event-driven architectures are just a few examples of the cloud-native primitives that have catapulted serverless computing to new heights. These tools enhance the serverless paradigm and make it easier to integrate with preexisting cloud ecosystems. These primitives provide a solid groundwork for large-scale data processing activities, allowing for the construction of resilient and scalable data processing pipelines capable of effortlessly handling petabytes of data. Companies may streamline development and reduce time-to-market by orchestrating complicated data processing processes using managed services like AWS Lambda, Google Cloud Functions, and Azure Functions.

There are a number of issues and trade-offs associated with serverless computing that must be carefully considered in light of its possible advantages. A major obstacle is cold start delay, which occurs when a function container is initialized and causes latency overheads. This is particularly true for functions that are not visited often or for workloads that are bursty. Data consistency and robustness may be compromised by using external state management or different architectural patterns due to the transient nature of serverless functions, which makes them unsuitable for stateful processing activities or those that operate for an extended period of time. Moreover, debugging, monitoring, and performance optimization might be made more difficult due to a lack of control and visibility over the underlying infrastructure. To alleviate operational complexity, strong tools and best practices are required.

For businesses looking to process, analyze, and extract insights from huge datasets, serverless computing is a great alternative due to its many appealing features in the context of big data processing. Organizations may construct cost-effective data processing pipelines that can adapt to changing workloads by using cloud-native primitives like managed storage, databases, and event-driven architectures. Serverless computing's pay-as-you-go pricing model also helps businesses save money by letting them tie expenses to real use rather than an initial investment in infrastructure.

II. Review Of Literature

Nazari, Maziyar et al., (2021) Academics and corporations alike are taking an interest in serverless computing as a new paradigm in cloud computing, with many offering suggestions for improvements and creating excellent programs to use it. Popular public cloud providers that provide Function-as-a-Service in addition to their Serverless Computing platforms include IBM, Microsoft Azure, Google Cloud, and Amazon Web Services (AWS). Despite

the many advantages it has given to software engineers and programmers, this paradigm change has now introduced new challenges for cloud providers. The platform's pay-as-you-go concept and granular pricing structure make it perfect for developers as they alleviate the stress of managing resources. The capacity to conduct embarrassingly parallel processes, fault tolerance, auto-scaling, highly available service, and elasticity are all promised. On the other hand, cloud providers are confronted with fresh challenges when it comes to managing resources effectively, delivering low-latency service, and guaranteeing proper security and isolation. In this work, we survey the literature on optimizations and extensions that people have proposed and implemented for current Serverless platforms, with the goal of making the most of the Serverless infrastructure. The paper will conclude by discussing the current limitations of the Serverless paradigm and outlining some possible future directions and research possibilities associated with Serverless Computing.

Eismann, Simon et al., (2020) Benefits claimed by the serverless computing paradigm for cloud applications include low-cost deployment, fine-grained control, and operation without administration, among others. Consequently, the paradigm has seen rapid growth; dozens of serverless systems are in use, and all major cloud providers now support it. In this paper, we lay out the details of a long-term effort to find, collect, and define 89 serverless use cases; this will help with adjusting existing platforms, guide the creation of new serverless alternatives, and deepen our knowledge of this paradigm. Our research for this survey's use cases is based on interviews with domain experts in areas such as scientific computing and reviews of relevant white and grey literature. Every use case is evaluated using a comprehensive set of 24 criteria that address both high-level concepts and more nuanced details, such as workload, application, and requirements. When it comes to processes, we go even farther into the features of the use cases. Finally, we hope that the academic and corporate communities find our work useful, and we encourage everyone engaged to continue sharing and presenting their use cases.

Werner, Sebastian et al., (2018) Cloud computing as a service, or serverless computing, is an alternate model to traditional on-demand models. Unlike traditional cloud consumption, which involves directly accessing the cloud infrastructure, function services are executed by a FaaS provider. Is processing large data with serverless computing an efficient, scalable, and cost-effective option? As an example, we go into this study area in this work using matrix multiplication. In this paper, we present a FaaS-based matrix multiplication prototype, describe the needs for a serverless big data application architecture, and analyze and synthesize findings from extensive experiments. We show that serverless big data processing has the potential to lower operational and infrastructure costs without compromising system quality. When it comes to speed and scalability, serverless computing may even outperform distributed compute frameworks that rely on clusters.

Perez, Alfonso et al., (2018) Thanks to innovations in cloud computing, such as auto-scaling, and new architectural patterns like microservices, developers have been able to break large, complex systems down into smaller, stateless services. Programs may now be defined as a series of event-triggered functions, thanks to serverless computing made feasible by cloud

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.5560-5567
https://publishoa.com
ISSN: 1309-3452

providers. However, serverless platforms such as AWS Lambda impose significant limitations on these applications by making it impossible to create and deploy alternative libraries or by requiring users to stick to a certain set of programming languages. Presented here is a methodology and framework for developing Serverless Container-aware Architectures (SCAR) to address these issues. With the SCAR framework, developers can build event-driven serverless apps with high parallelism by utilizing Docker images on top of AWS Lambda. The essay utilizes a large-scale image processing use case to illustrate SCAR's architecture and the cost-saving usage of cache-based improvements. Based on the findings, SCAR transforms AWS Lambda into an HTC platform that is ideal for high-parallelism; burst workloads that have brief stateless operations.

III. Experimental Setup

Metrics overview

When optimizing end-to-end MapReduce execution and finding the optimal configuration for serverless instances, we leverage a number of Key Performance Indicators (KPIs):

- Invocation requests - Incoming requests for Lambda function execution
- Invocation duration - Time it takes for an invocation request to run
- Concurrency - Invocation requests * Time required to invoke How many requests your function can process in parallel is called concurrency.
- Initialization duration - The initialization code is executed and the function runtime is loaded every time the Lambda service creates a new instance of the function.

IV. Results And Discussion

A single query based on airline data, such as the top 10 airlines by on-time arrival performance, was implemented for the performance data collection phase. Ingesting, mapping, and reducing were the three separate stages of the implementation that were monitored and recorded.

Ingest

An S3 bucket was used to upload all 240 raw data files, which amounted to 34,870MB, during the ingest phase. It is possible to set up an S3 event trigger in a production environment to call the ingest function whenever a s3: Object Created event occurs. Nevertheless, in order to conduct performance testing, a regular workflow on AWS was set up to call the intake function simultaneously with each ingest Lambda function handling one of the raw data files.

Optimal function for lambda: To find the sweet spot for memory consumption when weighing performance against price, we ran tests on the ingest function with 1, 2, and 3 GB of RAM. As a function of memory setup, Lambda distributes CPU and other resources proportionally. We set up the intake function's parallel processes appropriately, estimating that at the CPU levels mentioned above, we had around 2, 2, and 3 vCPU available for processing. There is some available research suggesting that CPU ceilings could be somewhat varied.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.5560-5567
https://publishoa.com
ISSN: 1309-3452

Table 1 shows the results that were obtained by calling the function numerous times. By providing a substantial performance boost, the 2GB lambda function allows us to maintain the same cost as the 1GB lambda function. We also got a comparable workpoint when we used the power adjusting tool.

Table 1: Lambda usage

Processed	LambdaMemMB	DurationMs	MaxMemUsedMB	InitDurMs	CapMB	TotalMB
436950	1024	121133.14	415	500.28	135.0045462	34869.82844
436950	2048	88402.71	415	489.43	135.0045462	34869.82844
436950	4096	90243.8	415	518.14	135.0045462	34869.82844

Map

We found that the read/write overhead for the map phase was too high when creating separate S3 objects for each row in the CSV files. Alternatively, records were written each microbatch after being aggregated by partition key. One map invocation yields four outputs—one for each partition key—in this case, A, B, C, and D—in a micro batch of 100 total rows, where 30 rows are assigned to A, 30 rows to B, 30 rows to C, and 10 rows to D. The total delay caused by writing and reading that many objects or items in the reduction phase was lowered as a result. For instance, there were around 460k rows, with an average of 300 bytes per item, in a single 135MB CSV file. The item payload was decreased to 4,600 objects after the 100x reduction.

Reduce Gate

We started the reduction phase with the implementation of a counter gate.

{
“id”: “c3f72186-2466-4daa-ae81-beb3f80a305e”,
”ingested”: 1299481,
”mapped”:1299481
}

Prior to commencing the reduction aggregations and ranking, it is crucial to finish all map executions using the decrease gate. The reduction gate will verify that the ingested and mapped counters are equivalent for this job's execution ID. We are aware that a race situation may occur if the ingest count and the mapped count are equal before the map phase is fully finished. In order to deal with this, the ingest phase waits to publish its values until after it has finished, but the map phase writes data while it is doing its micro- batch processing.

Reduce Aggregation

The reduction per airline code was the first step in the reduction phase. For each airline code, this requires calling a function that reads all of the mapped outputs. If you use S3 with a prefix like [execution id]/[airline code]/[instance id], the reduce1 method will read all the

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.5560-5567
https://publishoa.com
ISSN: 1309-3452

objects in that prefix that are connected with one airline.formats such as 07b8c1d8-4e89-4969-83f3-72df580132f9/AA/01a63ff8-88e0- 4ab9-aa0b-1451c5cb6f0e.json were used.
The reduce1 function would get all items from a DynamoDB that are linked to the same structure when using DynamoDB. Instance ids served as sort keys and execution ids as hash keys in a DynamoDB table; a local secondary index with execution ids as hash keys and airline keys as sort keys was then used to achieve this.

Reduce Ranking

Part two of the reduction phase involves ranking the initial reduction aggregates by on-time arrival performance and then producing a list of the top ten airlines. Since this code only sorts the results after iterating over an array, it runs rather fast. On average, this function took 61 milliseconds, as we saw.

End-to-End

We conducted extensive performance testing using S3 as shuffle storage, adjusting the amount of parallel processing and memory settings for each Lambda function to achieve incremental improvements.

Performance analysis

Information on S3's performance as shuffle storage is shown in Table 2. Because DynamoDB experiences such severe throttling, its performance is not taken into account in this study. It is worth noting that the intake phase became less of a bottleneck when the system was subjected to increasing data loads, as the map began to experience increased strain. Add to it the fact that Reduce Aggregate, a function in reduce1, needs to read a large amount of mapped data in some instances. Table 3 shows that this phase's proportion of the total task execution increased as a consequence.

Table 2: Performance data of S3 as shuffle storage: Overall execution time

Scenario	Ingest (s)	ReducePrep (s)	ReduceGate (s)	ReduceAggregate (s)	ReduceRank (s)	Overhead (s)	Total (s)
1	92.19	0.84	1.13	14.13	0.98	0.37	109.64
2	63.10	0.82	1.71	13.27	0.89	0.47	80.25
3	54.01	0.08	20.50	12.60	0.08	0.49	87.77
4	55.40	0.40	11.20	13.27	0.49	0.39	81.15
5	58.57	0.50	62.02	162.68	0.51	0.40	284.70

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 3, 2022, p.5560-5567
https://publishoa.com
ISSN: 1309-3452

Table 3: Performance data of S3 as shuffle storage: Percentage of overall execution time per stage

Scenario	Ingest (%)	ReducePrep (%)	ReduceGate (%)	ReduceAggregate (%)	ReduceAggregate (%)	Overhead (%)
1	84.1	0.8	1.0	12.9	0.9	0.1
2	78.7	1.0	2.1	16.6	1.1	0.6
3	63.5	0.1	21.0	14.8	0.1	0.6
4	70.0	0.5	11.6	16.7	0.6	0.5
5	20.6	0.2	21.8	57.1	0.2	0.1

V. Conclusion

Serverless computing has the capacity to revolutionize large-scale data processing in today's cloud systems by providing scalability, cost-efficiency, and operational simplicity. To stay ahead in today's data-driven market, enterprises need to embrace cloud-native primitives and architectural patterns. With serverless computing, they can drive innovation, get new insights, and gain a competitive advantage. Serverless computing's capabilities and suitability for large-scale data processing activities will surely be enhanced by future study and breakthroughs in the industry as it continues to mature.

References: -

[1] Dean Jeffrey, Ghemawat Sanjay “Mapreduce: simplified data processing on large clusters,” Commun. ACM, 51 (1) (2008), pp. 107-113

[2] Eismann, Simon & Scheuner, Joel & Eyk, Erwin & Schwinger, Maximilian & Grohmann, Johannes & Herbst, Nikolas & Abad, Cristina & Iosup, Alexandru. (2020). A Review of Serverless Use Cases and their Characteristics.

[3] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. “Occupy the Cloud: Distributed Computing for the 99%.” In Proceedings of the 2017 Symposium on Cloud Computing (SoCC’17). ACM, New York, NY, USA, 445–451

[4] Jangda, Abhinav & Pinckney, Donald & Baxter, Samuel & Devore-McDonald, Breanna & Spitze, Joseph & Brun, Yuriy & Guha, Arjun. (2019). Formal Foundations of Serverless Computing.

[5] K. Shvachko, H. Kuang, S. Radia and R. Chansler, “The Hadoop Distributed File System,” 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010, pp. 1-10, doi: 10.1109/MSST.2010.5496972

[6] L. Mai, E. Kalyvianaki, and P. Costa, “Exploiting time-malleability in cloud-based batch processing systems, ” in Proc. Int’l Workshop on Large-Scale Distributed Systems and Middleware (LADIS), Farmington, PA, USA, Nov. 2013.

[7] M. M. Rahman and M. Hasibul Hasan, “Serverless Architecture for Big Data Analytics,” 2019 Global Conference for Advancement in Technology (GCAT), 2019, pp. 1-5, doi: 10.1109/GCAT47503.2019.8978443.

[8] Nazari, Maziyar & Goodarzy, Sepideh & Keller, Eric & Rozner, Eric & Mishra, Shivakant. (2021). Optimizing and Extending Serverless Platforms: A Survey. 1-8. 10.1109/SDS54264.2021.9732138.

[9] Perez, Alfonso & Moltó, Germán & Caballer, Miguel & Calatrava, Amanda. (2018). Serverless computing for container-based architectures. Future Generation Computer Systems. 83. 10.1016/j.future.2018.01.022.

[10] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. ArpaciDusseau. 2016. "Serverless Computation with OpenLambda." In 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'16). USENIX Association, Denver, CO, 14–19

[11] Sebastian Werner, Jorn Kuhlenkamp, Markus Klems, Johannes M ¨ uller, ¨ and Stefan Tai. 2018. "Serverless Big Data Processing Using Matrix Multiplication as Example." In Proceedings of the IEEE International Conference on Big Data (Big Data'18). IEEE, Seattle, WA, USA, 358–365.

[12] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-aService) Platforms," Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom, vol. 2017–Decem, pp. 162–169, 2017.

[13] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly and S. Pallickara, "Serverless Computing: An Investigation of Factors Influencing Microservice Performance," 2018 IEEE International Conference on Cloud Engineering (IC2E), 2018, pp. 159-169, doi: 10.1109/IC2E.2018.00039.

[14] Werner, Sebastian & Kuhlenkamp, Jorn & Klems, Markus & Muller, Johannes & Tai, Stefan. (2018). Serverless Big Data Processing using Matrix Multiplication as Example. 358-365. 10.1109/BigData.2018.8622362.