BIG DATA PROCESSING USING SERVERLESS COMPUTING

[1]Fernandes Macklon Roshvin, [2]Dr. Andhe Dharani
[1]PG Student, [2]Director and Professor,
[1,2]Department of Master of Computer Application,
[1,2]RV College of Engineering, Bengaluru, India.

Abstract: Big data is a term that refers to a significant amount of structured, semi-structured, or unstructured data. In today's business world, data is critical to the success of a company. Big Data projects and technologies are being utilised to examine this huge amount of data in order to gain insights that can aid in strategic decision-making. That is why managing such a big amount of data will be difficult and complex. In practise, there are several challenges to processing and calculating this data, including server administration, storage, clustering, algorithm deployment, and so on. Because everything is done manually in the Cloud, it's impossible to design the correct architecture for data analysis. Serverless computing is a method of providing clients with pay-per-use backend services. Serverless computing is a method of providing clients with pay-per-use backend services. Users can write and deploy programmes without having to worry about running and managing servers with a serverless service. This paper illustrates how to implement, operate, and control a serverless big data application on AWS, as well as how to provide a serverless architecture for big data processing (Amazon Web Service). In addition, in a serverless system, we will show the distinction between standard data analytics and big data analytics.

IndexTerms: Cloud Computing, Big Data, Amazon Web Services.

I. Introduction

Serverless platforms are available from cloud service providers such as Amazon, Google, and Microsoft. Although the scope or functionality of the services may vary, the basic principle behind the services is nearly the same, namely, that by converting computation to a pay-as-you-go model, it aims to accomplish auto-scaling while providing more affordable compute services. The Cloud service providers use Functions as a Service (FaaS) to run the code on their containers, and they connect with the Backend as a Service (BaaS) for Data Storage. A function can be triggered by a variety of sources, depending on the provider and accessible triggers, such as S3 events, SNS, and SQS. AWS Lambda, Google Cloud Functions, and Azure Functions are all Faas solutions from the big cloud providers, namely AWS, GCP, and Azure.

Serverless computing is a computational approach in which the cloud service provider is responsible for dynamically assigning resources and executing services (Lambda, Glue, etc.). Serverless Computing makes it easier to create, deploy, and manage cloud-scale applications. Instead of worrying about data infrastructures such server procurement, configuration, and management, a data engineer may concentrate on the duties required to assure a fully functional data pipeline from beginning to finish. It's also worth noting that, despite the fact that the cloud provider abstracts the architecture, servers are still involved in the code execution.

The architecture for big data analytics will be simpler, and it will also save money. It will also be demonstrated how to build, deploy, and operate big data analytics apps on AWS (Amazon Web Service) using serverless technology. It starts with AWS Lambda which batches the files and triggers the AWS Glue job for 7 concurrent tasks which process the batched data results which are called a transformation service where join, aggregation, and partitioning are performed where cost and performance are calculated. Then we also look at the impact of serverless computing on an organization and finally conclude it with a comparison with a traditional system.

II. Literature Work

Serverless computing has risen to prominence as a new platform for deploying apps and services. The widespread use of cloud technologies has resulted in cloud modeling techniques, frameworks, and platforms, demonstrating the high maturity of the technology. To identify crucial traits and use cases, as well as investigate technological obstacles and open problems, a study of existing serverless systems from enterprise, academia, and open-source initiatives was done. [1][6] Addresses major flaws in first-generation serverless computing, which put its autoscaling potential at conflict with current computing's major tendencies, including data-centric and distributed computing, as well as open source and bespoke hardware. [3][4] How migrating an app to the AWS Lambda deployment architecture reduced hosting costs by 66% to 95%, and how this trend can affect traditional software architecture design approaches. [2][5]

When a partitioned task operates a tiny function instance and also presents performance data for simultaneous invocations in conjunction with throughput, network bandwidth, I/O, and compute performance, current server-less computing systems may allow paralleled dynamic applications. [7][8]. As far as hosting implications go, infrastructure flexibility, load balancing, provisioning variability, infrastructure maintenance, and memory restriction size have all been explored. There were also four stages of serverless architecture identified: provider cold, VM cold, container cold, and warm, with microservice throughput ranging by up to 15x depending on these stages. [9][10]. The research was conducted to understand the design, implementation, performance and also current trends and open problems of serverless computing. The literature survey findings were implemented to come to conclusion and also in creating the below case study conducted to show the cost and performance of serverless computing in big data.

III. Transformation Service

This section provides an overview of the Big Data application that was built on AWS's serverless rail. Transformation service is the name of the application, and it uses Lambda architecture. The data source is used to process batch pipelines.

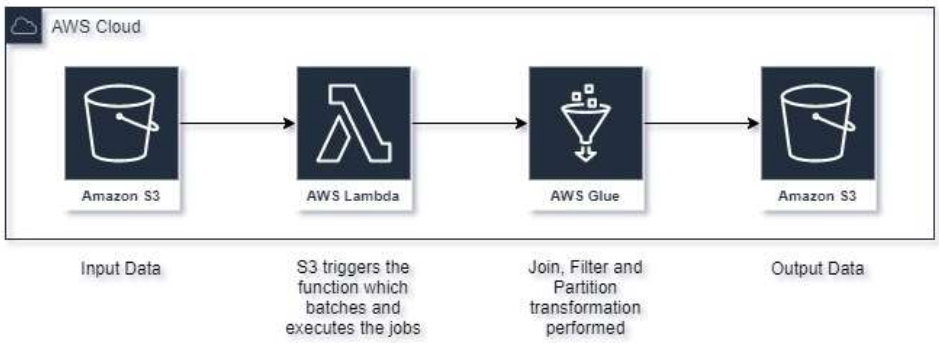

Fig 3.1 Transformation service Architecture

3.1 Approach

- **Extract** data of each resource type (Ex: Item, Replacement, Stock, and Demand) into the data frame from S3 and find the time taken for it.
- Perform following **Transforms**
  - left join all items from OEM resource types (Ex: item and replacement) which will give us Item and Replacement Details. Find the time taken for achieving this.
  - left join all items from OEM resource types (Ex: output and item) which will give us Item, Replacement & Replacing Items details. Find the time taken for achieving this.
  - OEM files - Group by all Replacement and Replacing Items. Find the time taken for achieving this.
  - Filter demand files based on the latest transaction in the following order i.e. demand date, sales order line number, sales order number, extraction date
  - Dealer files - outer join all items from dealer resource types (Ex: stock and demand). Find the time taken for achieving this.
  - Join the two OEM and Dealer on the fields which uniquely identify an Item. Find the time taken for achieving this.
  - Format the output accordingly
- Partition and **Load** the result into S3. Find the time for achieving this.

3.2 Understanding AWS Glue worker types

- AWS Glue includes three workers, also known as Data Processing Units (DPUs), which are available in Standard, G.1X, and G.2X configurations.
- With two Spark executors, the typical worker has 16 GB of memory, 4 vCPUs of computing capacity, and 50 GB of associated EBS storage.
- With one Spark executor, the G.1X worker has 16 GB of memory, 4 vCPUs, and 64 GB of associated EBS storage.
- With one Spark executor, the G.2X worker has 32 GB of memory, 8 vCPUs, and 128 GB of associated EBS storage.
- Both standard and G1.X workers are assigned to a single DPU, which may handle up to eight tasks at once.
- A G2.X worker corresponds to two DPUs, each of which can handle 16 tasks at once.
- As a result, horizontal scaling can benefit compute-intensive AWS Glue operations with a high degree of parallel computing (more standard or G1.X workers).
- Vertical scaling can help AWS Glue operations that require a lot of memory or a lot of disc space to store intermediate shuffle output (more G1.X or G2.X workers).

3.3 DPU Capacity Planning

Input: - Item 237.5 MB, Replacement 29 MB, Stock 676.8 MB & Demand 7.8 GB

Table 3.3.1 job details

	G1.X – 2 workers	G1.X – 4 workers	G1.X – 6 workers	G1.X – 8 workers	G1.X – 10 workers
Start-up time	6 secs	**6 sec**	6 secs	7 secs	7 secs
Execution time	15 mins	**6 mins**	4 mins	3 mins	3 mins
DPU hours	0.50	**0.40**	0.47	0.53	0.56
Cost *	$0.22	**$0.176**	$0.206	$0.233	0.246

Output: - 500 folders with each having 1 file of size 50.9 MB

As we see from the above test case results, increasing the capacity won't make much difference in execution time but DPU consumption increases (after 8 DPU's execution time is constant). Considering the sample input dataset size and test results concluded that **G1.X 4 workers** as the optimal DPU

3.4 AWS Glue job Results

Input: - Item 236 MB, Replacement 219 MB, Stock 7.30 GB & Demand 8.55 GB

Table 3.4.1 Input Dataset details

	Item	Replacement	Stock	Demand
File size	236 MB	219 MB	7.30 GB	8.55 GB
Record Count	18 lakh	37,57,185	105 Million	105 million

Table 3.4.2 Seven Concurrent Job results

G1.X – 2 workers	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7
Start time	**15:38:36**	15:38:37	15:38:39	15:38:40	15:38:42	15:38:44	15:38:45
Start-up time	4 secs	4 secs	9 secs	13 secs	6 secs	9 secs	6 secs
Execution time	13 mins	12 mins	15 mins	13 mins	14 mins	14 mins	13 mins
End time	15:51:46	15:51:27	**15:54:31**	15:52:43	15:52:51	15:53:07	15:52:34
DPU hours	0.44	0.43	0.52	0.46	0.47	0.47	0.46
Cost *	$0.19	$0.19	$0.29	$0.20	$0.21	$0.21	$0.20

Table 3.4.2 **7** Total Job results

Total DPU used	Total DPU hours	Total time	Total cost
14	3.25	14 mins	$1.42

Output: - Total number of objects is 3500 with each having 1 file of 124 MB

I.5 Detailed Job Analysis

- A **job** can be thought of as a physical component of your ETL code.
- If your task has five **stages**, there will be four shuffles. The number of shuffles is always one less than the number of stages.
- The number of **tasks** in each stage represents the number of partitions that spark will operate on, and each task inside a stage represents the same work that spark will perform but on a different piece of data.
- Shuffling is the process of moving data between stages, or it can also be defined as the process of moving data between numerous Spark stages.
    - Before transmitting, "**shuffle write**" refers to the sum of all written serialised data on all executors (normally at the end of a stage).
    - The aggregate of read serialised data on all executors at the start of a stage is referred to as "**shuffle read**."

Table 3.5.1 Job details

Job Id	Description	Submitted	Duration	Stages: Succeeded/ Total	Tasks (for all stages): Succeeded/Total
4	Transforms	2021/04/16 15:39:25	13 min	11/11	176/176
3	Replacement File load	2021/04/16 15:39:22	0.2 s	1/1	1/1
2	Item File load	2021/04/16 15:39:21	0.2 s	1/1	1/1
1	Stock File load	2021/04/16 15:39:20	0.2 s	1/1	1/1
0	Demand File load	2021/04/16 15:39:08	11 s	1/1	1/1

Table 3.5.2 Job Id 4 Task details

Stage Id	Description	Submitted	Duration	Tasks: Succeeded /Total	Input	Output	Shuffle Read	Shuffle Write
14	Write S3	2021/04/16 15:43:47	8.9 min	16/16		60.5 GB	2.2 GB	
13	OEM & dealer Join	2021/04/16 15:43:03	44 s	16/16			615.9 MB	2.2 GB
12	Stock & Demand Join	2021/04/16 15:40:40	28 s	16/16			205.3 MB	197.5 MB
11	Group by	2021/04/16 15:42:45	19 s	16/16			588.1 MB	418.4 MB
10	Item, Replacing items & Replacement Join + Group by	2021/04/16 15:41:13	1.4 min	16/16			297.6 MB	588.1 MB
9	Item & Replacement Join	2021/04/16 15:40:54	12 s	16/16			153.0 MB	267.3 MB
8	Replacement File Scan	2021/04/16 15:39:25	10 s	16/16	219.3 MB			122.8 MB
7	Item File Scan	2021/04/16 15:39:25	1.3 min	16/16	236.7 MB			30.2 MB
6	Filter Latest Demand	2021/04/16 15:40:05	14 s	16/16			254.0 MB	11.4 MB
5	Stock File Scan	2021/04/16 15:39:25	25 s	16/16	1073.0 MB			193.9 MB
4	Demand File Scan	2021/04/16 15:39:25	33 s	16/16	1255.7 MB			254.0 MB

IV. Impact

- A programmer can use a serverless platform to write code and have it operate in the cloud without having to worry about hardware, operating systems, or servers. They can develop the business logic in code and then run it without having to worry about deployment issues.
- By eliminating the requirement for server management, scaling, high availability, configurations, builds, and other issues, serverless systems enable citizen programming (low code). As a result, serverless platforms lower the entry barrier for developing complicated backend systems.
- Extremely high scalability isn't new in cloud services, but the no-service architecture takes this to the next level. Allow vendors to extend the app by not using a server, without explicitly adding or removing instances from the server array, and not using a server. There's no need to consider how many simultaneous requests can be processed before memory runs out as the cloud service provider manages extensions on a per-request basis.
- Without any additional work from the developer, applications achieve high availability and auto-scalability. This can drastically reduce development time and, as a result, costs.
- In a serverless environment, each Lambda function is completely isolated. The server does not crash if one of the features is disabled because it has no influence on the other features.

V. Comparison between serverless and traditional architecture

Servers are an essential component of traditional web applications. The application server is mostly done, notwithstanding the presence of load balancers or dedicated web servers in front of it. It provides all of an application's necessary functions, such as storing user data, completing security authentication, and regulating activities, among others. The majority of the application's pages serve primarily as a user interface for the backend, though some navigational functionalities are included. This is the multi-tier architecture that many people are familiar with. Browsers, application servers, and several post-systems make up the majority of the system.

All of these tiers can be deleted with a serverless design for a more straightforward solution. It is better to design a single-page web application that implements the application programs in the browser instead of utilising the web client as the application server's interface. All you'll need is a simple static web server to get started. The conveyance of application material is the only focus of all interactions. The browser serves as an application container. In contrast to typical web application architecture, the final design eliminates all intermediate layers, allowing the browser to connect directly to the services it requires.

Table 5.1. Table comparing serverless architecture to SOA architecture.

	Serverless architecture	SOA architecture
Angle of focus	One layer of business logic	Various components
Server management	Provider of cloud computing services	Technical personnel with a connection to the company.
Capability to expand	Expand automatically based on the current scenario	Scalability necessitates receiver queue and display capacity control.
Use and tolerance for failure	Usability and tolerance of failure are incorporated	Code need to be set up, and administered.
Mode of Payment	Pay on the basis of a request	Pay on the basis of the purchase price

VI. Conclusion

Serverless computing is a new technology that allows you to process large amounts of data at minimal cost and with excellent performance. Only appropriate solution architecture can be used to compute data in the cloud platform. A literature review was undertaken in order to better understand serverless computing, and cost and performance were demonstrated using an example. Using a real-world dataset, it was built using Amazon Web Services.

Serverless architecture is a new technique of generating, distributing and concentrating software for developers. This technique can minimise time for the market, operating costs and system complexity, especially for large data applications, in which 4 Vs those are volume, variety, velocity and truthfulness all are problems. As a result, a Serverless Architecture is an excellent approach for dealing with these issues.

REFERENCES

[1] Baldini, Ioana, et al. "Serverless computing: Current trends and open problems." Research Advances in Cloud Computing. Springer, Singapore, 2017. 1-20.
[2] McGrath, Garrett, and Paul R. Brenner. "Serverless computing: Design, implementation, and performance." IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2017. 405-410
[3] Fox, Geoffrey C., et al. "Status of serverless computing and function-as-a-service (faas) in industry and research." arXiv preprint arXiv:1708.08028 (2017)
[4] Hellerstein, Joseph M., et al. "Serverless computing: One step forward, two steps back." arXiv preprint arXiv:1812.03651 (2018).
[5] Jonas, Eric, et al. "Cloud programming simplified: A berkeley view on serverless computing." arXiv preprint arXiv:1902.03383 (2019).
[6] Adzic, Gojko, and Robert Chatley. "Serverless computing: economic and architectural impact." Proceedings of the 2017 11th joint meeting on foundations of software engineering. 2017. pp. 884-889
[7] Castro, Paul, et al. "The rise of serverless computing." Communications of the ACM 62.12 (2019): 44-54.
[8] Lee, Hyungro, Kumar Satyam, and Geoffrey Fox. "Evaluation of production serverless computing environments." 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 442-450.
[9] Lloyd, Wes, et al. "Serverless computing: An investigation of factors influencing microservice performance." IEEE International Conference on Cloud Engineering (IC2E). 2018. 159-169
[10] Akkus, Istemi Ekin, et al. "{SAND}: Towards High-Performance Serverless Computing." 2018 {Usenix} Annual Technical Conference ({USENIX}{ATC} 18). 2018.