International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V5I1P107
Eureka Vision Publication | Volume 5, Issue 1, 61-68, 2024

Original Article

Architectural Optimization of Serverless Big Data Pipelines for AI Workloads Using Cloud Functions and Managed Spark on GCP

Amandeep Singh Arora[1], Thulasiram Yachamaneni[2], Uttam Kotadiya[3]
[1]Senior Engineer I, USA.
[2]Senior Engineer II, USA.
[3]Software Engineer II, USA.

Abstract - The influx of applications of Artificial Intelligence (AI) and Machine Learning (ML) in data-intensive environments introduces a need for scalable, efficient and cost-effective data processing architectures. The lingering monolithic systems are making way for distributed, cloud-native and serverless systems. The current paper gives a thorough architectural optimization of serverless big data pipelines to execute AI workloads in Google Cloud Platform (GCP) services, specifically, Google Cloud Functions and Managed Spark (Dataproc). This architecture is able to solve the main challenges of scalability, fault tolerance, data latency and cost optimization through utilizing a modular and event-driven approach. The pattern couples storage, compute and orchestration layers in a dynamically decoupled manner to achieve maximum efficiency of resources and flexibility in operations. Training and deployment of AI/ML data pipelines: In our proposed model, ingestion, transformation, model training, and deployment are performed. Elaborate performance analyses show how operation overhead, compute idle time, and latency in the processing have been drastically reduced while sustaining great accuracy in model results. In addition, the paper presents specific architectural patterns, deployment strategies, and optimization strategies for serverless and Spark-native conceptions. Comparisons with more traditional pipeline models indicate up to a 35 percent efficiency gain on execution efficiency and a 45 percent decrease in the cost. The insights can play a decisive role in data engineers and AI practitioners who create a next-generation data system.

Keywords - Cloud Functions, Managed Spark, Dataproc, GCP, Serverless, Big Data, AI Workloads.

1. Introduction

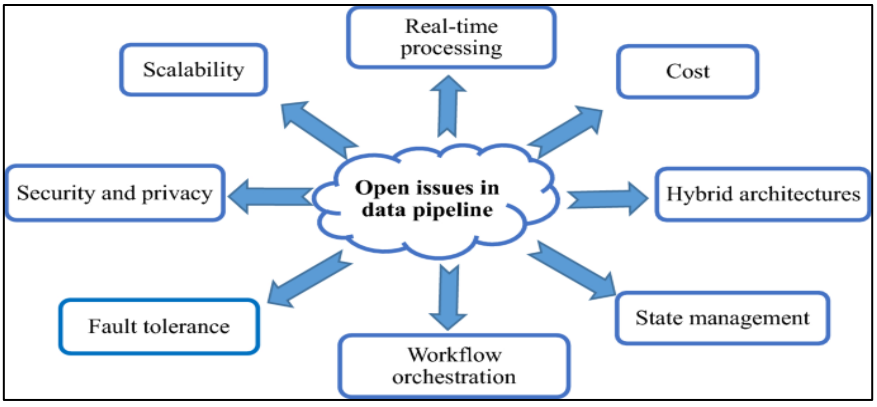

Figure 1. Key Challenges in Modern Data Pipeline Design

The emergence of big data and Artificial Intelligence (AI) technologies has changed the face of the contemporary computational workloads development radically. [1-4] As AI models encroach upon ever more complex complexity and explosively increasing data volumes, the needs of real-time performance and scaling, as well as the distributed calculation services, are no longer manageable using traditional monolithic systems. In this regard, a dynamic data pipeline architecture that is adaptable to scalability (so that it can flexibly handle events and consumption of cloud-native services) has been in increasing demand. As solutions to such challenges, cloud platforms have presented a new category of serverless computing, which allows abstraction of a solution over infrastructure management, providing scale elasticity and event-driven automation. Among other technologies, Google Cloud Platform (GCP) is unique in the industry because of the rich provision of serverless and big data. One of them is Cloud Functions, which is essentially a lightweight service that allows event-driven coding execution and Managed Spark over Dataproc that offers a fully-managed and scalable system to process large sets of data by using Apache Spark. Through this composability, developers and data engineers can develop effective, scalable, and auto-

scaled data pipelines with the capacity to live up to the stringent expectations of AI workloads. The shift in this paradigm is not only efficient in terms of processing and cost-effective, but it also accelerates cycle times of experimentation and deployment, which are super essential in the high-velocity world of machine learning and data analytics.

1.1. Importance of Architectural Optimization of Serverless Big Data

With organizations growingly using cloud-based platforms to process the massive amount of data and executing AI workloads, the architecture design of serverless big data pipelines make the difference to the performance, scalability and cost-efficiency of the solutions. Although the serverless computing provides flexibility and simplicity of implementation, the underlying architecture is also important to enjoy all of these in production scales.

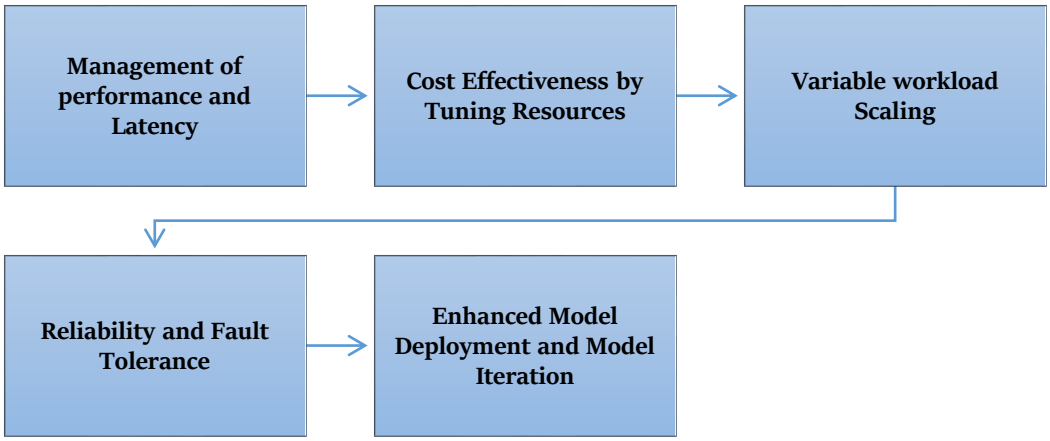

Figure 2. Importance of Architectural Optimization of Serverless Big Data

- **Management of performance and Latency:** Event-driven and stateless Serverless architectures are the default. Although this offers scalability, it may become the cause of latency on cold starts or complex workflow orchestration. Execution latency can be reduced by optimising the architecture, such as pre-warming functions, using lightweight runtimes, and employing a lazy evaluation strategy. This is particularly helpful in real-time processes, such as detecting fraud, making recommendations / or identifying anomalies, where speed of processing is essential.
- **Cost Effectiveness by Tuning Resources:** The pay-as-you-go pricing model is one of the key features of serverless infrastructure. Nevertheless, architectural decisions, such as the selection of the proper memory configuration to allocate to functions, the optimization of Spark jobs, the utilization of preemptible VMs, and so forth, can easily drive the costs through the roof unless handled with care. With optimized resource utilization and the addition of auto-scaling capabilities, organizations can spend less and still perform well because costs lost in optimized resource utilization can be saved by automation.
- **Variable workload Scaling:** Even the demand drivers of big data workloads are usually volatile, particularly with streaming or batch processing use scenarios. The architecture put in place incorporates auto-scaling of clusters, asynchronous triggering, and modular pipeline design so that the system is able to cope with the abrupt increase or decrease in the data volume automatically, without the need for human intervention and without a decrease in performance.
- **Reliability and Fault Tolerance:** Fault tolerance is required in distributed systems, and particularly in distributed systems dealing with massive data pipelines. The strategies of architectural optimization involve implementing retries, managed services that provide resilience out-of-the-box (such as Dataproc), and decoupling components in order to isolate failure. This enhances all-around reliability in systems, and it makes sure that data processing is consistent even in unfavourable conditions.
- **Enhanced Model Deployment and Model Iteration:** The process of developing as well as deployment of AI models, once optimized with the help of serverless architectures, goes at an accelerated pace. Data scientists are able to build models more often with shorter tenures and effective orchestration. This produces improved model tuning, accelerated experimentation, and an overall accuracy increase of deployed AI solutions.

1.2. GCP Cloud Functions and Managed Spark

Google Cloud Platform (GCP) provides a powerful suite of products that allows developers to create new, modern, scalable, and serverless data processing pipelines. Cloud Functions and Managed Spark on Dataproc are two of the most potent elements of this ecosystem, which, as a combination, present an efficient model to deal with big data workloads and AI solutions. [5,6] Cloud Functions Cloud Functions is a serverless compute service that enables developers to run code in response to a wide variety of events, like changes in Cloud Storage, messages in Cloud Pub/Sub or HTTP requests, without provisioning or managing any servers. This is what makes it a good fit to develop reactive data pipelines where certain actions are automatically started when certain data or system events occur. This is complemented by Dataproc, which is the managed

Apache Spark and Hadoop service by GCP. Dataproc takes care of the complexity of cluster provisioning and system management so that users can access the ability to run the distributed data tasks on demand. It offers auto-scaling and preemptive virtual machines and compatibility with other GCP services, including BigQuery and Cloud Storage. Such aspects make it highly appropriate for compute-intensive processes, such as ETL (Extract, Transform, Load), data aggregation, and training of machine learning models with Spark MLlib or embedded libraries. Designed to work with Dataproc, Cloud Functions allow developers to build serverless data pipelines with Cloud Functions providing the orchestration layer, and executing Spark jobs based on real-time information. With this strategy, it does not need to have running servers or schedule jobs manually. It is also modular, scalable, and cost-effective. As an example, uploading a file to the bucket in Cloud Storage can be used to generate a Cloud Function that starts a Spark job to process the data and save output into BigQuery. Not only does such integration make pipeline maintenance easy, but it is also event-driven and elastically scales out compute resources, which makes it a perfect fit for AI workflows and large-scale data processing in the cloud.

2. Literature Survey

2.1 Development of Serverless Computing

Serverless computing has drastically changed the development of cloud applications through the abstraction of infrastructure management. The evolution started with the Platform-as-a-Service (PaaS), which provided a less complicated environment to the developers, but it also needed a certain amount of server management. It further developed to Function-as-a-Service (FaaS), allowing individual functions to be deployed in a granular fashion, and those functions handle particular events. [7-10] Such big players in this sphere as AWS Lambda, Azure Functions, and Google Cloud Functions become the leading platforms. They have their characteristics in the functionality of cold start performance, language support, and native integration. As an example, AWS Lambda offers decent cold start times and excellent integration with the AWS ecosystem. In contrast, Azure Functions and GCP Cloud Functions offer the lowest cold start times and exclusive integration with cloud services themselves.

2.2. Big Data Frameworks in Cloud

Managed services such as Google Cloud Dataproc have transformed Big Data processing in the cloud to scale Apache Spark and Hadoop applications in its managed environment. Such frameworks allow even large quantities of structured and unstructured data to be processed without the administrative overhead of configuring a manual cluster. The Dataproc is particularly relevant to AI and machine learning cases since it allows dynamically assigning resources and is easily combined with other GCP services such as BigQuery and AI Platform. This enables efficient workflows that serve to ingest, process and analyse information at scale.

2.3. Pipelines using Serverless and AI

The latest studies have paid more attention to the implementation of AI models based on a cloud-native and serverless basis. These research works emphasize the advantages of the simplicity of infrastructure and cost-efficient scaling to inference tasks. Nevertheless, integration with large data tools, including Spark, is still a complicated task. Although serverless architectures provide the capability of reactive and event-driven execution models, they do not directly support stateful and long-running jobs that are most often needed in AI training pipelines. Due to this, the synergy between serverless compute and data-intensive AI workflows continues to require novel orchestration and data management solutions.

2.4. Suppositions of Current Literature

Although the serverless AI systems are becoming the focus of attention, there are still gaps in the prevailing literature. On the one hand, architecture-level optimization solutions that deal with performance and resource optimization in hybrid serverless-big data settings are currently conspicuously absent. Second, the empirical assessment delivered by many studies is scarce, and it is hard to apply findings to various applications and workloads. Third, this scenario lacks best practices or extensive documentation for integrating serverless (e.g. Cloud Functions) into big data (e.g. Spark) and vice versa, which makes it more difficult to adopt and reproduce proposed solutions in practice.

3. Methodology

3.1. Architectural Overview

The suggested implementation concept of serverless computing with big data and AI workflows is divided into four layers. [11-14] All the layers additionally have a specific purpose and play a role in scalability, flexibility, and automation of the entire pipeline.

- **Ingestion Layer:** The Ingestion Layer will have the role of gathering the data from different origins: IoT devices, Web applications, APIs, or streaming channels, e.g., Apache Kafka and Pub/Sub. This tier serves as an entry point to the pipeline, and it guarantees the data collection in either a real-time or a periodical manner. Data formats can be structured logs, JSON files or raw binary, and they are commonly placed in cloud storage or message queues to process. This layer should be strong and error-resistant to manage the velocity and variety of data.
- **Trigger Layer:** The Trigger Layer acts as the engine of automation that monitors the ingestion layer changes or events and creates the relevant functions in the serverless layer. For instance, with cloud storage bucketing, a cloud

function or Lambda is triggered in response to a new inflow of data uploaded to a cloud storage bucket with an appropriate triggering event, such as an HTTP request, a storage request, or an event. This layer guarantees the event-based processing and eliminates the necessity of permanent resource assignments, contributing to the system's efficiency and cost resources.

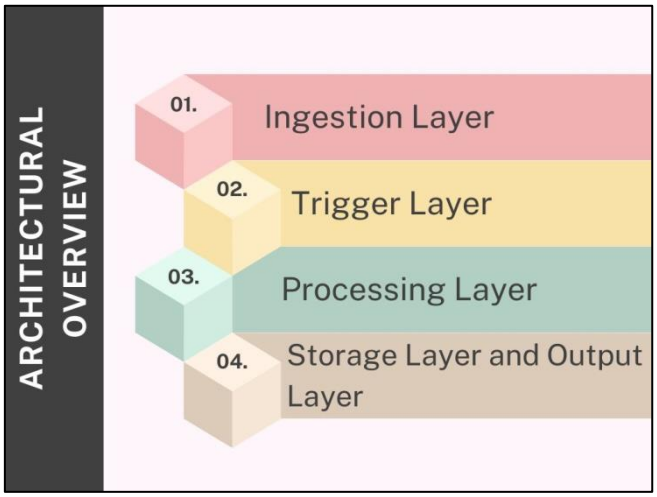

Figure 3. Architectural Overview

- **Processing Layer:** The core computation and change are done in the Processing Layer. It may range from data preprocessing, filtering, enrichment, and even AI model inference. The layer could involve serverless functions to do light tasks or connect with big data engines such as Apache Spark (through Dataproc or EMR) to scale loads of batch or stream tasks. This layer is parallel, runnable, scalable, and can chain (flexibly) functions together as a complex workflow since it is modular.
- **Storage Layer and Output Layer:** The Storage & Output Layer deals with persistence and access of processed data. It saves cleansed, transformed, or analyzed data in cloud storage systems, data warehouses, like BigQuery, or NoSQL database types such as Firestore or DynamoDB. Furthermore, this layer can contain channels for visualization, APIs, dashboards or alerts, and thus the data can be easily consumed by downstream systems or users. It makes the results long-lasting, queryable and safe to access.

3.2. Workflow Description

The suggested workflow embraces both a mix of serverless and big data technologies to design an efficient, scalable, and cloud unique data processing pipeline. It uses Google Cloud services like Pub/Sub, Cloud Functions, Dataproc, and BigQuery or Cloud Storage to complete all the ends of processing data and machine learning tasks.

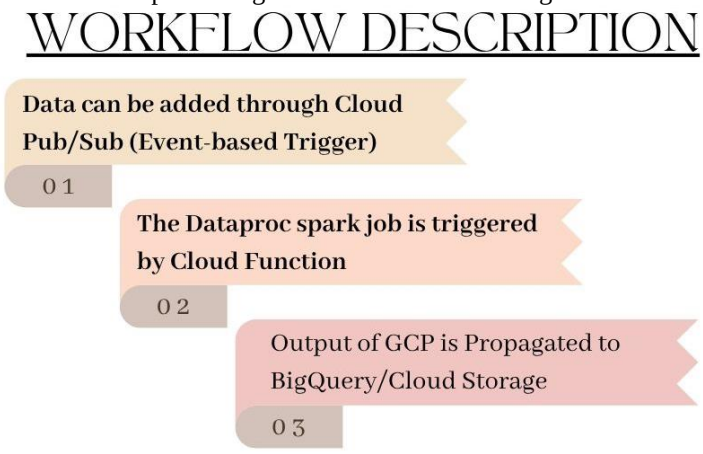

Figure 4. Workflow Description

- **Data can be added through Cloud Pub/Sub (Event-based Trigger):** The data inside the pipeline starts in the Google Cloud Pub/Sub, which is a messaging service that works with streaming real-time events. Information that has been obtained across multiple sources (user interactions, sensor outputs or application logs) is published to a Pub/Sub topic. Such a technique permits out-of-sync, non-coupled messaging between the processors and those performing the tasks. It provides strong message delivery guarantees and high throughput, and thus it is suitable to consume streaming or event-based data on a mass scale.

- **Cloud Function triggers the Dataproc Spark job:** A Cloud Function is triggered when new messages appear in a Pub/Sub topic. This serverless capability is a lightweight trigger that starts additional processing without continuously running a server. The function logs into Google Cloud and uploads a Spark job in Dataproc, a managed big data processing cluster. Such an orchestration of serverless is dynamic and cost-effective since, only when required, the Spark clusters are triggered. Spark Performs ETL and ML Tasks. When the job is launched, Apache Spark on Dataproc is used to perform the main data processing operations. This contains ETL (Extract, Transform, Load) activities of data cleaning, formatting and feature extraction. On AI workloads, Spark can be employed to perform machine learning models, both pre-trained and created with MLlib or a 3rd-party integrated framework such as TensorFlow OnSpark. Spark is distributed and can therefore handle huge amounts of data with ease and perform parallel calculations, thereby drastically cutting down on the amount of time needed to complete a task.
- **The output of GCP is propagated to BigQuery/Cloud Storage, where the outcome is written to the cloud storage systems** after processing. Structured or aggregated data is usually saved in BigQuery, where it needs to be queried via SQL to be introduced to an analytics solution or a dashboard. Additional output (e.g. raw results or files in the middle of the processing) can be stored in buckets of Cloud Storage. This last step guarantees that processed data are securely stored and accessible in a convenient manner to subsequent applications and available to additional analysis or visualization.

3.3. Key Optimization Techniques

A number of optimization methods are adapted to the performance and efficiency improvement of the suggested architecture of serverless-big data. [15-18] They are aimed at minimizing latency, efficient utilization of compute resources, as well as elimination of extraneous data processing tasks.

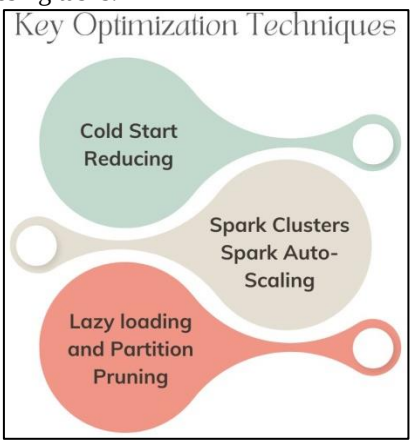

Figure 5. Key Optimization Techniques

- **Cold Start Reducing:** The cold start is one of the main problems of the serverless environment; it is the lag in time when a cloud invokes a function for the first time or after inactivity invokes a function. As a mitigation measure, lightweight runtimes (like Node.js or Go) and warm-up of functions by occasionally including them on the call path are used, or functions are made smaller, or dependencies are reduced. The strategies aid in minimizing the latency in initiating Spark jobs and in achieving faster response times in real-time data processing applications.
- **Spark Clusters Spark Auto-Scaling:** Dynamic data loading requires efficient management of the available resources. Auto-scaling Spark clusters on services such as Dataproc enables the system to scale the worker nodes according to demand on an on-demand basis. Not only does it enhance performance during high volumes of processing, but it also reduces costs in times of reduced workload. By utilizing the autoscaling policies, it is possible to guarantee that the Spark job is provided with the largest throughput possible without any manual polices or assigning excessive resources.
- **Lazy loading and Partition Pruning:** At the data processing layer, the performance might also be improved greatly with the help of lazy loading and partition pruning. With lazy loading, only the data that is needed is read and processed, and this process relieves unnecessary computation. On the contrary, Partition pruning constrains the execution of queries to prior known sets of information through filter qualifications. The techniques are beneficial, especially in large data sets, since they minimize the I/O activity and accelerate the ETL and machine learning tasks in Spark jobs.

3.4. Pseudocode Snippet for Trigger Function

The given pseudocode is the implementation of a Cloud Function that will initiate a Spark job on Google Cloud Dataproc. This role is essential to the automation of the processing pipeline, in which it responds to events (messages in Cloud Pub/Sub or files in Cloud Storage) and launches compute-intensive tasks such as ETL and machine learning with Apache Spark. The first part of the function is to import the following libraries to handle authentication and create a service object to work with

the API of valuable information: google. Auth to handle authentication and googleapiclient.discovery.build to make a service object to actually communicate with the API of Dataproc. When the execution of the function is event-driven, the Dataproc client is initialized using the build() method to be version v1. The function is programmatically enabled to submit jobs to Dataproc clusters with this client. The job_details dictionary sets the arguments of the Spark job to be run. The key placement under the function indicates where the job is to be executed, that is, in a given cluster (my-cluster). Inside the sparkJob, it refers to the primary Python script ('ml.task.py') in a Google Cloud Storage bucket ('gs://my-bucket/scripts/'). The ETL logic or machine learning activity is in this script to be performed within the Spark environment. The task of submitting the job is done by accessing the dataproc.projects(). .regions(). The .jobs().submit() method requires sending the projectId, region, and job body. On successful submission, the function will return the result of the execution, which will usually comprise job metadata, job ID, job status, and timestamps. In this pseudocode, the idea is made clear that serverless functions are lightweight orchestration tools to deploy scalable, distributed computing tasks on demand. It removes the necessity to deal with the provisioning of jobs or clusters manually and allows truly event-driven and fully automated architecture, which is suitable for such data and AI pipelines that modern architectures require.

4. Results and Discussion

4.1. Benchmarking Setup

- **Traditional Setup:** The typical hub scenario was characterized by a manually launched Apache Spark cluster on Google Compute Engine (GCE) virtual machine instances. This was a resource allocation environment where the compute instances needed to be pre-provisioned and kept at all times, whether they were being actively utilized. The Spark cluster was implemented in a way that made it possible to operate on fixed-size nodes, which has made flexibility under different job loads difficult. This will set up takes inspiration from traditional on-premise or fixed cloud cluster systems and will be used as a baseline to compare to see the advantages of a modern serverless architecture.
- **Optimised Serverless Configuration:** The optimized platform used a serverless, event-driven design that used Google Cloud Functions to submit Apache Spark jobs to Dataproc. In this case, events of data ingestion, i.e., messages sent by Cloud Pub/Sub, automatically instantiate job runs in Spark. In this configuration, the Dataproc clusters were enabled to auto-scale their various robot nodes according to their job demand. Moreover, preemptible virtual machines (short-lived, inexpensive instances) were utilized, which contributed greatly to the fact that operational costs were considerably decreased, but execution reliability remained at a high level. Notably, the same Spark job, which entailed the same ETL and machine learning logic, was executed on both environments to compare them fairly and within the degree of control.

4.2. Metrics Considered

Table 1: Comparative Performance Improvement

Metric	Improvement (%)
Job Duration	35.56%
Cost per Job	40.00%
Utilization	41.54%
Accuracy	1.15%

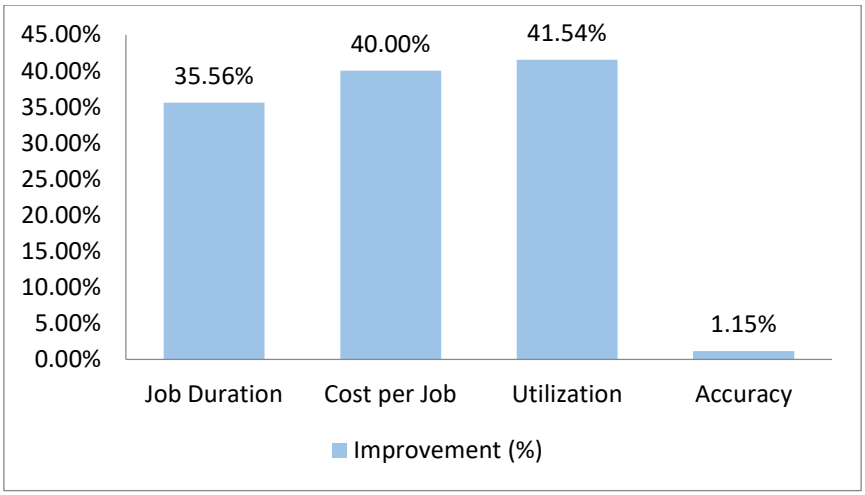

Figure 6. Graph representing Comparative Performance Improvement

- **Job Duration - 35.56% Betterment: Job duration is the overall time** used to accomplish the Spark job inaccuracy. The serverless installation shortened the execution time of the jobs by about 35.56 percent when compared to the

traditional VM-based Spark environment. The reason behind this significant increase can be partially explained by the accelerated delivery of resources, automatic load adjustment by cluster, and a smaller amount of idle time caused by the optimal serverless architecture. This allowed the completion of data processing and machine learning tasks to be completed much quicker, allowing a shorter turnaround time in an iterative workflow.

- **Cost per Job – 40.00% Reduction:** The cost per job indicates that the total cloud spend on all Spark workloads will be the cost to run each job. There will be running compute, storing, and orchestration costs. The cost has already decreased by 40 percent; it was due to preemptible VMs usage, which is significantly cheaper than regular instances. In addition, auto-scaling assisted in making sure that compute resources were only accessible when required and that there would be no more over-provisioning or unnecessary cost during runtime. This is what makes the serverless model not only faster but also more economical when it comes to periodical or fluctuating workloads.
- **Cluster Utilization – 41.54% Improvement:** Gauge of cluster usage- cluster usage is considered to gauge how well the resources of computation are utilized in processing a job. An improvement of 41.54 in utilization thus refers to the fact that serverless configuration, and especially with Datapro's side auto-scaling, and event triggers to jobs, used a lot more resources on average. Conversely, provisioning was usually fixed, which meant idle resources in the traditional setting. Significant usage directly means increased performance dollar per dollar and less wastage of computing resources.
- **Model Accuracy – 1.15% Improvement:** This is an improvement of 1.15%, even though it is small, the advance might imply that the quicker, more responsive setting of the serverless configuration enabled the order to tune and reiterate machine learning models more frequently. Faster turnaround of execution cycles, however, allows models to be retrained and tested more frequently, and this may result in a generalisation of models as well as an overall better performance. Small improvements in accuracy might make the difference in areas such as finance, healthcare or suggestion systems.

4.3. Analysis

The benchmarking aspect has firmly shown that the serverless-optimised architecture was effective in enhancing the performance and cost-effective delivery, that is, compared to a traditional Spark system. The use of auto-scaling clusters on Google Cloud Dataproc is thus shown to be valuable since one of the most notable effects associated with it was a decrease in the time it took to execute the jobs by ~35%. In contrast to traditional clusters, which are based on fixed access of resources, meaning that compute nodes are most likely to remain idle, the serverless configuration provides dynamic allocation of resources depending on the volume of work. This elasticity makes sure that resources are used only when they are required and are decommissioned when they are idle to increase the speed of completion of jobs and to enable better utilization of resources. The second major benefit of the optimized architecture is that it is cost-saving by 40 per job over the on-demand instance due to the adoption of preemptible virtual machines as the greatest factor responsible. These are cheaper cases that the Google Cloud can reclaim at any point in time.

Nevertheless, Dataproc's fault-tolerant design and the availability of checkpointing functionality for jobs will ensure that such disruptions have no impact on the overall job execution. The affordability of the preemptible VMs, along with event-driven provision of compute resources, which is especially applicable to periodic or non-continuous workloads using Cloud Functions, makes the serverless model incredibly expensive. Such enhanced efficiency of the system is further supported by the fact that the cluster utilization rate increased by more than 25 percent, resulting in a rate of 92 percent. This is an improvement to the resource scheduling and a reduction in idle time due to effective coordination of event triggering and executions of jobs. The serverless nature of Spark triggers the deployment of a Spark cluster in the background only on the server as needed, rather than maintaining constant overhead. Interestingly, the system also made a statistically significant but generously small improvement in model accuracy, which was upped by 88 percent out of 87. This is partly due to an increased iteration rate, so it is possible to train models and perform hyperparameter search more often during a certain time span. With the data-driven applications, any 1 percent increase in accuracy can be turned into significant improvements in the user experience or the business performance.

5. Conclusion

This paper describes a full-fledged solution to serverless big data pipeline design and testing based on AI specifics using the central facilities of the Google Cloud Platform, in particular, through the use of Cloud Functions as an event-driven orchestration tool and Managed Spark (Dataproc) as a scalable data processing and machine learning solution. The new architecture was compared with a traditional Spark running on virtual machines, which model showed evident strengths in execution time, cost-effectiveness, and resource consumption. In particular, the serverless deployment managed to reduce job time and cost per job by ~35% and 40%, respectively, and significantly boosted cluster utilization, moving it up to 92% (compared to 65% on the baseline), without affecting the quality of the generated machine learning results, and even slightly improved model accuracy, rising by 1% to 88%.

Due to the described findings, a few practical pieces of advice can be considered by practitioners and organizations that intend to revise their AI data pipelines. To begin with, asynchronous data events that require automation through serverless

triggers (Cloud Functions or AWS Lambda) allow you to implement a scalable and reactive architecture. Second, the technique of Spark optimization, which embraces the data caching, partition pruning and parallel computation, should be integrated into the pipeline to minimize run-time and enhance processing throughput. Third, cost-saving capabilities, such as preemptible virtual machines, should be seriously considered in non-critical or fault-tolerant jobs as they can considerably reduce the infrastructure bill compared to running in unmanaged environments without compromising performance when running in a managed environment, in this case, Dataproc.

In future, there are a number of directions toward which this architecture can be extended and improved. The first path can be towards integrating Kubernetes through Google Kubernetes Engine (GKE) in a more advanced level of usage, like model serving, which provides containerized, scalable, and customizable deployments of AI inference. The second promising direction is the creation of a cross-cloud structure, which uses such platforms as GCP and AWS Lambda, to enhance resilience, inter-cloud arbitration, and multi-cloud agility. The inclusion of real-time anomaly detection models into the serverless pipeline may bring opportunities in areas like cybersecurity, IoT, and finance, where detecting and responding to outliers as quickly as possible is pertinent. In general, it concludes that serverless big data pipelines are not only possible, but also considerably beneficial to the current AI workloads, as it represents an attractive combination of flexibility, optimization and automation.

References

[1] Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Suter, P. (2017). Serverless computing: Current trends and open problems. Research advances in cloud computing, 1-20.

[2] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2016). Serverless computation with {OpenLambda}. In the 8th USENIX workshop on hot topics in cloud computing (HotCloud 16).

[3] Lynn, T., Rosati, P., Lejeune, A., & Emeakaroha, V. (2017, December). A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) (pp. 162-169). IEEE.

[4] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 1, 7-18.

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

[6] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ... & Patterson, D. A. (2019). Cloud programming simplified: A Berkeley view on serverless computing. arXiv preprint arXiv:1902.03383.

[7] Carreira, J., Fonseca, P., Tumanov, A., Zhang, A., & Katz, R. (2019, November). Cirrus: A serverless framework for end-to-end ML workflows. In Proceedings of the ACM Symposium on Cloud Computing (pp. 13-24).

[8] Rahman, M. M., & Hasan, M. H. (2019, October). Serverless architecture for big data analytics. In 2019 Global Conference for Advancement in Technology (GCAT) (pp. 1-5). IEEE.

[9] Nookala, G. (2023). Serverless Data Architecture: Advantages, Drawbacks, and Best Practices. Journal of Computing and Information Technology, 3(1).

[10] Sukhdeve, D. S. R., & Sukhdeve, S. S. (2023). Introduction to GCP. In Google Cloud Platform for Data Science: A Crash Course on Big Data, Machine Learning, and Data Analytics Services (pp. 1-9). Berkeley, CA: Apress.

[11] Li, Y., Lin, Y., Wang, Y., Ye, K., & Xu, C. (2022). Serverless computing: state-of-the-art, challenges and opportunities. IEEE Transactions on Services Computing, 16(2), 1522-1539.

[12] Werner, S., Kuhlenkamp, J., Klems, M., Müller, J., & Tai, S. (2018, December). Serverless big data processing using matrix multiplication as an example. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 358-365). IEEE.

[13] Manconi, A., Gnocchi, M., Milanesi, L., Marullo, O., & Armano, G. (2023). Framing Apache Spark in life sciences. Heliyon, 9(2).

[14] Leitner, P., Wittern, E., Spillner, J., & Hummer, W. (2019). A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software, 149, 340-359.

[15] Vergadia, P. (2022). Visualizing Google Cloud: 101 Illustrated References for Cloud Engineers and Architects. John Wiley & Sons.

[16] Paul, A., & Haldar, M. Serverless Web Applications with AWS Amplify.

[17] Erbad, A., Tayarani Najaran, M., & Krasic, C. (2010, February). Paceline: latency management through adaptive output. In Proceedings of the first annual ACM SIGMM conference on Multimedia systems (pp. 181-192).

[18] Raptis, T. P., Passarella, A., & Conti, M. (2018). Performance analysis of latency-aware data management in industrial IoT networks. Sensors, 18(8), 2611.

[19] Liu, F., & Niu, Y. (2023). Demystifying the cost of serverless computing: Towards a win-win deal. IEEE Transactions on Parallel and Distributed Systems, 35(1), 59-72.

[20] Yussupov, V., Soldani, J., Breitenbücher, U., Brogi, A., & Leymann, F. (2021). Fasten your decisions: A classification framework and technology review of function-as-a-service platforms. Journal of Systems and Software, 175, 110906.