SANTA CATARINA STATE UNIVERSITY - UDESC
COLLEGE OF TECHNOLOGICAL SCIENCE - CCT
GRADUATE PROGRAM IN APPLIED COMPUTING - PPGCAP

GABRIEL PIMENTA ROBAINA

PERFORMANCE AND PREDICTABILITY ANALYSIS OF LOCAL FILE SYSTEM I/O WORKLOADS ON SERVERLESS CLOUD PLATFORMS

JOINVILLE

2024

GABRIEL PIMENTA ROBAINA

PERFORMANCE AND PREDICTABILITY ANALYSIS OF LOCAL FILE SYSTEM I/O WORKLOADS ON SERVERLESS CLOUD PLATFORMS

Master thesis presented to the Graduate Program in Applied Computing of the College of Technological Science from the Santa Catarina State University, as a partial requisite for receiving the Master's degree in Applied Computing

Supervisor: PhD Adriano Fiorese

JOINVILLE

2024

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Universitária Udesc,

com os dados fornecidos pelo(a) autor(a)

Robaina, Gabriel
PERFORMANCE AND PREDICTABILITY ANALYSIS OF LOCAL FILE SYSTEM I/O WORKLOADS ON SERVERLESS CLOUD PLATFORMS / Gabriel Robaina. -- 2024.
88 p.

Orientador: Adriano Fiorese
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2024.

1. Serverless. 2. AWS. 3. Google Cloud. 4. Análise de desempenho. 5. Previsibilidade. I. Fiorese, Adriano. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

Gabriel Pimenta Robaina

Performance and predictability analysis of local file system I/O workloads on serverless cloud platforms

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Computação Aplicada** área de de concentração em "Sistemas de Computação", e aprovada em sua forma final pelo Curso de Mestrado em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade ddo Estado de Santa Catarina.

Joinville, 01 de Julho de 2024

ACKNOWLEDGMENTS

I would like to thank my parents for all the support and opportunities that led me to being here. You made it all possible from the start. I am only here because of you. My love and gratitude to Bruna for being here ever since I started my transition from mechanical engineering to IT. You always believed in me and my potential to achieve my goals. You also brought Maggie into my life. Thank you for caring for and supporting me through the extensive hours of studying and working at home.

Finally, my sincere gratitude to all my teachers, especially my supervisor Adriano. You believed in me from the beginning of my work on GOTE, and most importantly during my transition to serverless performance analysis. You encouraged me when I questioned whether I should quit the master's program. Thank you.

“Every time the weight of who I wish I already was comes crashing down on who I still am. I try to be ready and excited to receive it, and even if all is said and done, I’m still meant to be an Honorary Astronaut.”

Honorary Astronaut

RESUMO

Serverless é um paradigma emergente para computação em nuvem que habilita o desenvolvimento de aplicações em nuvem com uma abstração do provisionamento e gerenciamento da infraestrutura computacional. AWS Lambda e Google Cloud Functions (GCF) são exemplos de grandes plataformas de nuvem que já oferecem produtos para *serverless*. Usuários de plataformas *serverless* são cobrados apenas pelos recursos de nuvem consumidos e tempo de execução de suas aplicações, chamadas nesse paradigma de funções, em um modelo *pay-per-use*. Embora a abstração de infraestrutura facilite e acelere o desenvolvimento de aplicações nas plataformas *serverless*, ela limita o controle e visibilidade de aspectos que são determinantes para o desempenho das aplicações. Esse cenário prejudica a previsibilidade de desempenho e custo das aplicações serverless uma vez que desempenho e tempo de execução tem impacto direto em custo. A falta de previsibilidade de custo é um fator de risco relevante principalmente na avaliação do potencial de retorno financeiro relacionado a migração de aplicações legadas em ambientes *on-premises* para plataformas *serverless* na nuvem. Atualmente as plataformas *serverless* dispõem de um sistema de arquivos efêmero que pode ser acessado pelas funções, habilitando assim a migração de uma classe de aplicações que interagem com o sistema de arquivos em operações de entrada e saída (I/O). Esse trabalho busca aumentar a visibilidade de aspectos caixa-preta das plataformas *serverless* e ajudar desenvolvedores a tomar decisões informadas sobre a migração de aplicações de I/O de arquivos para AWS Lambda e GCF. Para isso, este trabalho apresenta um agregado de fatores que impactam desempenho e previsibilidade em plataformas *serverless*, e uma análise comparativa de desempenho e previsibilidade entre AWS Lambda e GCF de segunda geração para cargas de trabalho de I/O no sistema de arquivos local das funções. Resultados mostraram que AWS Lambda tem desempenho igual ao GCF para escrita de arquivos pequenos (10 KB) e superior na escrita de arquivos grandes (1 GB). Esses resultados contradizem as expectativas de que o desempenho do GCF seria superior devido ao seu sistema de arquivos ser mantido em memória. Por outro lado, GCF teve desempenho superior para todas as operações de leitura com I/O direto. AWS Lambda foi mais previsível para leituras e escritas em arquivos pequenos com mínima alocação de recursos de memória e CPU. Ainda, GCF foi mais previsível para leituras com I/O direto em arquivos grandes. Por fim, esses resultados permitiram a construção de um conjunto de recomendações para desenvolvedores que usam ou buscam usar AWS Lambda e GCF para cargas de trabalho de I/O para o sistema de arquivos local.

Palavras-chaves: Serverless, Lambda, AWS, GCF, Google Cloud, I/O, IO, Performance, Desempenho, Análise de desempenho, Previsibilidade, Variabilidade.

ABSTRACT

Serverless is an emerging paradigm for cloud computing that enables the development of cloud-based applications while abstracting the provisioning and management of computational infrastructure. AWS Lambda and Google Cloud Functions (GCF) are examples of major cloud platforms that already offer serverless products. Users of serverless platforms are only charged for the cloud resources consumed and compute time in a pay-per-use model. In this paradigm applications are also known as functions. Although the infrastructure abstraction facilitates and accelerates application development on serverless platforms, it limits control and visibility of aspects that are crucial for application performance. This scenario hinders the predictability of performance and cost of serverless applications since performance and runtime have a direct impact on cost. The lack of cost predictability is a significant risk factor, especially when evaluating the potential financial return related to migrating legacy applications from on-premises environments to serverless platforms in the cloud. Currently, serverless platforms offer an ephemeral file system that can be accessed by functions, enabling the migration of a class of applications that interact with the file system in input and output (I/O) operations. This work seeks to increase the visibility of black-box aspects of serverless platforms and help developers make informed decisions about migrating file I/O applications to AWS Lambda and GCF. To this end, this work presents an aggregation of factors that impact performance and predictability on serverless platforms, and a comparative analysis of performance and predictability between second-generation AWS Lambda and GCF for file I/O workloads on the functions' local file system. Results showed that AWS Lambda and GCF have equivalent performance when writing small files (10 KB). However, AWS Lambda performs better when writing large files (1 GB), contrary to the expectation that GCF would have superior performance due to it's in memory file system. Conversely, GCF presented superior performance for all read operations with direct I/O. AWS Lambda was more predictable for reading and writing small files with minimal memory and CPU resources allocated. Additionally, GCF was more predictable for direct I/O reads in large files. Finally, these results allowed for the construction of a set of recommendations to developers using or seeking to use AWS Lambda and GCF for file I/O workloads to the local file system.

Key-words: Serverless, Lambda, AWS, GCF, Google Cloud, I/O, IO, Performance, Performance analysis, Predictability, Variability.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

ANOVA	Analysis of Variance
API	Application Programming Interface
AWS	Amazon Web Services
CR	Compatibility Rule
CV	Coefficient of Variation
dd	Device-to-Device copy
ECDF	Empirical Cumulative Distribution Function
EFS	Elastic File System
FIO	Flexible I/O Tester
GCF	Google Cloud Functions
GCP	Google Cloud Platform
GCR	Google Cloud Run
GCS	Google Cloud Storage
HTTP	Hypertext Transfer Protocol
INIT	Initialization phase
INVOKE	Invocation phase
I/O	Input/Output
IOPS	Input/Output Operations Per Second
LB	Lower is Better
MAF	Microsoft Azure Functions
RA	Research Answers
RQ	Research Questions
S3	Simple Storage Service
SSD	Solid State Drive
VM	Virtual Machine
WSS	Working Set Size

CONTENTS

1 INTRODUCTION

Serverless is a cloud computing paradigm that enables the development of cloud native applications while abstracting the underlying infrastructure management with a pay-per-use billing model. Developers can focus on writing business rules inside serverless functions while relying on serverless platforms for providing and maintaining highly available and auto scalable infrastructure components. Serverless functions can be triggered from Hypertext Transfer Protocol (HTTP) requests, messaging queues and business events among others. Amazon Web Services (AWS) Lambda, GCF are two of the most relevant serverless offerings from major cloud platforms in the market (AWS, 2023l; GCP, 2023i; CNCF, 2023).

This paradigm has been adopted by a wide range of applications that include machine learning, big data analytics and Internet of Things (WEN et al., 2023). Serverless has also been gaining increasing attention among software engineers and the industry(WEN et al., 2021). Companies such as Netflix, Codepen, and Coca-Cola have adopted serverless-based architecture stacks, leveraging serverless platforms to deploy parts of their applications (SOMU et al., 2020). According to a market report from 2023, while the global market for serverless was estimated at $9.8 Billion in 2022, it is expected to grow into $43.7 Billion by 2030 (RESEARCH; MARKETS, 2023). The serverless topic has received increasing attention in academic research, with a significant rise in the number of papers published in top-tier conferences, growing from 1 in 2017 to 22 in 2022 (WEN et al., 2023).

Although serverless applications are advantageous due to its unique characteristics, they are fundamentally different from typical legacy applications. Migrating an existing application to serverless involves decomposing the software into granular services, and refactoring existing code to adhere to the platform's API (RISTOV et al., 2020; SPILLNER; DORODKO, 2017). In addition, the black-box nature of serverless platforms lead to the lack of knowledge and control over the serverless environments, consequently adding complexity to deploying and testing applications in these platforms (MARTINS; ARAUJO; CUNHA, 2020). Since the amount of existing legacy code that must continue running is larger than new code created for serverless, the migration of existing code to serverless is an open research problem (CASTRO et al., 2019; BALDINI et al., 2017; WEN et al., 2023; WEN et al., 2021).

In this context, the complexity of migrating different application profiles varies significantly (GOLI et al., 2020). For example, applications that heavily rely on network communication will experience substantial latency impacts based on the chosen deployment region within the cloud provider. On the other hand, the performance of CPU bound applications will depend on the vertical scalability options in the serverless platform and the underlying hardware. In addition, serverless architectures are particularly cost-efficient for workloads with highly fluctuating demand, whereas hosting dedicated infrastructure is generally more economical for steady request patterns (PEKKALA, 2019; GOLI et al., 2020; LIU; NIU, 2023). It is important to note that these factors can give rise to escalating costs, further emphasizing the significance of careful consideration and analysis during the migration process. From a software migration standpoint, predicting serverless costs from existing non-serverless workloads is specially helpful when evaluating if the migration is financially worthwhile.

By definition, an I/O bound application spends more time doing I/O than performing computations or network communication (SILBERSCHATZ; PETERSON; GALVIN, 2012). I/O bound applications running on serverless such as MapReduce sort, and distributed video processing often rely on the local file system for temporary or intermediary storage (KLIMOVIC et al., 2018a). Serverless platforms support these use cases by providing functions with a local file system for temporary storage with potentially better performance when compared to other dedicated cloud storage services for having lower or non-existent network overhead (AWS, 2023f; HELLERSTEIN et al., 2018). Other applications can leverage it for caching static assets at lower costs when compared to in-memory alternatives or external caching services (AWS, 2023e). It can also be used to store extra libraries or tools while a function is running (LEE; SATYAM; FOX, 2018). On the other hand, the lack of configuration options for local storage on serverless creates a barrier for migrating an existing I/O bound application to serverless while reliably meeting performance requirements. Since the cost of running an application on serverless depends on function execution time, the local storage performance has direct impact on cost for I/O bound applications (KELLY; GLAVIN; BARRETT, 2020). In addition, the high-level infrastructure abstraction that serverless provides makes it difficult for developers to know how a given function will perform (KIM; LEE, 2020).

In this sense, performance variability is a major issue for serverless applications running in the cloud and can be explained by the usage of shared hardware by cloud services and configuration of compute and storage resources (ROY; PATEL; TIWARI, 2021; KLIMOVIC et al., 2018a). High variability leads to the lack of performance predictability. In addition, substantial disparities in performance and variability can be observed not only within the serverless execution environments of a single provider but also when comparing different providers with the same configuration (JACKSON; CLYNCH, 2018; COPIK et al., 2021). These disparities are even larger for I/O bound workloads (COPIK et al., 2021). Furthermore, serverless platforms do not offer any explicit assurances regarding execution performance, nor are they obligated to meet a predefined standard of performance (ELSAKHAWY; BAUER, 2021; HELLERSTEIN et al., 2018; MAISSEN et al., 2020; SINHA; KAFFES; YADWADKAR, 2024). From a software migration standpoint, and since cost depends on performance, having predictable I/O in serverless platforms is specially helpful when evaluating if porting existing applications to serverless is financially worthwhile.

The current academic body of knowledge around serverless performance has focused on the AWS Lambda platform and CPU-bound functions, at the expense of local file system I/O workloads. Analyzing GCF enables investigating whether the conclusions previously drawn for AWS Lambda can also be applied to GCF. In addition, there is a scarcity of studies that attempt to comparatively evaluate performance and predictability in the serverless context, while the few existing attempts did not provide enough statistical background to support significance claims (WEN et al., 2023). Finally, running a performance analysis in GCF in 2023 has the possibility of yielding different results from previous work due to the release of 2nd gen functions in 2022 or other platform enhancements (PARK; KIM; LEE, 2020; LEE; SATYAM; FOX, 2018; GCP, 2023a).

Previous work that studied local file system I/O workloads contributed in identifying isolated factors that can affect performance in this context. In addition, different studies drawn divergent conclusions regarding a coincident factor. For this reason, the lack of a comprehensive set of performance factors in the local file system I/O context is also a gap. Having an aggregation of these performance factors enables developers from gaining a deeper understanding of the underlying dynamics of serverless platforms and how it impacts cost.

The context of local file system I/O operations in serverless function environments, the black-box nature of serverless platforms and the aforementioned gaps led to the following set of Research Questions (RQ):

- **RQ1**: What is the aggregation of factors that affect performance and predictability in the context of local file system I/O operations from serverless functions?
- **RQ2**: Between AWS Lambda and GCF, which serverless provider yields the best performance for local file system I/O operations? In this context, the comparison should be evaluated from the perspective of a chosen performance metric.
- **RQ3**: Between AWS Lambda and GCF, which serverless provider has the most predictable performance for local file system I/O operations? In this context, more predictability means less variability and dispersion of performance.

Therefore, this work proposes a comparative analysis of performance and predictability between AWS Lambda and GCF functions for local file system workloads through experimentation. Within the context of migrating I/O applications to serverless environments, this analysis aim to offer insights that illuminate the black-box nature of serverless platforms in terms of performance and predictability for I/O workloads. Even though the aforementioned platforms provide access to a local file system from function code, they do not provide enough configuration options that allow for specifying the performance levels to be expected from I/O operations. This analysis also aims to provide insights into the performance behavior of serverless functions, while providing developers more information on the performance level that can be expected for local file system I/O operations in these platforms.

To the best of our knowledge, this work is the first to conduct this analysis on 2nd generation functions in GCF. Other cloud services that provide storage and can be used in conjunction with serverless functions, like Amazon Simple Storage Service (S3) and Google Cloud Storage (GCS), are not part of the scope for this work.

1.1 OBJECTIVES

This section presents the general and specific objectives of this work.

1.1.1 General

To perform a comparative analysis of the performance and predictability of local file system I/O operations in AWS Lambda and GCF by means of experimentation with the aim to help developers making informed decisions on migrating I/O bound applications to serverless.

1.1.2 Specific

- From the literature and related work, to aggregate the factors that influence serverless local file system I/O performance and predictability.
- To develop heuristics for creating a subset of factors that are relevant for experimentation.
- To select an experiment design along with a number of repetitions and response variables.
- To determine appropriate levels for the subset of relevant factors.
- To perform experiments using the selected factors and levels in AWS Lambda and GCF.

- To analyze the results of the experiments.
- To establish a comparison of performance and predictability of local file system I/O operations in AWS Lambda and GCF.

1.1.3 Contributions

This work aims to contribute by: 1) providing an aggregation of the factors that influence performance and predictability and 2) a performance and predictability comparison analysis of local file system I/O operations between AWS Lambda and 2nd generation functions in GCF.

1.2 WORK STRUCTURE

This work is structured as follows: Chapter 2 lays the necessary theoretical background to support the understanding of this work and the proposed performance analysis including the following: Serverless technologies, file system I/O performance, performance analysis and analysis of variance. Chapter 3 presents previous work related to this research including a discussion of their limitations and how this work aims to fulfill them. Chapter 4 describes the methodology utilized for the experiments. Chapter 5 presents the experiments and results to support the performance and predictability comparisons discussed in Chapter 6. Finally, Chapter 7 showcases the conclusions and opportunities for future work.

2 BACKGROUND

This chapter provides the necessary background to understand the proposed performance and predictability analysis in this work. First, Section 2.1 discusses aspects related to serverless applications in general, as well as specifics related to AWS Lambda and GCF. Next, Section 2.2 presents the foundational theory concerning performance analysis and relevant terminology used throughout this work, followed by an overview of Analysis of Variance (ANOVA) in Section 2.3 as applied to experiments involving one or more factors. Finally, Section 2.4 introduces concepts specific to evaluating performance in the context of file system I/O.

2.1 SERVERLESS APPLICATIONS

Serverless enables running a server-based application without having to manage a server. Consumers of serverless platforms do not need to deal with tasks such as provisioning, maintenance, updates, scaling and capacity planning of server resources. Instead, these responsibilities and functionalities are seamlessly managed by a serverless platform and entirely abstracted away from developers and Information Technology (IT) operation teams. This abstraction enables developers to concentrate their efforts on crafting the core business logic of their applications (ALLEN et al., 2023). Moreover, developers can implement stateless microservices as serverless functions that can cooperate to support complex business functionality (KATZER, 2020).

Functions are executed by the platforms based on event triggers such as HTTP requests or messages from managed queues. Every time a function is triggered in a cloud hosted serverless provider, it executes in a server host via a container provisioned and managed by the platform (JACKSON; CLYNCH, 2018). Functions in a serverless architecture are designed to be stateless and short-lived, meaning they do not retain any internal state and are automatically deprovisioned if idle for a certain duration. While idle, warm function instances can be reused to respond to new requests without launching new instances. The size of a function can be determined by the CPU and memory resources allocated to it, which developers configure during setup on the serverless cloud platform. Furthermore, functions have the capability to interact with temporary storage using the local file system, allowing them to handle data that is required during their execution but does not need to be persistent (SREERAM, 2017; AWS, 2023l; GCP, 2023i).

One of the main advantages of serverless is its pay-per-use billing model, in which cloud providers do not charge for idle time. Even though serverless pricing varies between providers, it generally depends on function size, temporary storage size, number of executions, data transfer and execution time. The minimum increment measured for function execution time also differs between cloud providers. In addition, serverless functions have automatic horizontal scalability depending on load while the infrastructure abstraction provides out-of-the-box security and reliability (KATZER, 2020).

Serverless function invocations can be categorized as either ”warm” or ”cold” starts. When a function is invoked for the first time or after a period of inactivity, a new function instance is created, resulting in a "cold start". This initial invocation incurs additional overhead due to the container instantiation process. However, subsequent invocations of the same function can leverage the existing instance, avoiding this overhead and they are referred to as "warm starts". Cloud platforms typically retain existing idle instances for a limited period of time to handle future invocations, after which they may evict them to reclaim resources (SCHIRMER et al., 2023; WEN et al., 2021). This period of time is not configurable by developers. Consequently, function instances can suddenly terminate leading to unexpected cold starts (WANG et al., 2018; EISMANN et al., 2022).

Serverless platforms share the same approach to executing functions. Each function operates within a designated function container on a host Virtual Machine (VM) located within a predefined region of the respective serverless platform’s cloud infrastructure. Usually, these execution VMs maintain isolation from other individual users. Nonetheless, it is possible for VMs of distinct users to coexist within the same region, potentially giving rise to instances of interference effects (KELLY; GLAVIN; BARRETT, 2020). In addition, it is possible for multiple function instances of the same user to coexist in a host VM, leading to interference effects due to concurrent executions on the same host (KIM; LEE, 2020; LEE; SATYAM; FOX, 2018). This lack of isolation can lead, for example, to network I/O throughput contention when multiple function instances are concurrently running on the same host.

Even though functions can connect to other cloud storage services to share data between executions, this connection is made through a network interface. For this reason, recent studies showed the bandwidth experienced for these connections are an order of magnitude slower than a single modern Solid State Drive (SSD) (HELLERSTEIN et al., 2018). Alternatively, serverless functions can leverage a local file system for temporary storage with better performance when compared to other storage alternatives (AWS, 2023f). In AWS Lambda, this local file system is shared between function instances running on the same host VM (MAISSEN et al., 2020), while GCF provides more isolation to memory, global variables, file systems and application state (GCP, 2023e). Still, none of these serverless platforms offer developers the ability to provision the desired amount of Input/Output Operations Per Second (IOPS) and bandwidth for local temporary storage.

While increasingly accessible, the serverless computing paradigm is relatively new and proves most advantageous for: Asynchronous and easily parallelized workloads, resources with infrequent or sporadic demand, ephemeral and stateless applications and for highly dynamic businesses that require agile developer responsiveness due to changes in requirements. Currently, it represents an emerging computing model that encounters challenges in terms of standardization, ecosystem maturity, and the availability of comprehensive and stable documentation, as well as a set of established best practices (ALLEN et al., 2023). The following sections present the main aspects of serverless offerings from AWS Lambda and GCF along with their specific features.

2.1.1 Features from AWS Lambda and GCF

In AWS Lambda, developers can configure the memory allocated to a function, which in turn determines the amount of virtual CPU available to it. Adding more memory proportionally increases the amount of CPU, making memory allocation a crucial factor in this context. Currently, functions have the equivalent of 1 vCPU for every 1769 MB of allocated memory (AWS, 2023j). Execution time is measured in 1 ms increments. In AWS Lambda developers are not able to configure bandwidth or IOPS for temporary storage while its size can be increased independently from memory. Developers can choose between using the local temporary storage or other storage offerings on AWS to support AWS Lambda functions.

AWS Lambda also supports external extensions to augment functions with monitoring, observability, security, and governance tools (AWS, 2023h). In this context, Fig. 1 presents the lifecycle of the execution environment of a AWS Lambda functions. The Initialization phase (INIT) starts upon a function's trigger, subsequently leading to the execution of the function's code in the Invocation phase (INVOKE). The runtime environment has the potential to be recycled for subsequent invocations and will run INVOKE without needing a new INIT. The runtime environment is reused until the platform determines it is appropriate for shutdown.

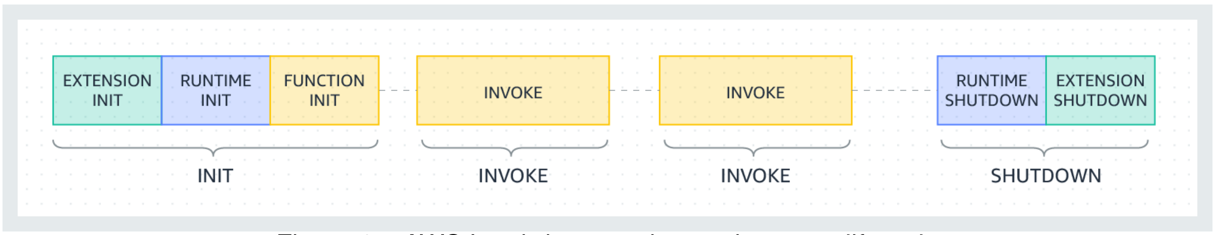

Figure 1 – AWS Lambda execution environment lifecycle
Source: The author

In GCF CPU allocation for a function is also derived from the memory allocated by developers. In contrast to AWS Lambda, the resource allocation options for GCF are divided in tiers (GCP, 2023g). The local file system is held in memory for GCF, which means that the size of its underlying storage cannot be increased independently and any consumed space is taken from the overall memory allocated of the function. Execution time is measured in 100 ms increments (GCP, 2023b). Similarly to AWS Lambda, GCF does not offer configuration options for bandwidth or IOPS related to temporary local storage. The lifecycle of functions in GCF is also represented by Fig. 1 with the exception of the external extensions steps (green) since it is an AWS Lambda specific feature.

Google announced the general availability of 2nd gen GCF in 2022 (GCP, 2023a). When compared to 1st gen functions, 2nd gen GCF has a wider range of resource allocation tiers, allows for a higher limit of function execution time, among other features. From the user's standpoint, GCF has progressed across various dimensions from its first generation while upholding its foundational principles. Consequently, this work adopts a broad reference to GCF, while exclusively distinguishing between 1st and 2nd generation functions for generation-specific attributes.

The support provided by serverless platforms to some features can vary. Table 1 was inspired by (COPIK et al., 2021) and presents an updated comparison between features in AWS Lambda and GCF (GCP, 2023f; AWS, 2023g; GCP, 2023j; AWS, 2023k; AWS, 2023c; GCP, 2023g; AWS, 2023d; GCP, 2023b; AWS, 2023i; GCP, 2023h; AWS, 2023j; AWS, 2023m; GCP, 2023c). The most notable additions are related to the 2nd generation functions in GCF.

AWS Lambda supports a total of 6 programming languages natively while GCF supports 7 being PHP the differentiating factor. Nevetherless, AWS Lambda supports custom runtimes through containers with a compatible Linux image, thus virtually allowing any language runtime to operate on the platform's environment. Google allows for similar customization along with container support outside of GCF in Google Cloud Run (GCR).

Even though the serverless paradigm aims for short-lived functions, AWS Lambda and GCF have been improving the time limits for function executions. AWS Lambda announced an increase in time limit from 5 to 15 minutes in 2018 (AWS, 2023b) while GCF increased this limit in the annoucement of the 2nd gen functions (GCP, 2023a).

The support for static memory allocation provided in AWS Lambda allows for configuring any value between 128 MB and 10 GB (AWS, 2023c), even though new AWS accounts are limited to 3 GB. From the documentation, it is not clear how or when an AWS account gets allowed for allocating 10 GB to Lambda functions (AWS, 2024). In contrast, GCF employs a tiered approach in which developers must select

Table 1 – Comparison of features between AWS Lambda and GCF

Feature	AWS Lambda	GCF
Native languages	Python, Node.js, C#, Java, Ruby and Go	Python, Node.js, C#, Java, Ruby, Go and PHP
Custom runtime support	Yes, through containers	No. Suggests Google Cloud Run as an alternative serverless platform with more customization options
Time limit	15 minutes	9 minutes for 1st gen functions and up to 60 minutes for 2nd gen
Memory allocation	Static. From 128 MB to 10 GB. New accounts are limited to 3 GB	Tiered. From 128 MB to 8192 MB in 1st gen. From 128 MB to 32 GB in 2nd gen
CPU allocation	Proportional to memory. 1 vCPU at 1769 MB	Proportional to memory tier. 1 vCPU at 2048 MB
Billing	Number of invocations, duration, allocated memory and temporary disk size	Number of invocations, duration and memory tier
Default maximum concurrent function invocations	1000	No limit for 1st gen. 1000 for 2nd gen
Temporary disk space	Between 512 MB and 10 GB, in 1 MB increments	File system held in memory. Cannot be provisioned independently and consumes space from memory

Source: Adapted and updated from (COPIK et al., 2021) a memory tier, with limited flexibility for in-between values. CPU is tied to memory allocation in both cases, with approximately 1.74 vCPU available in AWS Lambda when setting memory to 3 GB. Comparatively, selecting the highest resource tier of 2nd gen functions in GCF allows for allocating 32 GB of memory alongside 8 vCPU to a runtime.

In the context of serverless costs, billing is similar for both platforms: Price is calculated from the number of invocations, execution duration and allocated resources. The temporary disk space in AWS Lambda can be provisioned independently from memory and consequently is billed separately at lower rates when compared to memory (AWS, 2023e).

Automatic service scaling is one of the main benefits of serverless architectures. Platforms can horizontally scale environments up to thousands of function instances depending on application load. In this context, platforms employ a configurable limit for the maximum number of concurrent function invocation. This feature sets a scalability ceiling that prevents serverless costs from increasing unexpectedly. GCF 2nd gen evolved to match the default configuration for maximum concurrent function invocations of AWS Lambda.

2.2 PERFORMANCE ANALYSIS

Performance analysis is a combination of measurement, interpretation and communication (LILJA, 2005). Measurement involves determining a performance metric interesting and useful to measure. This selection depends on factors such as the goals of the performance analysis or the cost and complexity of collecting the selected metric (LILJA, 2005). Selecting an inappropriate performance metric can result in misleading claims about the result of the analysis. The execution time of programs is considered a consistent and reliable performance metric, and can be defined as the latency to complete a task including all associated subtasks (HENNESSY; PATTERSON, 2017). Measuring execution time in a computer system is analogous to using a stopwatch to observe the time required for some event to occur (LILJA, 2005). This metric can be categorized as Lower is Better (LB), meaning that smaller values are preferable (JAIN, 1991). In the context of file I/O, latency can be observed from the perspective of read or write operations.

The interpretation phase of the analysis involves utilizing the collected measurements to achieve the objectives of the study. Performance analysis commonly pursues goals such as comparing alternatives, assessing the impact of new features, system optimization, and tuning, among others. When comparing alternatives, the aim is to quantitatively determine the best configurations under specific conditions based on performance metrics. Finally, the output of the interpretation phase is employed to communicate and report the analysis results in a clear and consistent manner, while also providing sufficient information to ensure reproducibility (HENNESSY; PATTERSON, 2017; LILJA, 2005).

Performance variability is the measure of dispersion of a group of measurements with the aim to quantify how spread out these measurements are. More variability of performance data means less performance predictability. This index of dispersion can be used alongside an index of central tendency, such as the mean or median, to characterize a group of measurements. The standard deviation can be used as an index of dispersion even though it needs to be interpreted relative to the mean value of the group of measurements. In this context, the CV normalizes the standard deviation with respect to the mean to provide a dimensionless value. The CV compares the size of the variation with the mean value of the group of measurements (LILJA, 2005). In the context of serverless file I/O, this variability metric relates to the predictability of the performance of I/O operations. Eq. 2.1 defines the CV related to the standard deviation s and the mean $\bar{x}$. The CV is commonly represented as a percentage. It is an LB metric, meaning less CV results in more predictability.

$$CV = s/\bar{x} \tag{2.1}$$

Collecting observations for the chosen performance metric typically involves experimentation. The primary objective of experimentation is to maximize the amount of information acquired while minimizing the total number of experiments conducted. For the design and analysis of experiments, researchers often utilize the following terminology (JAIN, 1991; LILJA, 2005):

- Response variable: Can be defined as the measured performance metric or the outcome of the experiment.
- Factors: Variables that affect the response variable and consequently affect the performance metric.
- Levels: Indicate the possible values a factor can take (e.g., memory size levels such as 512 kB or 2 MB)
- Replication: Refers to the number of experiment repetitions. Replications are necessary for gauging measurement errors.
- Interaction: Occurs when two factors affect each other, leading to a dependency of one factor's effect on the response variable based on the level of the other. Conversely, a factor with no interaction independently influences the response variable, irrespective of the presence of other factors and their levels.

The full factorial experiment design involves using all possible combinations of levels and factors to observe their impact in the response variable. In an example, applying this technique for 5 factors and 3 levels per factor would result in 3^5 experiments. Adding replications would escalate the number of experiments even further (JAIN, 1991).

Alternatively, the $n2^m$ experiment design is used to determine the effect of m factors to the response variable. This technique involves n replications of the experiment and is restricted to two levels per factor. These levels are commonly chosen to be the minimum and maximum possible values of a factor (JAIN, 1991). This approach simplifies the analysis of the measured data while allowing for determining which factors have the largest impact on the response variable. The most important factors can then be examined in more detail in a full factorial experiment with a larger number of levels (LILJA, 2005). The number of experiments E can be defined as a function of m factors, l levels and n replications. Eqs. 2.2 and 2.3 present this relationship for full factorial and $n2^m$ designs, respectively.

$$E = nl^m \tag{2.2}$$

$$E = n2^m \tag{2.3}$$

In this context, ANOVA serves as a robust technique for comparing alternatives and can be used to quantify which fraction of the variation of the response variable can be explained by the factors, their interactions and measurement errors (LILJA, 2005).

2.3 ANALYSIS OF VARIANCE (ANOVA)

Section 2.3.1 describes ANOVA and its usage in one-factor experiments. Furthermore, Section 2.3.2 presents a generalization of its concept to two-factor and *m-factor* experiments.

2.3.1 ANOVA and one-factor experiments

ANOVA originates from the partition of the total variability of data into its component parts. The measure of total data variability in a single-factor experiment with a levels and n observations per level derives from the corrected sum of squares SS_T described in Eq. 2.4. In this context, y_{ij} represents the response variable measured for level i and observation j, and $\bar{y}_{..}$ stands for the grand average of all observations (MONTGOMERY, 2017).

$$SS_T = \Sigma_{i=1}^{a}\Sigma_{j=1}^{n}(y_{ij} - \bar{y}_{..})^2 \tag{2.4}$$

Furthermore, the total variability in the data represented by the corrected sum of squares SS_T can be partitioned into two other components. The first component SS_L, described in Eq. 2.5, relates to the sum of squares of the differences between the level averages $\bar{y}_{i.}$ and the grand average $\bar{y}_{..}$, and represents the variation due to the effects of the level differences. The second component SS_E, described in Eq. 2.6, relates to the sum of squares of the differences of observations within levels y_{ij} from the level average $\bar{y}_{i.}$, and represents the variation due to random errors. Finally, SS_T is presented as a combination of both components in Eq. 2.7 (MONTGOMERY, 2017; LILJA, 2005).

$$SS_L = n\Sigma_{i=1}^{a}(\bar{y}_{i.} - \bar{y}_{..})^2 \tag{2.5}$$

$$SS_E = \Sigma_{i=1}^{a}\Sigma_{j=1}^{n}(y_{ij} - \bar{y}_{i.})^2 \tag{2.6}$$

$$SS_T = SS_L + SS_E \tag{2.7}$$

In this context, the fraction of the total variation explained by the differences between levels can be represented by SS_L/SS_T. Similarly, SS_E/SS_T is the fraction of the total variation explained by random errors. The *F-test* can be used do determine if the differences between these fractions are statistically significant at a level of significance α. To test this hypothesis, the F_0 value is compared to a critical value F_{crit} obtained from the *F* distribution. This critical value depends on the significance level α, described by the $1 - \alpha$ parameter, as well as the number of degrees of freedom for both the factor being tested and the random errors, defined by $a - 1$ and $a(n - 1)$ respectively. If F_0 is bigger, then the effect of the factor being tested is statistically different from the effect of random errors (LILJA, 2005). This comparison is described in Eq. 2.8. The F_0 value is the computation of the ratio between the mean-square values of SS_L and SS_E. The mean-square value is the total variation for a component divided by its corresponding degrees of freedom, while the number of degrees of freedom for the levels and random errors are $a-1$ and $a(n-1)$, respectively. Finally, Eq. 2.9 defines F_0 (MONTGOMERY, 2017).

$$F_0 > F_{crit}[1 - \alpha, a - 1, a(n - 1)] \tag{2.8}$$

$$F_0 = \frac{SS_L/[a-1]}{SS_E/[a(n-1)]} \tag{2.9}$$

2.3.2 ANOVA applied to two-factor and *m-factor* experiments

The idea of the ANOVA for single-factor experiments can be extended to accommodate two factors, with the addition of the effects derived from the interaction between factors. Consequently, the total variation in the measurements SS_T should be partitioned into the variation explained by random error SS_E and the level differences in factors *A* and *B* as well as their interaction, represented by SS_A, SS_B and SS_{AB}, respectively. This relationship is presented in Eq. 2.10 (LILJA, 2005).

$$SS_T = SS_A + SS_B + SS_{AB} + SS_E \tag{2.10}$$

Conversely to the one-factor experiments, the response variable y for two-factor experiments are subject to a three dimensional matrix. Consequently, y_{ijk} is the *k*th measurement made with level i of factor *A* and level j of factor *B*. Assuming n measurements were made and factors *A* and *B* have a and b possible levels, respectively, the sums of squares SS_A and SS_B for these factors are defined in Eq. 2.11. The sum of squares SS_{AB} for the interaction between factors *A* and *B* is presented in Eq. 2.12 (LILJA, 2005).

Similarly to one-factor experiments, $\bar{y}_{i..}$ is the level average of all measurements made with any level of factor *B* while factor *A* is set to level i. Alternatively, $\bar{y}_{.j.}$ is the level average of all measurements made with any level of factor *A* while factor *B* is set to level j. Finally, $\bar{y}_{ij.}$ is the average of all measurements with levels i and j set to factors *A* and *B*, respectively, while $\bar{y}_{...}$ is the grand average of all observations with any level of factor *A* and *B*.

$$SS_A = bn\Sigma_{i=1}^{a}(\bar{y}_{i..} - \bar{y}_{...})^2,\ \ SS_B = an\Sigma_{j=1}^{b}(\bar{y}_{.j.} - \bar{y}_{...})^2 \tag{2.11}$$

$$SS_{AB} = n\Sigma_{i=1}^{a}\Sigma_{j=1}^{b}(\bar{y}_{ij.} - \bar{y}_{i..} - \bar{y}_{.j.} - \bar{y}_{...})^2 \tag{2.12}$$

In addition, the sum of squares for the random errors SS_E is defined in Eq. 2.13. Finally, the total variability present in the measurements can be represented by the sum of squares SS_T presented in Eq. 2.14.

$$SS_E = \Sigma_{i=1}^{a}\Sigma_{j=1}^{b}\Sigma_{k=1}^{n}(y_{ijk} - \bar{y}_{ij.})^2 \tag{2.13}$$

$$SS_T = \Sigma_{i=1}^{a}\Sigma_{j=1}^{b}\Sigma_{k=1}^{n}(y_{ijk} - \bar{y}_{...})^2 \tag{2.14}$$

In the context of the *F-test* applied to two-factor experiments, the F_A, F_B and F_{AB} terms correlate to the computation of the mean-square values of the factors and interactions divided by the mean-square value of random errors. The F values for the two-factor experiments are presented in Eq. 2.15. Similarly to the one-factor experiments, these values should be compared to a critical value F_{crit} from the F distribution, as shown in Eq. 2.8, that in turn depends on the level of significance α, the number of degrees of freedom of the factor or interaction being evaluated, and the number of degrees of freedom of random errors. The *F-test* applied to this context allows for determining whether the effects of the factors or interactions are statistically significant to the response variable. The number of degrees of freedom for random errors is $ab(n-1)$, while a and b are the number of levels factors *A* and *B* can assume. In addition, the number of degrees of freedom for factors *A* and *B* can be defined as $a-1$ and $b-1$, respectively. Consequently, the number of degrees of freedom for the interaction between factors *A* and *B* is the product $(a-1)(b-1)$ (MONTGOMERY, 2017; LILJA, 2005).

$$F_A = \frac{SS_A/[a-1]}{SS_E/[ab(n-1)]},\ F_B = \frac{SS_B/[b-1]}{SS_E/[ab(n-1)]},\ F_{AB} = \frac{SS_{AB}/[(a-1)(b-1)]}{SS_E/[ab(n-1)]} \tag{2.15}$$

The usage of ANOVA for two-factor experiments can be generalized for *m-factor* experiments given that $m > 2$. However, the computational complexity of performing this analysis progressively increases with more factors, and more importantly, with the increasing number of interactions between factors (LILJA, 2005). In practice, ANOVA is performed with the assistance of specialized software.

Even though ANOVA allows for identifying the statistically relevant factors and interactions, it does not allow for comparing them. The Tukey test is a method to compare pairs of means and can be used to compare levels, factors or interactions by providing a confidence level of the difference between the means being compared (MONTGOMERY, 2017).

The following section presents an analysis example in the context of performance experiments. This example was developed with the aim to facilitate the understanding of the analysis' steps, relevant variables, and resulting insights that can exist in a simplified scenario using ANOVA and the Tukey test.

2.3.3 ANOVA example applied to a performance experiment

The example consists of a two-factor performance experiment in the serverless context. The first factor is the amount of memory allocated to the tested function, with levels defined as 128 MB (small) and 8192 MB (large). The second factor is the serverless platform. In this case, it can be either AWS Lambda or GCF. The response variable is the execution time for a certain request measured in milliseconds. Table 2 presents the sample data used throughout the example with this setup.

Table 2 – Sample data for ANOVA example

	Serverless provider execution time (ms)	
Memory (MB)	AWS Lambda	GCF
128 (Small - S)	(59.31, 56.98, 58.26)	(80.62, 76.54, 81.08)
8192 (Large - L)	(67.84, 68.17, 67.32)	(65.53, 65.67, 64.81)

Source: The author

ANOVA calculations were performed for this example's sample data using R Studio (POSIT, 2023) with a confidence level of 99% ($\alpha = 0.01$). Table 3 presents the ANOVA table calculated for the sample data.

Table 3 – Resulting ANOVA table for the example

Source of Variation	Degrees of Freedom	Sum of Squares		F Value	$F_{crit}[0.99, 1, 8]$
		Absolute	% of total		
Memory		15.1	2.1	7.5	
Provider	1	264.8	36.9	132.3	11.3
Memory:Provider		420.2	58.7	210.0	
Residuals	8	16.0	2.2		
Total	11	716.1	100		

Source: The author

Comparing the F values to the F_{crit} allows for determining whether the effects of the factors or interactions are statistically significant to the response variable (serverless provider time) with a set confidence level. In this case, the effect of memory is not significant while the effects of the serverless provider choice and the interaction between memory and provider (Memory:Provider) are significant with a conficence level of 99%. A significant interaction means that the effect of memory to the response variable was different in each provider, even though memory was not significant by itself.

Analyzing the resulting sum of squares allow for determining which components are responsible for the variation in performance data. Residuals accounted for around 2% of total variation, meaning that most of the variation was due to the effect factors or interactions. Memory was responsible for 2.1% of total variation, which confirms the conclusion previously drawn for the F value that the effects of this factor are not significant to the response variable. Alternatively, the serverless provider and the interaction between provider and memory (Memory:Provider) were collectively responsible for 95.6% total variation and are significant to the response variable.

Even though this analysis allows for identifying the statistically relevant factors and interactions, it does not allow for comparing them. Therefore, the Tukey test can be used to compare factors within the data. Fig. 2 presents the confidence intervals for the means of the differences between factors or interactions. In this context, confidence intervals that include 0 mean that there are no evidences of statistically significant differences between the factors or interactions being compared.

The confidence interval for the difference of the means of AWS Lambda and GCF observations (Lambda-GCF) is negative, which means GCF had higher values for the response variable. Since execution time is a LB metric, it means AWS Lambda had better general performance for the experiment. Similarly, the confidence interval for the difference of the means of AWS Lambda and GCF observations with small memory values (S:Lambda-S:GCF) is also negative, which means AWS Lambda had better performance for small memory configurations.

Alternatively, the confidence interval for the difference of the means of AWS Lambda and GCF observations with large memory values (L:Lambda-L:GCF) includes zero, which means we cannot affirm there is a statistically significant difference of performance between these providers for this memory configuration.

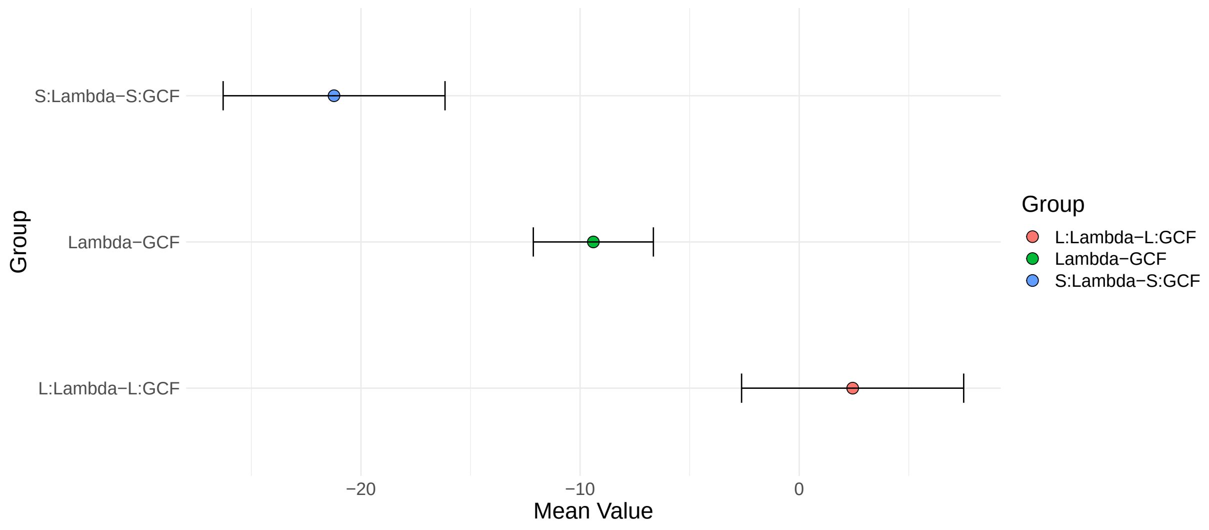

Figure 2 – Confidence intervals for the mean differences in Tukey test
Source: The author

This simplified example highlights how ANOVA can be used to check whether factors and interactions are statistically significant to the response variable, and how the Tukey test can be used for comparative analysis. This work leverages both statistics tools and the principles presented in this example to evaluate performance in a real scenario that includes a wider range of factors, interactions and levels in the serverless context.

2.4 FILE SYSTEM I/O PERFORMANCE

The performance of file systems often take precedence over storage devices performance for I/O applications. This is because applications directly interact with the file system rather than the storage device itself, creating an abstraction layer that enables optimizations and caching techniques aiming to reduce the time applications wait for an I/O operation to finish. In this context, logical I/O operations are issued by applications to the file system, while physical I/O operations are issued by file systems to storage devices. Latency is measured as the time it takes for a logical file system request to reach completion. One strategy to study performance in this context is to focus on the latency of logical I/O operations while treating the file system as a black box (GREGG, 2014).

Random read, sequential read, random write, and sequential write are commonly tested operations for file system I/O performance analysis (GREGG, 2014). In the context of file operations, the sequential access pattern involves the reading or writing of all bytes or records in a continuous order, progressing from the file's start to its end. In contrast, the random access pattern relates to the non-sequential retrieval or modification of data within the file, requiring an initial position from which the operation begins, rather than obligatorily starting from the first byte of the file (TANENBAUM; BOS, 2014). The Device-to-Device copy (dd) command can be used as an ad hoc benchmark tool for file system performance of sequential read and write operations (GNU, 2023).

The caching techniques employed by file systems often involve holding data in memory for faster read and write operations (TANENBAUM; BOS, 2014). For this reason, the latency experienced by I/O applications depends on the amount of memory available, the I/O size and the Working Set Size (WSS) being read or written. The WSS relates to the size of the file being writte or read, or the volume of data being accessed while I/O size relates to the amount of data transferred by each I/O. Since starting an I/O operation has a performance cost, applications executing random I/O operations might benefit from small I/O sizes, while large I/O sizes are more suited for sequentially reading or writing on large files. In the context of a read operation, a small WSS may return entirely from cache while a large WSS may need to be read from a storage device with bigger performance cost (GREGG, 2014).

Understanding aspects specific to file system operations, while narrowing down from general performance analysis concepts, is key to evaluating performance and predictability in this context. For instance, latency was used as the response variable for performance experiments in this work while having I/O access patterns and operation types alongside the file size as factors. In addition, the existence of caching behavior on the file system layer is a important factor when analyzing performance results and defining WSS levels.

3 RELATED WORK

This chapter presents previous studies related to the performance and predictability analysis performed in this work. First, we present previous research that identified and discussed factors that affect I/O performance in the serverless context. Then, we discuss other work that approached I/O performance evaluations of serverless-compatible shared file systems such as Elastic File System (EFS). Even though shared file systems are out of scope for this work, previous research around this topic assists in understanding general aspects of serverless I/O performance analysis. Finally, we present studies that closely relate to the scope of this work and evaluated I/O performance specifically for serverless local file system workloads followed by other work that discussed aspects related to serverless performance predictability. This collection of related work is then discussed in Section 3.5 along with an aggregation of factors identified on previous research that impact the performance and predictability of serverless local file system I/O workloads.

3.1 FACTORS AFFECTING SERVERLESS I/O PERFORMANCE

The work of (EISMANN et al., 2022) conducts a performance evaluation in AWS Lambda for a multi-function, network I/O bound serverless application. It was found that performance varies significantly over long periods of time even though it is stable within multiple function executions in the same day. For this reason, performance measurements should be performed periodically so as to keep up with optimizations and improvements to the underlying hardware of serverless providers. Long term performance fluctuations are also due to extended warm-up periods of function environments. Furthermore, the study revealed that the overall performance is adversely affected by the presence of multiple concurrent function executions on the same host. Serverless platforms control the allocation of function instances to hosts, consequently, this factor is not under the control of developers. Additionally, cold starts can unpredictably occur as a result of function instances being recycled by the platform. Although this research did not explicitly address GCF, nor did it specifically concentrate on file system I/O performance, its findings make a valuable contribution to the broader comprehension of performance behavior within AWS Lambda serverless environments.

Alternatively, the findings in (GINZBURG; FREEDMAN, 2020) suggest that serverless performance can be affected by time of day and deployment region. Benchmark results from a variety of application classes demonstrated spatial and temporal performance patterns in AWS Lambda meaning that developers can take advantage of function placement and time-shifting workloads to achieve better performance and minimize costs. Furthermore, the study revealed varying levels of performance variability among different application classes. Even though one of the applications tested involved file system I/O, the study was dedicated to exploring spatial and temporal performance patterns and not the latency of isolated file system operations. In addition, this research focused on AWS Lambda and did not extend its analysis to encompass comparisons across other serverless providers.

The work in (JACKSON; CLYNCH, 2018) had used empty functions deployed to AWS Lambda and Microsoft Azure Functions (MAF) to investigate the impact of programming language runtime to warm and cold start performances. The findings revealed significant performance differences not only among different language runtimes within the same provider but also across both serverless platforms for the same language runtime. The C# .NET runtime demonstrated the best overall performance when executed in MAF. This performance advantage can be attributed to differences in platform implementation. Specifically, MAF utilizes Windows container technology, while AWS Lambda relies on Linux-based containers. This study exposes the cost implications of choosing a poorly performing language runtime for a serverless provider and the impact of platform implementation to performance.

The impact of language runtime to performance was also evidenced in (BORTOLINI; OBELHEIRO, 2020) alongside the memory allocated to functions. The experiments were conducted in AWS Lambda, GCF and IBM Cloud Functions for a CPU-bound function. Results showed performance differences of up to 15x depending on the choice of programming language. It was also found that memory allocation has an impact in performance and cost with different magnitudes depending on the serverless provider.

The study in (SCHIRMER et al., 2023) has used a CPU-bound function deployed on GCF to analyze performance variability over several months. The results revealed performance discrepancies of up to 15% within a single day since executions during business hours tend to be slower. This observation aligns with the findings reported by (LAMBION et al., 2022) in a study of AWS Lambda across four deployment regions, where similar patterns were observed for executions within the same day. In addition, significant performance differences were found between measurements taken over the week and weekends. Cold start occurrences also increased during working hours, indicating higher instance recycling during periods of high demand. Additionally, it was observed that performance variability decreased as more memory and CPU resources were allocated to the function, making it more predictable. It is noteworthy that this research focused exclusively on performance within GCF and did not investigate file system I/O bound functions or has explored platforms beyond GCF. Nonetheless, these studies demonstrated how factors such as time of day and day of the week can impact function performance and highlighted the sensitivity of performance predictability to function resource configuration.

The SeBS benchmark suite introduced in (COPIK et al., 2021) covered a range of serverless applications including functions for multimedia and scientific workloads. The benchmarks in SeBS were evaluated in AWS Lambda, GCF and MAF to provide insights into platform performance and consistency. The results showed I/O related workloads had higher distribution of latencies and more outliers when compared to other experiments. In addition, it was found that I/O performance increased with more memory allocated to functions, and that AWS Lambda was faster in most of the tested scenarios. In contrast, the findings in (LLOYD et al., 2018) suggest the relationship between performance and function resources may not be linear, as adding additional resources have shown diminishing returns in improving function performance. Even though SeBS provided new insights to performance behavior in serverless, its benchmarks were not focused on I/O nor isolated file system I/O operations during the experiments.

A benchmarking suite was also introduced in (MARTINS; ARAUJO; CUNHA, 2020) with the aim to test the effects of memory allocation, the performance of CPU-bound cases, the impact of the choice of programming language, overall platform overhead, among others. The tested providers were AWS Lambda, GCF, MAF and IBM OpenWhisk. The findings suggested significant impact of memory allocation to function performance. In contrast, it was found that in general the choice of programming language does not affect warm start performance within the same serverless provider, even though different platforms might deliver different performance levels for the same programming language. The overhead imposed by serverless providers' platforms was tested by running a function with very low computational effort and measuring its latency. It was found that the platform overhead was negligible between AWS Lambda, GCF and MAF, even though it was significant for IBM OpenWhisk. The comparisons made for the tested factors were not statistically backed, and there was no evaluation of performance variation.

3.2 SERVERLESS SHARED FILE SYSTEM I/O PERFORMANCE

Even though serverless functions have access to a local file system, its underlying storage is not accessible by other function runtimes. Some applications require sharing data in a remote data storage. This capability is commonly achieved through storage offerings such as EFS and S3 from AWS, and Cloud Storage from Google Cloud Platform (GCP) (AWS, 2023f; GCP, 2023d). The following related work investigated file system I/O performance for serverless applications interacting with shared storage offerings in the cloud. Shared storage solutions for serverless are not part of the scope for this work since it focuses on local file system I/O performance. However, studies that performed measurements on shared file systems compatible with serverless applications can give relevant insights to be applied on functions interacting with the local file system.

The work in (ROY; PATEL; TIWARI, 2021) compares the performance of high concurrency I/O between S3 and EFS in the context of AWS Lambda and AWS using 3 benchmark applications. In this scenario, applications can interact with EFS through a standard file system interface while I/O operations targeting S3 are performed through an HTTP Application Programming Interface (API). It was found that concurrency levels negatively affects the performance of EFS but not S3, meaning that the latter is preferred for high concurrency workloads.

Some serverless applications may also consume temporary intermediate data in a shared storage. The *Pocket* distributed data store presented in (KLIMOVIC et al., 2018b) followed the motivations described in (KLIMOVIC et al., 2018a) to provide a low cost storage solution for serverless analytics with the capability to automatically scale CPU, network bandwidth and storage capacity so that applications are not bottlenecked by I/O. Results showed that *Pocket* achieved similar performance to ElastiCache Redis, which is a in-memory datastore service provided by AWS (AWS, 2023a), while reducing cost by almost 60%.

3.3 SERVERLESS LOCAL FILE SYSTEM I/O PERFORMANCE

Serverless functions can access temporary storage using the local file system. In contrast with other offerings like S3 and GCS, the underlying storage for the local file system of functions is not guaranteed to be shared between different runtimes (SREERAM, 2017; AWS, 2023l; GCP, 2023i), even though function instances will share it if they run concurrently in the same VM host (MAISSEN et al., 2020). The following work closely relate to the scope of this work and focused on investigating performance aspects of the local file system of serverless functions in the cloud.

The impact of concurrent function invocations in local file system I/O latency was investigated by (LEE; SATYAM; FOX, 2018) while comparing the performance overhead from concurrency between AWS Lambda, GCF, MAF and IBM OpenWhisk. The study found that the overhead for I/O operations to the temporary directory ranged from 91% in AWS Lambda to 338% in IBM OpenWhisk for 100 concurrent executions, while MAF failed to complete the executions within the time limit of 5 minutes. Even though the influence of other factors were not investigated in this work, these results illustrate how different levels of latency can be expected due to implementation differences in the underlying serverless platforms. In addition, it establishes the number of concurrent function executions as a relevant factor to file system I/O performance.

The work in (KIM; LEE, 2020) explored local file system read and write I/O performance in the serverless context using a suite of I/O benchmarks provided in FunctionBench (KIM; LEE, 2019) while evaluating the impact of memory size configurations and concurrent execution in AWS Lambda. Even though function placement cannot be controlled by users, I/O bandwidth is negatively impacted by the number of concurrent executions on a function host. It was also found that I/O performance is indirectly affected by CPU performance and increases proportionally to resource allocation. In addition, the performance of I/O bound applications like image processing and MapReduce was compared between AWS Lambda and GCF. It was found that I/O is a bottleneck for functions with high resource allocation, and not having isolation for allocating or limiting I/O resources negatively impacts overall performance in this context. On the other hand, not having this level of isolation is beneficial to I/O performance with low resource allocation. This work has not compared I/O performance between the providers.

The negative impact of concurrent function executions on the same host to local file system I/O performance was also found in (WANG et al., 2018). The results of experiments in AWS Lambda, GCF and MAF showed different I/O throughput contention characteristics for each provider. In all cases, the I/O throughput of function instances decreased as concurrency levels increased. I/O throughput measurements for AWS Lambda were stable between different instances, with CVs ranging from 1% to 6% for all concurrency levels. In GCF and MAF, the maximum measured throughput per instance increased as more resources were allocated to functions. In contrast, the maximum file system I/O throughput for an individual function instance in AWS Lambda was measured at low resource allocation, meaning that the throughput scaling is limited in this platform. This suggests that overallocating resources to functions lead to an increase on the available throughput for I/O operations per instance in GCF and MAF but not AWS Lambda. This study did not cover a performance comparison between providers. It also did not provide enough performance variability information for assessing the predictability of functions in GCF and MAF.

The performance impact of the adoption of microVMs by AWS Lambda and GCF was investigated in (PARK; KIM; LEE, 2020). Performance was evaluated with CPU, network and file system I/O intensive workloads while varying memory configurations and the level of concurrent executions. Consistent with prior research, this study observed a proportional enhancement in performance corresponding to the resource allocation of the function. However, in contrast to earlier findings, this study discovered that performance degradation was not substantial as concurrency levels increased. This can be attributed to recent improvements on function runtime scheduling algorithms, which ensured the execution of functions instances in separate virtual machine hosts and mitigated any negative impact on performance. These findings highlight the evolving aspect of serverless as an emerging paradigm and the importance of repeating performance evaluations periodically. This study did not establish a statistically supported comparison of performance and predictability among the different serverless providers.

A comparison of latency and bandwidth between EFS and local storage in AWS Lambda was explored in (CHOI; LEE, 2020). This evaluation was performed using the *FunctionBench* I/O benchmarks (KIM; LEE, 2019) under different two memory configurations: 512 MB and 2 GB. The outcomes revealed that augmenting the configured RAM size of AWS Lambda does not improve performance with EFS, whereas the benefit for local storage is significant. Local storage also had higher bandwidth and lower latency when compared to EFS for all memory configurations. The authors suggested that the results of the comparison were heavily influenced by network performance affecting communication between function runtimes and the remote EFS storage. This bottleneck is not present while accessing function local storage.

The work in (KELLY; GLAVIN; BARRETT, 2020) measured local file system I/O performance in AWS Lambda, GCF and MAF with the goal to unveil the architecture characteristics of the underlying platforms, as well as the effect of factors like memory allocation and environment isolation to overall performance. Its results confirmed the findings in (WANG et al., 2018): The file system I/O throughput increased proportionally to the resource allocation of functions in GCF, while in AWS Lambda the throughput scaling reached a plateau at low memory allocation (512 MB). In addition, GCF had superior overall I/O throughput when compared to AWS Lambda and MAF, while the latter had the lowest throughput. This performance was attributed to GCF having an in-memory file system. The range of performance data was higher in GCF when compared to other platforms because of the heterogeneity identified in underlying hardware. The results from MAF were limited since the authors were not able to configure function resources due to MAF dynamic allocation characteristics. The performance claims were not supported by statistical data: No confidence intervals were reported for performance, and no CVs were reported for data variability. Finally, results showed temporal patterns similar to those found in (GINZBURG; FREEDMAN, 2020; SCHIRMER et al., 2023; LAMBION et al., 2022) with latency peaks measured at around 12:00pm, which correlates to higher use during business hours.

3.4 SERVERLESS PERFORMANCE PREDICTABILITY

Even though the serverless topic increased in popularity over the years, the minority of published papers analyzed the statistical variation of the performance data. However, it is the case for the research in (WEN et al., 2023) which has investigated the end-to-end response latency variance of 65 open source serverless functions in AWS Lambda and GCF platforms. It was found that function performance tends to have higher CV on warm starts compared to cold starts. It is also recommended that function executions are repeated at least 50 times to achieve tight confidence intervals around the mean. While this research examined performance variability on serverless platforms, it focused solely on executing the tested functions on their respective serverless platforms. It did not account for the application classes or consider the impact of platform-specific characteristics on each class. Additionally, it did not conduct a comparative analysis of performance predictability among different providers.

Performance predictability can also be affected by the variability introduced from factors outside of developer's control. The study in (ELSAKHAWY; BAUER, 2021) employed a CPU bound serverless function written in Python to examine how the selection of an operating system, Python engine compiler, and language runtime version influences the overall performance and variability. The outcomes of the study indicated considerable variations in performance across different providers, with CVs approximately at 73% in GCF, 7% in AWS Lambda and 18% in MAF for the Python 3.7 runtime. It was observed that the factors under evaluation had a noteworthy influence on the overall performance, while having no impact on performance variability. Instead, the variability was attributed to platform-specific architectural or implementation decisions. Even though this work solely evaluated a CPU bound workload while isolating it from I/O latency, it provides an indication of the impact of cloud platform choice on performance predictability.

3.5 CONSIDERATIONS ABOUT THE RELATED WORK

The infrastructure abstraction offered by serverless computing can indeed be advantageous, but it also limits developers from gaining a deeper understanding of the underlying server architecture. Consequently, previous research in this domain has primarily focused on identifying isolated factors that significantly impact performance, with the aim of shedding light on the black-box nature of serverless platforms.

The first contribution of this work is to provide an aggregation of the factors that affect local file system I/O performance and predictability. Experimenting with these factors allow for investigating and comparing their impacts in performance and predictability for a local file system I/O workload across providers like AWS Lambda and

GCF. This aggregation answers RQ 1, and was made possible by analyzing prior research and the literature around file system I/O. In addition, the findings in (KIM; LEE, 2020; PARK; KIM; LEE, 2020; CHOI; LEE, 2020; WANG et al., 2018) suggest CPU allocation affects the performance of I/O operations in serverless environments. For this reason, related work that investigated the performance of CPU bound workloads were also taken into consideration for understanding the factors that can indirectly impact the latency of I/O operations in the context of serverless. The following list presents the resulting aggregation of factors that affect local file system I/O performance and predictability:

1. Cloud serverless platform: The work in (COPIK et al., 2021; MARTINS; ARAUJO; CUNHA, 2020; WEN et al., 2023; ELSAKHAWY; BAUER, 2021; LEE; SATYAM; FOX, 2018; KIM; LEE, 2020; WANG et al., 2018; PARK; KIM; LEE, 2020; KELLY; GLAVIN; BARRETT, 2020; BORTOLINI; OBELHEIRO, 2020) experimented with more than one serverless platform and reported performance differences between providers.

2. Time of day: While (EISMANN et al., 2022) reported stable performance for multiple executions on the same day, the studies in (GINZBURG; FREEDMAN, 2020; SCHIRMER et al., 2023; LAMBION et al., 2022; KELLY; GLAVIN; BARRETT, 2020) found temporal patterns suggesting that latency tends to be higher during business hours.

3. Day of the week: The findings presented in (SCHIRMER et al., 2023) indicate that performance tends to improve during weekends. This observation implies a potential interaction with the time of day, suggesting that measurements taken during daytime on weekends might yield different results compared to those taken on weekdays.

4. Concurrent executions on the same host: The works in (KIM; LEE, 2020; LEE; SATYAM; FOX, 2018; WANG et al., 2018) found that concurrent executions on the same host can negatively impact performance, while (PARK; KIM; LEE, 2020) found that the existence of this impact depends on the function runtime scheduling algorithms employed by the platforms.

5. Deployment region: The findings in (GINZBURG; FREEDMAN, 2020) reveal performance differences depending on the deployment region for an AWS Lambda function by measuring executions in South Korea and the United States. Still, it is not clear how much of the difference was due to infrastructure heterogeneity between regions or the time of day in different time zones.

6. Hardware heterogeneity: The effect of hardware heterogeneity to performance variability was reported by (KELLY; GLAVIN; BARRETT, 2020). In addition, the variability results in (ELSAKHAWY; BAUER, 2021) also suggest a similar effect. In general, higher levels of hardware heterogeneity leads to less performance predictability.

7. Memory and CPU allocation: CPU resources are derived from memory allocation in AWS Lambda and GCF. Consequently, function memory allocation can have an effect on I/O performance (COPIK et al., 2021; MARTINS; ARAUJO; CUNHA, 2020; BORTOLINI; OBELHEIRO, 2020) even though it has diminishing performance returns when increased beyond a threshold (LLOYD et al., 2018). In addition, the share of file system I/O throughput for individual function instances is affected by memory allocation as reported by (WANG et al., 2018; KELLY; GLAVIN; BARRETT, 2020). The work in (KIM; LEE, 2020; PARK; KIM; LEE, 2020) found significant impact of CPU allocation in local file system I/O performance respectively for AWS Lambda and GCF, while (SCHIRMER et al., 2023) found variability decreased as more resources were added to the function instances in GCF, thus increasing predictability. Besides, the amount of available memory can dictate the number of cache hits from file system I/O operations (GREGG, 2014).

8. Programming language runtime: As investigated by (ELSAKHAWY; BAUER, 2021; JACKSON; CLYNCH, 2018; BORTOLINI; OBELHEIRO, 2020), the choice of language runtime interacts with the underlying serverless platform implementation and affects function performance most notably for cold starts. Alternatively, (MARTINS; ARAUJO; CUNHA, 2020) found that the selection of a programming language typically does not have an effect on warm start performance within a serverless provider, even though different platforms may yield varying performance outcomes for the same programming language.

While previous studies have identified isolated factors that influence performance, most of them did not conduct statistical analysis to examine the interactions between these factors. Conducting an analysis of interactions between factors requires simultaneously experimenting with all factors with varying levels, thus requiring a high amount of experiments and repetitions. The analysis of interactions between factors is not part of the scope for this study, but is a suggestion for future work.

Some performance factors are not controllable by developers. For example, there is no guarantee that sequential function triggers will lead to concurrent executions on the same host since the runtime allocation and host VM lifecycles are managed by the serverless platforms. Also, some workloads are time sensitive and cannot be shifted outside of business hours or to weekends, and are vulnerable to interference due to the lack of resource isolation in the platforms. This highlights how the lack of control over the infrastructure and platform implementation can prevent developers and users to develop accurate performance expectations. The effect of these factors can introduce interference that in turn affect the level of performance predictability.

The level of hardware heterogeneity is a characteristic of the underlying platform and may change over time and across regions. The deployment region and hardware heterogeneity factors may be interchangeable, assuming the serverless platform implementation and function allocation algorithms are the same between regions. Since developers can choose the deployment region for functions, they can use this as a tool to make a choice with the least hardware heterogeneity. In contrast, the choice of deployment region usually maximizes the proximity to users in order to minimize the effects of communication latency. In addition, cloud providers do not document the expected levels of hardware heterogeneity in each region. As a result, the practicality of tuning performance and predictability via the selection of a deployment region is limited.

Similarly, the choice of programming language will often be guided by cold start performance for short lived functions. The differences observed for warm start performance will be more noticeable the longer a function executes. Functions developed in C# .NET are an exception to this rule when executed in Microsoft' platform. Also, measuring the latency from file system operations enables gauging performance without interference of the programming language choice.

Table 4 presents a comparison of related work regarding the existence of file system I/O performance investigation, the serverless providers involved in the studies and the type of comparisons conducted. Even though performance and predictability comparisons between serverless providers were established in previous work, none of them supported the comparisons with statistical evidence and confidence levels. In addition, none of the previous work made comparative claims for predictability of performance in the context of file system I/O.

Most previous work has focused on AWS Lambda as it is the most popular serverless platform to the detriment of other platforms such as GCF. This work aims to bridge this gap by comparing performance between providers and investigating whether the conclusions previously drawn for AWS Lambda can also be applied to GCF. Additionally, the majority of prior research focused on assessing CPU bound functions, which represents a general use case, while local file system I/O constitutes a relatively specialized and niche application within the serverless domain. This work aims to bridge this gap by contributing with insights around the underrepresented local file system I/O use case. To the best of our knowledge, the most recent work that investigated local file system I/O in GCF was published in 2020 when 2nd gen functions

Table 4 – Comparison of related work

Related work	Local file system I/O	Serverless provider		Comparison between providers	
		AWS Lambda	GCF	Predictability	Performance
(EISMANN et al., 2022)	X	✓	X	X	X
(GINZBURG; FREEDMAN, 2020)	X	✓	X	X	X
(JACKSON; CLYNCH, 2018)	X	✓	X	X	✓
(SCHIRMER et al., 2023)	X	X	✓	X	X
(LAMBION et al., 2022)	X	✓	X	X	X
(COPIK et al., 2021)	X	✓	✓	✓	✓
(LLOYD et al., 2018)	X	✓	X	X	X
(MARTINS; ARAUJO; CUNHA, 2020)	X	✓	✓	X	✓
(WEN et al., 2023)	X	✓	✓	X	X
(ELSAKHAWY; BAUER, 2021)	X	✓	✓	✓	X
(LEE; SATYAM; FOX, 2018)	✓	✓	✓	X	✓
(KIM; LEE, 2020)	✓	✓	✓	X	X
(WANG et al., 2018)	✓	✓	✓	X	X
(PARK; KIM; LEE, 2020)	✓	✓	✓	X	X
(CHOI; LEE, 2020)	✓	✓	X	X	X
(KELLY; GLAVIN; BARRETT, 2020)	✓	✓	✓	X	✓

Source: The author

were not yet available. This work also bridges this gap by conducting an up-to-date analysis of 2nd gen functions in GCF.

As described by (WEN et al., 2023), previous published papers lacked statistical analysis over performance variation and nearly 60% of them did not provide enough information about the number of repetitions used during performance experiments. This work also aims to provide statistical background to performance comparisons and factor significance claims.

The study in (KELLY; GLAVIN; BARRETT, 2020) had the highest number of similarities with the objectives of this research. It evaluated local file system I/O workloads in AWS Lambda and GCF while comparing performance. In addition, it was published in 2020 when 2nd gen functions were not available in GCF. Like other previous work, it did not provide statistical background for the performance claims or enough variation data in the form of coefficients of variance, standard deviation or other dispersion metrics. This work differentiates itself from that study by running the experiments in the context of 2nd gen functions in GCF and deepening the predictability analysis while making comparisons with statistical support.

The most recent works that investigated the performance of local file system I/O in the serverless context were published in 2020 (KELLY; GLAVIN; BARRETT, 2020; CHOI; LEE, 2020; PARK; KIM; LEE, 2020; KIM; LEE, 2020). The serverless platforms and implementations evolve over time suggesting experiments can yield different results when repeated in the future (PARK; KIM; LEE, 2020). Nevertheless, this study introduces innovation by not only assessing performance, but also using variability to compare predictability across local file system workloads in the serverless context.

4 METHODOLOGY

This chapter presents the methodology used to achieve the objectives listed in Section 1.1.The general objective relates to establishing a performance and predictability comparison between the local storage offerings from AWS Lambda and GCF in serverless functions. The first specific objective of this work relates to aggregating the factors that influence local file system I/O performance and predictability in the serverless context, and was presented in Section 3.5.

The second specific objective is the development of heuristics to narrow down a subset of factors from the aforementioned aggregation that are relevant for experimentation. To achieve this objective, this work developed two heuristics for factor selection presented in the following list:

- **Heuristic 1: Remove factors that cannot be controlled or configured.** In practice, the usefulness of the information gained by experimenting with these factors is limited since developers cannot benefit from an optimal configuration or fine tuning technique.
- **Heuristic 2: Remove factors that do not have a clear minimum or maximum level.** Experimenting with the minimum and maximum levels of a factor is useful in the context of a $n2^m$ experiment design. The selection of two extreme values enables capturing whether the factor had an effect on the response variable. Not having a clear choice of minimum and maximum levels can bias the results depending on the chosen levels. For example, the result of an analysis of variance has a high chance of indicating the factor is not relevant if both measured levels have a similar effect on the response variable. For this reason, a full factorial experiment design is a better fit for information gain on these factors.

Concurrent executions on the same host and hardware heterogeneity factors were removed from the experiment through Heuristic 1. A discussion on why these factors are not configurable by developers was presented in Section 3.5. Alternatively, the deployment region factor was removed by means of Heuristic 2 since it is not clear what is the choice of level to minimize or maximize the latency of the tested I/O operations. In addition, it is important to note that the choice of language runtime investigated in (ELSAKHAWY; BAUER, 2021; JACKSON; CLYNCH, 2018; MARTINS; ARAUJO; CUNHA, 2020) was also not considered as a factor for experimentation. This decision was made because the measurement of file system I/O latency was carried out using the dd benchmark tool (GNU, 2023), which allowed for a consistent and controlled assessment of the performance independently of the language runtime. Consequently, the performance overhead imposed by platforms to starting functions was also not considered since running dd enables the experiments to focus on specific operations performed after the execution environment is ready. It also means that cold and warm starts do not interfere with the I/O experiments. A similar strategy for assessing performance isolated from cold and warm start effects was also employed on previous work (BORTOLINI; OBELHEIRO, 2020).

Experiments in this work followed the $n2^m$ design for capturing the set of relevant factors and interactions for all platforms. The main advantage of this design is minimizing the number of experiments performed by using the minimum and maximum level values for each factor. Then, future work can further investigate the relevant factors and interactions in more detail through a full factorial experiment. In addition, this work aims to replicate each experiment configuration at least 50 times as suggested by (WEN et al., 2023). Finally, if all 8 performance factors listed in the aggregation presented in Section 3.5 were to be used in the experiment, it would require 12800 experiments in the context of a $n2^m$ design with 50 repetitions before considering any additional parameters required by the benchmark tool. Applying the heuristics to derive a subset of relevant factors has the benefit of further reducing the number of experiments required and the analysis scope.

The latency of I/O operations was used as the performance metric and response variable for the experiments. It is an LB metric and smaller values mean better performance. In this context, latency can be defined as the time it takes for an I/O operation to complete from the perspective of an application including all associated subtasks. It is a measure of time and has resolution of 1 ms. ANOVA was then used to process the latency measurements and evaluate the effects of the different factors to performance. The performance comparisons were performed for each factor level combination using confidence intervals from the experiment repetitions.

The CV was used as the predictability metric for latency. It is an LB metric and a dimensionless measure of the variability present on the performance data relative to a mean value as defined in Eq. 2.1. In this context, less variability means more performance predictability. The observation of the latency alongside the CV allows for characterizing the performance and predictability of local file system I/O operations in the context of serverless. Calculating a CV requires a collection of latency measurements. Consequently, each CV measurement was taken out of a collection of latency measurements over all combinations of factor levels in the $n2^m$ experiment design. For this reason, this work compared the predictability between both providers directly through the calculated CV values from latency observations. In addition, small CV differences (inferior to 10%) were considered negligible for developers intending to use

AWS Lambda or GCF for local file system workloads. This study focused on shedding light on the large CV differences observed on the experiments.

Even though AWS Lambda (AWS Lambda) allows for a higher level of customization through the use of Docker images, other microbenchmarks like Flexible I/O Tester (FIO) are not available from function code by default and cannot be installed. In addition, the FunctionBench serverless benchmark tool does not have a file I/O benchmark that supports sequential and random I/O for both AWS Lambda and GCF (KIM; LEE, 2019). For these reasons, the dd benchmark was used for collecting latency measurements from local file system I/O operations (GNU, 2023). Even though it does not support random I/O operations, dd is a convenient choice since it is installed on the underlying operating systems of both AWS Lambda and GCF platforms by default and can be called from function code without any customization that can introduce noise or other side effects to the performance results. The dd benchmark took the following parameters during the experiments in this work:

1. I/O size: Random operations typically yield better performance with smaller I/O sizes, while sequential operations on large files may benefit from using larger I/O sizes (GREGG, 2014). For this reason, it is expected that this factor impact I/O performance depending on the file size.
2. File size: Caching strategies employed to file systems can interact with the amount of memory available and the size of the files being written or read (GREGG, 2014). File sizes smaller than the main memory favors caching and enables the investigation of the file system software. In contrast, file sizes larger than the main memory minimizes the caching effects and drives the benchmark toward testing disk I/O.
3. Operation type: Read or write operations. The performance of file system operations can vary based on their type. Similarly to access patterns, different file system optimization techniques may apply depending on the type of I/O operation (GREGG, 2014).

All read operations were performed using direct I/O, while leaving the write operations subject to caching at the operating system level. Opting for direct I/O on read operations prevented any residuals from previous write operations on a same function runtime. Allowing for operating system caching effects on write operations aims to replicate the perceived performance of a real application writing a file. Additionally, */dev/urandom* was used as the input file for write operations. While reading from this file introduces latency, this overhead is expected to be similar across both providers, ensuring that it does not interfere with the performance comparisons of this work.

As described on Section 2.1.1, AWS Lambda allows for configuring any memory value between 128 MB and 3 GB to functions in new accounts and CPU allocation is derived from memory. Conversely, GCF has a tiered approach to memory settings such that each tier has a corresponding memory and CPU allocation with little flexibility within any given tier. From the perspective of AWS Lambda memory allocation limit of 3 GB to new accounts, the closest GCF memory tiers immediately below and above are, respectively, 2 GB and 4 GB (AWS, 2024; AWS, 2023c; GCP, 2023g). Consequently, the maximum matching memory allocation between both providers is 2 GB. This is a limitation imposed by AWS Lambda memory allocation limits. As discussed in 2.1.1, it is not clear when or how an AWS account gets allowed for allocating functions beyond 3 GB.

Regarding CPU, GCF has a limiting factor for compatibility of allocation settings between providers due to the restrictive tiered approach. Similarly, the amount of CPUs allocated to functions in AWS Lambda is determined by its memory settings. This results in a limitation that prevents independent control of CPU allocated to functions. Consequently, instead of using memory and CPU allocation as an experiment factor, we applied a tiered approach so that resource settings between AWS Lambda and GCF are compatible. The tiers were built by picking a memory allocation size starting from 128 MB, deriving the corresponding CPU allocation in AWS Lambda and adjusting the settings in any given GCF tier to minimize the CPU difference between both providers. This allowed for full compatibility of memory settings while maximizing compatibility of CPU. This approach resulted in the resource allocation tiers 1 to 5 presented in Table 5 along with corresponding percent difference of CPU allocation on each tier. Tiers 1 and 5 were used in the context of a $n2^m$ experiment design with minimum and maximum levels.

Table 5 – Resource allocation compatibility tiers

		vCPU			
Tier	Memory (MB)	AWS Lambda (Derived)	GCF	Compatible	% Difference
1	128	$\approx$ 0.072	0.083	0.080	-10.562
2	256	$\approx$ 0.145	0.167	0.145	-0.197
3	512	$\approx$ 0.289	0.333	0.289	0.148
4	1024	$\approx$ 0.579	0.583	0.579	-0.024
5	2048	$\approx$ 1.158	1	1	13.623

Source: The author

Finally, Table 6 presents the final subset of relevant factors to be considered in the $n2^m$ experiment design along with the variable benchmark parameters and general remarks. In addition, even though the long-term performance measurements suggested in (EISMANN et al., 2022) are not part of the scope for this work, a reproducibility package is provided to assist with this task. This package increases the longevity of the results by allowing the experiments to be repeated in the future.

Table 6 – Factors, levels and considerations for the $n2^m$ experiment

	Levels		
Factors	Min	Max	General considerations
Platform	AWS Lambda	GCF	In this context, AWS Lambda and GCF are not directly related to minimum or maximum levels. Instead, these are the only two alternatives for the cloud serverless platform factor.
Time of day	Business hours	Off-hours	The results presented in (SCHIRMER et al., 2023) indicate that the performance of warm calls to the tested function reached its maximum value before dawn while the minimum occurred during business hours (9 AM to 6 PM).
Day of week	Workdays	Weekends	The findings in (SCHIRMER et al., 2023) suggest performance differences between workdays and weekends. In this context, we collected data over arbitrary days during both periods.
Resource tier	1	5	We modeled resource tiers to maximize resource allocation compatibility between AWS Lambda and GCF. We presented these tiers in Table 5.
I/O size	512 B	128 KB	This choice of minimum and maximum I/O sizes allow for testing the maximum realistic IOPS and throughput in the context of sequential read and write operations (GREGG, 2014)

Table 6 – continued from previous page

	Levels		
Factors	Min	Max	General considerations
File size	10 KB	1 GB	This work's goal is to analyze the performance of I/O operations from the perspective of an application interacting with the local file system. The author believes this file size range is enough for benchmarking both the file system software and the underlying disk. It is also representative of what applications would consider small and large files.
Operation type	Read	Write	Since the performance of file system operations depend on the operation type, we experimented with read and write operations.

Source: The author

Even though Table 6 presented a comprehensive list of relevant factors for experimentation, the domain of I/O and serverless introduces compatibility aspects that limit the possible interactions between factors. These aspects are modeled as Compatibility Rules (CRs) as follows:

- **CR 1: I/O size must not be higher than file size.** Applications use a number of I/O operations to perform reads or writes. The sum of I/O sizes multiplied by the number of operations must equal the intended file size. For example, writing a 10 KB file requires 2 I/O operations with 5 KB size. Consequently, it is not possible to experiment with I/O sizes higher than file sizes.
- **CR 2: File size must be smaller than allocated memory.** As shown in Table 1 and discussed in 2.1, GCF file system is held in memory. Consequently, it is not possible to experiment with file sizes higher than the memory allocated to a GCF instance.

In the context of the list of factors presented on Table 6, CR 1 enforces that file sizes of 10 KB can only operate with I/O size equal to 512 B. In addition, due to CR

2, large files (1 GB) can only be manipulated on tier 5 functions, according to Table 5. The interaction between incompatible factors should be disregarded. This results in a reduced number of experiments relative to a $n2^m$ experiment with all factors on Table 6 and their interactions.

5 EXPERIMENTS

This chapter presents the experiments performed on the AWS Lambda and GCF platforms regarding performance and predictability of local file system I/O operations in serverless functions. The already mentioned Table 6 presents the 7 factors identified from previous work and the benchmark tool that was used to drive the $n2^m$ experiment design. Besides the aforementioned heuristics for factor selection, other opportunities to reduce the number of factors required for the $n2^m$ experiment design were explored in a preliminary experiment.

Even though previous results in (SCHIRMER et al., 2023) identified time of day and day of week as relevant factors for serverless performance, it reported performance variations of $\sim$15% between business and off-hours, and $\sim$4% between weekdays and weekends. These results were also not backed by statistical evidence. In addition, these factors increase the complexity of the experiments since they require measurements to be performed at specific times and days of the week. For this reason, and due to the magnitude of reported performance differences and the lack of statistical background for these results, a preliminary experiment was conducted to evaluate if time of day and day of week factors are relevant enough to further experimentation in the main $n2^m$ experiment design proposed for this work. Section 5.1 describes this preliminary experiment.

On the sequence, results from the preliminary experiment were taken into consideration for the main experiment with the remaining factors. Section 5.2 presents the main experiment.

5.1 PRELIMINARY EXPERIMENT: TIME OF DAY AND DAY OF WEEK FACTORS

The goal of this preliminary experiment is to evaluate the effect of the time of day and day of week factors for local file system I/O performance in both serverless platforms. For this reason, constant values were set to the factors and levels in Table 6, excepting for cloud serverless platform, time of day and day of week. Table 7 presents the resulting factor and level values for this preliminary experiment. This experiment makes use of the same benchmark tool and response variable as the main experiment described in Chapter 4.

Measurements for time of day and day of week factors are time sensitive. Consequently, measurements were taken every minute over 4 days while varying the factor levels equally. This means each factor level combination accounted for $1/2^3$ of the total measurements for 3 varying factors. This approach allowed for a wider coverage of

Table 7 – Factors and levels for the preliminary experiment

	Levels		
Factors	Min	Max	Constant
Cloud serverless platform	AWS Lambda	GCF	
Time of day	Business hours	Off-hours	
Day of week	Workdays	Weekends	
Memory and CPU allocation			2 GB
I/O size			1 MB
File size			1 GB
Operation type			Write

Source: The author

time over a weekend and the beginning of the following week. It also allowed for more data points than a $n2^m$ experiment design. This experimentation approach allowed for collecting 5700 measurements from 00:00 on Saturday 11/11/2023 to 23:59 on Tuesday 11/14/2023. Fig. 3 presents histograms for the collected data on both serverless platforms.

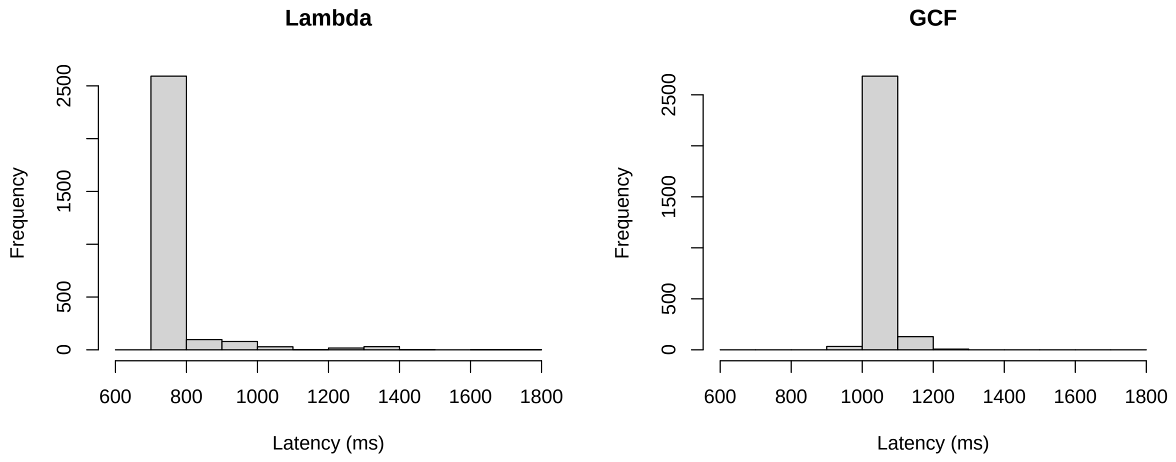

Figure 3 – Histograms for collected data on AWS Lambda and GCF in the preliminary experiment
Source: The author

Both distributions are right-skewed, asymmetric and unimodal. AWS Lambda has a lower mode in the interval (700,800] ms compared to GCF between (1000,1100] ms. The CVs for AWS Lambda and GCF are ~11.6% and ~2.8%, respectively. In this context, data for AWS Lambda presented more outliers distant from its mode than GCF. Even though this data brings some insights on what to expect from experiments in both platforms, this preliminary experiment's goal is to evaluate the time of day and day of week factors.

First, the confidence intervals for the mean latency were compared between AWS Lambda and GCF for weekdays and weekends. Then, a comparison was also established for business hours and off-hours. Fig. 4 presents the confidence intervals supporting these comparisons with a confidence level of 95%.

No pair of confidence intervals have an overlap that includes a mean value. For this reason, based solely on the confidence intervals, it is not possible to affirm that there are no statistically significant differences between latencies measured during the week and weekends on either platform. Similarly, the same can be said for latencies measured during business hours and off-hours in both platforms. These evidences so far confirm the findings in (SCHIRMER et al., 2023).

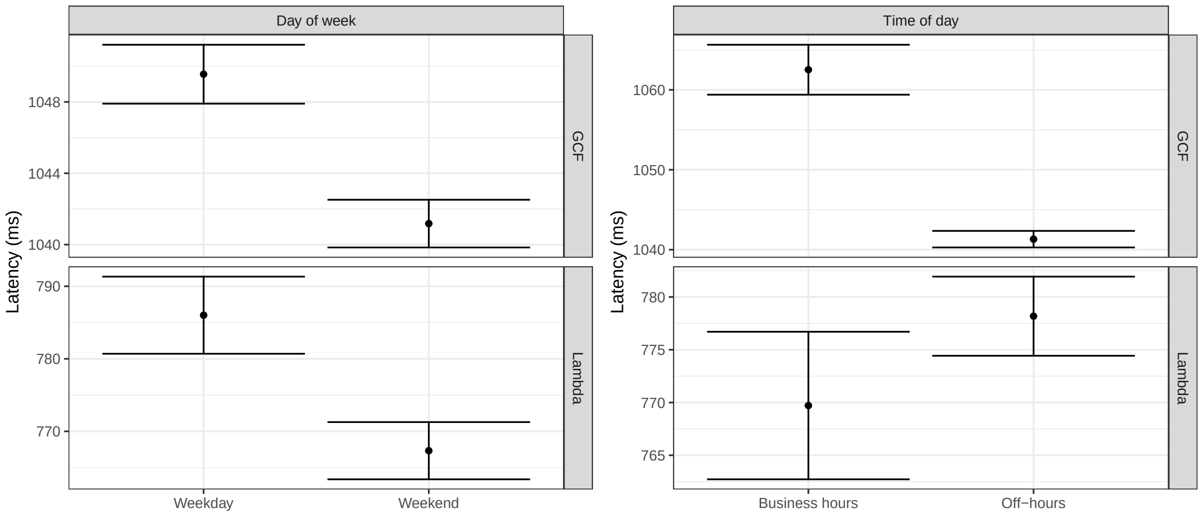

Figure 4 – Confidence intervals for the mean latencies in AWS Lambda and GCF for day of week and time of day factors in the preliminary experiment

Source: The author

ANOVA was also performed for latency data in AWS Lambda and GCF. As presented in the ANOVA table for AWS Lambda in Table 8, the F values for both time of day and day of week are higher than F_{crit}, meaning that both factors are statistically significant at a confidence level of 95%. In this context, conversely to day of week, the F value for the time of day factor is close to F_{crit} and would not be significant at the confidence level of 99% where F_{crit} is 6.64. Table 9 presents the ANOVA table for GCF. Similarly to AWS Lambda, the time of day factor is statistically significant at a confidence level of 95%. Alternatively, day of week is not a statistically significant factor at the same confidence level.

Finally, the goal is to evaluate the magnitude of the latency differences measured for each level in each factor. Since both distributions are asymmetric and unimodal, its more appropriate to use medians for this evaluation instead of the mean values. Table 10 presents the level medians for both factors measured in AWS Lambda and GCF. The results for this latency difference evaluation are also showed in Table 11 alongside a summary of the factor significance from the aforementioned ANOVA results and the

Table 8 – Resulting ANOVA in AWS Lambda for the preliminary experiment

Source of Variation	Degrees of Freedom	Sum of Squares (%)	F Value	$F_{crit}[0.95, 1, 2847]$
Time of day	1	0.14	3.92	3.84
Day of week		1.93	56.127	
Residuals	2847	97.93		
Total	2849	100		

Source: The author

Table 9 – Resulting ANOVA in GCF for the preliminary experiment

Source of Variation	Degrees of Freedom	Sum of Squares (%)	F Value	$F_{crit}[0.95, 1, 2847]$
Time of day	1	8.11	251.55	3.84
Day of week		0.01	0.09	
Residuals	2847	91.88		
Total	2849	100		

Source: The author

calculated percent differences between the level medians for time of day and day of week factors.

Table 10 – Level medians for time of day and day of week factors in the preliminary experiment

	AWS Lambda (ms)		GCF (ms)	
	Business hours	Off hours	Business hours	Off hours
Time of day	756.497	756.399	1055.700	1038.595
	Weekdays	Weekends	Weekdays	Weekends
Day of week	758.693	755.125	1044.550	1038.290

Source: The author

Table 11 – Factor significance and difference between level medians for each cloud provider in the preliminary experiment

Factor	ANOVA Statistically significant?		Difference between level medians (%)	
	AWS Lambda	GCF	AWS Lambda	GCF
Time of day	✓	✓	0.01	1.65
Day of week	✓	X	0.47	0.60

Source: The author

Even though the work in (SCHIRMER et al., 2023) did not encompass AWS Lambda, it found that time of day and day of week factors are significant for GCF. Nevertheless, this preliminary experiment found that these factors are also statistically significant for AWS Lambda and a local file system I/O workload. Conversely, the day of week factor is not statistically significant for GCF, to the contrary of what was previously reported. Finally, it is possible to conclude that, for data collected over 4 days, these factors have small contributions to the overall latency values as suggested by a maximum difference between level medians of 1.65% for time of day in GCF. In addition, the maximum total variation attributed to one of these factors was 8.11% for time of day in GCF. For these reasons, these factors were discarded from the main $n2^m$ experiment design of this work in favor of more repetitions with the remaining factors depending on time, availability and quantity limitations for experiments.

5.2 MAIN EXPERIMENT

The results from the main experiment presented in this section relate to the specific objectives described on Section 1.1.2. The goal is to establish a comparison of performance and predictability of local file system I/O operations in AWS Lambda and GCF. Each experiment configuration was repeated 150 times, going beyond the recommendation of 50 repetitions from (WEN et al., 2023). As seen on Section 5.1, the time of day and day of week factors were discarded from the main experiment due to insufficient contributions to overall latency values.

It is expected that I/O operations for small files are faster compared to large files. Similarly, performance is not comparable between writes and reads. These are different in nature and are subject to different caching techniques employed by the operating system. For these reasons, this section is divided into 4 subsections, exploring the combination of operation type and file size factor levels in each subsection. Small and large files are referred to as the minimum and maximum levels of the file size factor. Latency measurements of I/O operations on large and small files were reported in seconds and milliseconds, respectively.

As described in Chapter 4, the CV was used as the predictability metric for latency. CV results and comparisons were presented at the end of each subsection. CV is an LB metric and all comparisons were made directly through the calculated CV values from 150 repetitions on each experiment configuration. In addition, all results were produced and observed using a 95% confidence level.

Finally, the results presented in the following subsections made use of the Empirical Cumulative Distribution Function (ECDF) plot as a tool to assess the percentiles of the collected data and identify where most values occur. This plot depicts the latency, as the response variable for the performance experiments, on the X axis and its corresponding percentile on the Y axis. In an example, a data point on (1,0.75) means the 75% percentile for the presented data is 1 second (or millisecond, depending on the unit used). Consequently, 75% of the latency data is less than 1 second. It is important to note that the ECDF results were presented in pairs while some pairs have different scales for the X axis. This was necessary to properly demonstrate the tightness of results involving a particular factor.

5.2.1 Write operations on large files

Write operations on large files are compatible with all I/O size levels. Conversely, due to CR 2, only the maximum level of memory and CPU resource allocation is compatible (tier 5). The list of factors and levels used for write operations on large files is presented on Table 12.

Table 12 – Factors and levels for write operations on large files

	Levels		
Factors	Min	Max	Constant
Cloud serverless platform	AWS Lambda	GCF	
I/O size	512 B	128 KB	
Resource tier			Tier 5
File size			1 GB
Operation type			Write

Source: The author

An observation of the ECDF on Figure 5 suggests AWS Lambda shows up more performance both for the minimum and maximum I/O size levels. The latency on the 50th percentiles (medians) of AWS Lambda were 7.85 and 3.60 seconds for the minimum and maximum I/O sizes, respectively. GCF presented medians of 9.54 and 4.07 seconds under the same settings. Additionally, Appendix A presents histograms of write latency on large files.

The performance difference suggested by the ECDF observation is confirmed by the confidence intervals in Figure 6. AWS Lambda had mean latencies of 7.79 and 3.62 seconds for the minimum and maximum I/O sizes, respectively. Comparatively, GCF had mean latencies of 9.55 and 4.06 seconds for the same settings. These results also show how the overall latency for write operations on large files decreased as the I/O size increased.

Table 13 presents the ANOVA table for the experiment with write operations on large files. All factors and interactions were statistically significant (F Value > F_{crit}). The interaction between provider and I/O size is significant, suggesting the effect of I/O size levels on performance is different on each provider. Nevertheless, this does not challenge AWS Lambda from having higher performance compared to GCF in this setting.

Table 13 – Resulting ANOVA for write operations on large files

Source of Variation	Degrees of Freedom	Sum of Squares (%)	F Value	$F_{crit}[0.95, 1, 596]$
Platform		5.89	2619	
I/O size	1	90.18	40109	3.86
Platform:I/O size		2.59	1152	
Residuals	596	1.34		
Total	599	100		

Source: The author

On large files, as presented on Figure 7, AWS Lambda was more predictable than GCF when using I/O sizes of 512 B. Conversely, AWS Lambda was less predictable when using an I/O size of 128 KB.

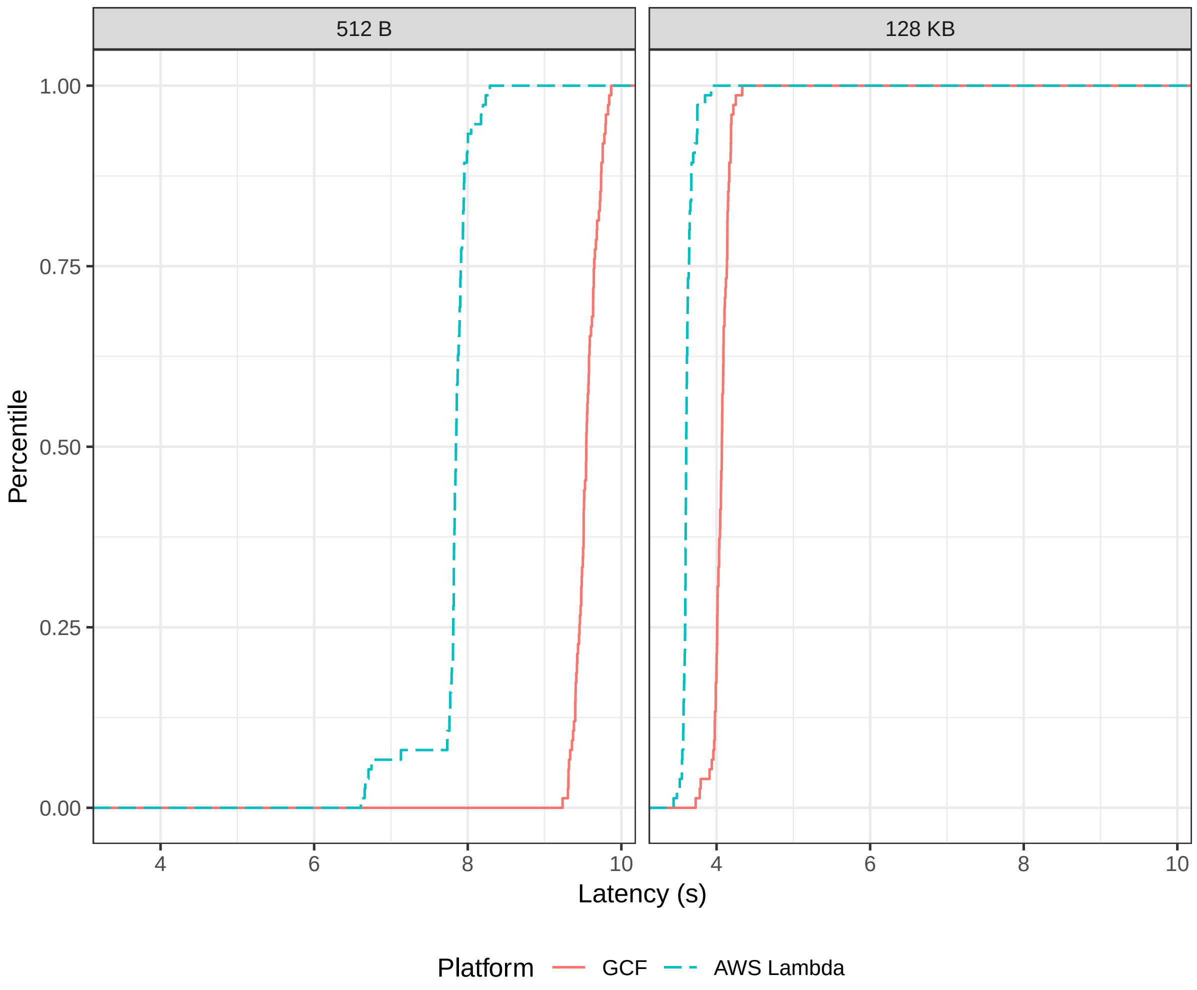

Figure 5 – ECDF of write latency for an 1 GB file in AWS Lambda and GCF using 512 B and 128 KB I/O sizes

Source: The author

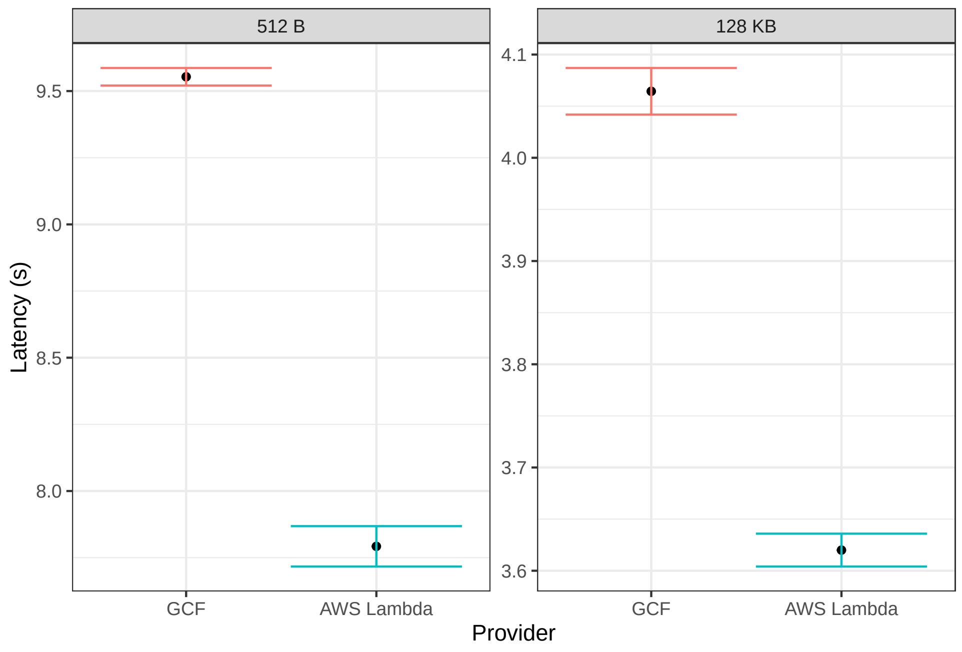

Figure 6 – Confidence intervals of write latency for an 1 GB file in AWS Lambda and GCF using 512 B and 128 B I/O sizes

Source: The author

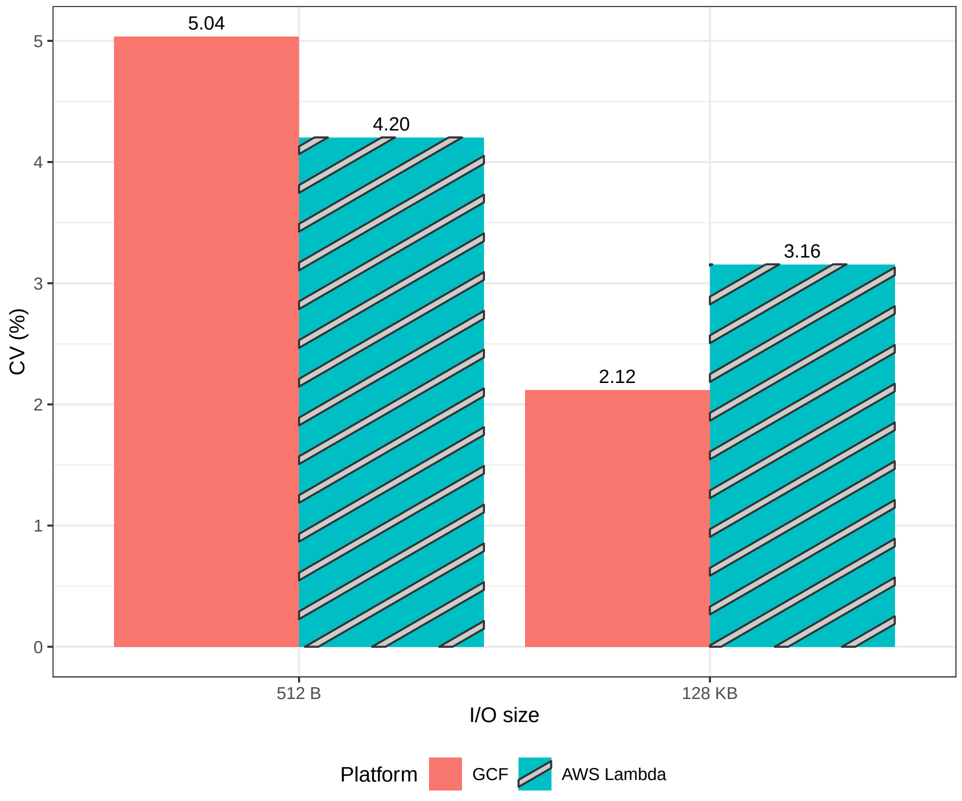

Figure 7 – CVs for write operations on large files

Source: The author

5.2.2 Write operations on small files

Write operations on small files are compatible with all memory and CPU resource allocation levels. Conversely, due to CR 1, only the minimum I/O size level is compatible. Table 14 presents the list of factors and levels used for write operations on small files.

Table 14 – Factors and levels for write operations on small files

	Levels		
Factors	Min	Max	Constant
Cloud serverless platform	AWS Lambda	GCF	
Resource tier	Tier 1	Tier 5	
I/O size			512 B
File size			1 GB
Operation type			Write

Source: The author

Even though GCF performance is higher for the minimum resource allocation level, as can be observed on the ECDF in Figure 8, AWS Lambda performance approaches at high resource settings. The 50th percentile (medians) on AWS Lambda was approximately 0.26 ms for both tiers 1 and 5. GCF had medians of 0.27 and 0.28 ms under the same settings. Additionally, Appendix B presents histograms of write latency on small files.

Figure 9 presents confidence intervals for the latency observed on both resource tiers. Mean latencies for AWS Lambda were 7.90 and 0.28 ms for tiers 1 and 5, respectively. GCF had mean latencies of 4.22 and 0.31 ms for the same settings. Since the interval overlap in tier 5 contains both means, it is not possible to statistically affirm performance is different between AWS Lambda and GCF at high resource allocation for write operations on small files. Even though there is an overlap on the interval in tier 1, it does not include mean values. For this reason, further investigation is needed to statistically determine if GCF is faster on low resource settings.

Table 15 presents the ANOVA table for this experiment. The resource tier was the only statistically significant factor in this setting (F Value > F_{crit}). Consequently, the effect of the chosen cloud platform is not statistically significant to latency on this confidence level. Even though results shown in Figure 9 are not conclusive for low resource allocation, the ANOVA results indicate performance differences between both platforms are mostly due to the resource tier and random error.

Figure 10 presents the CVs for write operations on small files. In this context, AWS Lambda was generally more predictable than GCF, with a large CV difference of

Table 15 – Resulting ANOVA for write operations on small files

Source of Variation	Degrees of Freedom	Sum of Squares (%)	F Value	$F_{crit}[0.95, 1, 596]$
Platform		0.57	3.62	
Resource tier	1	5.64	36.09	3.86
Platform:Resource tier		0.58	3.72	
Residuals	596	93.21		
Total	599	100		

Source: The author

333.46 percentage points on tier 1. The CV difference between providers was smaller on tier 5 when compared to tier 1 (4.67 percentage points).

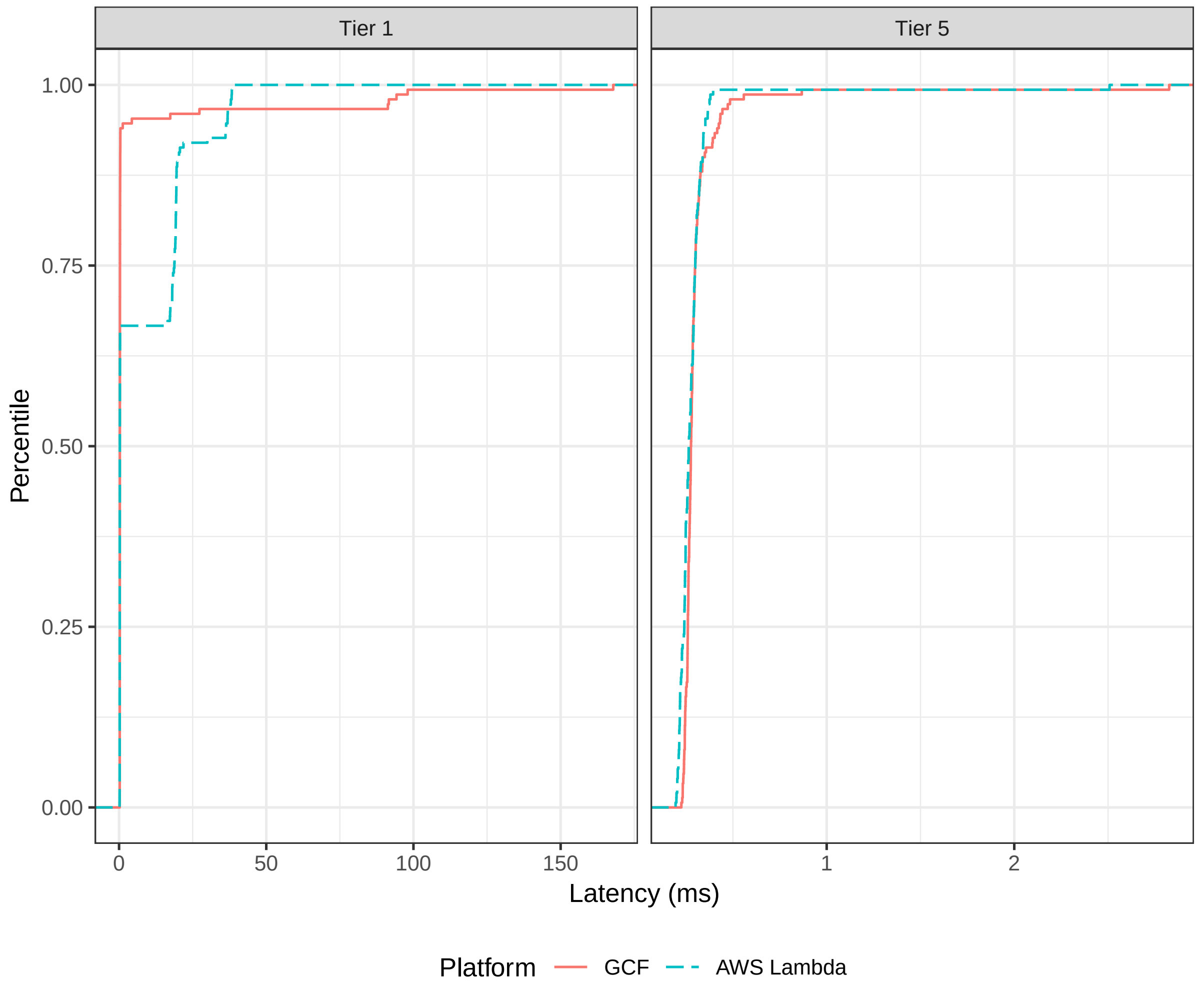

Figure 8 – ECDF of write latency for a 10 KB file with 512 B I/O size in AWS Lambda and GCF using minimum and maximum resource tiers

Source: The author

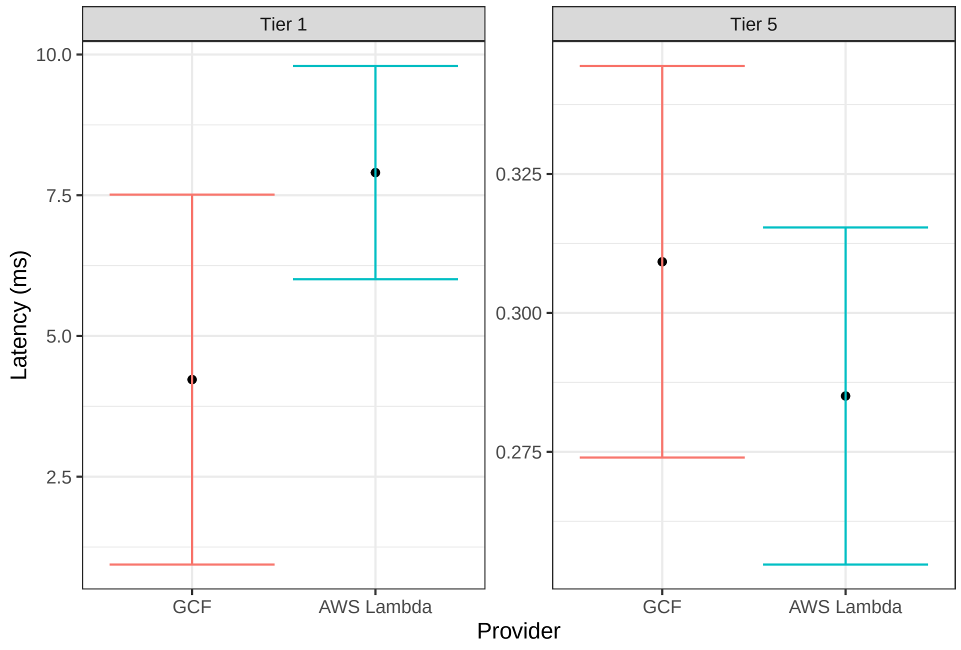

Figure 9 – Confidence intervals of write latency for a 10 KB file with 512 B I/O size in AWS Lambda and GCF using minimum and maximum resource tiers

Source: The author

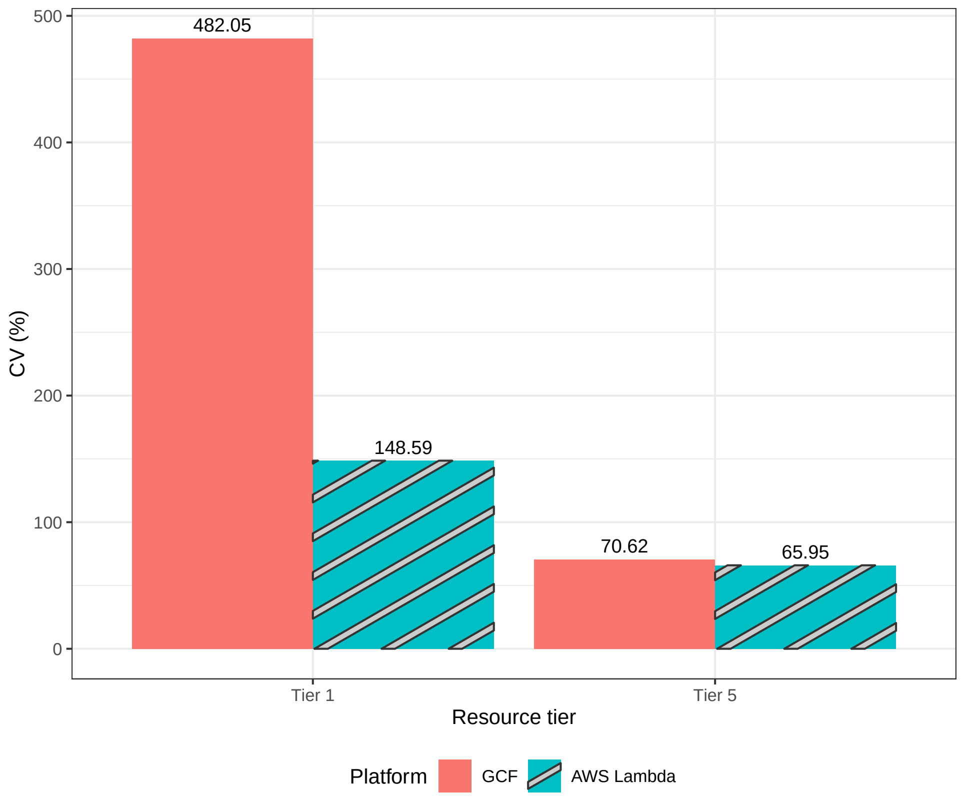

Figure 10 – CVs for write operations on small files

Source: The author

5.2.3 Read operations on large files

Similarly to Section 5.2.1, as a result of CR 2, read operations on large files are compatible with all I/O size levels but only compatible with the maximum level of memory and CPU resource allocation (tier 5). The list of factors and levels used for read operations on large files is presented on Table 16.

Table 16 – Factors and levels for read operations on large files

	Levels		
Factors	Min	Max	Constant
Cloud serverless platform	AWS Lambda	GCF	
I/O size	512 B	128 KB	
Resource tier			Tier 5
File size			1 GB
Operation type			Read

Source: The author

ECDF results in Figure 11 indicate that GCF has better performance on all cases even though the latency difference between providers reduced when increasing the I/O size. Median latencies for AWS Lambda were 265.31 and 29.80 seconds for the minimum and maximum I/O sizes, respectively. On GCF, the medians were 4.79 and 0.08 seconds under the same settings. Additionally, Appendix C presents histograms of read latency on large files.

This performance difference suggested by the ECDF is confirmed by the confidence intervals in Figure 12. The mean latencies for AWS Lambda were 265.25 and 23.39 seconds for the minimum and maximum I/O sizes, respectively. The mean latencies observed on GCF were 4.79 and 0.08 seconds under the same settings. In addition, similarly to the observations on write operations, increasing the I/O size level also increased the overall performance of read operations on large files.

Table 17 presents the ANOVA table for this experiment. All factors and interactions are significant (F Value > F_{crit}).

Figure 13 presents the CV results for read operations on large files. AWS Lambda is more predictable than GCF for I/O sizes of 512 B but less predictable on I/O sizes of 128 KB by an absolute difference of 40.14 percentage points.

Table 17 – Resulting ANOVA for read operations on large files

Source of Variation	Degrees of Freedom	Sum of Squares (%)	F Value	$F_{crit}[0.95, 1, 596]$
Platform	1	40.06	98403	3.86
I/O size		31.07	76326	
Platform:I/O size		28.62	70304	
Residuals	596	0.25		
Total	599	100		

Source: The author

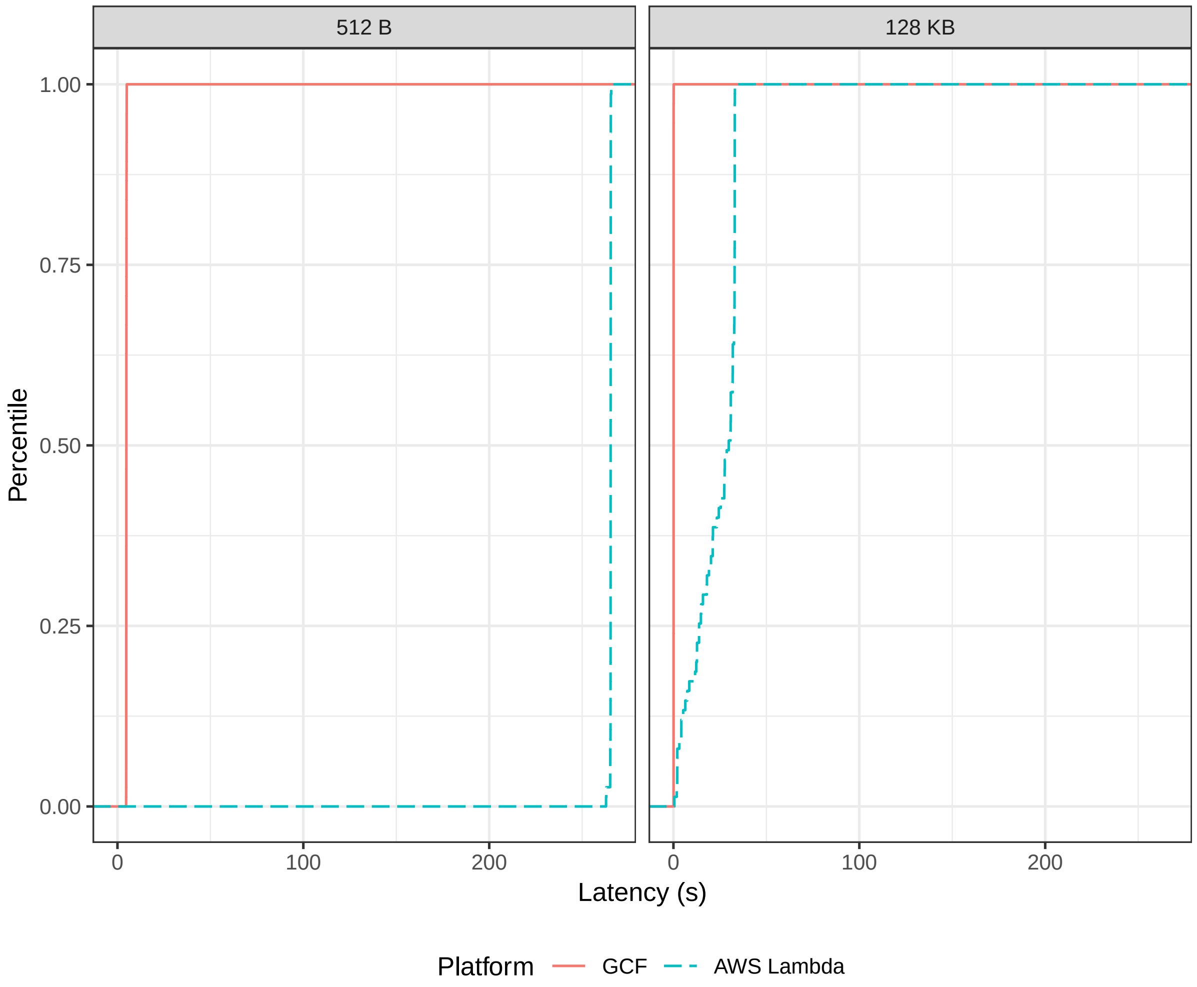

Figure 11 – ECDF of read latency for an 1 GB file in AWS Lambda and GCF using 512 B and 128 B I/O sizes

Source: The author

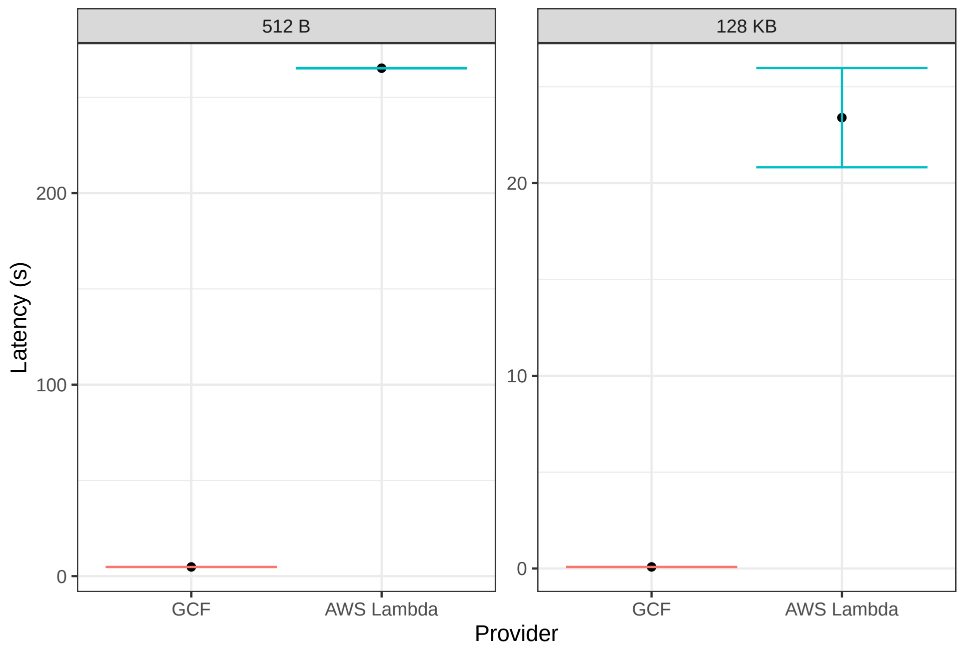

Figure 12 – Confidence intervals of read latency for an 1 GB file in AWS Lambda and GCF using 512 B and 128 KB I/O sizes

Source: The author

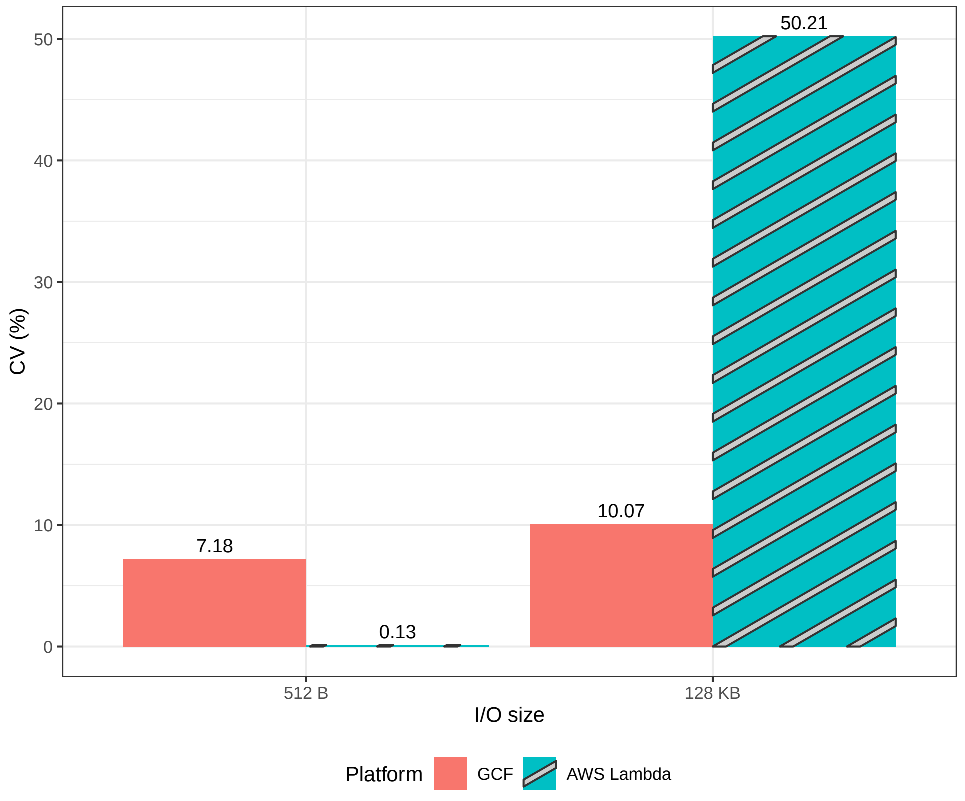

Figure 13 – CVs for read operations on large files

Source: The author

5.2.4 Read operations on small files

Similarly to Section 5.2.2, due to CR 1, read operations on small files are compatible with all memory and CPU resource allocation levels, but only with the minimum I/O size level. The list of factors and levels used for read operations on small files is presented on Table 18.

Table 18 – Factors and levels for read operations on small files

	Levels		
Factors	Min	Max	Constant
Cloud serverless platform	AWS Lambda	GCF	
Resource tier	Tier 1	Tier 5	
I/O size			512 B
File size			1 GB
Operation type			Read

Source: The author

Visual analysis of the ECDF on Figure 14 indicates that GCF performs better at the minimum resource allocation settings (tier 1), despite having more outliers compared to AWS Lambda. At maximum resource settings, AWS Lambda achieved similar but still inferior performance.

The median latencies for AWS Lambda were, respectively, 20.30 and 0.84 ms for tiers 1 and 5. GCF had median latencies of approximately 0.21 ms on both resource tiers. A closer look on the confidence intervals in Figure 15 indicate GCF had better performance on all resource allocation settings with mean latencies of 6.18 and 0.23 ms for resource tiers 1 and 5, respectively. Mean latencies observed on AWS Lambda were 26.36 and 0.89 ms under the same settings. Additionally, Appendix D presents histograms of write latency on small files.

ANOVA results in Table 19 shows all factors are statistically significant (F Value > F_{crit}), confirming the observations on the ECDF and confidence intervals. Figure 16 presents the CVs for read operations on small files. In this context, when compared to GCF, results showed AWS Lambda was more predictable on tier 1 but less predictable on tier 5 by an absolute difference of 10.43 percentage points.

Table 19 – Resulting ANOVA for read operations on small files

Source of Variation	Degrees of Freedom	Sum of Squares (%)	F Value	$F_{crit}[0.95, 1, 596]$
Platform		10.21	105.39	
Resource tier	1	23.16	259.30	3.86
Platform:Resource tier		8.94	92.34	
Residuals	596	57.69		
Total	599	100		

Source: The author

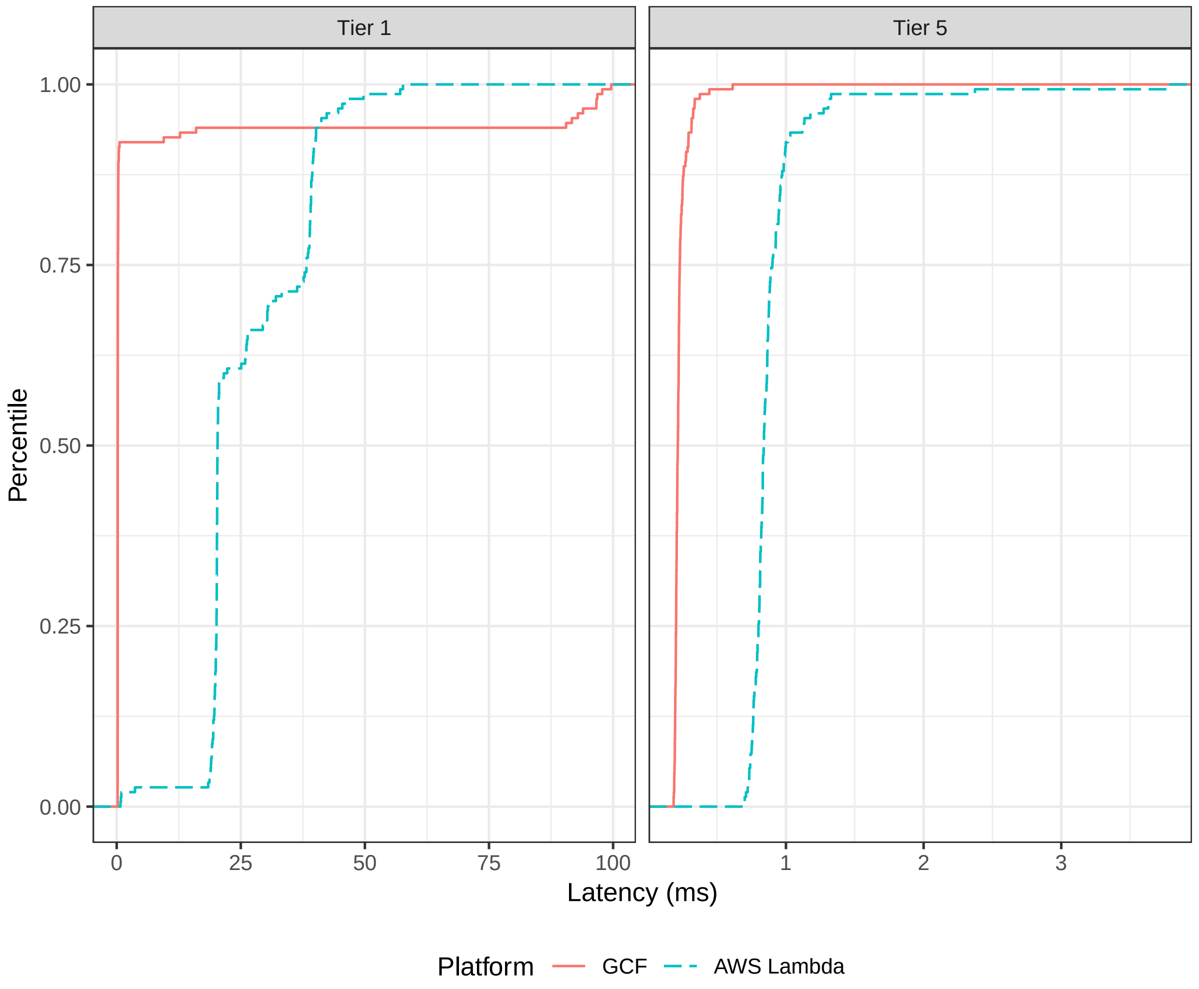

Figure 14 – ECDF of read latency for a 10 KB file with 512 B I/O size in AWS Lambda and GCF using minimum and maximum resource tiers

Source: The author

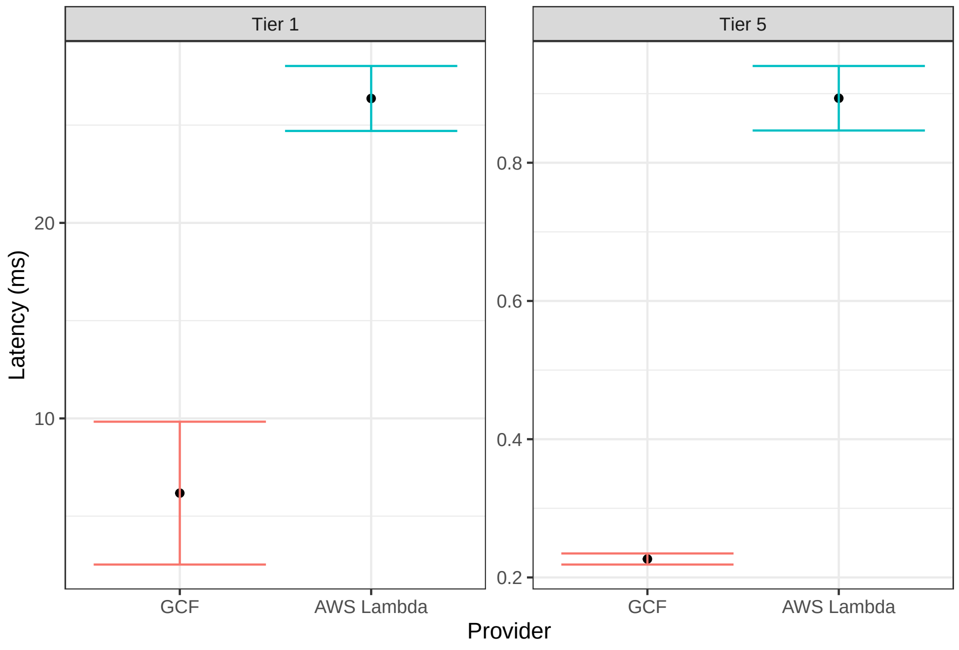

Figure 15 – Confidence intervals of read latency for a 10 KB file with 512 B I/O size in AWS Lambda and GCF using minimum and maximum resource tiers

Source: The author

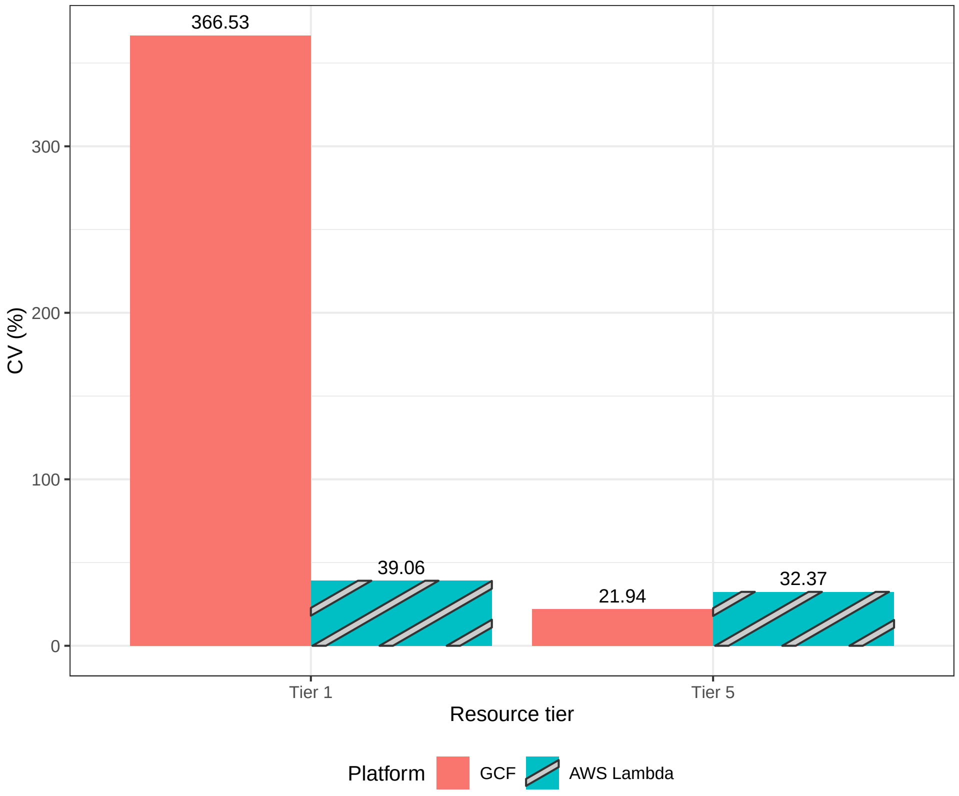

Figure 16 – CVs for read operations on small files

Source: The author

6 DISCUSSION

As presented on Section 2.1, GCF uses an in-memory file system to support local I/O operations on serverless environments. Conversely, even though AWS Lambda does not disclosure the inner workings of its local file system on functions, it is not implemented in-memory since its size can be configured independently from memory allocation. The in-memory file system of GCF might give an expectation of highly performant I/O operations. In reality, applications communicating with the local file system of serverless functions will do so through the operating systems. The caching techniques employed by these systems can have an effect on the performance observed by applications. Unlike read, experiments with write operations in this work did not use direct I/O and were subject to caching. In this context, results for write operations on large files showed better latency performance for AWS Lambda on all cases. For small files, the serverless provider's effect on performance was not statistically significant for a confidence level of 95%.

The aforementioned results showed that the performance expectations for the in-memory file system of GCF were not met when compared to AWS Lambda. The observed performance is due to file system caching techniques. As a consequence, applications communicating with the local file system of these serverless environments will perceive a latency level from AWS Lambda that is at least competitive, and sometimes better, when compared to GCF's in-memory file system.

On the other hand, experiments with read operations used direct I/O and are not subject to caching at the operating system level. In this scenario, GCF had significantly better latency performance on all cases when compared to AWS Lambda. The lack of a file system caching layer made the advantages of the in-memory file system implementation of GCF more apparent, reinforcing the caching effects also observed on write operations. This granted GCF better latency for direct I/O read operations.

The number of I/O operations performed to write or read a file is inversely proportional to the I/O size used. Consequently, more I/O operations are performed when writing a large file (1 GB) with 512 B I/O size when compared to 128 KB. In addition, these operations are more vulnerable to the interference of external factors and network overhead as the number of I/O operation increases. For this reason, the performance differences between AWS Lambda and GCF were larger when using the minimum I/O size. This can be observed on Figures 5 and 11 for write and read operations, respectively. This pattern was only observed for large files since CR 1 limits the I/O size options for small files.

Both write and read operations had performance improvements when increasing their resource allocation levels from tier 1 to tier 5. This confirms observations from previous work that the resource allocation was a significant factor to the performance of file I/O operations. These observations were presented on Section 3.5. Besides resource allocation, increasing the I/O size for write and read operations with large files also increased performance, even though the aggregation of factors from previous work presented on Section 3.5 did not observe the I/O size as a relevant factor. This is a result of the lower number of I/O operations required to process (read or write) a large file when using an I/O size of 128 KB. It also contributes to reducing the vulnerability to external factors interfering with I/O communication.

This work used the CV for measuring and driving the predictability comparisons between providers. In the context of predictability, even though AWS Lambda was generally more predictable for write operations on small files, there was a large CV difference of 333.46 percentage points observed with the minimum resource allocation level (tier 1). This difference was significantly smaller for tier 5 (4.67 percentage points). Besides this difference, both providers had smaller CVs on tier 5 when compared to tier 1. In general, CV results in tier 1 were significantly higher than tier 5. This indicates that increasing the resource allocation has an effect of increasing predictability under these settings and confirms the findings in (SCHIRMER et al., 2023). This observed behavior is attributed to higher resource contention present on tier 1 when compared to tier 5. In addition, when compared to tier 5, there is a high likelihood that providers will group a higher number of tier 1 functions on the same physical host. This increases the sources of interference, which can raise variability even without direct contention (KIM; LEE, 2020; LEE; SATYAM; FOX, 2018; WANG et al., 2018). CV differences between AWS Lambda and GCF were negligible for other experiments with write operations.

When comparing the latency predictability of small and large files, with the exception of AWS Lambda's CV for read operations on large files with 128 KB I/O size (50.21%), small files showed higher CVs. The highest and smallest CVs for small files are 482.05% and 21.94%, respectively. Comparatively, the highest and smallest CVs for large files are 50.21% and 0.13%, respectively. One possible reason for this behavior is that latency overhead introduced by I/O communication, as a source of variability, was more impactful for operations on small files when compared to large files.

Even though GCF had better overall latency performance for read operations when compared to AWS Lambda, AWS Lambda had significantly better predictability with a CV difference of 327.47% for small files with tier 1 resource allocation. Even so, the latency performance difference under these settings is on the order of 20 ms.

In this case, developers can make a decision of whether higher predictability is worth a loss of latency performance. Conversely, a noticeable CV difference of 40.14% in favor of GCF is observed for large files with 128 KB I/O size. In this case, GCF had better predictability besides better latency performance. The CV for AWS Lambda on read operations for large files with 512 B of I/O size is noticeably low (0.13%). Regardless of the low CV, AWS Lambda had a mean latency of 265.25 seconds, significantly higher than GCF with 4.79 seconds under the same settings. This highlights how high predictability may lose its value if performance levels are not reasonable.

Table 20 presents a summary of the comparative analysis of performance and predictability between AWS Lambda and GCF. Small (10 KB) and large (1 GB) files represent the minimum and maximum levels of the file size factor.

Table 20 – Summary of performance and predictability results

	Best performance		Best predictability	
	Write	Read (Direct I/O)	Write	Read (Direct I/O)
Small files	-	GCF	AWS Lambda[a]	AWS Lambda[a]
Large files	AWS Lambda	GCF	-	GCF[b]

[a] Only on small resource allocation (tier 1)
[b] Only on high I/O size (128 KB)

Source: The author

The use of dd as the benchmark tool was successful for this experiment. It was a reliable tool for collecting latency data from AWS Lambda and GCF serverless environments, and offered options for direct I/O. Even so, it has the limitation of only supporting sequential I/O. Still, sequential I/O is representative of the majority of serverless use cases since random I/O requires manipulating offsets and is consequently a less common access pattern. For instance, the default code examples for the file system module on NodeJS indicate sequential I/O instead of a random access pattern (NODE.JS, 2024).

In addition, there is a trade off of customizing the serverless environments with other benchmark tools that are not native to the platform, via a Docker image for example, versus leveraging limited native tooling like dd. Installing custom tooling on the environments while preventing systematic error is also challenging since platforms do not offer a common way of customizing the environments. Future work may leverage new ways of customizing the serverless environments so that other tools besides dd can be used for experimentation while preventing the introduction of systematic error.

Using the default file system module of a programming language would be an alternative to dd for the goal of measuring the performance and predictability of I/O operations. Even though using these modules is representative of what applications would do in reality, it also introduces the programming language as a factor. There might be significant performance differences between file system modules of different languages. For this reason, this work made option for dd as a language agnostic benchmarking tool.

Chapter 4 presented the concept of resource tiers as a model to achieve resource compatibility between AWS Lambda and GCF. By using this strategy, this work was able to leverage full memory compatibility between both providers with a CPU difference of approximately 13.62% in the worst case. Therefore, this strategy was successful for modeling a common resource allocation structure for both providers with negligible CPU allocation differences. Also, the need for a compatibility strategy highlights the heterogeneity of configuration options between providers.

Extending the performance and predictability comparisons of this work to other serverless providers poses challenges related to a common resource compatibility model. Microsoft Azure Functions (MAF), for example, enforces a dynamic resource allocation strategy for functions depending on demand (AZURE, 2019). On MAFs, developers have no control of memory and CPU allocation. Extending the performance and predictability comparisons to MAFs would be significantly more complex in terms of achieving resource compatibility between all three providers.

This work employed a $n2^m$ experiment design to maximize the gain of information with less experiments when compared to a full factorial design even though the CRs limited possible interactions between factors. These restrictions reduced the number of possible level combinations, and consequently the number of experiments, beyond the level limitations introduced by the $n2^m$ experiment design. Even so, this work successfully used the $n2^m$ experiment design within the CR limitations as seen in the factors and levels used in Sections 5.2.1 to 5.2.4.

In the context of serverless platforms, the usage of AWS Lambda with a new account introduced a memory ceiling of 3 GB as described in Section 2.1.1. Consequently, this work was not able to comparatively explore high memory allocation scenarios such as 32 GB GCF functions. Even within the aforementioned limitations, this work successfully explored the performance and predictability comparisons of the compatible factors using their minimum and maximum levels in a $n2^m$ experiment design.

The experiments in this work employed direct I/O for read operations only. When writing files, applications will perceive performance with the effect of caching techniques applied at the operating system levels. When reading, there is a possibility that a portion of the files exists in memory, enabling applications to leverage the effects of caching. When reading after writing, there is a high probability of cache hit since the file was just written, but there is no guarantee of the amount of data living in cache at the time of reading. It depends on the state of the system and memory available, among other factors. Consequently, it would be difficult to minimize systematic error when measuring read operations without direct I/O. At the same time, GCF's file system is implemented in-memory, and comparing it to AWS Lambda with direct I/O gives GCF an advantage. This advantage has been proven true in this work since GCF had better performance on all read experiments.

Conversely, when looking at write operations, AWS Lambda had equal and superior performance results for small and large files, respectively. These results give an indication that AWS Lambda applications communicating with the local file system through the operating system may leverage performance levels that are competitive with in-memory file systems. In addition, AWS Lambda has the advantage of allowing for a configurable size for the local file system with lower cost when compared to memory allocation. Even though these write results cannot be directly transposed to read operations, a new experiment that explores sequential read operations on the same file might shed more light on this difference and address the use case of using files in the local file system as a cache for API calls. Even so, it is expected for GCF to have similar read performance to what was found in this work since the absence of a particular storage device, external storage or disk, results in communication with the memory regardless of whether the I/O operation is direct or not.

Results indicated performance is higher and more predictable with a high allocation of CPU and memory resources. Similarly, it was found that the performance of I/O operations depends on the choice of I/O size. In this context, even though serverless pricing depends on the amount of allocated resources, as seen on Section 2.1, the I/O size is a configurable parameter that does not increase costs. Developers should conduct specific analysis to determine the optimal configuration for any given workload considering aspects such as cost and performance levels.

7 CONCLUSION

Serverless computing is an emerging paradigm with increasing popularity due to its ease of deployment and pay-per-use billing model. However, the lack of control that results from the computing infrastructure abstraction makes it difficult for developers to reliably meet or fine tune performance requirements specially for I/O bound applications that interact with a local file system. In the context of migrating legacy software to serverless, performance predictability is key when evaluating if the migration is financially worthwhile.

A preliminary experiment was performed to evaluate if the time of day and day of week factors are statistically significant to local file system I/O performance in AWS Lambda and GCF as reported by (SCHIRMER et al., 2023) in GCF. It was found that, besides the time of day factor for GCF, all the other factors are statistically significant for both serverless platforms. Still, this preliminary experiment concludes that these factors have small contributions to the overall latency magnitude of file I/O operations as suggested by a maximum difference between level medians of 1.65% for the time of day factor in GCF. In addition, the maximum total variation attributed to one of these factors was 8.11% for time of day in GCF. For these reasons, these factors were discarded from this work's main $n2^m$ experiment design in favor of more repetitions with the remaining factors.

On the main experiment, even though GCF had better overall performance for read operations with direct I/O, AWS Lambda had better performance for write operations on large files (1 GB). No statistically significant differences were found for write operation performance between GCF and AWS Lambda on small files (10 KB). These results challenge the expectation that GCF would have better performance across read and write operations since its local file system is held in-memory.

Small CV differences (inferior to 10%) were considered negligible for developers intending to use AWS Lambda or GCF for their local file system workloads. From this perspective, AWS Lambda had better predictability than GCF for reads and write operations to small files (10 KB) when operating with low allocation of resources (tier 1). This is not the case for large files (10 GB). Alternatively, performance on read operations for GCF was significantly better than AWS Lambda on all cases due to the direct I/O aspect. In this context, superior predictability may not be valuable to developers if the performance is not competitive, and even though AWS Lambda had better predictability under specific read scenarios, it does not hold value since GCF's performance was significantly higher. Nevertheless, knowing that AWS Lambda is more predictable for write operations under low resource allocation for small files with competitive performance to GCF is valuable to developers. In addition, AWS Lambda has better performance on write operations on large files with similar predictability to GCF.

One of the contributions of this work is to shed some light into the black-box nature of serverless platforms and help developers making informed decisions on using these platforms. The discussion and results found in this work allow for the following recommendations to developers considering AWS Lambda and GCF platforms for local file system I/O workloads:

1. Prefer GCF for read heavy workloads. It has an in-memory file system, and is able to consistently deliver performance levels equivalent to I/O cache hits.
2. Prefer AWS Lambda for write heavy workloads. It delivers equal or better performance to GCF and allows for configuring the local file system size independently from memory at a lower cost. It is also significantly more predictable than GCF at low resource allocation levels.
3. Performance is more predictable with more memory and CPU resources. Our results showed that the CV values from operations with small files decreased when moving the resource allocation tier from 1 to 5.
4. Even though I/O operations for small files (10 KB) are fast, they are less predictable. Our results showed mostly higher CV’s for operations with small files when compared to large files.
5. I/O performance increases with more memory and CPU resources allocated to functions. Specific analysis should be conducted to determine the optimal resource allocation for any given workload from the perspective of cost. Results in this work showed that small files written and read with high resources (tier 5) had better performance when compared to low resources (tier 1) on both platforms.
6. I/O performance depends on I/O size. Differently from the resource allocation, increasing or decreasing the I/O size does not directly impact cost. Specific analysis should be conducted to determine the optimal I/O size for any given file I/O workload. Results in this work showed that large files (1 GB) written and read with 128 KB of I/O size had better performance when compared to 512 B on both platforms.

Significant performance differences between AWS Lambda and GCF were found for read operations using direct I/O. Developers have no control of the memory state or execution environment of serverless functions. For this reason, direct I/O was needed in this case to isolate the reads from any previous write operations on the same function and consequently minimize bias and systematic error. Nevertheless, future work should further investigate read operations and the impact of consecutive reads on a same file. This would better address the static assets caching use case suggested by Amazon Web Services on (AWS, 2023e). Even though GCF had better performance on reads with direct I/O, the results will possibly be different for consecutive reads on the same file when allowing for cache on the operating system level.

This work used the $n2^m$ experiment design as a strategy to maximize the information gained from experimentation while reducing the number of factor level combinations required. Future work should consider expanding the experiments to a full factorial design, and analyze the interaction present on a broader combination of factors. Although it requires a high level of experimentation and replications, this analysis contributes to discovering if any intermediate factor interaction is relevant to determining performance and predictability. It also allows for further investigating if the trends seen in this work are also valid when looking at more intermediate levels. For example, even though this work found that predictability increases when increasing resources from tier 1 to tier 5, more data and analysis is required to check if this holds true also for tier 3.

In addition, future work can use experimentation to further explore and compare the impact of deployment region choice in performance and predictability across different serverless providers. Finally, future work can also extend these investigations and other aforementioned investigations to more serverless platforms beyond AWS Lambda and GCF.

The general and specific objectives presented on Section 1.1 aim to answer the Research Questions (RQ) established in Chapter 1. This work's results allowed for Research Answers (RA) to each question, as follows:

- **RQ1**: What is the aggregation of factors that affect performance and predictability in the context of local file system I/O operations from serverless functions?

  RA1: Besides the aggregation of factors that influence serverless local file system I/O performance and predictability presented in Section 3.5, results in this work also showed I/O size is a configurable factor that influences performance.

- **RQ2**: Between AWS Lambda and GCF, which serverless provider yields the best performance for local file system I/O operations?

  RA2: The performance comparisons were discussed on Chapter 6 based on the results presented on Chapter 5.2. These comparisons were summarized on Table 20.

- **RQ3**: Between AWS Lambda and GCF, which serverless provider has the most predictable performance for local file system I/O operations? In this context, more predictability means less variability and dispersion of performance.

  RA3: Similarly to RA2, the predictability comparisons were also discussed on Chapter 6 and summarized on Table 20.

Finally, the findings on this work led to recommendations targeting developers using or intending to use either AWS Lambda or GCF. Therefore, this work successfully achieved the intended specific and general objectives alongside the corresponding answers to the research questions. In addition, a reproducibility package[1] was published including this work's data for both the main and preliminary experiments alongside all code related to orchestrating and analyzing these experiments.

[1] Available on: https://github.com/gpr-indevelopment/dissert-serverless-local-io

BIBLIOGRAPHY

ALLEN, S. et al. **CNCF Serverless whitepaper v1.0**. 2023. Disponível em: <https://github.com/cncf/wg-serverless/blob/master/whitepapers/serverless-overview/cncf_serverless_whitepaper_v1.0.pdf>.

AWS. **Amazon ElastiCache for Redis**. 2023. Disponível em: <https://aws.amazon.com/elasticache/redis/>.

AWS. **AWS Lambda enables functions that can run up to 15 minutes**. 2023. Disponível em: <https://aws.amazon.com/about-aws/whats-new/2018/10/aws-lambda-supports-functions-that-can-run-up-to-15-minutes/>.

AWS. **AWS Lambda now supports up to 10 GB of memory and 6 vCPU cores for Lambda Functions**. 2023. Disponível em: <https://aws.amazon.com/about-aws/whats-new/2020/12/aws-lambda-supports-10gb-memory-6-vcpu-cores-lambda-functions/>.

AWS. **AWS Lambda Pricing**. 2023. Disponível em: <https://aws.amazon.com/lambda/pricing/>.

AWS. **Best practices for working with AWS Lambda functions**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html>.

AWS. **Choosing between AWS Lambda data storage options in web apps**. 2023. Disponível em: <https://aws.amazon.com/blogs/compute/choosing-between-aws-lambda-data-storage-options-in-web-apps/>.

AWS. **Configuring function timeout (console)**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html#configuration-timeout-console>.

AWS. **Lambda extensions**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-extensions.html>.

AWS. **Lambda function scaling**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html>.

AWS. **Lambda quotas**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>.

AWS. **Lambda runtimes**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html>.

AWS. **Serverless on AWS**. 2023. Disponível em: <https://aws.amazon.com/serverless/>.

AWS. **Working with Lambda container images**. 2023. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/images-create.html>.

AWS. **Lambda: Concurrency and memory quotas**. 2024. Disponível em: <https://docs.aws.amazon.com/lambda/latest/dg/troubleshooting-deployment.html#troubleshooting-deployment-quotas>.

AZURE. **Making Azure Functions more “serverless”**. 2019. Disponível em: <https://blogs.msdn.microsoft.com/appserviceteam/2016/11/15/making-azure-functions-more-serverless/>.

BALDINI, I. et al. Serverless computing: Current trends and open problems. **Research advances in cloud computing**, Springer, p. 1–20, 2017.

BORTOLINI, D.; OBELHEIRO, R. R. Investigating performance and cost in function-as-a-service platforms. In: BAROLLI, L.; HELLINCKX, P.; NATWICHAI, J. (Ed.). **Advances on P2P, Parallel, Grid, Cloud and Internet Computing**. Cham: Springer International Publishing, 2020. p. 174–185. ISBN 978-3-030-33509-0.

CASTRO, P. et al. The server is dead, long live the server: Rise of serverless computing, overview of current state and future trends in research and industry. **arXiv preprint arXiv:1906.02888**, 2019.

CHOI, J.; LEE, K. Evaluation of network file system as a shared data storage in serverless computing. In: **Proceedings of the 2020 Sixth International Workshop on Serverless Computing**. [S.l.: s.n.], 2020. p. 25–30.

CNCF. **CNCF Cloud Native Interactive Landscape - Serverless**. 2023. Disponível em: <https://landscape.cncf.io/serverless>.

COPIK, M. et al. Sebs: A serverless benchmark suite for function-as-a-service computing. In: **Proceedings of the 22nd International Middleware Conference**. [S.l.: s.n.], 2021. p. 64–78.

EISMANN, S. et al. A case study on the stability of performance tests for serverless applications. **Journal of Systems and Software**, Elsevier, v. 189, p. 111294, 2022.

ELSAKHAWY, M.; BAUER, M. Performance analysis of serverless execution environments. In: IEEE. **2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)**. [S.l.], 2021. p. 1–6.

GCP. **Cloud Functions 2nd gen is GA, delivering more events, compute and control**. 2023. Disponível em: <https://cloud.google.com/blog/products/serverless/cloud-functions-2nd-generation-now-generally-available>.

GCP. **Cloud Functions pricing**. 2023. Disponível em: <https://cloud.google.com/functions/pricing>. Acesso em: 15.5.2023.

GCP. **Cloud Functions vs. Cloud Run: when to use one over the other**. 2023. Disponível em: <https://cloud.google.com/blog/products/serverless/cloud-run-vs-cloud-functions-for-serverless>.

GCP. **Cloud Storage**. 2023. Disponível em: <https://cloud.google.com/storage>.

GCP. **Function isolation**. 2023. Disponível em: <https://cloud.google.com/functions/docs/concepts/execution-environment#function-isolation>.

GCP. **Function timeout**. 2023. Disponível em: <https://cloud.google.com/functions/docs/configuring/timeout>.

GCP. **Memory limits**. 2023. Disponível em: <https://cloud.google.com/functions/docs/configuring/memory>.

GCP. **Scalability**. 2023. Disponível em: <https://cloud.google.com/functions/quotas#scalability>.

GCP. **Serverless computing**. 2023. Disponível em: <https://cloud.google.com/serverless>.

GCP. **Write Cloud Functions**. 2023. Disponível em: <https://cloud.google.com/functions/docs/writing>.

GINZBURG, S.; FREEDMAN, M. J. Serverless isn't server-less: Measuring and exploiting resource variability on cloud faas platforms. In: **Proceedings of the 2020 Sixth International Workshop on Serverless Computing**. [S.l.: s.n.], 2020. p. 43–48.

GNU. **dd: Convert and copy a file**. 2023. Disponível em: <https://www.gnu.org/software/coreutils/manual/html_node/dd-invocation.html>.

GOLI, A. et al. Migrating from monolithic to serverless: A fintech case study. In: **Companion of the ACM/SPEC International Conference on Performance Engineering**. [S.l.: s.n.], 2020. p. 20–25.

GREGG, B. **Systems performance: enterprise and the cloud**. 2. ed. [S.l.]: Pearson Education, 2014.

HELLERSTEIN, J. M. et al. Serverless computing: One step forward, two steps back. **arXiv preprint arXiv:1812.03651**, 2018.

HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture: a quantitative approach**. 6. ed. [S.l.]: Elsevier, 2017.

JACKSON, D.; CLYNCH, G. An investigation of the impact of language runtime on the performance and cost of serverless functions. In: IEEE. **2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)**. [S.l.], 2018. p. 154–160.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: Wiley New York, 1991. v. 1.

KATZER, J. **Learning Serverless**. [S.l.]: "O'Reilly Media, Inc.", 2020.

KELLY, D.; GLAVIN, F.; BARRETT, E. Serverless computing: Behind the scenes of major platforms. In: IEEE. **2020 IEEE 13th International Conference on Cloud Computing (CLOUD)**. [S.l.], 2020. p. 304–312.

KIM, J.; LEE, K. Practical cloud workloads for serverless faas. In: **Proceedings of the ACM Symposium on Cloud Computing**. [S.l.: s.n.], 2019. p. 477–477.

KIM, J.; LEE, K. I/o resource isolation of public cloud serverless function runtimes for data-intensive applications. **Cluster Computing**, Springer, v. 23, p. 2249–2259, 2020.

KLIMOVIC, A. et al. Understanding ephemeral storage for serverless analytics. In: **2018 USENIX Annual Technical Conference (USENIX ATC 18)**. [S.l.: s.n.], 2018. p. 789–794.

KLIMOVIC, A. et al. Pocket: Elastic ephemeral storage for serverless analytics. In: **13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)**. [S.l.: s.n.], 2018. p. 427–444.

LAMBION, D. et al. Characterizing x86 and arm serverless performance variation: A natural language processing case study. In: **Companion of the 2022 ACM/SPEC International Conference on Performance Engineering**. [S.l.: s.n.], 2022. p. 69–75.

LEE, H.; SATYAM, K.; FOX, G. Evaluation of production serverless computing environments. In: IEEE. **2018 IEEE 11th International Conference on Cloud Computing (CLOUD)**. [S.l.], 2018. p. 442–450.

LILJA, D. J. **Measuring computer performance: a practitioner's guide**. [S.l.]: Cambridge university press, 2005.

LIU, F.; NIU, Y. Demystifying the cost of serverless computing: Towards a win-win deal. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, 2023.

LLOYD, W. et al. Serverless computing: An investigation of factors influencing microservice performance. In: IEEE. **2018 IEEE international conference on cloud engineering (IC2E)**. [S.l.], 2018. p. 159–169.

MAISSEN, P. et al. Faasdom: A benchmark suite for serverless computing. In: **Proceedings of the 14th ACM international conference on distributed and event-based systems**. [S.l.: s.n.], 2020. p. 73–84.

MARTINS, H.; ARAUJO, F.; CUNHA, P. R. da. Benchmarking serverless computing platforms. **Journal of Grid Computing**, Springer, v. 18, p. 691–709, 2020.

MONTGOMERY, D. C. **Design and analysis of experiments**. 9th. ed. [S.l.]: John wiley & sons, 2017.

NODE.JS. **Node.js v22.2.0 fs documentation**. 2024. Disponível em: <https://nodejs.org/api/fs.html>.

PARK, J.; KIM, H.; LEE, K. Evaluating concurrent executions of multiple function-as-a-service runtimes with microvm. In: IEEE. **2020 IEEE 13th International Conference on Cloud Computing (CLOUD)**. [S.l.], 2020. p. 532–536.

PEKKALA, A. Migrating a web application to serverless architecture. 2019.

POSIT. **RSTUDIO IDE**. 2023. Disponível em: <https://posit.co/products/open-source/rstudio/>.

RESEARCH; MARKETS. **Serverless Architecture - Global Strategic Business Report**. 2023. Disponível em: <https://www.researchandmarkets.com/reports/4806010/serverless-architecture-global-strategic>.

RISTOV, S. et al. Daf: Dependency-aware faasifier for node. js monolithic applications. **IEEE Software**, IEEE, v. 38, n. 1, p. 48–53, 2020.

ROY, R. B.; PATEL, T.; TIWARI, D. Characterizing and mitigating the i/o scalability challenges for serverless applications. In: IEEE. **2021 IEEE International Symposium on Workload Characterization (IISWC)**. [S.l.], 2021. p. 74–86.

SCHIRMER, T. et al. The night shift: Understanding performance variability of cloud serverless platforms. In: **Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies**. [S.l.: s.n.], 2023. p. 27–33.

SILBERSCHATZ, A.; PETERSON, J. L.; GALVIN, P. B. **Operating system concepts**. [S.l.]: John Wiley & Sons, Inc., 2012.

SINHA, P.; KAFFES, K.; YADWADKAR, N. J. Shabari: Delayed decision-making for faster and efficient serverless function. **arXiv preprint arXiv:2401.08859**, 2024.

SOMU, N. et al. Panopticon: A comprehensive benchmarking tool for serverless applications. In: IEEE. **2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)**. [S.l.], 2020. p. 144–151.

SPILLNER, J.; DORODKO, S. Java code analysis and transformation into aws lambda functions. **arXiv preprint arXiv:1702.05510**, 2017.

SREERAM, P. K. **Azure serverless computing cookbook**. [S.l.]: Packt Publishing Ltd, 2017.

TANENBAUM, A.; BOS, H. **Modern Operating Systems**. 4. ed. [S.l.]: Pearson, 2014.

WANG, L. et al. Peeking behind the curtains of serverless platforms. In: **2018 USENIX Annual Technical Conference (USENIX ATC 18)**. [S.l.: s.n.], 2018. p. 133–146.

WEN, J. et al. Rise of the planet of serverless computing: A systematic review. **ACM Transactions on Software Engineering and Methodology**, ACM New York, NY, 2023.

WEN, J. et al. An empirical study on challenges of application development in serverless computing. In: **Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering**. [S.l.: s.n.], 2021. p. 416–428.

WEN, J. et al. Revisiting the performance of serverless computing: An analysis of variance. **arXiv preprint arXiv:2305.04309**, 2023.

APPENDIX A – HISTOGRAMS OF WRITE LATENCY ON LARGE FILES (1 GB)

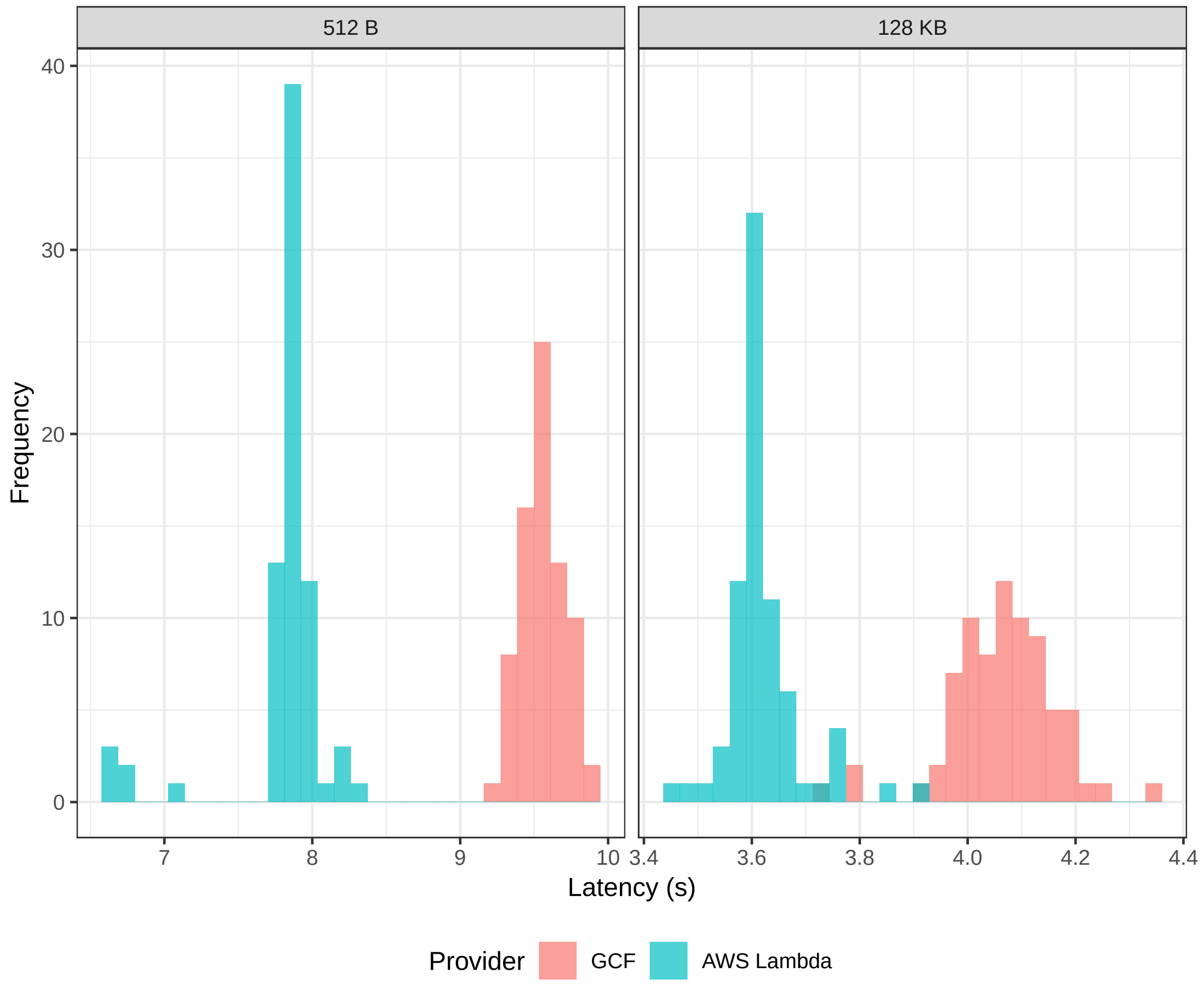

Figure 17 – Histograms of write latency for an 1 GB file in AWS Lambda and GCF using 512 B and 128 KB I/O sizes

Source: The author

APPENDIX B – HISTOGRAMS OF WRITE LATENCY ON SMALL FILES (10 KB)

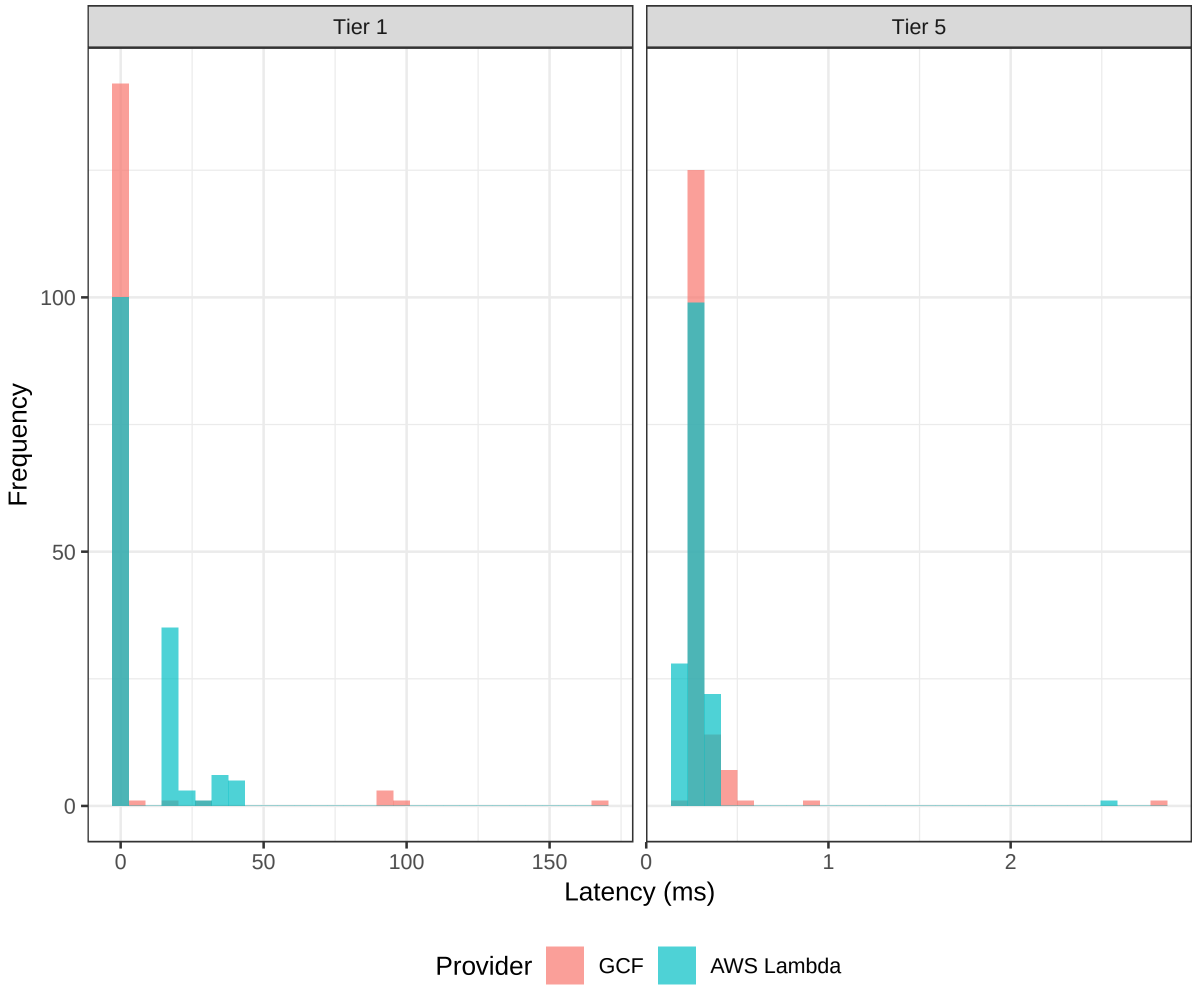

Figure 18 – Histograms of write latency for a 10 KB file in AWS Lambda and GCF using minimum and maximum resource tiers

Source: The author

APPENDIX C – HISTOGRAMS OF READ LATENCY ON LARGE FILES (1 GB)

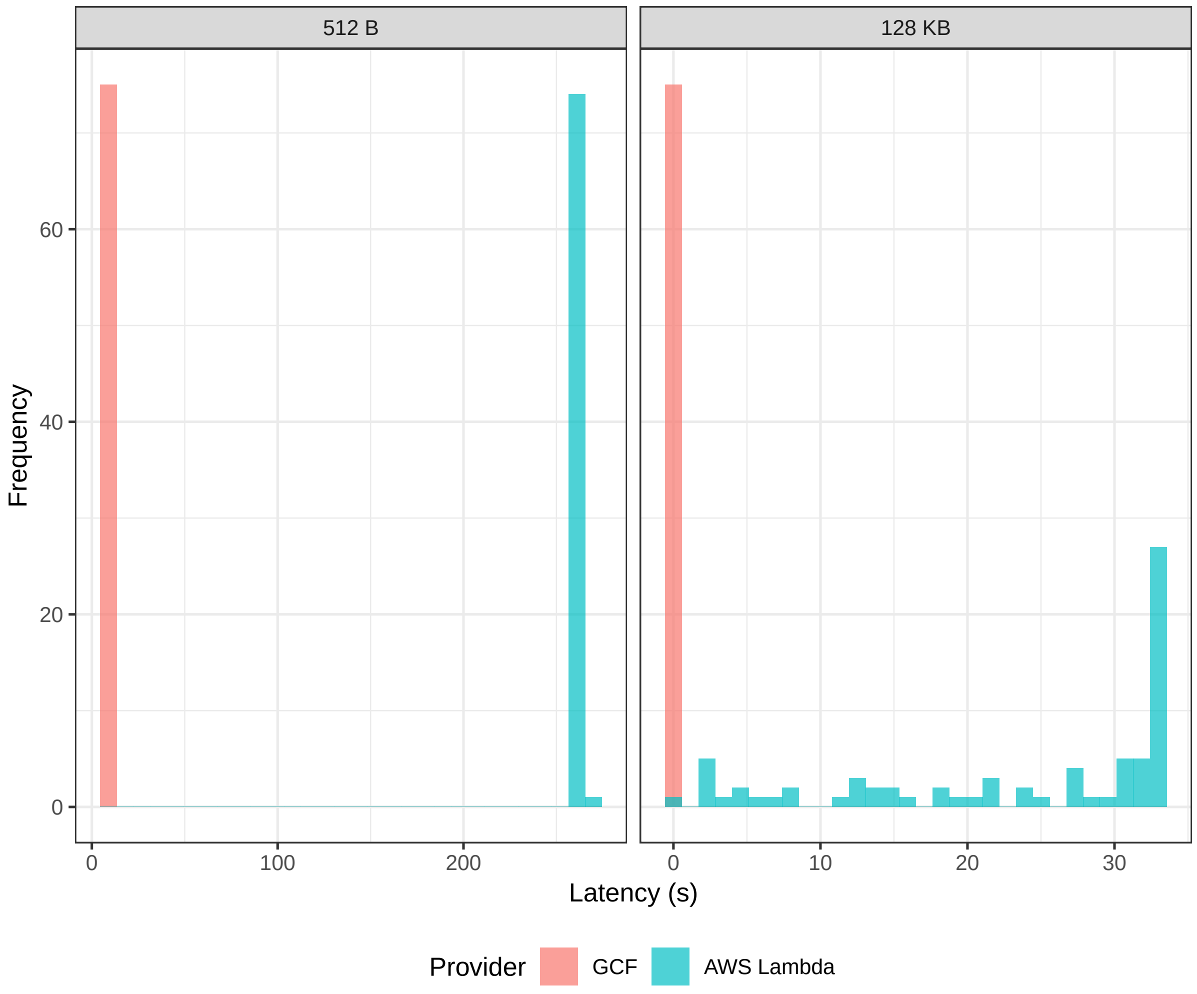

Figure 19 – Histograms of read latency for an 1 GB file in AWS Lambda and GCF using 512 B and 128 KB I/O sizes

Source: The author

APPENDIX D – HISTOGRAMS OF READ LATENCY ON SMALL FILES (10 KB)

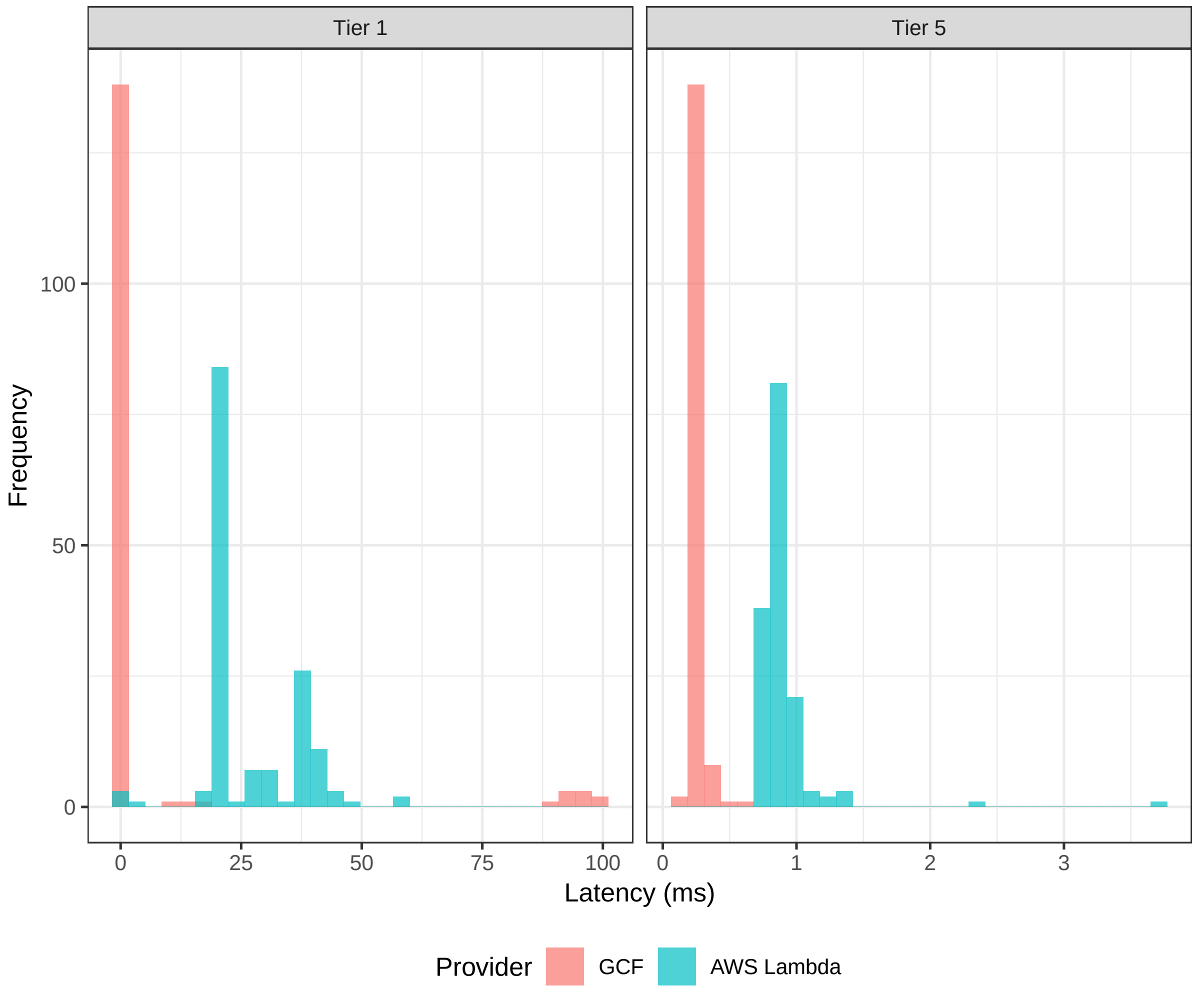

Figure 20 – Histograms of read latency for a 10 KB file in AWS Lambda and GCF using minimum and maximum resource tiers

Source: The author