`

Optimising Inter-Function Communication in Serverless Computing through Network-Aware Adaptive Data Compression

MSc Research Project

Msc. in Cloud Computing

Gurnam Sunil Arora

Student ID: 22142525

School of Computing

National College of Ireland

Supervisor: Dr. Punit Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Gurnam Sunil Arora		
Student ID:	22142525		
Programme:	Cloud Computing	**Year:**	2023-2024
Module:	MSc Research Project		
Supervisor:	Dr. Punit Gupta		
Submission Due Date:	14/12/2023		
Project Title:	Optimising Inter-Function Communication in Serverless Computing through Network-Aware Adaptive Data Compression		
Word Count:	6971		
Page Count:	20		

Signature: Gurnam Sunil Arora

Date: 13/12/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimising Inter-Function Communication in Serverless Computing through Network-Aware Adaptive Data Compression

Gurnam Sunil Arora

22142525

Abstract

Serverless computing has emerged as a popular cloud computing paradigm for offering scalability, cost-effectiveness and abstracting away the complexities of infrastructure management. However, its wide-spread adoption for mainstream and mission-critical applications is limited by critical challenges like inter-function communication latency. This research proposes adaptive data compression which dynamically compresses data based on payload characteristics and runtime metrics as a potential solution to improve inter-function communication latency. The efficacy of the proposed solution was thoroughly validated through extensive simulations and a real-world deployment. Findings showed substantial reductions in key workflow execution metrics within simulations and significant boosts in responsiveness and resilience were observed under peak load in the real-world scenario, proving its practical viability. This research thus sets the stage for building holistic strategies to further unlock the potential of serverless computing. The positive results open exciting new possibilities for enhancing performance across serverless architectures via adaptive data transformations.

Keywords: Serverless Computing, Inter-Function Communication latency, Data Compression.

1. Introduction

Serverless computing is an emerging cloud computing paradigm that is geared towards deployment of massive event driven systems in the cloud. Serverless platforms like Google Cloud Functions, AWS Lambda are notable examples that provide automatic scaling, pay-per-use billing model and abstract away complexities of infrastructure management to provide ease of development. Consequently, these attributes of serverless computing have led this cloud computing paradigm to gain significant prevalence in both academic and industry settings. However, the current state of serverless computing is not without its challenges. Current issues concerning coldstart latencies, state management, isolation, fault tolerance substantially limit the feasibility and hence wide-spread adoption of serverless computing for mainstream applications. While challenges like cold-start latency, scheduling have been studied fairly adequately, the challenges related to network and inter-function communication still remain open(LI et al. 2023). Shafiei et al. (2022) emphasise on the critical nature of "Network, sharing, and Intra communications" challenges in serverless computing and also mention these challenges as a cause of various issues. Hellerstein et al. (2018) argue that efficient and fast communication between serverless functions is crucial for unlocking the true potential, scale and adoption of serverless computing. They argue that these limitations obstruct the development of efficient distributed systems and parallel compute applications on serverless platforms, thereby impeding innovation in data processing and distributed computing. Despite advancements in cloud storage, brokers, and direct networking for inter-function communication, these methods often fall short in ensuring low latency across various workloads and application scenarios. To address these limitations, the proposed research explores a novel approach: adaptive data compression tailored for serverless computing. This method aims to reduce communication overheads in diverse use-cases by dynamically compressing data and payloads according to payload attributes, network conditions, and other relevant metrics. The central research question of this study is:

"Can an innovative adaptive data compression technique, which dynamically adjusts compression based on payload attributes and runtime metrics, significantly enhance the performance and efficiency of communication between distributed serverless functions?"

The goal of this research is to demonstrate that adaptive data compression can notably improve communication within serverless applications. By doing so, it aims to bolster the widespread adoption of serverless computing for both mainstream and mission-critical applications, showcasing its effectiveness in optimising inter-function communication in a serverless environment.

2. Related Work

Serverless architectures implement communication between functions using several methods. These methods include Shared Memory, where functions utilise a shared memory region on the host machine; Cloud Storage, using services like AWS S3 or object storage; Brokers, which are services/functions that act as an intermediary and facilitate communication between serverless functions, and Direct Networking, where functions communicate directly through intelligent techniques like TCP/UDP hole punching. All of these paradigms have their own benefits and tradeoffs and are examined in this section and are summarised in *Table-1.*

Shared Memory: Shared Memory is a common and long-established approach in computing paradigms such as multithreaded and parallel computing. It has effectively demonstrated to achieve highly efficient communication within these paradigms. The SPRIGHT framework (Qi, S. et al., 2022) implements inter-function communication between co-located functions (i.e. functions located on the same host) via Shared Memory, where functions located on the same host share a memory region to exchange data with each other. However, in such a situation where functions are not co located, the benefits of The SPRIGHT framework are lost due to the fact that in such a situation there cannot be a shared memory region. This limitation is bypassed by the SAND platform presented by Akkus, I. E. et al., (2018) through the use of a hierarchical memory bus. This hierarchical Memory bus consists of a local and global message bus for co-located and non co-located functions respectively. Nevertheless, the global message bus' pub-sub nature incurs multiple overheads like serialisation along with message broker look up. Another major disadvantage is that SAND decreases scheduling flexibility because the middleware makes an effort to place functions on the same host for better communication. However, the shared memory approach provides very poor scalability because only a limited number of functions can be crammed onto a single host. Both frameworks, SAND and SPRIGHT, propose significant optimizations and augmentation for a shared memory based communication but don't eliminate the inherent architectural limit with shared memory demanding research on alternate methods for faster communication among serverless functions.

Cloud Storage: One of the more popular methods of facilitating communication between serverless functions is Cloud Storage. This is due to the reason that serverless platforms like AWS restrict direct communication between serverless functions. A few research efforts have implemented optimisations to the utilisation of cloud storage. For instance, in the works of Fouladi, S. et al., (2019), a framework called 'gg' is proposed to parallelize distributed applications by leveraging serverless functions on AWS. 'gg' utilises services like S3 (AWS) as a medium to transfer immutable blobs, facilitating communication between stateless functions. A notable feature of 'gg' is its memoization technique (remembers output for a particular input), which stores the outcomes of functions based on content hashes. This approach helps 'gg' avoid repeating executions for identical inputs and save time. This approach however loses its benefits when new, non-memoized inputs are introduced. Starling (Perron, M et al,. 2020) is a query execution framework that focuses on distributed SQL query execution on serverless functions. This proposed framework also utilises cloud storage for interim stages during query execution. The optimization Starling makes is that it efficiently organises and partitions data in S3 and optimises complex data shuffle operations ensuring functions only read the necessary data partition. However, Starling's optimizations are specific to analytical and query workloads and have limited applicability for generic workloads compared to 'gg'.
While both these papers put forward very intelligent optimizations for cloud storage, they come with distinct trade-offs. These optimizations don't fully address the inherent latency issues and limitations of using cloud storage as a communication medium. Services like S3 (AWS) or Cloud Storage (GCP) are primarily designed for scalability, durability and cost-effectiveness. While these cloud storage services have been repurposed as a

communication medium in serverless functions, they are not ideally suitable for this role, often causing inherent latency and other limitations to serverless architectures.

***Brokers*:** The use of Brokers or intermediaries for inter-function communication is an emerging method in serverless computing. InfiniCache, introduced in the paper "InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache" by Wang, A. et al. (2020) is a prominent example of such an implementation. InfiniCache is an in-memory object caching system that uses serverless functions and employs a broker based architecture for facilitating communication between these functions. The system consists of three main components: Client libraries, Proxy servers and Lambda functions which act as primary cache nodes.
Since AWS does not allow direct communication between serverless function, proxies serve as a broker to facilitate this communication. The lambda functions serve as small data nodes to store cached data. When clients request data, the proxies use the cache metadata to determine which lambda function has this requested data. They retrieve this data from lambda functions and return it to the client. This broker based architecture has several advantages. Primarily, it overcomes the limitations imposed by cloud platforms of inbound connection to serverless functions. Additionally, the proxies enhance network efficiency by parallelizing streaming of cached data. However, there are some drawbacks, as these proxies can become a single point of failure and become a bottle-neck when the system is scaled.

Direct Networking: Direct networking is the most effective and preferred method for achieving data exchange between serverless functions. It allows for functions to communicate with each other directly on different hosts or even cloud regions using network protocols and channels. It is the most preferred way due to its support for high scalability in distributed serverless architecture. The first significant development in this area is detailed in "Punching Holes in the Cloud: Direct Communication between Serverless Functions Using NAT Traversal" by Moyer, D. W. (2021). This paper introduces the implementation of direct communication between serverless functions, and overcoming serverless platform restrictions using NAT traversal techniques. This approach shows impressive ultra low-latency communication with high throughput and is supported by a communication library based on TCP sockets. However, implementing this Socket-Level API can be exceedingly complicated for serverless applications. Conversely, The FMI framework (Copik, M. et al., 2023) implements an MPI (Message Passing Interface) inspired interface for communication between serverless functions. This interface provides a more developer-friendly approach to implement inter-function communication by message passing. The FMI framework offers a more flexible, efficient and high-throughput communication within serverless environments and supports various channels like TCP, object storage and in-memory data stores, alongside MPI collective communication operations. However, this approach struggles in a dynamic serverless environment since it assumes a fixed communicator group. Given that serverless functions are transient and scalable, assuming a fixed communicator group limits the scalability potential of serverless architectures. The Boxer framework (Wawrzoniak, M. et al., 2021) utilises TCP hole-punching for inter-function communication. Boxer employs a separate TCP hole-punching service within each serverless function and uses a coordinator service for function discovery. This framework implements a clever optimization which is connection multiplexing (reusing the same TCP connections instead of creating new ones) and further enhances scalability and efficiency. However, implementing hole-punching logic within serverless functions leads to larger function size which affects deployment and performance. Furthermore, the coordination service becomes a bottleneck when the system is scaled and hence limits the scalability of the serverless architecture. Finally, Particle (Thomas, S. et al., 2020) and Floki (Nestorov, A. M. et al., 2022) are frameworks that are designed to enhance network performance in container-based serverless environments like Kubernetes for burst-parallel workloads. Particle minimises the overhead for namespace creation and allows rapid container setup. Alternatively, Floki uses pipes and sockets for communication between co-located and non co-located containers. Essentially, both these frameworks put forward optimizations that improvise the network aspect of container based serverless environments, but have their limitations. Floki is tailored for data-intensive workloads and unsuitable for general serverless scenarios. Additionally, Floki's implementation of pipes and sockets restricts multi-tenancy and reduces fault-tolerance. Particles' approach of consolidating namespaces and network interfaces has an effect on function isolation that degrades security.

Framework	**Paradigm**	**Benefits**	**Limitations**
SPRIGHT	Shared Memory	Highly efficient for co-located functions	Doesn't work for distributed functions, poor scalability
SAND	Shared Memory	Works for distributed functions	Overheads from serialisation and lookup, reduces scheduling flexibility, limited scalability
gg	Cloud Storage	Parallelization, memoization to avoid re-computation	Loses benefits with new non-memoized inputs
Starling	Cloud Storage	Optimised data partitioning and shuffle	Specific to analytical workloads
InfiniCache	Brokers/Proxies	Overcomes platform limits on connections, parallel streaming	Proxies become bottleneck, single point of failure
NAT Traversal	Direct Networking	Ultra low latency, high throughput	Complex socket-level API
FMI	Direct Networking	Flexible, efficient, developer-friendly	Assumes fixed group, limits scalability
Boxer	Direct Networking	Connection multiplexing for efficiency	Bigger function size, coordination service becomes a bottleneck
Particle	Direct Networking	Rapid container setup	Affects isolation and security
Floki	Direct Networking	Data-intensive workload optimization	Restricts multi-tenancy, reduces fault tolerance

Table 1: Literature Review Summary

3. Research Methodology

3.1 Proposed Solution: The current body of work detailed in the Related Work section made some strides but has not been able to fully eliminate latency issues within serverless computing. This proposed solution aims to accomplish this by incorporating intelligent data compression at its core, tailored to optimise serverless inter-function communication. It is designed to intelligently detect the payload's type, size, and other characteristics to select the most optimal compression algorithm. Additionally, this approach takes into account compute availability for algorithm selection, ensuring efficient resource utilisation. Furthermore, the system is equipped to estimate network congestion by analysing function execution metrics, enabling adaptive compression. This ensures that compression is employed only when necessary, enhancing overall efficiency and reducing unnecessary computational overhead.

The methodology for this research endeavour employed a dual approach: Simulation and a Real-World Data-Driven Application. This methodology was specifically chosen to comprehensively and thoroughly validate the efficacy of the proposed solution. The simulation setup provided for a controlled environment for testing the solution in a wide range of scenarios, while the real world application provided practical validation in actual operating settings.

3.2 Approach 1: Simulation

Objective: The objective of running simulations was to test the efficacy of the solution of using Adaptive Data Compression in a wide variety of scenarios and test-cases without the complexity and variability of real-world settings. The simulation allowed for creation of multiple controlled environments with different complexities and sizes so that the solution could be thoroughly tested in a variety of scenarios including edge or extreme conditions, which may not be feasible to produce in a real serverless environment. This approach was crucial in initial evaluation of the solution and provided a proof of concept before it could be replicated and deployed in a real-world setting.

Simulation software: WorkflowSim (Chen and Deelman, 2012) was leveraged to replicate a serverless environment for simulations. WorkflowSim which is an extension of Cloudsim toolkit, is specifically designed to simulate and study workflow algorithms in cloud and grid computing, was chosen and repurposed here to simulate a serverless environment because of its relevant capabilities and robust features for modelling, such as:

- *Data Transfer Modelling:* The most crucial aspect for simulations within this research endeavour is to model data transfer within serverless functions. WorkflowSim allows for accurate modelling of data transfer between tasks (analogous to serverless functions), which is crucial for observing the impact of the proposed solution.
- *Diverse Workflows*: WorkflowSim supports a variety of workflow patterns, which enables testing of the solution in a variety of simulations.
- *Resource Management:* Since it is an extension of CloudSim, WorkflowSim is capable of simulating different computing resources and their management.
- *Task Scheduling and Execution*: workflowSim can emulate the behaviour of task scheduling and execution which is a core aspect of serverless computing.

Mimicking a serverless environment within WorkflowSim: To effectively replicate a serverless environment within WorkflowSim simulations, several key aspects of the serverless architecture were considered.

- *Tasks as Analogous to serverless functions*: WorkflowSim simulates workflows (Jain A. and Kumari R., 2017) which comprise several dependent tasks. These tasks were used to represent serverless functions within the simulation. This analogy is crucial as it allows us to simulate interactions between serverless functions within a cloud environment. Like serverless functions, each task in the workflow can be defined using specific inputs, outputs, and computing requirements, mirroring the functionality of serverless functions in the serverless architecture.
- *Function Isolation and VM Mapping:* Serverless functions typically run in isolation within a cloud environment. This isolation was simulated by assigning each task to a distinct VM (Virtual Machine) within the simulations. This assignment of tasks to distinct VM's not only allowed for isolation but also enabled the modelling of data transfer times and hence assessment of inter function communication latency.
- *Ephemeral Nature of Functions:* In serverless computing, serverless functions are ephemeral, i.e operating only when triggered and not retaining operation status between executions. Using WorkflowSim tasks to represent serverless functions incorporates this characteristic within simulations as WorkflowSim tasks are also transient in nature.

By choosing WorkflowSim for simulations and leveraging workflow tasks to represent serverless functions, we were able to closely emulate the operational dynamics of a serverless environment. This in turn allowed for a thorough investigation into the performance of Adaptive Data compression within a serverless context.

Experiments: Two scientific workflows were utilised to create simulation scenarios, namely: Montage and Inspiral. Montage (De Prado, R. P. *et al.,* 2014) is a data intensive application for generating sky mosaics and Inspiral is an application that analyses gravitational waves from merging black holes and neutron stars. The number of tasks for both these workflows were varied to create different test scenarios. For Montage workflow, Montage_25, Montage_50 and Montage_100 were utilised (the number represents the number of tasks within the workflow). Similarly, Inspiral, Inspiral_30, Inspiral_50, Inspiral_100 were used.

Two primary test cases were defined for all these scenarios. The first with 'Compression Off' runs the workflow simulations, without adaptive data compression. The second case with 'Compression On' executes the same workflows with adaptive data compression enabled. The goal of defining comparative test cases was to isolate the impact of the proposed solution i.e Adaptive Data Compression.

Metrics such as Total execution time, Average Task execution time, Average Task Start-time and Average Task Finish-time were collected to study the performance of the proposed solution. These metrics were chosen as they enable quantification of efficacy of the solution with respect to various aspects of workflow execution, especially when comparatively analysing the baseline 'Compression off' case with 'Compression On' case.

3.3 Approach 2: Real-World Validation

Motivation: This approach with testing the Adaptive Data Compression by integrating it within a real-world data-driven application was vital for assessing the efficacy of the solution in actual serverless settings. This phase aimed to validate and extend the findings from the simulation experiments and provided insights into how the Adaptive Data Compression Library performed under real-world operation conditions. This allowed for testing this solution in a dynamic and uncontrolled environment where factors such as network variability, resource availability, coldstart latency and actual data characteristics could significantly impact performance.

Application use-case selection: The choice of creating a Data-driven application as a use-case for real-world testing was made to closely mirror a common serverless scenario. Since Data- driven applications are quite prevalent in cloud computing and often require efficient data-transfer between components, a data-driven application becomes an ideal candidate for demonstrating the effectiveness of Adaptive Data Compression as a solution to reduce inter-function communication latency.

Experiments: To validate the efficacy of Adaptive Data Compression in a real world setting, it was integrated with a Data-Driven application deployed using serverless functions on Google Cloud Platform (GCP) and a rigorous load test was conducted using Locust. This methodology allowed us to simulate user requests and measure response time (Total execution time) for both primary test-cases (i.e With Adaptive Data compression enabled and disabled). Performance metrics like 95th percentile response time, and failure rates were gathered by mimicking peak load conditions.

- *95th Percentile Response Time*: This measures the response time below which 95% of the requests are fulfilled.
- *50th Percentile Response Time*: Often referred to as the median response time, this metric represents the middle value of the response time distribution.
- *Failure rate:* Refers to the proportion of requests to the application that fail compared to the total number of requests made during a test period

These metrics effectively quantify the efficacy of Adaptive Data compression in a real-world setting beyond theoretical advantages, and allow a comprehensive insight into the value of this solution as an optimization technique

4. Design Specification

Overview of the Adaptive Data Compression (ADC) Library: This library has been meticulously engineered to enhance data transfer speeds and hence improve inter- function communication latency in serverless environments. It estimates the amount of network congestion by monitoring the average response time of serverless functions over a recent time series data. If the average response time exceeds a certain predefined threshold, the library enables data compression. The choice of which compression algorithm is to be utilised is made intelligently by looking at factors like payload type, payload size and availability of compute resources within the serverless function. This comprehensive approach allows this library to significantly improve inter-function communication by optimising the data-transfer process.

Design choices: Design choices around threshold determination and compression algorithm selection have been detailed below. The pseudocode of the Adaptive Data Compression Algorithm and the System Architecture has been presented in *Figure 1*.

Threshold Determination: The library employs a monitoring mechanism to estimate the prevailing conditions of the network by monitoring function response times. Using the Google Cloud Monitoring API, the library fetches average response time for recent time series metrics data and compares this average to a predefined threshold to enable compression. This predefined threshold can be tweaked according to the operational needs by the users/developers, allowing for greater flexibility and catering to specific application requirements

Compression Algorithm Selection Logic: The ADC Library's compression algorithm is intelligently selected based on data size, type, and available resources (Jeong, G. *et al.,* 2023). For data under 1MB, LZ4 is used for its fast processing, while Snappy is chosen for data sizes between 1MB and 10MB, offering a balance of speed and efficiency. For larger datasets over 10MB, Zstd is employed for optimal compression. The library uses Brotli for textual data to maximise efficiency and Zstd or zlib for binary/non-textual or general/mixed data types, ensuring versatility and broad compatibility. Depending on resource availability, LZ4 or Snappy is used in resource-constrained environments for their low overhead, while Zstd or Brotli is preferred in scenarios with adequate resources for higher compression effectiveness.

Algorithm 1 Adaptive Data Compression

```
for each data transfer between functions do
    measure currentResponseTime and resourceAvailability
    if currentResponseTime > threshold then
        if resourceAvailability is Constrained then
            if data size < 1 MB then
                Compress with LZ4
            else
                Compress with Snappy
            end if
        else if resourceAvailability is adequate then
            if data size < 1 MB then
                Compress with LZ4
            else if 1 MB < data size < 10 MB then
                Compress with Snappy
            else if data size > 10 MB then
                Compress with Zstd
            else if textual data then
                Compress with Brotli
            else
                Compress with zlib
            end if
        end if
        Send compressed data
    else
        Send original data
    end if
end for
Upon receiving data
if data compressed then
    Decompress data
end if
Process data
```

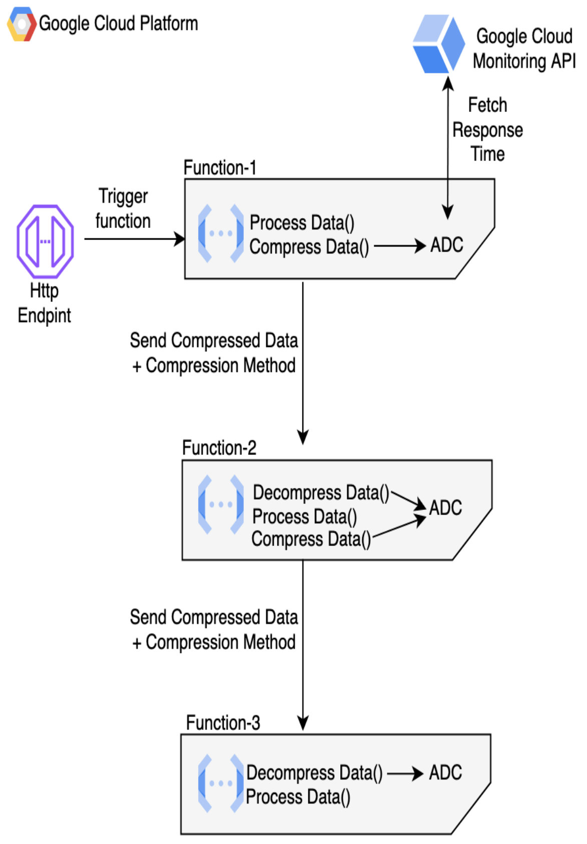


Figure 1: Pseudocode for Adaptive Data Compression And System Architecture

System Architecture:

- *Integration with Serverless Functions*: The architecture of the ADC library is designed for seamless integration with serverless functions within cloud infrastructures, ensuring minimal disruption.
- *Data Flow Optimization*: The library is structured to intercept and compress data as it moves between serverless functions, thereby reducing data transfer times and overall latency.

5. Implementation

The implementation phase of this research endeavour was pivotal in materialising the Adaptive Data Compression algorithm to a practical solution and evaluate its effectiveness. The implementation phase also employed a twofold approach. The first approach entailed conducting preliminary experimentation to gather crucial data regarding important metrics (compression factor, compression time and decompression time) which was followed by a comprehensive simulation setup to evaluate the algorithm's effectiveness in a controlled and varied environments. The second part of this implementation entailed the development of the Real-World Data-Driven application within which the Adaptive Data Compression Library was integrated. This phase was crucial to test the solution in a real serverless environment. Each step within this phase was meticulously executed to ensure that the Adaptive Data Compression library met its intended design, and also adapted efficiently to dynamic needs of serverless computing, thereby providing a robust solution for mitigating latency in inter-function communication within serverless computing.

5.1 Preliminary Experimentation:

Objective: The objective of the preliminary experimentation was to gather critical metrics that enable the ‘Compression On’ (i.e with Adaptive Data Compression enabled) scenario in the simulations. This specifically involved determining the average compression factor, compression time, and decompression time for a wide range of payload sizes and payload types.

Methodology :

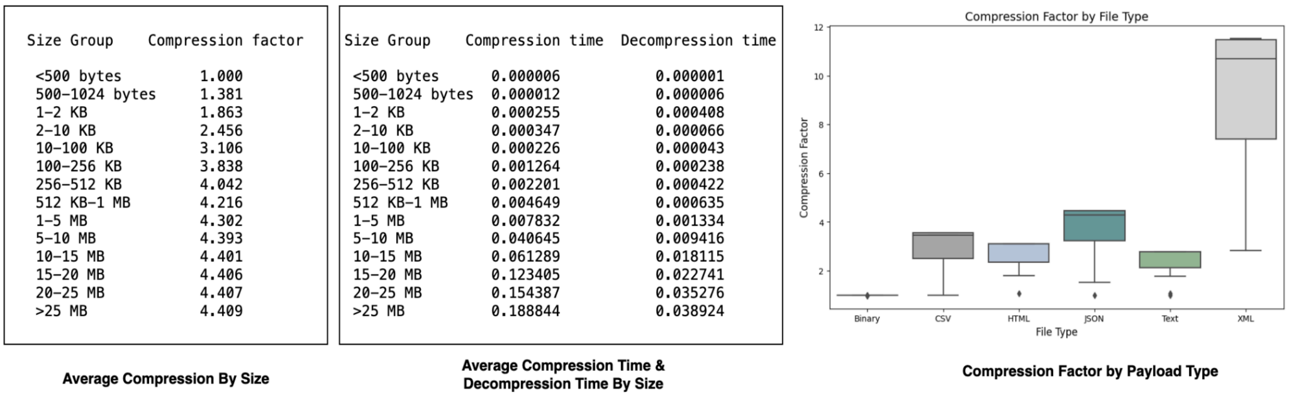
- *Synthetic payload creation:* Synthetic Payloads of various types (Binary, JSON, CSV, HTML, XML & Text) and various sizes (from 500 bytes to 25 MB) were created using the Faker Library in Python. These payloads were meticulously created to resemble real payloads encountered in serverless computing, ensuring relevance of the simulation experiments to real world scenarios.
- *Setting up Cloud Storage* : Google cloud Storage was utilised to store these payloads.
- *Deployment of Cloud Function:* A serverless function was created using Google Cloud Functions (which is a FaaS offering from Google). This cloud function (1 CPU, 1GB RAM) was implemented in Python 3.9 and developed to automate the process of

fetching each payload from the cloud storage, applying compression/decompression and collecting critical metrics (compression factor, compression time and decompression time) for each payload.

- *Compression and Decompression Process:* For each file, the cloud function executed compression and decompression using the Adaptive Data Compression algorithm. The algorithm employed various compression libraries (LZ4 for smaller files, Brolti for Text, and Zstd for other types) and intelligently selected the optimum library for compression. The serverless function subsequently recorded the compression factor, compression time and decompression time for each payload.
- *Data Diversity and Coverage*: The methodology for this experimentation particularly focused on ensuring a broad payload type coverage, to simulate real world scenarios where serverless functions may interact with various data types and sizes.

Utilisation of Collected Metrics in Simulation:

Data Analysis.: Post-experimentation, this metrics data was analysed (see Figure 2) to determine the average compression factors, the average compression time and decompression time for a particular payload size across various types (i.e Binary, JSON, CSV, HTML, XML, Text).



```
 Size Group    Compression factor

  <500 bytes         1.000
  500-1024 bytes     1.381
  1-2 KB             1.863
  2-10 KB            2.456
  10-100 KB          3.106
  100-256 KB         3.838
  256-512 KB         4.042
  512 KB-1 MB        4.216
  1-5 MB             4.302
  5-10 MB            4.393
  10-15 MB           4.401
  15-20 MB           4.406
  20-25 MB           4.407
  >25 MB             4.409
```

Average Compression By Size

```
Size Group    Compression time  Decompression time

 <500 bytes      0.000006          0.000001
 500-1024 bytes  0.000012          0.000006
 1-2 KB          0.000255          0.000408
 2-10 KB         0.000347          0.000066
 10-100 KB       0.000226          0.000043
 100-256 KB      0.001264          0.000238
 256-512 KB      0.002201          0.000422
 512 KB-1 MB     0.004649          0.000635
 1-5 MB          0.007832          0.001334
 5-10 MB         0.040645          0.009416
 10-15 MB        0.061289          0.018115
 15-20 MB        0.123405          0.022741
 20-25 MB        0.154387          0.035276
 >25 MB          0.188844          0.038924
```

Average Compression Time & Decompression Time By Size

Compression Factor by Payload Type

Figure 2: Data Analysis from Preliminary Experiments deriving Critical Metrics

Application in Simulation: The derived metrics were then utilised to enable the 'Compression on' test-case for the simulations. For example, adjustments were made to the Montage_25 workflow DAX (Directed Acyclic Graph XML) to scale down the 'size' attribute according to the average compression factor, similarly the 'runtime' attribute was adjusted to account for average compression and decompression time for that particular payload size.

Enhancing realism and accuracy in Simulations: The determination of these critical metrics (ie. average compression factor, compression time and decompression time) using a real cloud function enhances the realism of the simulations. Imputing these metrics in the workflow DAX ensured the accurate modelling of compression/decompression in the simulation environment.

5.2 Simulation Setup:

Development Environment: The simulations were conducted using the WorkflowSim simulator. This simulator was chosen for its capabilities to accurately simulate serverless environments, especially because of this capability to accurately model data transfer between tasks (serverless functions). The development was carried out in Java and leveraged WorkflowSim's, extensive support for cloud and grid computing simulations.

Implementation of the Planning Algorithm: A new planning algorithm was developed and implemented to enable scheduling of tasks (functions) to distinct Vm's. This was done to observe key serverless architecture characteristics like function isolation and allowed for modelling data transfer between tasks(functions). The pseudocode for this algorithm is given in Figure 3.

```
Algorithm 2 Task to Distinct VM Assignment
 1: Initialize randomGenerator with current system time
 2: Create availableVms list from VM list
 3: for each task in task list do
 4:     if availableVms is empty then
 5:         Break from loop
 6:     end if
 7:     Select random index from availableVms using randomGenerator
 8:     Get the VM at the random index as vm
 9:     Assign the task to vm
10:     Set VM ID of the task to vm's ID
11:     Remove the selected vm from availableVms
12: end for
```

Figure 3: Pseudocode For the 'Distinct' Planning Algorithm

Configuration of simulation scenarios: Two primary test-cases were configured across a variety of workflows : 'Compression Off' and 'Compression On'.

- *Compression Off Scenario*: This served as a baseline test-case for all workflows, and represented a system where-in the system operated without the use of adaptive data compression. This baseline test-case was implemented for all the workflows: Montage_25, Montage_50, Montage_100, Inspiral_30, Insiral_50, and Inspiral_100. The numbers here denote the count of tasks within each workflow (for example, Montage_25 has 25 total tasks, similarly Inpiral_50 has 50 total tasks and so on.). The number of Virtual Machine's (VM's) were set equal to the number of tasks within each workflow and the bandwidth within the data-centre was set to 15MBPS to standardise network conditions across all tests.
- *Compression On Scenario:* This test- case represents a system where-in the adaptive data compression is enabled. The same workflows were used as in the 'Compression Off' scenario but with modifications of 'size' and 'runtime' attribute in the workflow DAX to reflect the effects of data compression. For all tasks in the workflow, the payload size was scaled down to size after compression using the average compression factor obtained from the preliminary experiment. Additionally the

runtime attribute was adjusted to account for compression and decompression time, which were also obtained from the preliminary experiment.

Essentially, Both the test-cases were executed under the exact same simulation configuration except for the deliberate change in size and runtime attributes. The critical metrics that determine new values for 'size' and 'runtime' were derived from the preliminary experiment which was conducted using a cloud function that employed the Adaptive Data Compression library to compress/decompress data and record relevant metrics.

Metrics Collection and Evaluation: The simulation output was used to collect a range of metrics, which enabled the performance assessment of the simulation. A thorough performance study of the solution has been presented in the Evaluation Section using these metrics:

- *Total Execution Time*: Measures the total time taken for workflow execution in both test-cases ('Compression On' and 'Compression Off')
- *Average Task Execution Time:* Measures the average execution time for all tasks within the workflow
- *Average Task Start-Time*: Measures the average time at which tasks in workflowSim begin execution.
- *Average Task Finish-Time*: Measures the average time at which tasks finish execution in workflowsim

Each of these metrics provide valuable insight about the effects of the proposed solution into various aspects of workflow execution and performance. Comparative analysis of these metrics for both primary test-cases (i.e 'Compression On' and 'Off') provide a clear understanding of the impact that the proposed solution has on overall performance workflows within the simulation.

5.3 Real-World Application

Adaptive Data Compression Library Development: The adaptive data compression library was developed in Python 3.9 and was focused on creating a robust and efficient tool for data compression. The library was designed to intelligently select the optimum compression algorithm (from zlib, brotli, LZ4, Zstd, snappy) to compress data. This library utilised the Google Cloud Monitoring API (formerly known as Stackdriver) to monitor the response time of serverless functions and estimate the amount of network congestion. If the response time exceeds a predefined threshold, the library dynamically enables data compression and hence mitigates the increased latency caused due to network congestion. Special precautions were taken to maintain a low algorithmic complexity of this library to keep the overheads low.

Integration with cloud functions and Application: The Adaptive Data compression library was integrated with serverless function runtime by packaging it with the function source-code as a separate python file. This integration was designed to be minimally invasive and modular to ensure that the solution could be adopted with minimal change to the codebase. This design allowed for scalable integration of the library with the serverless functions.

Deployment and Enhancements: The library was evaluated and incrementally enhanced following the deployment. Several changes were made to the monitoring and compression algorithm selection logic to improve the performance of the adaptive data compression library within serverless environments.

Challenges and Solutions: A major challenge was to balance the trade-off between compression efficiency and computational overhead. This challenge was addressed by carefully optimising the algorithm to introduce minimum overhead and performance fine tuning. This involved fine tuning the compression algorithm selection logic and also reducing the response time monitoring overhead, which was done by reducing API calls to Google Cloud monitoring API and pulling aggregate response time data over larger time-series intervals, instead of frequently gathering time-series data for smaller intervals. This approach reduced the computational burden of locally aggregating the time-series data and network traffic, leading to faster and more efficient execution of algorithm logic.

6. Evaluation

This section delves into the empirical analysis conducted using two distinct types of workflows: Montage and Inspiral. Both these workflows represent real-world data intensive scientific applications and serve as a cornerstone for these experiments.

6.1 Experiment 1 : Montage Workflows

Description: Montage workflows are data-intensive applications that are used for creating sky mosaics. In the context of this research, the Montage workflows represent a complex serverless application with complex tasks involving a significant amount of data transfer, making them ideal to assess the efficacy of the proposed solution.

Objective: This experimentation scenario considers three variants of the Montage workflow (Montage_25, Montage_50 and Montage_100) and records Total Execution Time, Average Task Execution Time, Average Task Start-Time and Average Task Finish-Time for both primary test cases i.e 'Compression Off' and 'Compression On'.

Simulation Configuration:
Number of VMs: Equal to the number of task in each workflow variant.
Datacenter: Standard Datacenter Setup with 1 Datacenter
Hosts: Configured to accommodate the number of VM's for each workflow variant.
Bandwidth: Datacenter Bandwidth is standardised to 15MBPS across all test to maintain uniform network conditions
Payload Size : 10Mb
Compression On Specific Configuration :
    Compression Factor : 4.401x
    Compression Time : 0.061 Seconds
    Decompression Time : 0.018 Seconds

Results:

- Total Execution Time: Reduced from 356.72 seconds (without compression) to 118.49 seconds (with compression) for Montage_25. Approximately 3x reduction for all variants
- Average Task Execution Time:Minimum improvement was observed to be 61% indicating substantial enhancement in task execution efficiency.
- Average Task Start and Finish Times: Similarly, 2.5-3 times reduction was observed indicating accelerated start and completion of tasks.

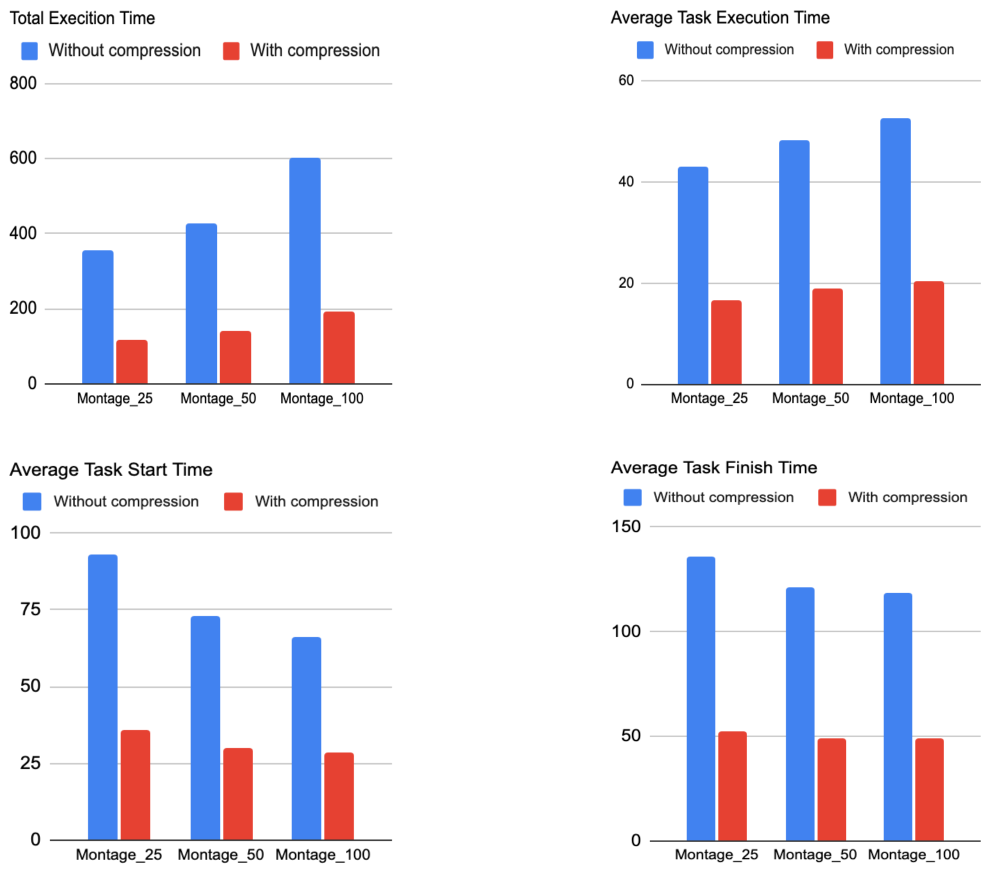


Figure 4 : Total Execution Time, Average Task Execution Time, Average Task Start and Finish Times for Montage Workflows (Seconds)

Inference : Considerable enhancements are observed across all four metrics measured in the 'Compression On' scenario, demonstrating a marked increase in data-transfer efficiency and, consequently, quicker task completion times when data compression is enabled. However, the performance benefits do not scale proportionally with the addition of tasks. This phenomenon can be attributed to the increase in the number of concurrent (parallel) tasks, as opposed to an increase in sequential (dependent) tasks within the workflow. As the workflow expands horizontally with more parallel tasks, the potential for ADC to reduce the execution time for each task diminishes, resulting in a less pronounced overall improvement in performance.

6.2 Experiment 2: Inspiral Workflows

Description: Inspiral workflow is used in analysis of gravitational waves from black holes and neutron stars, these workflows are computationally intensive and also involve complex data analysis. In the context of this research, this workflow represents a distinct serverless scenario where there are more parallel tasks than dependent or sequential tasks. This means that the workflow has more breadth than depth (Inspiral has depth of 5 compared to 9 of Montage). This structural characteristic allows to study the effectiveness of the proposed solution under a different workflow configuration compared to Montage

Objective: Similar to Experiment 1, this experiment also assess three variants of Inspiral workflow (Inspiral_30, Inspiral_50 and Inspiral_100) and collects the same metrics for the same two primary test-cases i.e 'Compression Off ' and 'Compression On'

Simulation Configuration:
Number of VMs: Equal to the number of tasks in each workflow variant.
Datacenter: Standard Datacenter Setup with 1 Datacenter
Hosts: Configured to accommodate the number of VM's for each workflow variant.
Bandwidth: Datacenter Bandwidth is standardised to 15MBPS across all test to maintain uniform network conditions
Payload Size : 10Mb
Compression On Specific Configuration :
Compression Factor : 4.401x ; Compression Time : 0.061 Seconds ; Decompression Time : 0.018 Seconds

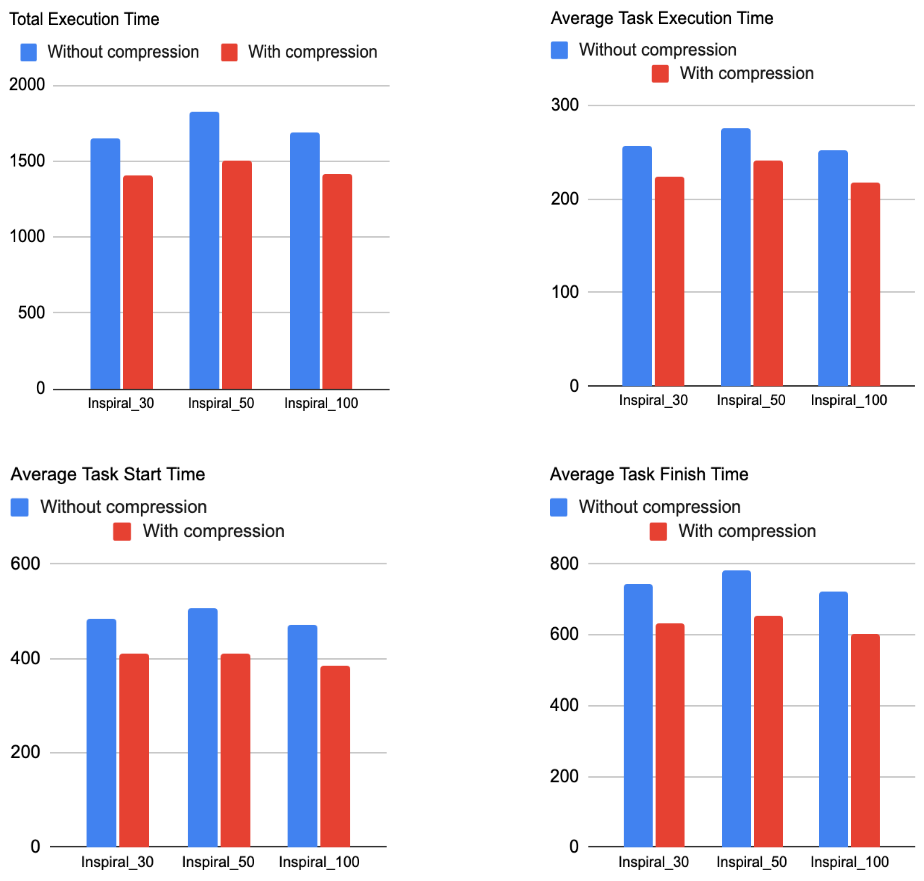


Figure 5 : Total Execution Time, Average Task Execution Time, Average Task Start and Finish Times for Inspiral Workflows

Results:

- Total Execution Time: There was a significant average reduction of 16% for all variants, although not as substantial as Montage workflows

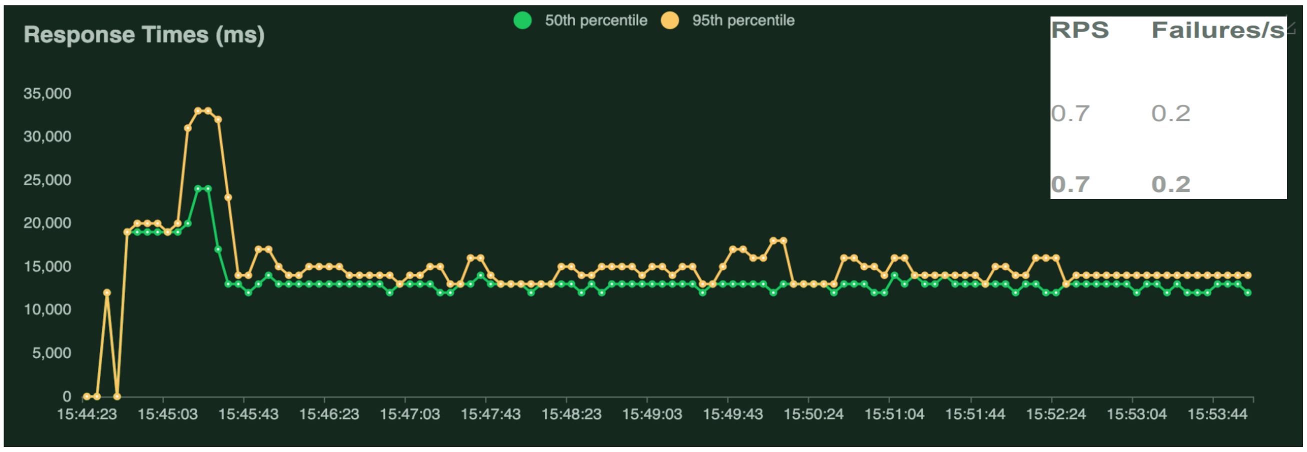
- Average Task Execution Time: signifcant reduction of about 12% across all variants indicating improved task efficiency. but the effect of data compression is somewhat moderated
- Average Task Start and Finish Times: Approximately 15 - 18% improvement across variants showing modest acceleration in overall task completion

Inference: The results show good improvements in task execution times which resulted from improved data transfer efficiency, however the performance gains were modest compared to the Montage workflows. This can be attributed to the structural characteristics of the Inspiral workflow. Contrasting to the Montage workflow, Inspiral workflow has more breadth and less depth i.e there are more parallel tasks than dependent or sequential tasks, compared to the Montage workflow. This results in a lower gain in performance as data compression has less opportunity to reduce transfer times. This underscores the importance of the workflow structure in determining the effectiveness of data compression and highlights that the true strength of data compression lies in sequential workflows.

6.3 Experiment 3: Real-World Application Performance Evaluation

Description: This experiment was conducted on the real-world data driven application to assess the impact of Adaptive Data Compression in actual operational settings. Locust, a versatile load testing tool was employed to measure the applications execution time and responsiveness under two scenarios- ADC enabled ('Compression On') and ADC disabled ('Compression Off').

Objective: The main objective of this experiment was to monitor applications performance in terms of overall response time (execution time) and failure rates during a 10 minute window for both scenarios to isolate the impact of adaptive data compression.



RPS	Failures/s
0.7	0.2
0.7	0.2

Figure 6: 95th & 50th Percentile Response Time vs Time and Failure rate for Compression Off

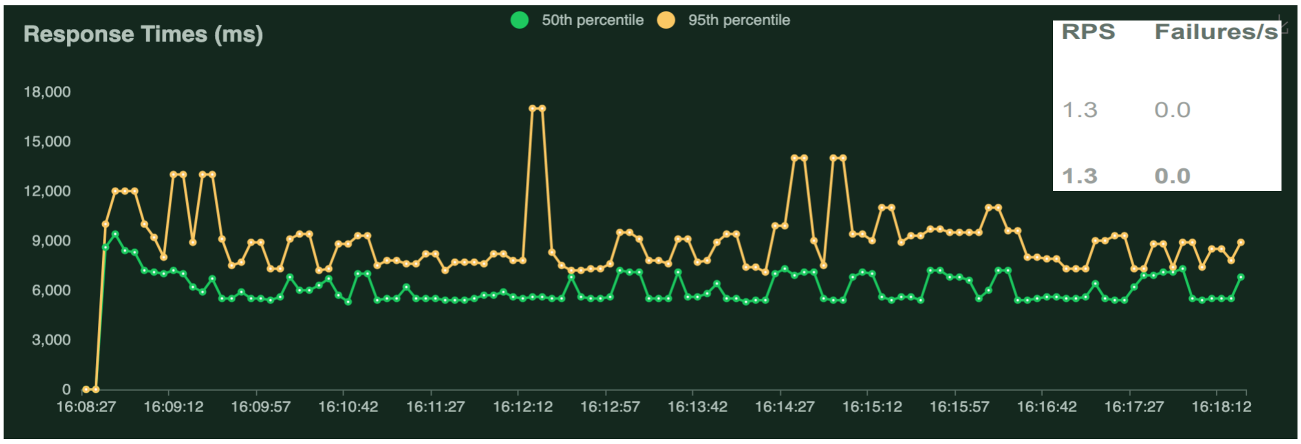


Figure 7: 95th & 50th Percentile Response Time vs Time and Failure rate for Compression On

Results:

- **95th Percentile Response Time:** With compression enabled the response time was reduced from 17000 ms in 'Compression Off' scenario to 9000 ms in 'Compression on 'Scenario
- **50th Percentile Response time**: This metric representing the median response time dropped from 13000 ms in the 'Compression Off' scenario to 5600ms in 'Compression On' scenario.
- **Failure Rate**:The failure rate dropped from 20% in 'Compression Off' scenario to 0% in 'Compression On'

Inference: The experiment's findings underscore the effectiveness of Adaptive Data Compression in enhancing both the responsiveness and reliability of real-world applications under load conditions. These results affirm the practical applicability of Adaptive Data Compression, reinforcing its value as a critical optimization technique for real-world applications in serverless environments.

6.4 Discussion

This research endeavour encompassed various experiments focusing on adaptive data compression for reducing inter-function communication latency in serverless computing. The results highlighted both strengths and weaknesses of the proposed solution. While the proposed solution provides significant data transfer latency in sequential workflows, it was not as effective in parallel task workflows. This limitation means that the solution is more specialised and focused in its scope, very effective in specific scenarios but less in others, particularly those that do not follow a sequential workflow pattern. As noted in the literature review, serverless computing faces inherent latency issues, and while this proposed solution partly addresses these concerns, it also underscores the need for a broader strategy to tackle the inherent latency issues in serverless computing. However, This research contributes to the existing body of knowledge by demonstrating the potential of adaptive data compression in serverless computing, albeit with room for further optimization.

7. Conclusion and Future Work

This research endeavour aimed to explore the effectiveness of Adaptive Data Compression in enhancing efficiency and reducing latency in inter-function communication. The main findings suggest that approach significantly improves latency within inter-function communication by enhancing data transfer efficiency. However, the findings also suggest that the impact of this optimization depends on the workflow structure. The research successfully addressed the research question by demonstrating the potential of adaptive data compression, but also revealed its limitation in parallel workflow scenarios.
For future work, development of more context-aware and dynamic compression strategies can be explored. Additionally, examining the application of this solution within different cloud computing paradigms like cloud and distributed computing can be a valuable avenue of exploration. Furthermore, applying the optimization of adaptive data compression to other communication mechanisms like pub-sub messaging systems, or cloud storage could reveal the versatility of this technique. Exploring all these research avenues could provide an in-depth understanding of the solutions efficacy and potential across various cloud scenarios and communication models which can lead to development of a broader and more versatile optimization strategy for cloud computing environments. This research lays the groundwork for future innovations in communication strategies and optimizations within the cloud computing context.

References

Akkus, I. E. *et al.* (2018) "{SAND}: Towards {high-performance} serverless computing," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 923–935.

Chen, W. and Deelman, E. (2012) "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th International Conference on E-Science*. IEEE.

Copik, M. *et al.* (2023) "FMI: Fast and cheap message passing for serverless functions," in *Proceedings of the 37th International Conference on Supercomputing*. New York, NY, USA: ACM.

De Prado, R. P. *et al.* (2014) "Processing astronomical image mosaic workflows with an expert broker in cloud computing," *Image Processing & Communications*, 19(4), pp. 5–20. doi: 10.1515/ipc-2015-0020.

Fouladi, S. *et al.* (2019) "From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 475–488.

Hellerstein, J. M. *et al.* (2018) "Serverless computing: One step forward, two steps back," *arXiv [cs.DC]*. Available at: http://arxiv.org/abs/1812.03651

Jain, A. and Kumari, R. (2017) "A review on comparison of workflow scheduling algorithms with scientific workflows," in *Advances in Intelligent Systems and Computing*. Singapore: Springer Singapore, pp. 613–622.

Jeong, G. *et al.* (2023) “Characterization of Data Compression in Datacenters,” in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE.

Li, Y. *et al.* (2023) “Serverless computing: State-of-the-art, challenges and opportunities,” *IEEE transactions on services computing*, 16(2), pp. 1522–1539. doi: 10.1109/tsc.2022.3166553.

Moyer, D. W. (2021) *Punching holes in the cloud: Direct communication between serverless functions using NAT traversal*. Virginia Tech.

Nestorov, A. M. *et al.* (2022) “Floki: A proactive data forwarding system for direct inter-function communication for serverless workflows,” in *Proceedings of the Eighth International Workshop on Container Technologies and Container Clouds*. New York, NY, USA: ACM.

Perron, M. *et al.* (2020) “Starling: A scalable query engine on cloud functions,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM.

Qi, S. *et al.* (2022) “SPRIGHT: Extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing,” in *Proceedings of the ACM SIGCOMM 2022 Conference*. New York, NY, USA: ACM.

Shafiei, H., Khonsari, A. and Mousavi, P. (2022) “Serverless computing: A survey of opportunities, challenges, and applications,” *ACM computing surveys*, 54(11s), pp. 1–32. doi: 10.1145/3510611.

Thomas, S. *et al.* (2020) “Particle: Ephemeral endpoints for serverless networking,” in *Proceedings of the 11th ACM Symposium on Cloud Computing*. New York, NY, USA: ACM.

Wang, A. *et al.* (2020) “{InfiniCache}: Exploiting ephemeral serverless functions to build a {cost-effective} memory cache,” in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 267–281.

Wawrzoniak, M. *et al.* (2021) “Boxer: Data analytics on network-enabled serverless platforms,” in *11th Annual Conference on Innovative Data Systems Research (CIDR 2021)*. ETH Zurich.