National College of Ireland

Cost optimization for I/O intensive workload in FaaS

MSc Research Project
Cloud Computing

Utkarsh kumar singh
Student ID: x22229698

School of Computing
National College of Ireland

Supervisor: Dr. Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Utkarsh kumar singh
Student ID:	x22229698
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr. Shivani Jaswal
Submission Due Date:	12/12/2024
Project Title:	Cost optimization for I/O intensive workload in FaaS
Word Count:	XXX
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Utkarsh kumar singh
Date:	29th January 2025

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):	

Cost optimization for I/O intensive workload in FaaS

Utkarsh kumar singh
x22229698

Abstract

FaaS is the most popular software paradigm in cloud computing world, and it's adoption is increasing day by day. FaaS provides a lucrative option for companies, government and individual developers as it is based on pay as you go model instead of traditional model where customers has to reserve the resource for specific period irrespective of whether the resource is idle or in use. FaaS billing model is based on function execution time, memory and CPU usage. This model works well for CPU computation workload whereas it fails when workload is I/O intensive where CPU and memory are not utilized fully thus leading to unfair charges to customer for these underutilized resources. In this project an framework is proposed where workload is divided into two parts, where first one is CPU intensive workload and other is I/O intensive workload which are executed separately. Amazon Kinesis is leveraged for its scalability, deduplication and high throughput for dynamically adjusting resources of lambda function with the help of machine learning to predict memory and timeout of lambda function. AWS Cloudwatch is used to monitor timeout of lambda function if lambda function is timeout, alarm is triggered and model is trained again. The proposed framework has saved cost by 16%-32% for specific type of I/O intensive workload.

1 Introduction

In today's world cloud computing is becoming mainstream platform for everyone where companies are relocating resource from in-house to cloud due to several benefits offered by it in terms of scalability, reliability, cost, security and ease of use. AWS EC2 (Elastic Compute Cloud) is first cloud service offered by AWS which revolutionized the computing world, with EC2 amazon has solved two problems one of which is additional infrastructure for handling demand during peak season of Christmas, while in off season period rent out the excess infrastructure to others. EC2 was introduced nearly two decades ago, computing world has gone through many changes and the latest offering is serverless computing. Serverless computing lets developer focus on coding part only where all the infrastructure is managed by cloud service providers while also ensuring scalability , reliability and security. These benefits make it very lucrative options for everyone.

One of the most important factor is the cost of hosting on cloud and it is vastly cheaper when compared to in-house due to economy of scale. One of the latest offering from cloud service providers is serverless, where pay-as-you-go model is implemented in which customer is charged only for duration of program (function) execution, thus leading to larger cost saving due to not being charged for idle periods. Serverless function offload the responsibility of managing infrastructure to cloud service providers, while also offering

to quickly launch product by reducing development time significantly but it can also raise the development cost rapidly if serverless functions are not managed properly. If serverless function is allocated large amount of memory it will drive up the cost if memory is not utilized efficiently, whereas if too little memory is allocated it will slow down the execution which will increase the total execution time of function thus increasing the cost. Hence, there is need to find optimal configuration of serverless function to get most out of it.

Though most of cost optimization research is done in field is related to scheduling algorithms, optimal resource allocations, reducing cold start time etc. which benefits mainly CPU intensive workloads as the pricing model is based on CPU, memory and function runtime. This research focuses on reducing cost of I/O intensive workloads in serverless paradigm which is overlooked due to nature of serverless pricing model, thus leading to unfair bill due to under utilization of CPU, memory which is primary factor in current billing model. The aim is to reduce to the cost of I/O intensive workloads while also providing an alternative billing model which is fair to both users and cloud service providers.

In present most of research is focused on optimizing the cost on the basis of resource utilization which are CPU usage, memory as these two resource are main deciding factors in current billing model of cloud service providers. Various approaches are taken for optimizing cost such as function fusion where authors have combined set of function into one function thus reducing the total number of invocations. One of approaches is to use combination of Infrastructure as a Service (IaaS) and FaaS to lower cost at reasonable performance. Another study explore in area of hybrid cloud, by the help of public and private clouds where cost is reduced by placement of function in private cloud from public cloud while maintaining sufficient performance, it is achieved by greedy algorithm which decides whether function will be placed in public or private cloud. Lastly some studies have implemented optimal configuration with the help of machine learning to reduce cost.

In this research, a framework is proposed where workload is made up of CPU intensive and I/O intensive tasks which will be executed by different lambda function with optimal configuration. The workload has sample CPU intensive calculation and performing data transformation. The execution time along with other parameters are pulled from serverless function logs to form dataset on which machine learning model is trained which will suggest optimal configuration on subsequent invocations. By providing insights gained from this research, the goal is to maximize utilization of resources while keeping cost in mind. This research aims to provide key findings in field of serverless computing which can be valuable for both academic and practical works.

1.1 Research Question

What are the ways of cost optimization for I/O intensive workload in Function as a service paradigm?

- Divide the complex workload into CPU intensive workload and I/O intensive workload.
- Dynamically adjust configuration of lambda function.
- Using Sagemaker to build,train and deploy Machine learning model which will predict optimal configuration.

- Using Elastic file system to share CPU intensive workload code between lambda function.
- Evaluate the optimization strategies implemented and their effectiveness.
- Leveraging event driven architecture to coordinate between different AWS services.

1.2 Ethics Consideration

This study does not include human subjects or private/public datasets, as per Table 1.

Declaration of Ethics Consideration Table	Yes / No
This project involves human participants	No
The project makes use of secondary dataset(s) created by the researcher	No
The project makes use of public secondary dataset(s)	No
The project makes use of non-public secondary dataset(s)	No
Approval letter from non-public secondary dataset(s) owner received	No

Table 1: Declaration of Ethics Consideration Table

1.3 Document Structure

The document is structured as follows in this research paper, where section 1 does in detail about background of cloud computing and evolution of from IaaS to FaaS. What are the current problems in serverless computing? Gives overview of proposed solution. Literature review is in Section 2 where various papers referenced each solving different parts of research question, it serves as starting point of research as it provide summary of research till date. It is important point of reference for different strategies implemented by researcher, along with what approaches have been taken along with its finding and gap in research. In Section 3 goes in detail about Methodology and Design Specification. Design specification is discussed in Section 4. Implementation is discussed in Section 5. In Section 6 experimentation results are discussed and lastly conclusion and future work is mentioned in 7.

2 Literature Review

2.1 Resource and cost management

Following papers Dittakavi; 2021 and Spillner (2020) have worked on resources management. In Spillner; 2020 paper author discussed on current problem faced by customers where they can only choose from predefined configuration offered from cloud service providers. To reduce the wastage of unused memory authors have proposed a solution which consist of three parts. First is to closely monitor the resource usage of function over period of time which will be used to create trace profile. Second is auto tuning of memory allocation for docker based applications where container can free up memory in advance which could be utilized by another container which result in increase of containers numbers thus allowing vertical scaling. Third is cost comparison of optimization vs base FaaS.

In paper Dittakavi (2021) there are 3 strategies is discussed to reduce the cost and managing resources effectively. First is cloud/VM based technique where various configuration is explored for different use cases and based on these different models and algorithm are proposed. Second is workflow techniques, where workflow is analysed also known as pre-processing stage then it is continuously monitored where resource is allocated/deallocated ensuring efficient execution depending on current workload. Lastly, placement of task ensuring sufficient resources are allocated, i.e. memory, CPU etc. this achieved by ILP technique and Multi-cost job routing and scheduling. In third strategy two distinct approaches are discussed one focused on efficient consumption of energy and other focuses on where scheduler focused on budget constraint and allocated resources which is under user budget.

Management of cloud resources is necessary for reducing expense of cloud , in data intensive workloads where most part of execution is read, write, create, delete and update of large volume of files, heavy media files and operation in database.

2.2 Cloud storage

In data intensive I/O workloads storage is key component as data, files are being read and written throughout the process. By optimizing the cost of storing of files and data it will bring down the overall cost of data intensive I/O workloads. The following papers (Borovica-Gajić et al.; 2016; Klimovic et al.; 2018) are in area of optimizing cost for data analytics applications using multi tier storage system.

This paper (Borovica-Gajić et al.; 2016) discuss in detail about how multi tier storage architecture could be used to bring down the cost when performing data analytics. In paper author have proposed Skipper architecture where it scale and perform better than traditional architecture when cold storage devices are used as primary storage. It takes advantage of greedy heuristics algorithm for efficient caching, scheduling algorithm which is based on rank and leveraging out of order execution for reducing switch latency. The paper discusses in details about using cold storage devices for data analytics to reduce the overall cost. Same could be implemented in FaaS using cold storage devices to lower the overall cost.

Following study (Borovica-Gajić et al.; 2016) is about Pocket which provides high throughput while using multi tier storage system for serverless platform with minimal latency. Pocket is able to adjust its parameter dynamically based on current usage pattern.

2.3 Caching

Caching is used to solve one of the biggest problem in serverless computing that is cold start. In cold start whenever serverless function is called for first time, there is delay because function instance need to be created first then boot it up to serve request. The delay is commonly known as cold start. Cold start causes overhead in current system which results in degraded performance of serverless function due to increase in latency. Cold start problem is solved by various researcher, one of such studies (Fuerst and Sharma; 2021) uses greedy dual algorithm to achieve 3 times more reduction compare to other approaches by introducing FaasCache. FaasCache can dynamically adjust its cache size based on workload and its keep alive is based on greedy dual policy resulting in increase in request capacity by two fold.

In (Wu et al.; 2020) study, a caching layer is introduced along with FaaS compute layer. This caching layer is known as HydroCache. It is built upon Anna(Wu et al.; 2019), a key-value store. Anna focuses on four things, one is sharding of key value store across various nodes for data scaling, second is using multiple masters for workload scaling where single key can be accessed to multiple threads, third is asynchronous message passing where there is no locking to ensure no waiting period for execution and lastly support for large number of system. In FaaSTCC(Lykhenko et al.; 2021), authors enhances on HydroCache(Wu et al.; 2020) by storing multiple copies of cache in storage then picking the most recent one to reduce execution latency of transaction, this is achieved by reducing the size of metadata which is then passed between function through workers. Consistency in FaaSTTC is achieved by using storage layer which ensures consistent value is returned.

2.4 Pricing models

Cost optimization is main objective of this research to understand how cloud service providers implement it. This topic is researched by various researchers with different objective but common point is pricing model. In one of paper (Mahajan et al.; 2019) authors have explored the pricing of service offered by cloud service providers from perspective of customers and providers. By using Nash equilibrium theory(Chen; 2007) to find optimal strategy for player while keeping others strategy in mind, so that it will be win-win situation for every player involved. Authors objective is to find optimal price at which it is affordable for customer while remains profitable for cloud service providers. For comparison, authors have taken 3 categories first is serverless only, second is virtual machine only and last is combination of both serverless and virtual machines.

In paper (Liu and Niu; 2023) authors have explored auctioning of serverless function similar to auctioning of EC2 instances currently. An analytical model is for maximum profit and utilization of resources which takes both users and cloud service providers interest in account. It compares pricing scheme of all major cloud service providers. Authors have compared IaaS (Infrastructure as a Service) and FaaS on various metrics such as memory utilization and expense and came up with future function pricing.

2.5 Performance

One of best area to optimize cost is to do optimization in area of performance. In paper(Suo et al.; 2021), HotC framework is proposed by authors to mitigate cold start of serverless functions. HotC achieved this by wrapping all function in alpine container which is light weight in nature. It uses translation lookaside buffer and hot cache taking advantage of storage volume provided for each container. These container are kept in memory by HotC framework.

Performance is indirectly related to cost, it is been observed that in cases of performance optimization there is net positive in cost reduction in most cases. Authors in paper(Suo et al.; 2021) used containers for serving request between client and backend servers, where existing container with suitable configuration is searched if its not available new container with desired configuration is booted up. After executing request HotC framework will cleanup the environment for future request, this container is maintained in pool of containers. Even though there are number of containers are in active state in pool, the overhead for running these resources is very less. By using same container to server requests caching can be leverage to further improve resource utilization.

2.6 Big data

In serverless architecture it is very difficult to implement big data framework due to timeout limitation which in case of AWS lambda is 15 minutes. Serverless functions are stateless in nature which is another drawback as big data need to keep the context and metadata. Both stateless nature and timeout issue are two major drawbacks when performing big data task in serverless environment. In many studies, various researcher have addressed this problem by proposing different frameworks. One of such study (Cai et al.; 2024) have implemented SPSC framework to process data in real time. It breaks down data into small parts, which are coming from AWS S3 in real time via streaming, these small parts are called atoms. These atoms are then sent to AWS SQS (Simple Queue Service) which is then forwarded to serverless function for further processing. In the end, processed result is then saved into AWS Dynamo database. In paper (Werner et al.; 2018), using AWS step function as central component to perform matrix multiplication using serverless functions.

2.7 Related works findings

Research Papers	**Problem Areas**	**Potential Finding**	**Gaps**
Spillner (2020)	Inefficient memory allocation in serverless	Developed tools for tracing memory, profiling it, and auto tune it.	Limited to Docker based system.
Dittakavi (2021)	Resource and cost management in cloud computing environments	Optimization strategies for VM, workflow-based.	Lack real time adaptability and large computation power needed for model.
Ana Klimovic et al. (2018)	Challenges in storage for serverless analytics	Pocket framework for data intensive workload using FaaS	For specific type of workload only
Renata Borovica-Gajic et al. (2016)	Cost effective and low latency solution for cold data	Skipper framework is created for querying over cold storage device	Requires modification in database system
Alexander Fuerst and Prateek Sharma (2021)	Cold-start problems in FaaS	Tackled cold start problem by using caching layer and keep alive policy for serverless function	Only works on selected workload type
Chenggang Wu, Vikram Sreekanti, and Joseph M. Hellerstein (2020)	Caching solution in serverless paradigm.	HydroCache is proposed where cache layer is shared among serverless functions.	Increase in overhead in dynamic transaction.

Taras Lykhenko, Rafael Soares, and Luis Rodrigues (2020)	Built upon HydroCache for any type of workload	FaaSTTC ensure transactional causal consistency in cache layer for dynamic workload	Complex setup and integration with existing cloud service providers.
Chenggang Wu, Jose M. Faleiro, Yihan Lin, and Joseph M. Hellerstein (2019)	Key-value store in cloud	Implemented Key-Value store called Anna which is consistent and scalable	Due to coarse grained replication and manual deployment it is inefficient.
Kunal Mahajan et al. (2019)	Pricing models in serverless computing for customers and cloud service providers	Using FaaS, VM and FaaS -VM hybrid to achieve best outcome for both customers and cloud service providers	Lack of research in case of hybrid cloud
Fangming Liu and Yipei Niu (2023)	Pricing models in serverless computing for customers and cloud service providers	Developed analytical model for auctioning serverless functions.	Complexity in implementing dynamic pricing models
Sebastian Werner et al. (2018)	Matrix multiplication in big data using FaaS	Cost optimization of big data workload in FaaS	Limited to specific data type which need to considered beforehand.
Kun Suo et al. (2021)	Cold start problem in FaaS	HotC framework to solve cold latency with the help of containers	Unable to handle complex workloads and load balancing between containers is missing
Zinuo Cai et al. (2024)	Big data stream processing using serverless computing	Using AWS S3, SQS, Lambda and DynamoDB to process big data in real time.	SQS is limiting factor in terms of latency.

Table 2: Related works findings

3 Methodology

3.1 Research approach

In this research various AWS service are used to investigate and evaluate whether optimizations is effective. The primary objective is to reduce the cost in lambda function with the help of other AWS services while making sure there is no significant drop in performance and reliability.

Static configuration of lambda function can either lead to high bills due to excessive memory allocated or performance degradation if very little memory is allocated .First to predict the configuration of lambda function based on file metadata such as size a

dataset is derived from previous run manually. This dataset is then used as input in XGBoost with MultiOutputRegressor model to predict both memory and timeout. Next step is preprocessing which make sure that data quality is good .After training the model , it is deployed on Sagemaker endpoint using Sagemaker studio. Model make prediction based on file metadata and configuration of lambda function is updated in real time. Cloudwatch logs are continuously monitored and model is trained at regular interval.

In addition, for handling CPU intensive workload the code is extracted from main lambda function saved into EFS then loaded from their to another lambda function where it is finally executed. This separation of CPU intensive workload and I/O intensive workload ensures that less resource are used for I/O intensive workload thus saving cost.

3.2 Services

Following AWS services are used in this research.

3.2.1 AWS Kinesis

AWS Kinesis is selected for its high throughput data streaming for message passing where it ensure every message is processed exactly once and independently. AWS Kinesis offer ultra low latency which is capable of handling large amount of data streamed from various different sources. Due to mentioned feature it is used for create application which stream data in real time for analyzing thus allowing for real time data analysis instead of batch processing of data after some time.

3.2.2 AWS Lambda

AWS lambda is main component of this research, as objective of this research is to do cost optimization in serverless computing. In AWS lambda developers can write and execute code without worrying about managing infrastructure. In this user is only billed for duration of function execution. It allows deployed function to automatically scale up or down depending on demand. It is used to create event driven application.

3.2.3 AWS Cloudwatch

AWS Cloudwatch is monitoring service provides by AWS which is used to measure resources or service performance in form of metrics and logs. In Cloudwatch, there are alarms which can be used to notify through AWS SNS (Simple Notification Service), email and trigger various services such as lambda function to based on condition set in alarm. AWS Cloudwatch allow user to visualize data in form of graphs. It can be used for live trailing of logs from resources such as lambda function. Cloudwatch allows developer to automate operation based on logs, alarms, insights and network monitoring. Due to these feature it allows top down view of whole system.

3.2.4 AWS Simple Notification Service

AWS SNS is publisher and subscriber messaging service offered by AWS. It is used to send encrypted message across various subscriber at once, these subscribers can be other application or person. With the help of Dead letter queue failed message can be processed later along with analysis of why its failed. AWS SNS allow message retires if it failed

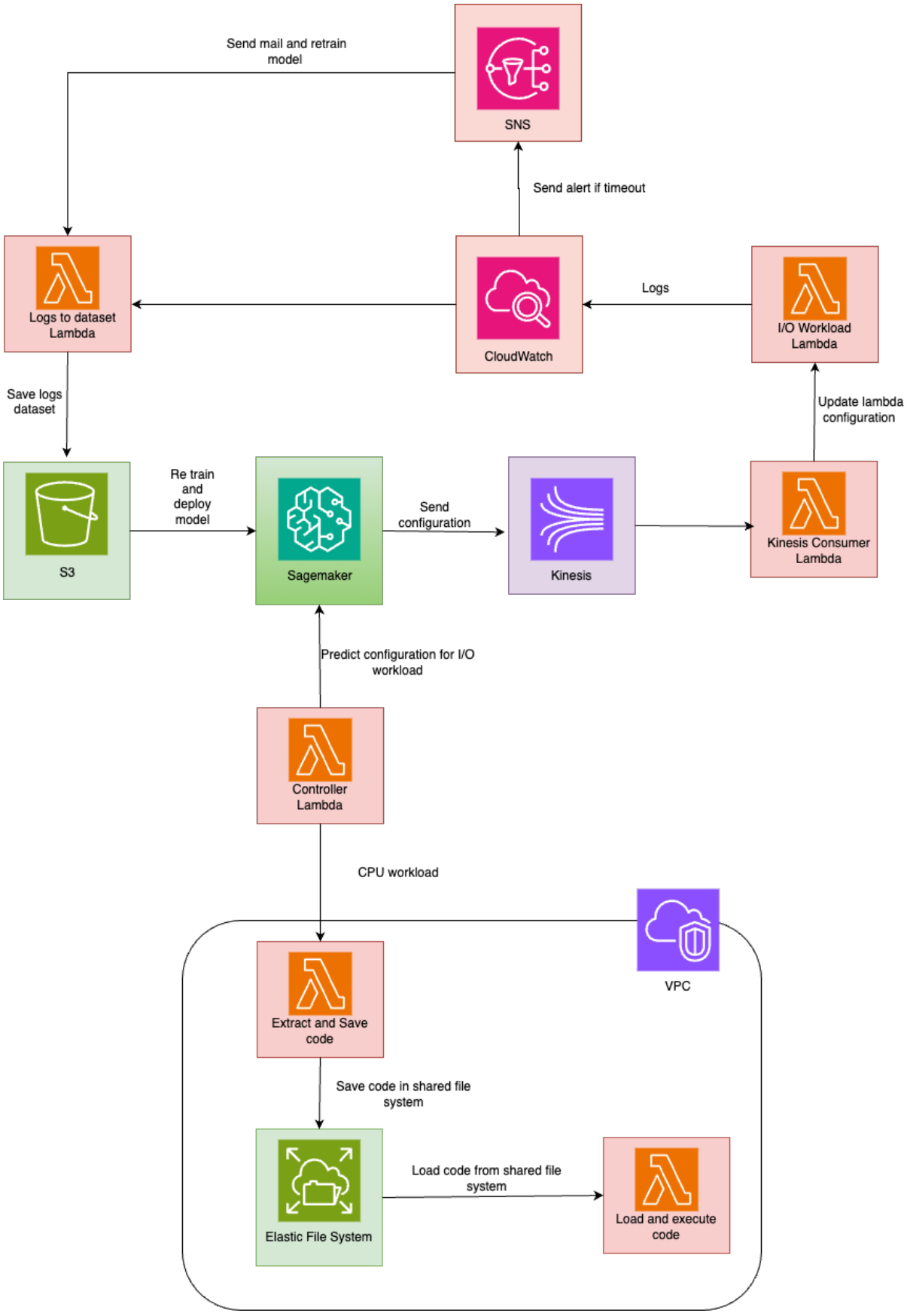

Figure 1: Architecture

once, saving it for future analysis or replaying it. SNS allow developer to set policies to discover personal identifiable information in accordance with regulation and compliance. SNS offer filter, using this only selected subscriber receive message based on filter criteria. It works well with event driven architecture such as AWS Lambda.

3.2.5 AWS S3

AWS S3 (Simple Storage Service) is known for durability, scalability and availability. It offer security in form of encryption and policies. S3 also allow to audit request made to S3 and along with it is complaint with programs such as GDPR, HIPAA, PCI etc. S3 also offer archival service at minimal cost where objects can be attached with life cycle policies for regulatory and compliance purpose. S3 allows minimum 3500 request per second for write operations and 5500 request per second for read operation. AWS S3 works on read after write policy which ensure data consistency, where latest version of object is provided for all request after write operation.

3.2.6 AWS Sagemaker

AWS Sagemaker is serviced offered by AWS for machine learning and AI (Artificial intelligence). It offer additional feature when compared to traditional machine learning tools where everything is at one place such as model building, training and deployment. It incorporates other service such as AWS Bedrock, AWS Athena, AWS Glue etc. It offer various applications and IDE (Integrated Development Environment) such as Notebooks, RStudio, Profiler etc. It allows to monitor model performance, management of model version etc.

3.2.7 AWS EFS

AWS EFS (Elastic File System) is storage service offered by AWS which is fully elastic in nature and built on serverless paradigm. It provides data throughput of gigabytes per second and allow to store petabytes of data. It gives 99.999999999 percent durability and offer 99.9999 availability. Due to it serverless nature it also uses pay as you go model. AWS EFS is fully managed service. It also offer three tier of storage classes depending on frequency of access of data which are standard, archive and infrequent access. It offer lifecycle management of data , where if data is not accessed recently it is moved to lower tier of EFS i.e. EFS Infrequent Access and EFS Archive. It provide other feature such as replication of data to another file system, backup service, security with the help of VPC and encryption of data at rest and in flight data.

3.2.8 AWS VPC

AWS VPC (Virtual Private Cloud) is a service where various different resources are contained in environment and separated logically. Due to its separation it provides improve security and control over environment by allowing advance traffic management.

4 Design Specification

In figure Figure 1 the architecture of framework is shown.The following are design specification for optimizing cost of complex workload, especially for I/O part. It consist of

AWS services,machine learning model and feedback loop. At core AWS lambda as orchestrator where it performs three task, first is using machine learning to make configuration (memory and timeout) prediction of lambda function, second is periodic learning of machine learning model through logs saved in AWS S3 collected from AWS Cloudwatch and lastly separation of CPU and I/O intensive workloads, where CPU intensive workload is extracted and saved into EFS then load from EFS and execute it thus decoupling the workload allows better scalability and efficiency.

AWS Cloudwatch is used for gathering metrics such as billed duration, maximum memory usage, initialization duration and errors. Whenever timeout happens in lambda function , an automated trigger will notify about failure via AWS SNS and this incident will be added in dataset to further enhance machine learning model. This feedback loop ensure that machine learning model is trained with new data at regular interval for learning new workload patterns thus improving machine learning model accuracy.

AWS Sagemaker studio is used to build,train and deploy machine learning model to predict optimal configuration of lambda function for I/O intensive workloads. It takes file size and latency as input and predict memory and timeout parameters. Using AWS Sagemaker, model is deployed at endpoint so that lambda function can query it for real-time prediction. AWS S3 is used to store dataset and models artifact. By combining machine learning, dynamic execution of workload and feedback loop ensures that system is cost effective and scalable.

5 Implementation

The implementation of Cost optimization for I/O intensive workload in FaaS has been described in this section.

5.1 Dataset Preparation

The first step is dataset preparation, which involves collecting logs from lambda function using AWS Cloudwatch. Metrics such as file size, latency, memory usage, billed duration and error are collected from log as shown in Table 3. For this research, dataset was created by running lambda function which is handling I/O intensive workload by creating dummy text files with small case characters. These files are of different sizes such as 50 MB, 100 MB, 150 MB, 200 MB, 250 MB etc. These files are processed against different memory configuration ranging from 128 MB - 2034 MB. After execution of lambda function there are some cases where lambda function timed out which are recorded in error column. Latency column tell about latency between AWS S3 and lambda function. Column max_memory_used record the maximum amount of memory used by lambda function during execution. Column billed_duration is used to calculate cost column for each lambda request. The total cost of lambda is calculated using formula 1. Feature scaling is done to ensure consistency and to avoid bias in model training with the help of standardization and normalization.

$$\text{Total Cost} = \text{billed\_duration} \times \text{max\_memory\_used} \times 0.0000166667 + 0.0000002 \quad (1)$$

where:

- **billed_duration**: Execution time in seconds (rounded up).

- **max_memory_used**: Memory allocated during execution (in GB).

Table 3: Logs DataFrame

Column	Dtype
request_id	object
function_name	object
memory_size	int64
duration	float64
billed_duration	int64
max_memory_used	int64
error	bool
cost_per_request	float64
latency_ms	float64
file_size_mb	int64
init_duration_ms	float64

5.2 Model Training and Deployment

Dataset is pulled from AWS S3 and it is trained using XGBoost because it can capture non-linear relation between input and output feature as show in Table 4. MultiOutputRegressor is used as wrapper around XGBoost algorithm for predicting multiple value. The data is then split into 80-20 ratio for training and testing. After training is completed model, standard scaler and min max scaler are serialized using joblib library, then these are packaged into tar.gz archive and uploaded to AWS S3. Later, this archive is pulled from AWS S3 for deploying to Sagemaker endpoint.

Table 4: Features

Input Features	Output Targets
file_size_mb	max_memory_used
latency	billed_duration

5.3 Controller Lambda

Controller lambda function is main lambda function which is main orchestrator, where CPU workload and I/O workload is identified. These workload are then forwarded to different lambda function each specializes in handling respective workload. Controller also calculate the I/O lambda configuration using AWS Sagemaker endpoint which is mentioned in Table 4. Controller then forward the predicted configuration to AWS Kinesis stream. For CPU intensive workload, a payload is sent to lambda function which contains function name which need to be executed along with updating configuration same.

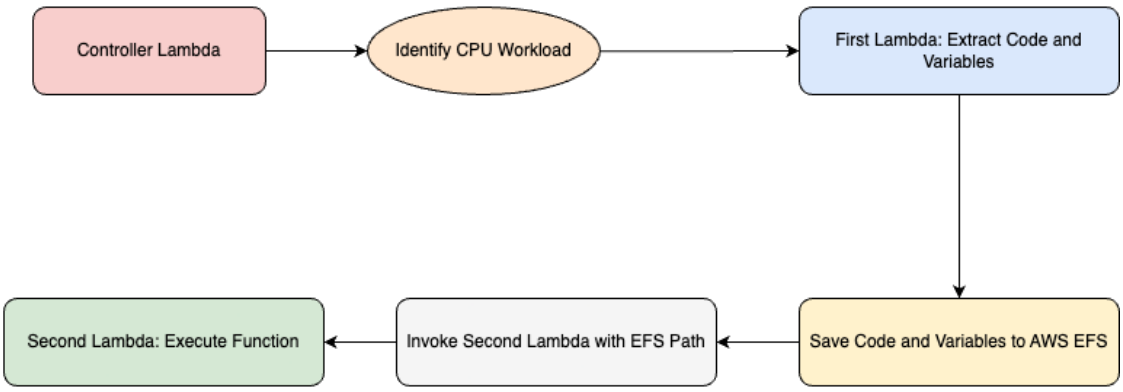

Figure 2: CPU intensive Workload

5.4 CPU intensive Workload

This workload is carried out by two lambda functions which have shared AWS EFS for sharing code and variables. First lambda extract the workload code from controller with the help of function name received from controller lambda in payload. It will extract the code and save it to AWS EFS in python executable file along with json which will contain variable information. At last, the First lambda will invoke second lambda by sending AWS EFS file path of extract code and json in payload.

Second lambda will extract function and variable json file path from AWS EFS. Using Python importlib and used it to create module from specification and execute it. Workflow is shown in Figure 2.

5.5 Kinesis consumer lambda

Consumer lambda is used to consume records from AWS Kinesis which contains the configuration of I/O intensive workload lambda discussed in 5.6. It ensure that each record is processed only once based on sequence_number. Once the configuration of I/O intensive workload lambda is updated successfully, it will invoke it.

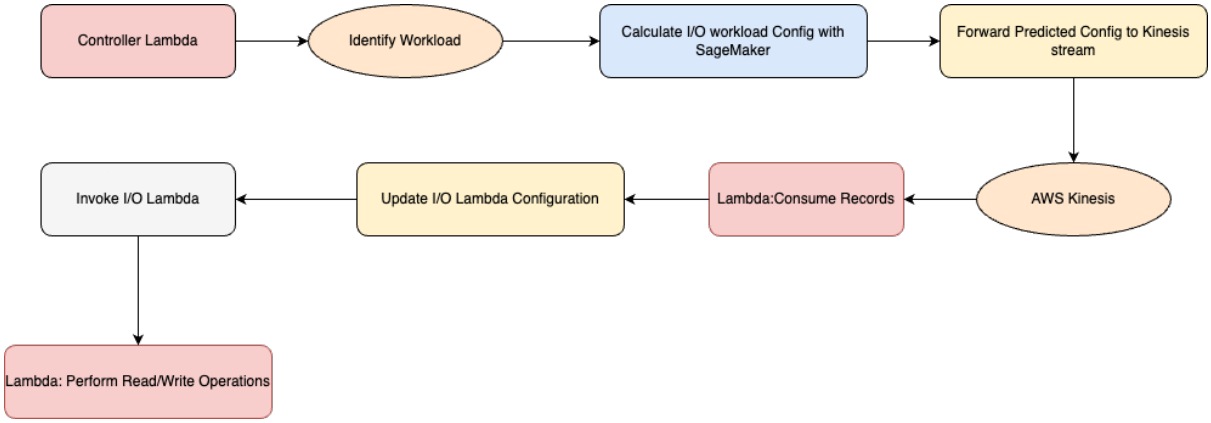

Figure 3: I/O intensive workload

5.6 I/O intensive Workload

I/O intensive Workload lambda will then perform read and write operation on disk by transforming data. Finally, logs will all metrics will be printed in AWS Cloudwatch for further training the model. The workflow is shown in Figure 3

5.7 Feedback

Model needs to be trained regularly on updated dataset, if predicted configuration causes timeout issue in I/O intensive Workload lambda function it need to be taken in account for training the model next time to avoid making same mistake. Feedback mechanism is implemented using AWS Cloudwatch. An alert is setup where it will go through logs of I/O intensive Workload function and check if there is any timeout. If there is timeout it will send alert using SNS which will send email and lambda function will be triggered

which will automatically update the AWS S3 dataset file with latest data. Using AWS Sagemaker to retrain model by fetching latest dataset from AWS S3 and deploy endpoint as shown in Figure 4.

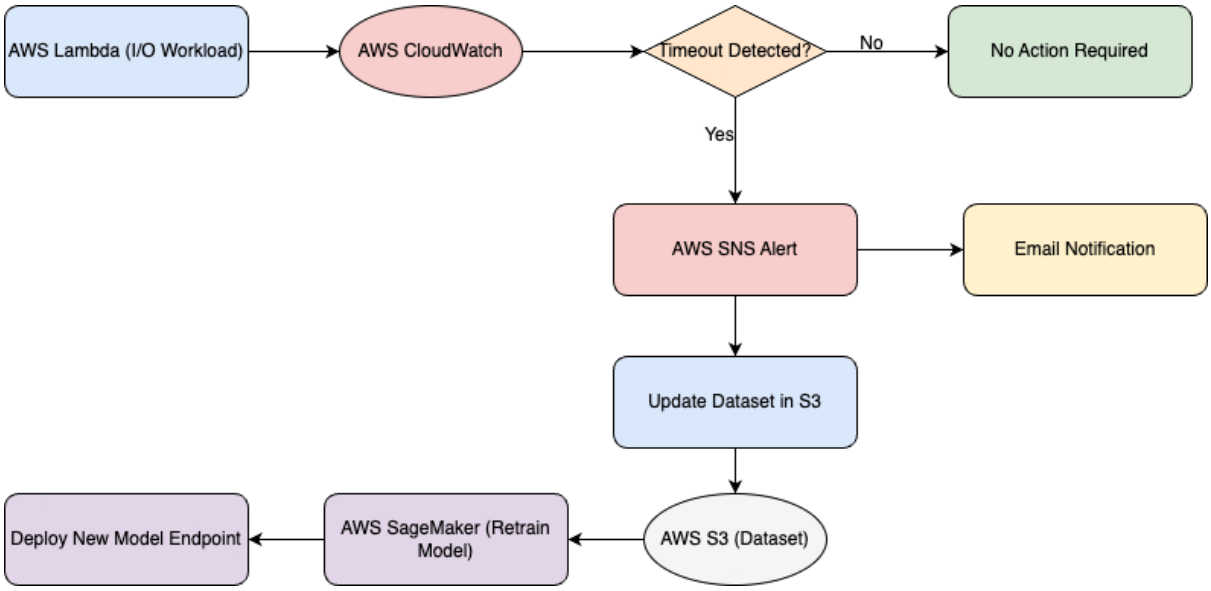

Figure 4: Feedback Loop

6 Evaluation

This section list the important findings of research. It evaluate the effectiveness of framework in coat saving for FaaS.

6.1 Cost Optimization

In this there is a complex workload which is executed both on vanilla serverless function and on framework where it uses divide and conquer method. Framework divides the workload in CPU and I/O intensive. Following was the result shown in Table 5. Billed duration and Max memory usage will be considered and cost will be calculated formula 1.

Table 5: Cost comparison per million request

File size	Vanilla CPU+I/O	Framework CPU	Framework I/O	Saving%
50 MB	$477.19	$384.62	$12.96	16.69%
100 MB	$641.57	$392.48	$42.24	32.24%
150 MB	$691.22	$392.20	$81.16	31.65%
200 MB	$776.48	$404.37	$123.53	32.04%

As evident from Table 5 framework saves 16%-32% because of I/O workload memory is dynamically adjusted as its not memory intensive. Figure 5 shows few observation which are:

- Both Vanilla and Framework shows linear growth, though vanilla has steeper curve thus indicating higher cost per MB.
- As the file size increase, saving% also increase till 32% where its constant.

Framework outperform Vanilla by large margin making it ideal for complex workload.

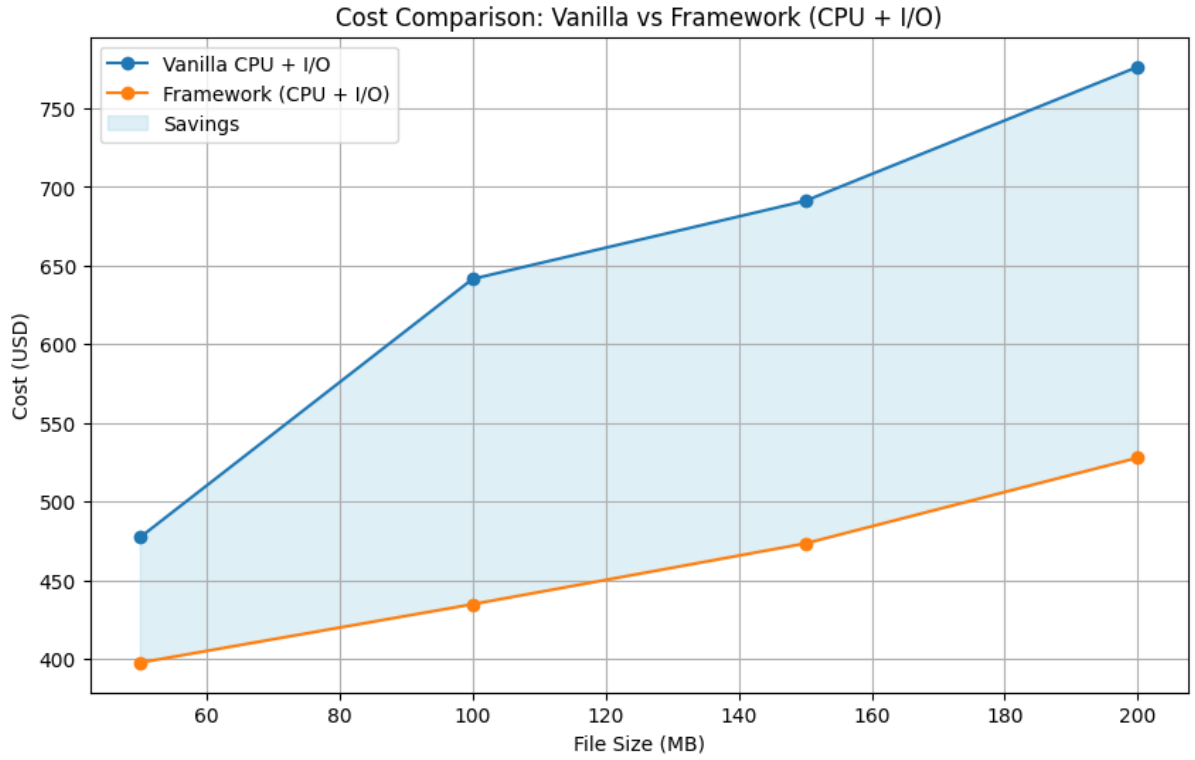

Figure 5: Vanilla vs Framework

6.2 Prediction Accuracy

The accuracy of XGBoost with MultiOutputRegressor is shown in Figure 6. The high MSE scores tell that there are edge cases , which needs to be removed as these outliers are affecting MSE score. The R^2 score is 0.80 which tells model is able to capture relationship between input and output, thus model's prediction is reliable. Although R^2 is high but there is still work need to done for lowering MSE score in handling outlier scenarios.

6.3 System Scalability

In Figure7, Framework was invoked 10 times which is limit of concurrent execution in AWS, blue line indicate concurrent invocation and orange line indicate errors. It is shown that framework is able to handle 10 concurrent invocation , though further testing is need for bigger payload.

```
  data["init_duration_ms"].fillna(0, inplace=True)
Mean Squared Error (MSE): 1536652.7774935954
R^2 Score: 0.8042429685592651
------!
```

Figure 6: Model Accuracy

Figure 7: System Scalability

6.4 Discussion

The experiment confirms that the proposed framework reduces the cost for complex workload in FaaS, as in traditional serverless function resource utilization is very less in I/O intensive workload. In FaaS configuration is fixed before execution of function, thus sometime it leads to over allocation of resource which lead to wastage or under allocation which will cause degradation in serverless performance. The proposed framework divides the workloads into I/O and CPU which is allocated to different lambda function. Using machine learning to configure the lambda function dynamically based on input parameters while Cloudwatch is used for monitoring framework reliability. There is up to 32% cost saving when lambda function configuration is predicated by machine learning model by taking file size and latency in account for adjusting memory and timeout. The machine learning model was able to accurately predict the configuration with high R^2 but high MSE score indicates there is noise in data and outliers are present. From scalability perspective there is only 10 lambda function can be invoked at given time thus in research there was limited scalability test.

Feedback loop is one of most important feature which run whenever lambda function timeout. Loop will save the recent logs in S3 , from here model will train on latest data from S3 thus allowing to further improve the model. One of the biggest limitation is lack of variety of I/O operations in research , there is overhead of training machine learning model every time and cold start problem where its takes some time for function to boot up. While the framework produces the desired result, addressing the limitation would vastly improved the framework which is applicable to different type of workloads.

7 Conclusion and Future Work

This research focuses on cost optimization of I/O intensive workload in FaaS, by implementing dynamic workload classification, dynamic configuration of lambda function by machine learning and feedback loop for continuously train machine learning to further improve it.

In the Section 6 framework was evaluated for cost optimization , prediction accuracy

and system scalability. Though there was 16% -32% saving, it is for very specific type of I/O workload and CPU workload which is common across all experiment which lead to higher accuracy from machine learning model. There is need to include other metadata of files such as file type and model need to be train with other features. The research proved two point , first there is need of dynamic resource management in FaaS as the framework has produced positive results. Second is that framework can be implemented on other cloud service providers.

Limitations of research are Machine learning model needs continuous training, model need to be able to handle new type of files and cold start problem of serverless function .

In summary, research is able to achieve the object of cost optimization in serverless computing for I/O workload although with limitation to workload types.

For future works , there is need to include different combinations of I/O and CPU workloads. Scalability also need to be tested for example 500 invocation. Other metadata of files such as type need to be included as feature in machine learning model. Framework needs to consider cold start problem for further cost optimization.

References

Borovica-Gajić, R., Appuswamy, R. and Ailamaki, A. (2016). Cheap data analytics using cold storage devices, *Proceedings of the VLDB Endowment* **9**(12): 1029–1040.

Cai, Z., Chen, Z., Chen, X., Ma, R., Guan, H. and Buyya, R. (2024). Spsc: Stream processing framework atop serverless computing for industrial big data, *IEEE Transactions on Cybernetics* .

Chen, J. (2007). Nash equilibrium: How it works in game theory, examples, plus prisoner's dilemma. [Accessed 04-08-2024].
URL: *https://www.investopedia.com/terms/n/nash-equilibrium.asp*

Dittakavi, R. S. S. (2021). An extensive exploration of techniques for resource and cost management in contemporary cloud computing environments, *Applied Research in Artificial Intelligence and Cloud Computing* **4**(1): 45–61.

Fuerst, A. and Sharma, P. (2021). Faascache: keeping serverless computing alive with greedy-dual caching, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 386–400.

Klimovic, A., Wang, Y., Stuedi, P., Trivedi, A., Pfefferle, J. and Kozyrakis, C. (2018). Pocket: Elastic ephemeral storage for serverless analytics, *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 427–444.

Liu, F. and Niu, Y. (2023). Demystifying the cost of serverless computing: Towards a win-win deal, *IEEE Transactions on Parallel and Distributed Systems* .

Lykhenko, T., Soares, R. and Rodrigues, L. (2021). Faastcc: Efficient transactional causal consistency for serverless computing, *Proceedings of the 22nd International Middleware Conference*, pp. 159–171.

Mahajan, K., Figueiredo, D., Misra, V. and Rubenstein, D. (2019). Optimal pricing for serverless computing, *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, pp. 1–6.

Spillner, J. (2020). Resource management for cloud functions with memory tracing, profiling and autotuning, *Proceedings of the 2020 Sixth International Workshop on Serverless Computing*, pp. 13–18.

Suo, K., Son, J., Cheng, D., Chen, W. and Baidya, S. (2021). Tackling cold start of serverless applications by efficient and adaptive container runtime reusing, *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, pp. 433–443.

Werner, S., Kuhlenkamp, J., Klems, M., Müller, J. and Tai, S. (2018). Serverless big data processing using matrix multiplication as example, *2018 IEEE international conference on big data (Big Data)*, IEEE, pp. 358–365.

Wu, C., Faleiro, J. M., Lin, Y. and Hellerstein, J. M. (2019). Anna: A kvs for any scale, *IEEE Transactions on Knowledge and Data Engineering* **33**(2): 344–358.

Wu, C., Sreekanti, V. and Hellerstein, J. M. (2020). Transactional causal consistency for serverless computing, *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 83–97.