National College of Ireland

A Serverless Architecture to optimize support for Stateful Applications

MSc Research Project
Cloud Computing

Gurpreet Kaur
Student ID: X18129293

School of Computing
National College of Ireland

Supervisor: Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Gurpreet Kaur
Student ID:	X18129293
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Muhammad Iqbal
Submission Due Date:	23/04/2020
Project Title:	A Serverless Architecture to optimize support for Stateful Applications
Word Count:	5024
Page Count:	16

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.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	26th May 2020

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):	

A Serverless Architecture to optimize support for Stateful Applications

Gurpreet Kaur
X18129293

Abstract

The serverless computing model has been rapidly adopted as it gives users the freedom of uploading application code without having to manage the allocation of resources. The cloud provider manages the deployment, scaling and execution of resources dynamically. This cloud paradigm is also known as Function as a Service (FaaS). The stateless nature of serverless functions allows them to scale elastically on demand. However, deploying a complex stateful application on serverless can cause workflows which requires state information to be shared with other functions. Serverless functions do not well support the execution of stateful applications as it cannot manage and coordinate the state of multiple functions. This research focuses on optimizing the coordination among functions to provide support for stateful applications. Therefore, we proposed an approach for designing stateful applications on a serverless platform by introducing a coordinator function and a monitoring function which will run on virtual machines (VM). The new functionalities are included in the design to provide a mechanism for the serverless functions to be able to report their execution state by utilising monitoring function and help coordinator function to effectively manage the application workflow.

1 Introduction

Cloud Computing has emerged as a building block in the world of information and technology for modern enterprises. The global adoption of cloud technology is the result of its wide range of benefits, such as deployment at scale and elasticity. In comparison to traditional infrastructure, cloud resources are easier to avail, access, manage and discard unused components with notably less cost. Similarly, the serverless model of cloud provides another layer of abstraction by relieving users from infrastructure management and focusing only on the development of application. These services allow application software to be deployed in the form of functions which are executed in response to an event such as a new file being uploaded, arrival of a message in the queue or direct function calls via HTTP or cloud API calls.[1]

Serverless functions are best suited for small event driven applications which can execute and scale independently. Cloud providers offering FaaS model, has some limitations such as, maximum size of each function, maximum memory allocation for executing the function and function execution time. Often, there are applications which are required to perform high frequency database transactions by multiple nodes, such functions are

considered to be heavy workload applications. The limitations in FaaS model make it challenging to use the platform for deploying heavy workload applications. To understand the limitations of serverless for running complex workflows, a study has been performed and some of the key issues are highlighted in this paper. For example, deploying a complex data analytics pipeline on serverless would require two or more functions interacting through local calls resulting in a communication overhead due to network latency, as the functions might be running on different hosts.[2] Most of the applications have state sharing needs, with serverless functions it is required to use object storage services for state sharing. This adds further database access latency in the serverless model of application. Additionally, the performance of the application deployed on serverless cannot be predicted due to cold start latency. Likewise, scaling of functions is often hard to predict. Among a number of functions, a given workflow might not require all the functions to scale up during peak workloads.[3]

In a distributed application, functions are required to run in coordination with other functions. For example, in a trip booking application, there could be different functions for accommodation booking and transport booking. Additionally, there might be different functions for booking in different geographical areas. Often, these functions are required to communicate or execute based on the response of its peer function. For example, function A might utilise output generated from function B. Let's consider, if function A failed to execute or function B was unable to receive output generated by function A. These kind of issues are hard to recover in the current state of serverless platform due to poor coordination between functions. Therefore, designing a serverless architecture for a complex stateful application is difficult. The given research was conducted with the aim of finding answer to the following question:

How to optimize support for deploying stateful applications on a serverless platform?

In this research, a novel serverless application design model has been proposed for implementing stateful applications. The proposed model adds two additional components to the design namely, coordinator function and monitoring function. The coordinator function provides orchestration capabilities to manage workflow of serverless functions. Where as, the monitoring function evaluates the execution of the serverless function. The monitoring function can also trigger an app recovery logic based on the status of the serverless function. The aim of the research is to improve efficiency of the stateful application running on a serverless platform. In this paper, we consider a scenario of failure of one or more functional steps in the application execution workflow. We evaluate how the proposed model can help serverless functions in designing efficient application workflows on serverless. The new model utilises both serverless platform and virtual machine based components.

2 Related Work

In this subsection, the related work has been elaborated and its applicability has been discussed to the above problem statement.

Xu et al. [4] attempted to reduce the cold start latency issue in serverless platform by predicting the serverless function invoking time and warming up the functions. The research also focussed on reducing the resource utilisation while trying to reduce the cold start latency. Two different techniques were used called as Adaptive Warm-up strategy (AWU) and Adaptive container pool scaling (ACPS). In the first technique, time series of function invoking was collected periodically. Based on this, the functions were started before the predicted time to minimize latency. The second technique was more like a backup for the first one. In case the predicted value is not close to the actual value, ACPS would help as it initializes a container pool where the containers are pre-launched and configured. Lin and Glikson [5] also proposed a similar strategy by maintaining a pool of function instances.

In another research by Deshpande Umesh [6], a scheduling approach is presented to improve performance of containerized stateful applications during load spikes. It reduces the evictions of application containers and provides an opportunity to use the spare resources. The scheduling strategy is based on the nature of two type of applications, stateful and stateless. The architecture is called as caravel and consists of components such as master for cluster managements tasks, worker to host the containers, configDB to store application configuration details given by the user while deploying the application and MetricsDB to store the cluster resource usage. The aforementioned databases are used by the eviction process running on each worker node.

Lee et al. [7] presented an evaluation on throughput, network bandwidth and performance of compute while running concurrent invocations. The result shows that serverless platform can perform better as compared to traditional VM platform if a task is well distributed and is small enough to execute under the function limitations. The cost optimization is high as the function can scale down to zero cost as compared to traditional VM. Winzinger and Wirtz [8] proposed a model for developing complex application workflow on a serverless architecture. The proposed model develops a dependency graph with each component of application, showing the workflow and data flow. Then, it shows mapping of the resources with each other and helps find the potential race condition in a workflow. This form of graph based analysis claimed to help designing stateful applications on a serverless platform and help in detection of several hot spots.

Palade et al. [9] evaluated the use of IoT edge devices to implement open-source serverless frameworks, to leverage the computing power of local network devices. The deployment model consisted of Internet of Things (IoT) Device layer and Edge Computing Layer. The frameworks chosen for evaluation were Kubeless, OpenFaaS, Knative and Apache Open Whisk. As a result, it was observed that Kubeless was more efficient in response time and throughput. Serverless Programming comes with a number of restrictions on application design, Perez et al. [10] introduces the concept of container aware architecture for serverless to address some of the limitations of serverless. The architecture consists of two components majorly, a Serverless Container-aware Architecture

(SCAR) client and a supervisor. The former is responsible for taking input from user, defining lambda functions and creating the deployment package. Whereas, the supervisor takes care of retrieving the docker image from Hub, creating the containers and managing input and output of staging data. The given design provides means to run containers on AWS Lambda and also manages function lifecycle. However, it only supports providers which have python serverless functions available.

In recent times, researchers have also explored the idea of running deep learning models on serverless platform. Ishakian et al. [11] evaluated the efficiency of serverless environments on executing large neural network models. For the experiment, they used popular image recognition models SqueezeNet, ResNet and ResNeXt-50. The tests focused on cold start and warm start effects. AWS lambda was used to create functions and the metrics evaluated were, response time, prediction time and cost. The results suggested that with cold starts, the latency is significantly higher as compared to warm starts. It also shows that the limitation of resources in serverless can be a blocker for running inference applications. Zhang et al. [12], evaluated the implementation of video processing apps on serverless. They used two parameters, resources configurations and function implementation approach to test function execution time and cost. The results show that the function execution time does not decrease with increase in memory size. The implementation scheme such as using other services (AWS Rekognition) can provide benefit on running video processing workloads.

Garcia et al. [13] describes the important trade-offs between serverless programming models and data analytics. The three trade-offs namely, isolation, dis-aggregation and simple scheduling shows difference between serverful and serverless architectures. This paper defines the term "ServerMix" and highlights its use in real world applications. ServerMix is nothing but the midpoint of serverful and serverless design models. An effective servermix design should focus on transparent resource provisioning and should prioritize cost-performance ratio.

Another factor that complicates the deployment of stateful applications on serverless is that it is difficult to decide which application component will benefit from the serverless architecture. Given that, Yussupov et al. [1] have tried to solve this problem by annotating functions for executing rare workloads by extracting from the source code of app and deploying them as FaaS. This does not change the architecture of the original application. This way complex applications can utilize serverless platform without the need of re-engineering the entire application.

Damkevala et al. [14] describes a technique for performing serverless data analytics using Watson machine learning API on IBM Cloud services. The idea here is to enable developers to use advanced analytics services on a serverless platform, without worrying about the hardware on which the analytics model would run. The research focused on user behaviour analysis by using a modified version of Mahalanobis Distance algorithm for correlation data synthesis. It further used a classifier model to refine the correlation data. The result states that the Watson machine learning API was the most efficient choice for running machine learning based algorithm variants without the hardware maintenance overhead. In [2], Ghosh et al. compared serverless and VM based system for implementing

complex services which involves databases, file stores and multiple functions. The results showed that the latency in serverless system was high even if the cold starts are ignored. The research also proposes an in-memory cache by using lambda container's persistent memory during multiple consecutive requests with in short intervals.

Serverless paradigm has been tested with a number of applications. One such implementation of evaluating user network profiles was proposed by Parres et al. [15], where a cyber security technique of anomaly detection based on TopK rankings of network user profiles was implemented using AWS Lambda, DynamoDB, Simple Queue Service (SQS) and S3. The serverless architecture builds user network profiles and evaluates local network traffic, resulting in cost and time reduction, scalability and improved availability without additional resource configuration. Another application by Luckow and Jha [16], introduces pilot-streaming, which gives an abstraction for managing resources of High-performance Computing (HPC), cloud and serverless by allocating resource containers that are independent of the application workload. Their work also demonstrates the use of StreamInsight, for performing experiments to evaluate the performance of streaming applications, infrastructure and scalability on serverless.

Jackson and Clynch [17], presents the impact of language run-time on the cost and performance of serverless function execution. The research also implements a new design of serverless performance testing framework. The testing framework consists of some common components such as, recording data from serverless platform, API for calculating performance and cost. The other components is specific to provider, AWS or Azure. The evaluation of metrics show that python is the most efficient for running functions on AWS. Whereas, C# and .NET work best with Azure Functions. Another language based research on serverless was conducted by Moczurad and Malawski [18], by leveraging the use of Luna, a visual-textual programming language. The paper aims at integrating luna with serverless model and exploring its benefits over other languages. It is based on the concept of extending the standard library and utilizing the language features to build API for serverless functions.

In the cost optimization side, Mahajan et al. [19] demonstrated an analysis of cost benefits of serverless for the cloud providers and end users. The aim of this paper was to identify and characterize three optimal working solutions for both the parties, given that they utilize either serverless model only, VM only, or both in a hybrid environment. One of the insights provided was that depending on system parameters the optimal solution may vary. The proposed framework can help understand the trade-offs and challenges of using a hybrid system.

Apostolopoulos et al. [20] introduced a flexible resource sharing method to allocate the users computing task among virtual machines (VMs) and serverless computing (SC's) in a social cloud computing environment. The dependent factors involved in deciding resource sharing methods are user satisfaction, cost and interaction among users. Considering the uncertainty of the environment, user behaviour is characterized by using Prospect Theory and the theory of tragedy of commons.

Implementing blockchain on a serverless cloud was analysed by Kaplunovich et al. [21] in terms of scalability. The paper describes the result of automating the configuration

steps of Hyperledger Fabric Blockchain on AWS and benchmarking the scalability of the system. For the automation, the University of California Riverside (UCR) time series dataset was used. AWS lambda was used to upload data in parts in to multiple instances and triggered the load by SQS messaging. Kritikos and Skrzypek [22], worked on simulation support with serverless. Simulation system allows execution of simulation of any application configuration. In the research, a novel simulation as a service architecture was proposed using serverless computing. The aim of the research was to optimize the cost and time it takes to run the simulation. The implementation method uses parallel invocation of functions and the case study was a business application offering artificial intelligence investments.

3 Methodology

Serverless computing has emerged as a simple cloud based solution where the cloud provider manages the infrastructure allocation and its scalability whereas the end user is only responsible for writing the application code. The applications are normally categorized as either stateless or stateful. The two types are different in terms of how the components scale and interact with other dependent components. In case of stateless applications, a function can independently run on its own as soon as it is triggered and can be horizontally scaled. However, stateful applications are complex as it consists of more than one functions and other dependent components such as storage. This brings complexity in deploying such applications on a serverless platform.

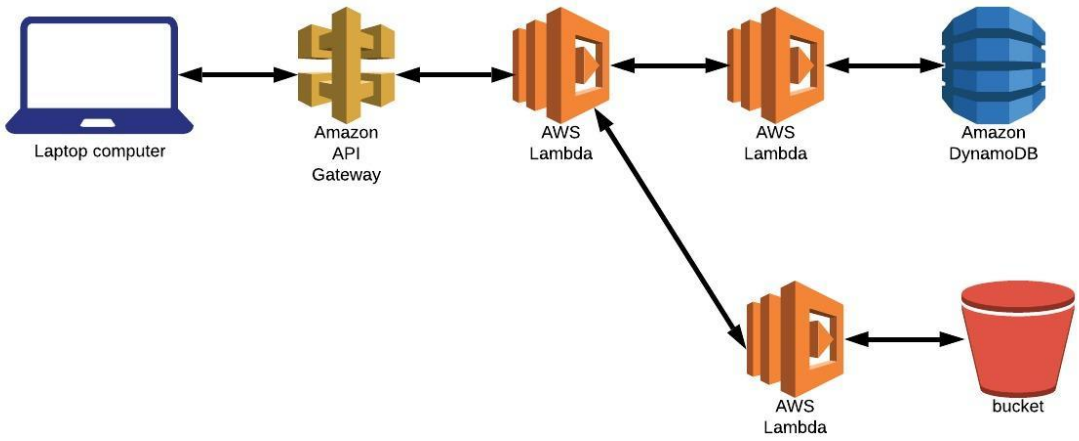

Figure 1: Traditional Architecture for Stateful Applications on Serverless

Stateful applications are often deployed on FaaS platform by distributing the application logic into multiple functions or components. The figure 1 shows the traditional approach of deploying stateful application on a serverless platform. The serverless function is either invoked by an event or directly through invoke function call. One function's output is generally used to invoke another function. An external storage is used to perform any read-write operation. Although, the application is able to make use of serverless

function to run the compute logic, there is a significant performance degradation in terms of latency, scalability and efficiency. For example, lambda function execution time is within the acceptable range. However, the latency is caused due to the time it takes to access the database [14].

The reason behind the aforementioned problem of latency is that the function and the database are not hosted on the same physical device. Similarly, the issue with scalability occurs as it is difficult to decide which component of the application needs to be scaled. As discussed before, most of the stateful application on serverless platform are deployed by dividing the function logic into multiple different chain of serverless functions. This type of system design incurs dependency of one function on another and requires effective coordination among the functions.

Implementing a multi-function application on serverless platform segregates the compute and storage in two different components, making it difficult to manage the workflow in the environment [3]. The existing cloud services does not offer a way for serverless functions to coordinate their state amongst each other. While, there are some available services which help in orchestration or management of function's workflow, such as AWS Step Functions [23], these services do not support effective communication for complex application logic. For example, on deploying a Convolutional Neural Network (CNN) based machine learning model on a FaaS platform, it requires computation for each CNN layer. Each layer consists of n functions and there could be up to k layers. In the given scenario, if one or more function's fail, the application will either result into an incorrect outcome or may fail completely. We can also consider an example of database sensitive application where two functions are performing read/write transactions on the same database. It is crucial that the read-function is always reading the updated data from the storage. In this scenario, if the write-function fails to update the database at a given point of time, there is no mechanism for the read-function to know that. This will result in an inconsistent state of the database and could give incorrect output to the user.

Thus, in order to improve coordination among functions in case of a function failure and improve performance of the application on serverless platform, this research proposes a solution.The solution focuses on monitoring and detecting such failure and improve response of the application to recover from the failure.

We introduce two new components in the design of serverless systems for stateful applications, namely, Coordination and Monitoring Function. The coordination function is responsible to maintain the workflow of the application. When an application starts, c-node should invoke the initial function and manage the execution of any other function that is required by the application. It can also be utilised to return results back to the user. On the other hand, monitoring function will read the metrics and logs of the functions to verify if the execution was successful. On the basis of the execution result, a function called as recovery logic will be triggered. The recovery logic can be considered as a disaster management trigger and it should define steps that must be taken to avoid any invalid outcome due to the failure of a function.

4 Design Specification

In this section, a detailed description of the proposed model is provided.

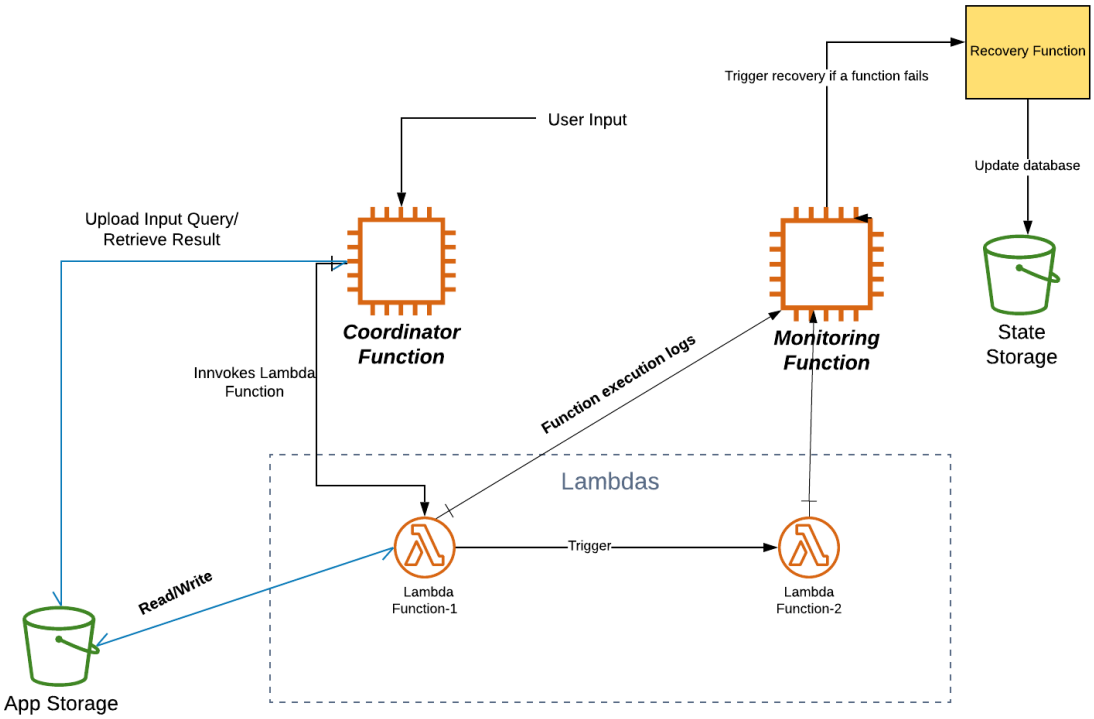

Figure 2: Proposed Architecture for Stateful Applications on Serverless

The figure 2 shows the architecture of the proposed model. The coordinator function (c-node) acts as an orchestration mechanism for the serverless functions. The c-node should be utilised as a component to manage all the application specific serverless functions and the associated database interactions. App-storage should be used to store application specific data. Based on the requirement of the application, c-node can also process the input before writing to the storage. While deciding which storage service to be used, one should consider factors like the type of data that is being processed and the latency involved in accessing the storage service from the serverless function. The c-node adds user defined input to the app-storage bucket. The event of putting input in the storage triggers the first serverless function. In this paper, we consider a serverless application model where a chain of functions are triggered one after the other, the output of the first function is utilised to trigger the second function and so on.

The monitoring function (m-node), should be configured to read and evaluate metrics and logs after each function execution to verify if the function was executed successfully or resulted in to an error. The m-node can run a number of tests to identify the state of the function. For example, create a test to check if the function was able to successfully perform read operation from a database. The m-node should be designed in such a way that, if it detects that a particular function or operation was failed, it will call a backup function called as app recovery logic.

The app recovery logic is a function executed by m-node as a response to the failure. The goal of the recovery logic is to clean up background operations which are impacted by the failure of the function. This helps in achieving the correct state of the system. For each application, the app recovery logic can perform different operation. For example, in the aforementioned case of a database sensitive application, app-recovery logic can notify the read-function that the write-function has failed and the database is not up to date.

As discussed in previous section, on a serverless platform a shared storage service is utilised by functions to share results of operations performed. The state storage is used for storing each function's execution result. It stores temporary data. The state storage plays an important role in this model because whenever there is a failure of an intermediate function, the recovery logic tries to update the state information. This operation is crucial to the workflow of the application as the state information is used to feed the running functions.

5 Implementation

The proposed design has been implemented on the AWS cloud platform. This section briefly describes the method and cloud services used for implementation. As each application design would be different on serverless, in the current scope of research we consider an online booking system application. Online booking system app generally has three phases which includes selecting item, confirming order and making payment. Such applications can run on serverless by splitting one task into multiple functions, where each function may require output from previously executed functions. In the given workflow, the output of confirming an item is consumed as input while processing the payment. The current services offered by cloud provider does not have a mechanism to allow function-to-function coordination. The given implementation plan helps in improving the overall response of the application in case the workflow deviates from the desired path.

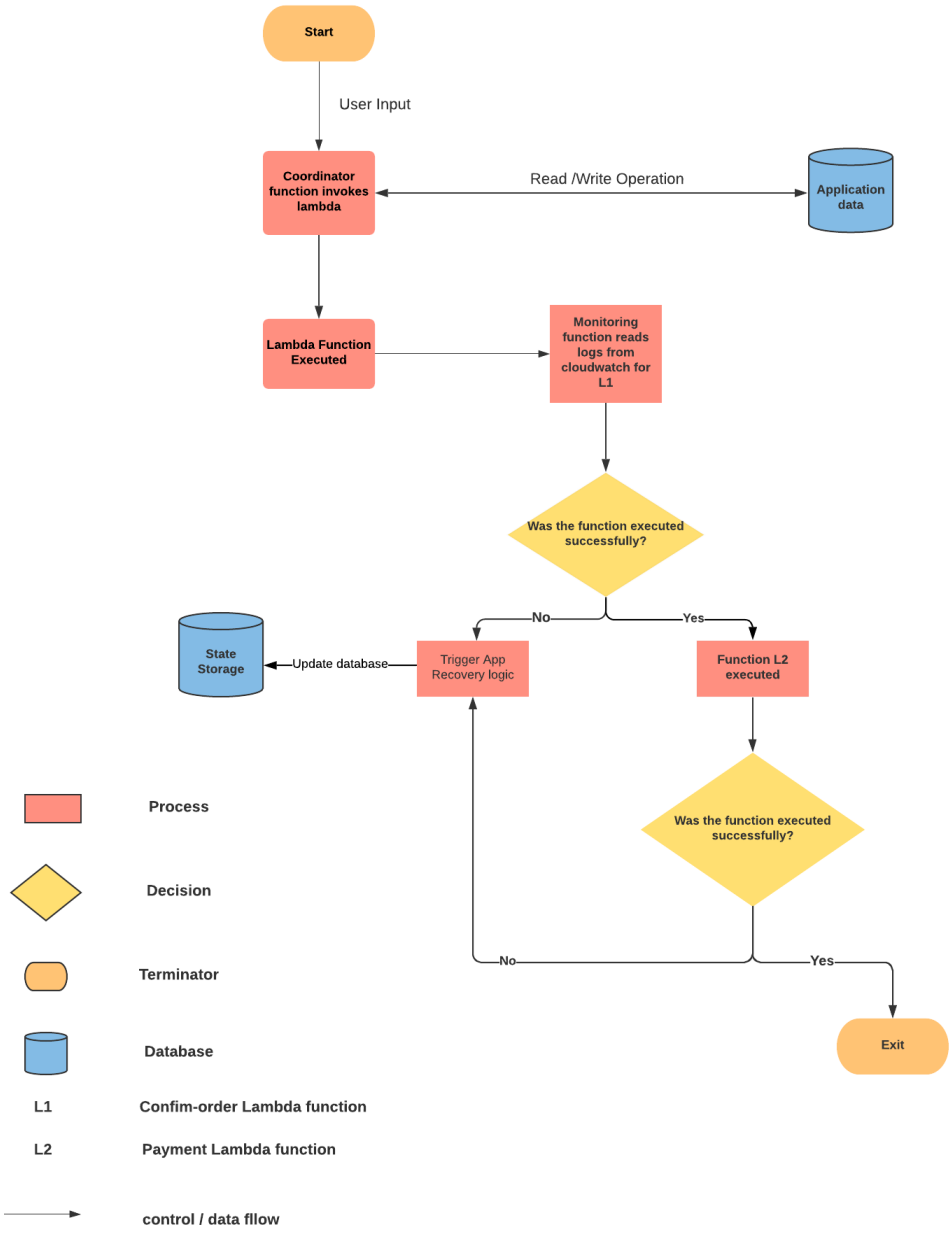

Figure 3: Flow Chart for proposed design

The figure 3 shows a workflow of the architecture developed during this research. The following steps describe the workflow of the architecture :

1. The coordinator function runs on AWS EC2 instance. It will take input from the end user and push it further to app-store, which is an s3 bucket.

2. The PUT event in S3 will act as an event trigger to invoke lambda function "confirm-order".

3. The lambda function "confirm-order" will execute and send its output to the state-storage bucket. It will also push logs to the AWS CloudWatch log group.

4. The monitoring function will read lambda function execution log from cloudwatch log events. If the log is successful. The function will proceed to the next step. However, if the lambda function fails, an app recovery logic will be executed.

The monitoring function is programmed using boto3 to get the latest log stream data. The latest log stream is fetched by using filter of last event time. From the log stream, the recent event data is taken to search if the response ended in error. If the logs show that lambda function was failed due to some error, the App Recovery logic defines the next step to be taken. Depending on the application, this logic can differ. In the scope of this research, we define the App Recovery logic to update the state storage data after a function has failed. Each application may handle failure scenario in different ways.

6 Evaluation

This section gives details about the tests that were implemented to evaluate the proposed model and the result obtained. A number of failure scenarios were studied and considered to evaluate the model. The scenarios have been divided into two categories :

1. Lambda function failure - As stated in the [24] , lambda function failure can happen due to multiple errors like deployment, invocation, execution or networking error. We test our model with function invocation and execution errors. Lambda invocation errors are caused due to the issues in request parameters, structure of event that invoked the function, function configuration or permission issues. Lambda execution errors are a result of issues with function code or due the resource that the function is trying to access.

2. Database access failure - The origin of these issues could be mainly due to the network errors. For example, storage service was not accessible and the function writing to database operation was lost. We can also configure lambda function to access resources in a local virtual private cloud (VPC). Network connectivity in issues in routing to VPC, can result in time out errors.

6.1 Case Study

As discussed in the implementation section, we consider use case of an online order application deployed on AWS lambda. For the experiment, we have executed two lambda functions and tested with three different function errors. The figure 4 shows the error of InvalidBucketName, this issue occurs when the function is trying to access a resource which does not exist. In figure 5, lambda function was failed with a function error called as type error for *getobject*() operation. Similarly, figure 6 shows access denied error which is caused when the lambda function does not have necessary rights to perform operations on other resources. All these errors were introduced in the second function called as "payment-function".

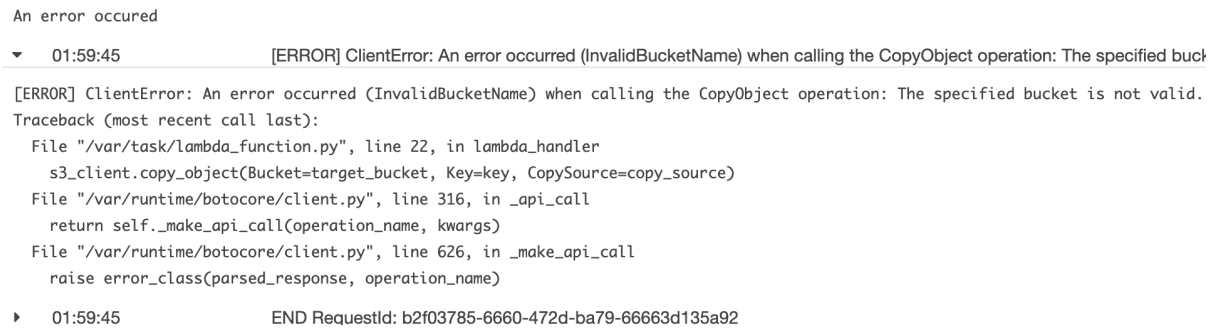

Figure 4: Invalid resource error in Lambda Function

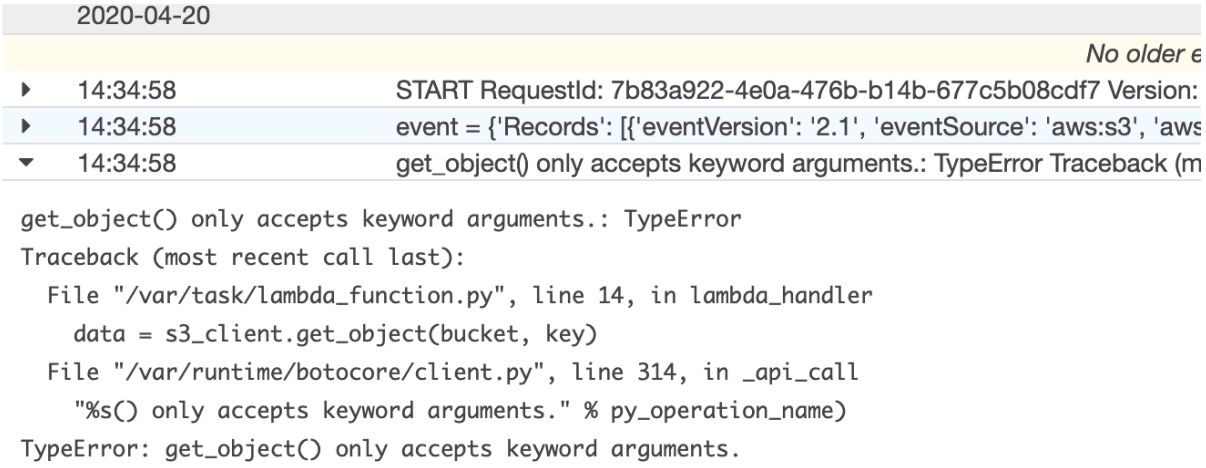

Figure 5: Type error in lambda function

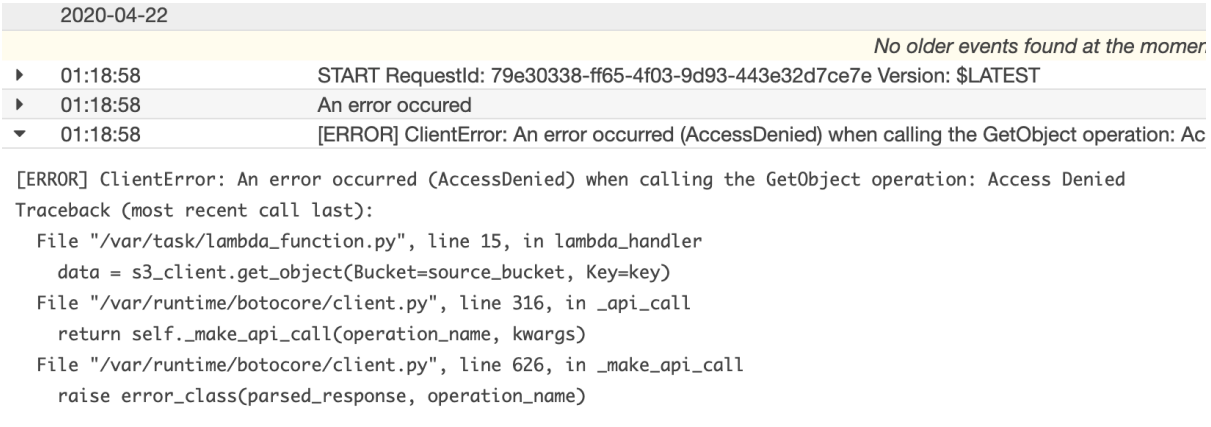

Figure 6: Permission error in lambda function

Both the Lambda functions namely, "confirm-order" which is the first function and "payment-function" is the second function are deployed. The first function is executed

successfully and updates the state storage. However, we introduce an error in the second function to test the behaviour of the monitoring function. The monitoring function executes and reports a success response HTTPS status code 200 for the first function and error for the second function as shown in figure 7 and 8 respectively. After the failure has been detected, app recovery logic is triggered which cleans up the state storage.

```
AWS Access Key ID [****************7WRS]: AWS Secret Access Key [****************yp9A]:
Default region name [us-west-2]:
Default output format [text]:
IEM1X43TUJG5M:April2020-code gkaur$
IEM1X43TUJG5M:April2020-code gkaur$
IEM1X43TUJG5M:April2020-code gkaur$ python3 M-node.py
Running M-node
Getting lambda execution logs for "confirm-order" function..
logstreamName= 2020/04/22/[$LATEST]6e00322afdf245f0b89ccd359e68a811
searching for error..
searching for error..
searching for error..
searching for error..
searching for error..
searching for error..
searching for error..
searching for error..
searching for error..
searching for error..
{'RequestId': 'e99b0dac-8301-41c2-a39c-14bc5cd9b676', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid':
mz-json-1.1', 'content-length': '906', 'date': 'Wed, 22 Apr 2020 01:41:02 GMT'}, 'RetryAttempts': 0}
HTTPSStatusCode ->  200
Lambda function request success..
IEM1X43TUJG5M:April2020-code gkaur$
```

Figure 7: Output of monitoring function executed for lambda function without error

```
IEM1X43TUJG5M:April2020-code gkaur$ python3 M-node.py
Running M-node
Getting lambda execution logs for "payment-function" function..
logstreamName= 2020/04/22/[$LATEST]f093cfc05f76412384647ffc2a8fabd4
searching for error..
Event Message is -  An error occured

executing recover_logic..
```

Figure 8: Output of monitoring function executed for lambda function with error

Based on the experiment, we observed that effective management and monitoring of the serverless functions can be achieved by designing external functionalities to work with serverless platform. We also note that failure of functions can be handled if the application is able to detect the failure. From a practitioner's perspective, as many stateful applications are using serverless, a function level failure detection and recovery can benefit the real world application design. The results obtained from our model also agree with the design approach suggested in ServerMix [13]. A combination of both serverless compute functions with traditional VM's can help overcome some limitations of function coordination.

7 Conclusion and Discussion

Serverless platform is powerful in terms of rapidly developing and deploying complex applications. However, a major issue for such applications running on FaaS is accessing and managing databases which is located in a separate storage device. One such issue is the latency in accessing due to network calls. Similarly, when multiple functions are trying to access the same database, there is a possibility of inconsistent data. Additionally, there is a lack of function to function coordination as there is no effective mechanism to communicate between the two functions. All the aforementioned issues can have a major impact on the performance of a stateful application running on serverless platform. The research question addressed during this work was created considering such issues.

In this work, we analysed multiple serverless architectures which are designed to work with heavy workload applications such as data analytics engine. Alongside, we also studied possible failure scenarios of the function and the impact of failure on the performance of an application. We further proposed an architecture for deploying stateful applications on serverless. The design includes two important components namely coordinator and monitoring function. Coordinator is responsible for managing the workflow of the serverless functions and the monitoring function evaluates serverless function execution metrics to make a decision for the next step in application workflow. The architecture is designed with a goal of providing support for optimizing performance of stateful applications on serverless. In order to evaluate the design model, we created a simple prototype using AWS serverless platform and tested the functionality of the model in case of failure of a lambda function. We observed that, failure of a function can be detected and responded with a recovery logic. The approach of the proposed model suggests that the monitoring function should be integrated with the application logic to help functions coordinate based on the peer function's execution results.

An immediate extension of this work is to develop a complex application with multiple functions using heavy database transactions such as data analytics and machine learning models. In addition, there will be a future scope towards developing a solution for fine grained coordination of concurrent function invocations.

References

[1] V. Yussupov, U. Breitenbücher, M. Hahn, and F. Leymann, "Serverless parachutes: Preparing chosen functionalities for exceptional workloads," in *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2019, pp. 226–235.

[2] B. C. Ghosh, S. K. Addya, N. B. Somy, S. B. Nath, S. Chakraborty, and S. K. Ghosh, "Caching techniques to improve latency in serverless architectures," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2020, pp. 666–669.

[3] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming sim-

plified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[4] Z. Xu, H. Zhang, X. Geng, Q. Wu, and H. Ma, "Adaptive function launching acceleration in serverless computing platforms," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019, pp. 9–16.

[5] P.-M. Lin and A. Glikson, "Mitigating cold starts in serverless platforms: A pool-based approach," *arXiv preprint arXiv:1903.12221*, 2019.

[6] U. Deshpande, "Caravel: Burst tolerant scheduling for containerized stateful applications," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1432–1442.

[7] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 442–450.

[8] S. Winzinger and G. Wirtz, "Model-based analysis of serverless applications," in *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*. IEEE, 2019, pp. 82–88.

[9] A. Palade, A. Kazmi, and S. Clarke, "An evaluation of open source serverless computing frameworks support at the edge," in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642. IEEE, 2019, pp. 206–211.

[10] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Generation Computer Systems*, vol. 83, pp. 50–59, 2018.

[11] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 257–262.

[12] M. Zhang, Y. Zhu, C. Zhang, and J. Liu, "Video processing with serverless computing: a measurement study," in *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2019, pp. 61–66.

[13] P. García-López, M. Sánchez-Artigas, S. Shillaker, P. Pietzuch, D. Breitgand, G. Vernik, P. Sutra, T. Tarrant, and A. J. Ferrer, "Servermix: Tradeoffs and challenges of serverless data analytics," *arXiv preprint arXiv:1907.11465*, 2019.

[14] D. Damkevala, R. Lunavara, M. Kosamkar, and S. Jayachandran, "Behavior analysis using serverless machine learning," in *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2019, pp. 1068–1072.

[15] A. Parres-Peredo, I. Piza-Davila, and F. Cervantes, "Building and evaluating user network profiles for cybersecurity using serverless architecture," in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2019, pp. 164–167.

[16] A. Luckow and S. Jha, "Performance characterization and modeling of serverless and hpc streaming applications," *arXiv preprint arXiv:1909.06055*, 2019.

[17] D. Jackson and G. Clynch, "An investigation of the impact of language runtime on the performance and cost of serverless functions," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 154–160.

[18] P. Moczurad and M. Malawski, "Visual-textual framework for serverless computation: a luna language approach," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 169–174.

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

[20] P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou, "Risk-aware social cloud computing based on serverless computing model," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[21] A. Kaplunovich, K. P. Joshi, and Y. Yesha, "Scalability analysis of blockchain on a serverless cloud," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4214–4222.

[22] K. Kritikos and P. Skrzypek, "Simulation-as-a-service with serverless computing," in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642. IEEE, 2019, pp. 200–205.

[23] "What Is AWS Step Functions?" 2020, accessed on: April 22, 2020. [Online]. Available: http://docs.aws.amazon.com/step-functions/latest/dg/welcome.html

[24] "Troubleshoot Invocation Issues in AWS Lambda," 2020, accessed on: April 20, 2020. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/troubleshooting-invocation.html