AWS Lambda TLQ (Transform, Load, Query) Data Pipeline

Jyoti, Shankar and Surbhi, Goyal
School of Engineering and Technology
University of Washington
Tacoma, Washington USA
jyotis@uw.edu, sgoyal3@uw.edu

***Abstract*—Implemented a cloud application using Function-as-a-service(FaaS) platform. The goal of this study is to compare the performance and cost of different Application Flow Control. The cloud application is implemented in two ways to compare the performance experiments done by using the same input data. And this experiment showed that Within Lambda performs better than Step Function.**

Keywords - Application Flow Control, StepFunctions, SQS, Lambda, AWS.

I. Introduction

Cloud Computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. This technology is widely accepted as it offers scalable and flexible resources without the need for direct management at a low cost. It includes Infrastructure as a Service, Software as a Service and Platform as a Service. This cloud technology is provided by many cloud providers and one of them is Amazon Web Services(AWS).

AWS provides reliable, scalable and secure global cloud services to deploy an application, set up a database, authenticate users. We have implemented a Transform, Load, and Query data pipeline services to test the performance of different Application Flow Control.

In this project, we have created three services named.

Service 1: Extract and Transform- This service extracts the CSV file from S3, transforms it and uploads it back to S3. Before uploading it back to S3 three changes are made. Order Priority column showing L, M, H and C is transformed to Low, Medium, High and Critical. Two Columns, order processing time[ship date - order date] and gross margin [total profit / total revenue] are added. And all the duplicate rows identified by order id is removed.

Service 2: Load- This service takes the transformed data as input and loads it into a table in AWS Amazon Aurora.

Service 3: Query- This service performs meaningful queries and is backed by Amazon Aurora. Query service facilitates a dynamic query and also validates the input query and designed to fail fast. Filtering and aggregation such as GROUP BY Clause and function SUM, AVG, MIN, MAX, and Count are also supported by the query.

A. Research questions

We investigate the following research questions:

RQ-1: How does Application flow control impact the application performance, throughput, and cost when hosted using Function-as-a-Service platforms?

RQ-2: How does parallel performance testing affect the run time and throughput of TLQ lambda functions?

II. Comparison Study

Comparison is done in between Within Lambda and With AWS Step Functions.

Within Lambda - In this approach, one lambda is initiated by the client and the other two are being triggered by the first lambda. All Lambda are getting input in JSON format. Transform uses cloud trail to trigger Load and Load uses SNS to trigger Query. A client can view the Query output in SQS. The output of Query service is sent to SQS and can be accessed by the client from there.

AWS Step Functions - It is a way to coordinate services and step through the functions of an application. In this approach, the input is provided in JSON format. Then the output of service 1, Transform, is passed to the Load service task and finally, the output of load service is passed to the final task i.e. Query. The final JSON output can be read either using the AWS GUI or command line.

The challenging part of this approach is the lambda function's maximum time limit which is 15s and the connection of lambda function to S3 while using VPC. For the large data size file, load service gets time out. To overcome this, the Batch method is used to load the data from S3 to the database.

Another challenge faced was the connection between Query lambda function from Load since the query needs to be passed in JSON format through SNS and has to be read correctly by Query Lambda function. To solve this issue RequestStreamHandler has to be used instead of HandleRequest.

A. *Design Tradeoffs*

Step Function flow control and within lambda flow diagram is given below. All the results are recorded in Cloudwatch.

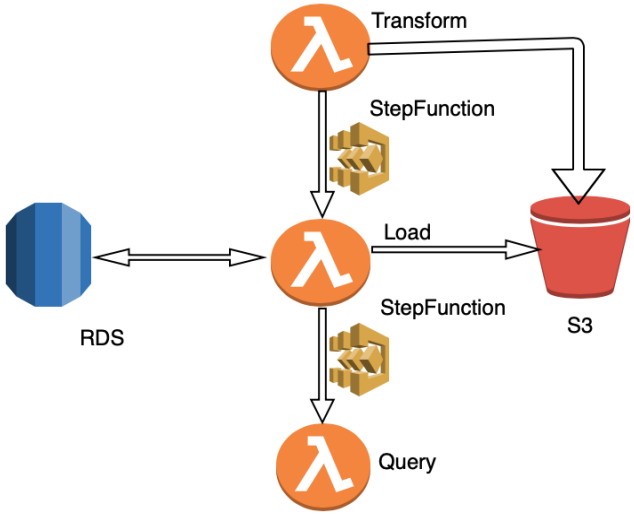

Fig 1. Step Function

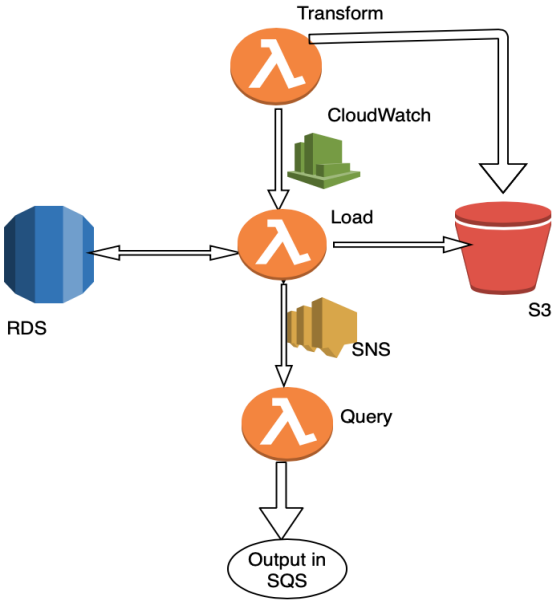

Fig 2. Within Lambda

B.Application Implementation

All the implementation is done using the JAVA 8 language. Multiple java libraries are being used such as Jackson-databind (library for serialization and deserialization of input and output stream), Google GSON. Load service creates the table and writes all data of CSV in RDS in a batch of 10,000. RequestStreamHandler is used in Query Service for byte stream implementation because HandleRequest supports only POJO.

For Within Lambda, state is passed using Cloud watch (between Transform and Load) and SNS (between Load and Query). For With Step Function, state is passed using Step function.

For Step Function, AWS CLI(command line),"aws stepfunctions start-execution" is used to launch an asynchronous execution of the state machine. The execution ARN is captured by the script and then, to determine when the state machine has completed, successive calls are made to "aws stepfunctions describe-execution" using the execution ARN to check the status. When the state machine is no longer running, a call is made to describe-execution to capture the JSON result.

B. *Experimental Approach*

All the experiments are run on EC2 C5d.large machine. Lambda for load service was time adjusted to 15 mins. Memory allocated for all the lambdas was 512MB. Load service and Query Service Lambda functions are assigned to run on VPC. All Lambda functions are in the same availability zone and same cloud region (N Virginia (us-east-1)). For the within Lambda approach, Transform and Load communicates using cloud Trail and Load and Query are communicating using SNS.

Testing is performed on sales data containing rows between 100 to 100000. Average of multiple runs are taken to get precise results. A test script 'callstepfunction_TLQ.sh' is written to perform the experiments.

To calculate the Cold service performance, each call is made after at least 30mins of inactivity. Warm performance is calculated when the cloud machines were extremely active.

III. Experimental Results

1. Step Function experiments results are mentioned below for each individual service and for the complete sequence of services. Fig 1 graph shows the log scale data.

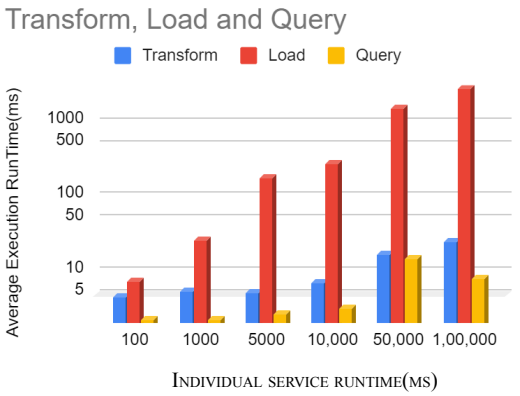

FIG 3.

Fig 3. StepFunction TLQ EXECUTION RUNTIME (in log(milliseconds)).

Figure 3 shows the average runtime of each individual service when a step function is executed. Each reading mentioned in the table is the average of multiple runs. And it doesn't include the service initialization time which is 22-23 ms on average for each service.

Most of the time is taken by the load service to write data from the text file(stored in S3) to the Aurora database table. Query runtime depends on the result data size. As we can see, query runtime for 50,000 records is 1451ms and for 100,000, it is 781ms only. During this experiment, it was found that the AWS step function has a maximum input or result data size limit of 32,768 characters. Due to it, the query was changed to get less data size results.

All the data has been taken from the Cloudwatch logs.

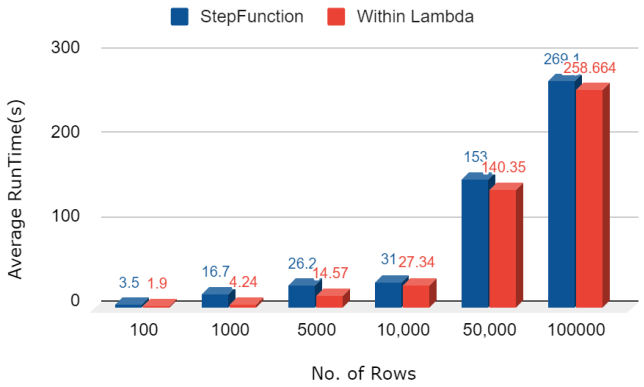

Fig 4. StepFunction WORKFLOW EXECUTION RUNTIME (in seconds).

Figure4 shows the average execution time of a complete sequence of services using a state machine and Within Lambda for different data sizes.

The comparison of each individual service and complete service shows that Step Function takes more time to execute an application. Step function execution time has scheduling and initiation overheads. As noticed, each service takes approximately 20seconds to start the next service(task). Overall, 1 min time just to initiate, schedule and enter the next task.

Cold and Warm service performance result set is mentioned in the given below table.

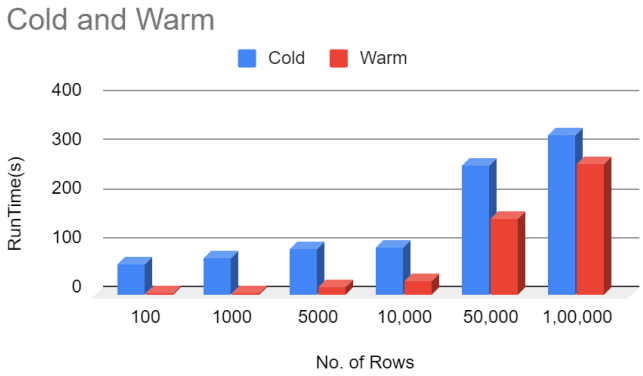

Fig5. COLD AND WARM RUNTIME (S)

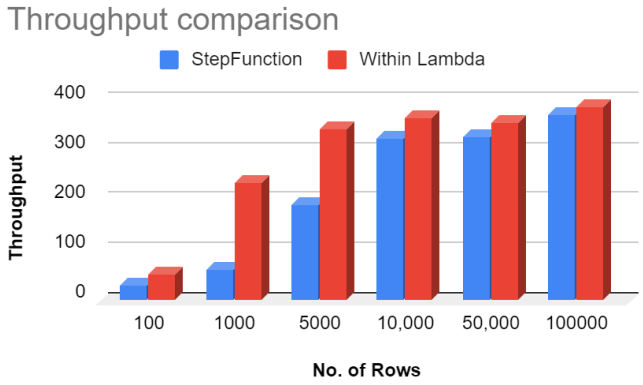

Fig 6. THROUGHPUT OF STEP FUNCTION AND WITHIN LAMBDA (NO OF ROWS/SECOND)

Figure 4 shows the throughput data of both application flow i.e. Step Function and Within Lambda. As per the results, Within Lambda has a higher throughput than StepFunction.

Billing Duration is a multiple of 100. For example, if the execution duration is 301ms then the billing would be done for 400ms. As we have used a free tier and educate account, the total cost for step function application flow experiment took only $7 and for Within Lambda is $13.

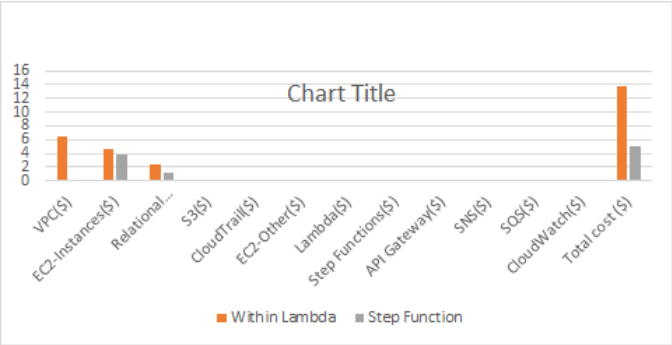

Fig 7. Cost Details

Figure 7 shows that Within Lambda implementation is costlier in comparison to Step Function implementation.

The total number of requests and executions lies in the range of free usage tier so this result may not be used to infer that step function is cheaper than Within Lambda.

2. Parallel Performance of TLQ Data pipeline.

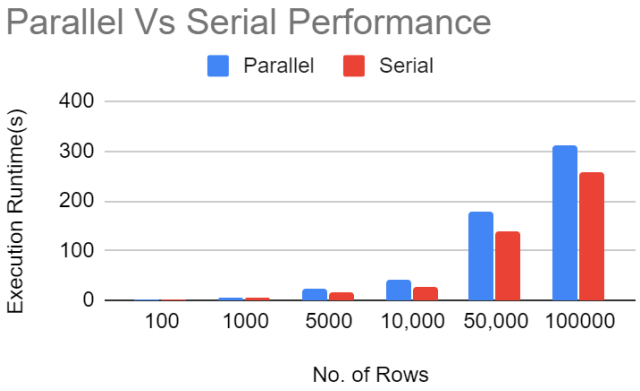

Fig 8. Parallel Vs Serial Performance of TLQ lambda functions.

This figure shows that parallel (concurrent calls) execution of multiple calls of a Transform, Load and Query lambda functions is slower than serial execution.

Figure 9 shows that the throughput of concurrent calls is much less than the throughput of a single call.

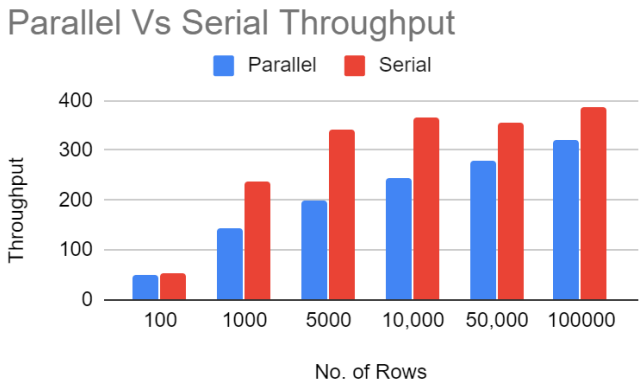

Fig 9. Throughput comparison of parallel Vs Serial calls

Conclusions

The experiment results show that Application flow control impacts application performance, throughput, and cost when hosted using Function-as-a-Service platforms.

Within Lamda application performs better than Step Function in terms of application execution time and throughput as Step function has execution overheads. But with respect to cost, the Step Function is cheaper.

AWS Step Function has a maximum limit of 32,768 characters input or result data size for a task, state, or execution as given in [1]. It affects the execution of a task, lambda function, query service result in our case. This data size limit is not adjustable, hence can't be used in case of large data size.

And Parallel execution increases the runtime and decreases the throughput of lambda functions.

Acknowledgment

AWS Educate Credits and Free Tier: The AWS Lambda free usage tier provides 1M free requests per month and 400,000 GB-seconds of compute time per month as in [2].

FUTURE WORK

Another option to direct load data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket is available. In the future, performance comparison of "LOAD DATA FROM S3" and load data using lambda function can be done.

References

[1] https://docs.aws.amazon.com/step-functions/latest/dg/limits.html

[2] https://aws.amazon.com/lambda/pricing/

[3] https://aws.amazon.com/step-functions/pricing/

[4] https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html/

[5] https://aws.amazon.com/rds/aurora/pricing/

[6] https://docs.aws.amazon.com/lambda/latest/dg/java-handler-io-type-stream.html