Prototype Big Data Archive in a Public Cloud

Group 56: Pathfinder of Big Data

Zhi Jiang, Isaac T Chan, Zhaoheng Wang

CS 462: Senior Capstone winter 2017

Oregon State University

Abstract

This document presents a review, summary, and timeline of our group's progress in the last 16 weeks, with a focus on the last 6 weeks. It includes the purpose of this project and details of where we currently are, as well as what is left to do.

CONTENTS

1 INTRODUCTION

1.1 Purpose

OSU campuses generate data constantly from multiples sources, including computer labs, wireless usage, student devices, and many others. This quantity of data, also known as big data, can effectively represent student behaviors for information technology. For example, analysis can be run to determine common student behaviors in order to allocate OSU resources to more often used utilities. Currently, the data is very difficult to manage because it is collected from multiple sources and is impossible to analyze. The data is neither stored in the same formats nor in the same locations, meaning it is inaccessible and useful information is unable to be extracted. The purpose of this project is to unify the data onto the consistent cloud platform of Amazon Web Services, which additionally provides utilities to manage and analyze.

1.2 Where we currently are and what is ahead

In order to complete the project in a timely manner, we utilize a timeline to schedule when different tasks of the project should be finished. For week 1 and 2 of winter term, we began with data storage implementation and database table creation. Then EC2 cluster configuration starting in week 3, Python script implementation in week 5 and both completed at the end of week 6. Following week 6 we have data visualization, testing, performance and security optimizations, and finally a cost comparison to a locally hosted solution. Currently we are ahead of schedule.

2 IMPORTING DATA - ZHI JIANG

My portion is to import data from data storage into a database in this project. Specifically, we need to read the file containing data in S3 and then import these data into a specified table in DynamoDB. In this portion, there are three main parts. First of all, I am going to describe what problems that have impeded my progress and my solutions for them. Secondly, I am going to talk about how I implement functions of my portion individually. Thirdly, I am going to discuss what I plan to improve on my portion.

2.1 Problem and Solution

When I start to do my portion, the primary I consider is what service I can use to operate these data between S3 and DynamoDB. The primary solution we decided in Technology Review document is we is EMR. The reason is EMR, as a managed cluster platform that simplifies running big data frameworks can provide many tools to process big data such as Hive and Pig efficiently. In my original plan, I would like to run Hive command on EMR cluster for my portion, but there are some problems I encountered. The first issue is I was not able to create an EMR cluster. Two necessary conditions are creating key pair and roles for creating a cluster, but the technical assistance of our client as an administrator did not authorize these two services for me at the initial stage. According to our communication, I got the permission of creating a key pair, but I still need to permission of setting up the role. Our administrator thought it would be equivalent to being system administrator if she gave me full access to create a role. She also told me she would try to use other ways to help resolve this problem, but in fact, I have wasted a lot of time on this part. Therefore, I started to consider other solution.

I went back to watch technology review document, and I planned to use an alternative solution, which is an AWS SDK for Python. According to my research, I believed Boto 3 as an AWS SDK for Python should be the most appropriate method to solve this problem. There are three benefits; one is I am familiar with Python, so I do not need to spend more time to learn it. Another is SDK provides useful methods which can quickly complete correlative tasks such as accessing S3 and DynamoDB. The third benefit the cost of this solution is less than the previous one because it does not require additional environment configuration like creating a cluster.

2.2 Function Implementation

When we implement this function, I need to read data from a file stored in S3 and then import these data to a table in DyanmoDB firstly. I also spent lots of time on this part because I always used improper method until I found the correct one. The following code is an inappropriate method I used before.

```
s3 = boto3. resource('s3')
obj = s3.Object('bucket_name', 'file_name')
content = obj.get()['Body'].read()
```

Listing 1: inappropriate method to access S3 and read data

I found this method in Boto 3 document. I was not familiar with this SDK, so I through each file should be regarded as an object, and then I just need to read the content of this object. But eventually, I failed to read data from variable content. In fact, I did not completely understand the role of an object type in this SDK. Object type is a complex constructor, and it contains much information. I tried to find other solution. The difference between this two method is the file we want to process is loaded into a temporary file and extension of its text.

```
s3 = boto3.resource('s3')
s3.download_file("bucket_name", "file_name", "tmp.txt")
```

Listing 2: appropriate method to access S3 and read data

The benefit of this approach is it made this problem more familiar to me because I have done many similar tasks, which I/O for a text file. So now what I should do was using a conventional method to read this temporary file in Python. When I wrote data in a table, there are two things I need to mention. One is I used method table.batch writer() to fo do it. The purpose of this approach is you can both speed up the process and reduce the number of write requests made to the service if you are loading a lot of data at a time, so it is very appropriate to this project related big data. Second is the primary key. The basic idea to create a table is that it must contain a primary key. The purpose of this way is to make each row unique. In our implementation, we set an int variable as primary key, and then to increase it as creating a new item.

```
# open this file contains content of CSV
f = open('tmp.txt')

# set a variable as primary key
num = 0

# use table.batch_writer() method to load a lot of data
with table.batch_writer() as batch:
  for line in f:
    num += 1
```

```
    list_line = line.split(',')

    # create a new item and put it into a table
    batch.put_item(
      Item = {
        'primarykey': str(num),
        'attribute_name0': list_line[0],
        'attribute_name1': list_line[1],
        ...
        ...
        ...
      }
    )
f.close()
```

Listing 3: importing data

We went to check results after we run this script. Fortunately, the result met our expectations. The following picture is showing the result of importing data. In this table "DataForCapstone", we can find the item count is 671, which means there are 671 items in our table. When we went to back check our CSV file, it still has 671 rows. So we successfully import data from S3 into DynamoDB.

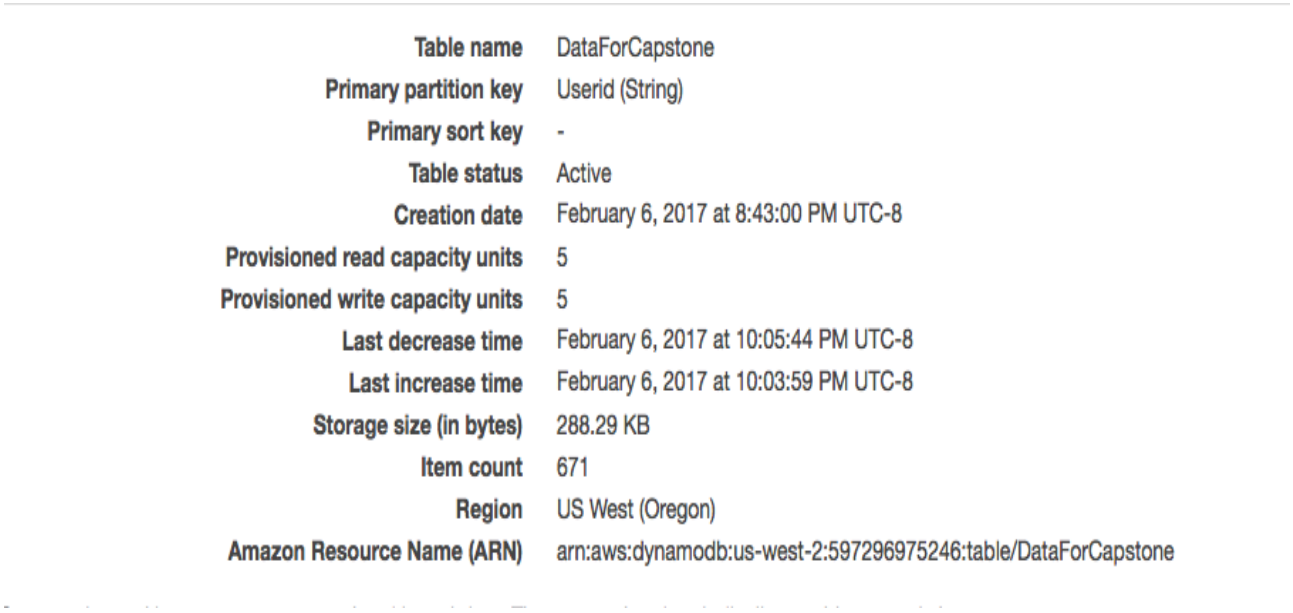


Table name	DataForCapstone
Primary partition key	Userid (String)
Primary sort key	-
Table status	Active
Creation date	February 6, 2017 at 8:43:00 PM UTC-8
Provisioned read capacity units	5
Provisioned write capacity units	5
Last decrease time	February 6, 2017 at 10:05:44 PM UTC-8
Last increase time	February 6, 2017 at 10:03:59 PM UTC-8
Storage size (in bytes)	288.29 KB
Item count	671
Region	US West (Oregon)
Amazon Resource Name (ARN)	arn:aws:dynamodb:us-west-2:597296975246:table/DataForCapstone

Figure 1: result after importing data

2.3 Subsequent Improvement

Although we have implement function of this part, we should improve our python script. Our current condition is that we need to create a table in Dynamo before importing data, in the other word, we import data into an existing table, but we hope our python script can create a table automatically as import data.

The one problem we have to encounter is the name of each attribute. In fact, the sample data file our client provided to us does not have a header, so we have to set a name of the attribute to each column according to our understanding. If a CSV file has a header includes all attribute name, we can use this header to create a table directly. Specifically, we can separate CSV files into two parts, one only includes a header, and another includes data. Then we just need to process part including a header for creating a table.

3 DynamoDB and QuickSight - Zhaoheng Wang

3.1 Data structure organization

The general process of our project is to parse the file which contains sample data and load into database. After that, we will do the analysis base on the processed data. Before loading data into DynamoDB, the table and the data structure should be organized. Therefore, the first job for me is to figure out the data structure before creating table. DynamoDB is actually a key-value type NoSQL which is different from the Mysql (the key-value type relational database). In relational database, the ER-diagram is usually used to convert data into schema. In the key-value type of NoSQL database, the ER-diagram is not suitable to represent the table. So I have do some research about how to organizing data structure in DynamoDB. From the research, there are two ways I find to organizing data structure in DynamoDB. The first one is to convert from ER-diagrams however, it only gives general idea about this not specific step. Therefore, I choose the second way for organizing data structure. In this way, we treat each rows of sample data as an individual item and each columns will be the attributes for the item. When we load the data into table, we are actually load the items into the table in DynamoDB. For instance, we need to store the wireless information for different users based on sample data such as connecting time, disconnecting time, information of device, average usage and operating system they use. In this case, we treat each user as individual items and the wireless information for that user will be the attributes.

3.2 Table creation

After data structure have been organized, the next job for me is to creating table in DynamoDB. The tool we choose is AWS java SDK. AWS Java SDK is similar with the command line prompt in Windows. we could use it to do different implementations for the table such as creating table, do updating. For instance, if we need to update some items(not large quality) in the table, we could use the APIs in AWS Java SDK to create table. Because of the APIS offered by AWS java SDK document, this step will be much easier than before. However, AWS Java SDK is not efficient for uploading the items if there are very large quality of items. As a result, we will only use it for simple operations such as creating table, deleting table, updating table. We finish creating table and loading sample data on Wednesday of week 5.

3.3 QuickSight

3.3.1 QuickSight graph creation

After the table is created and the data has been load into DynamoDB. We step to make visualization on QuickSight at the end of week 5. We follow step on the Quicksight document for creating the visual. For our project, the first step is to generate a data source in s3 or local pc. These data source are very important for Quicksight because it will require these data source for creating data set and do the analyzing. After that, the Quicksight needs to connect with the data source. In this step, if the database is on the Amazon web services the data source will be easily to connect. However, if the data source is not on the Amazon web services it requires database access information to connect the data source in

database. For our project, since the s3 is on AWS it is easy to connect with data source. The next step is to create the data set according to data source and do an analyzing. After the data set has been created, the visual can be generated by the data set. There are different types of visual on the Quicksight and we can choose one to generate the suitable graph for the analyzing results.

3.3.2 Improvement in future

Currently, we have finished data structure organizing, creating table and make visualization based on the sample data. However, there still have some improvement for these parts in our project. Firstly, the way I choose for design the data structure may not be the optimal choice. Therefore, I would like to check with the data analyzer whether it is the efficient way or not. Secondly, the graph which made by QuickSight needs to improve because some of sample data needs to be filter. Furthermore, the QuickSight seems not handle the complex analysis and our client doesnt give specific topic that we need to analysis. As a result, I would like to check with the data analyzer what kind of analysis we need to do.

3.3.3 Screenshot of the image on QuickSight

The figure 2 shows Proportion of Different OS which users use to access.Based on the graph, most of users are using Mac OS or IOS to access the wireless.The figure 3 the relationship between different OS and total Traffic. As we can see, the ios system will be the highest one.The figure 4 the Relationship of Total Traffic in different connecting time and Avg Usage in different Connecting Time.The figure 5 shows the relationship Between Total Traffic, Avg Usage and Device Name

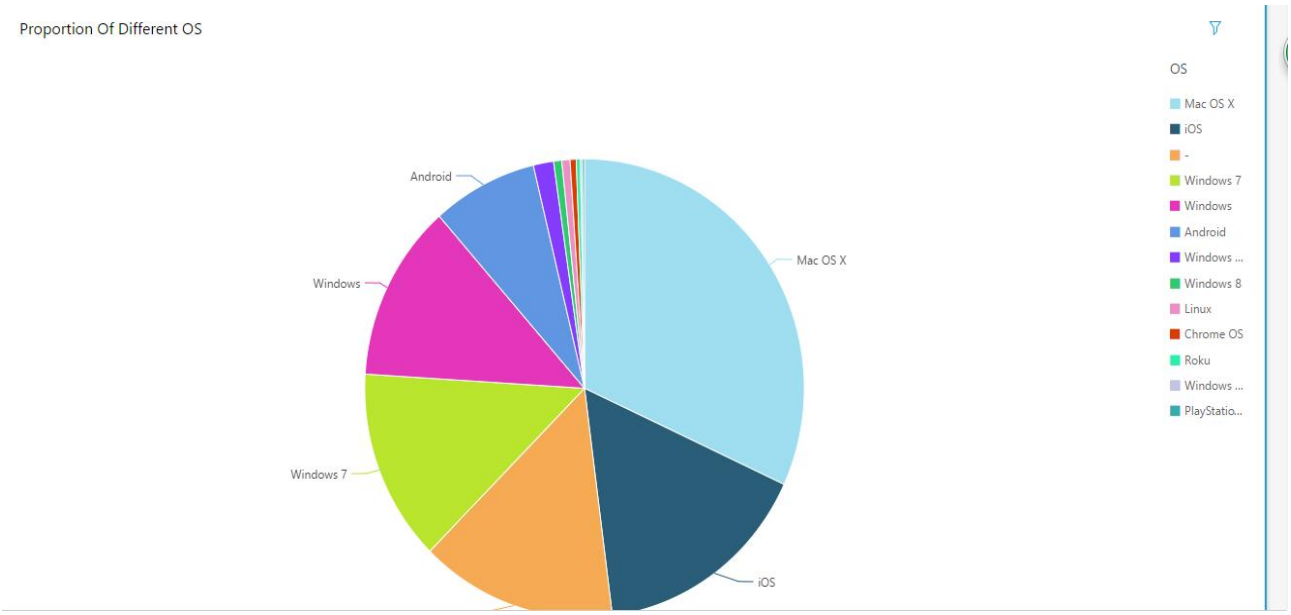


Figure 2: Proportion of Different OS

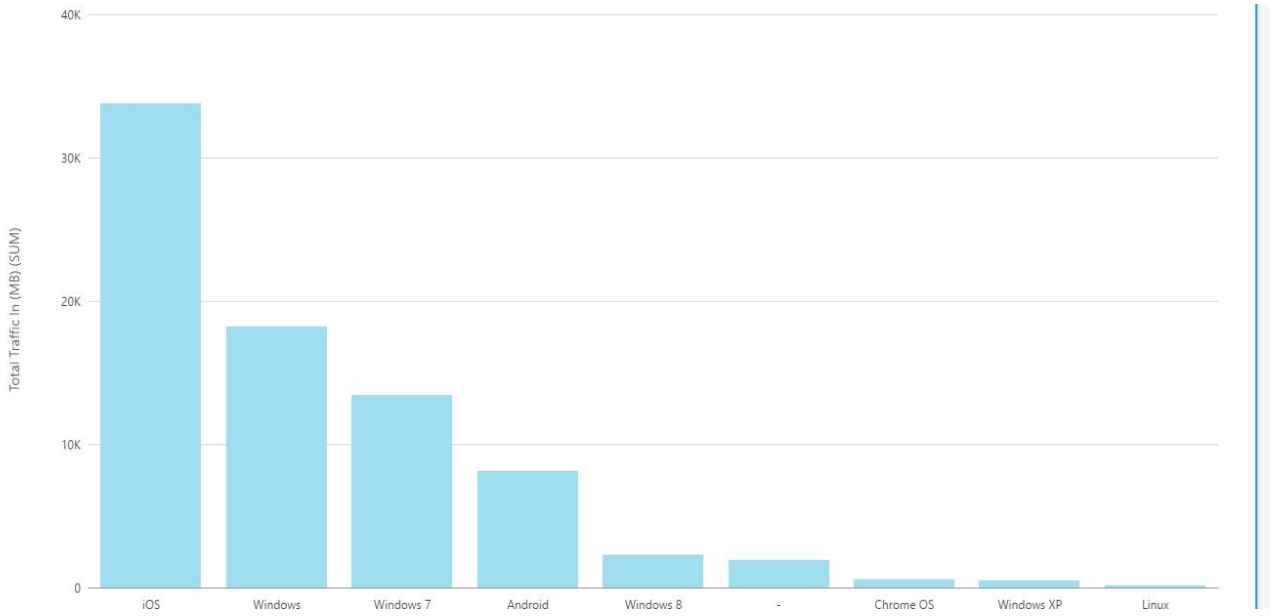


Figure 3: relationship between different OS and total Traffic

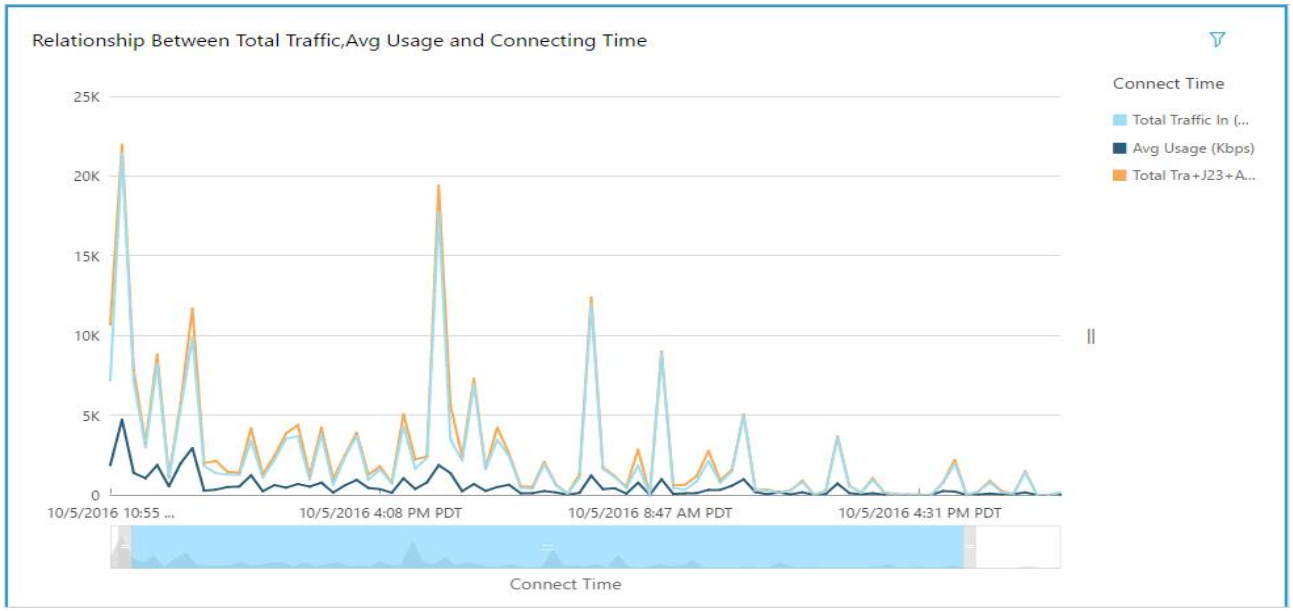


Figure 4: relationship between different OS and total Traffic

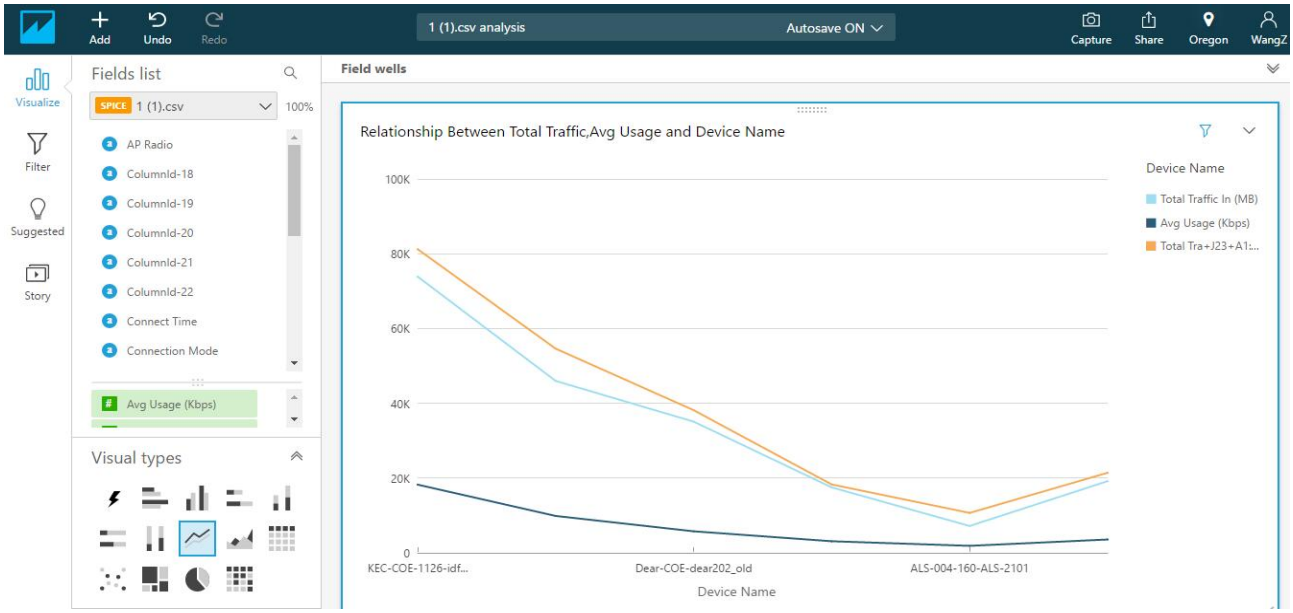


Figure 5: relationship between different OS and total Traffic

4 What remains to be done - Isaac Chan

Due to our project task division, my tasks all fall at the end of winter term. I have kept up with the project as a whole, as well as the details of specific implementations that my teammates have completed. There are still several important parts of this project that remain to be completed.

4.1 Tests for Functionality

Tests for functionality ensures that users are able to interact with the system. They will need to use different utilities for different interactions, such as monitoring resources, monitoring performance, managing data, and performing different methods of analysis. As shown previously, AWS Quicksight is able to build visualizations from the data. However, its unable to perform more complex analysis, such as machine learning. We plan to test basic data analysis code from AWS EMR, or Elastic MapReduce, to ensure data analysis is possible. For example, we can run a linear regression script to determine the relationship between time a user stays on a specific wireless network compared with their location on campus.

In the end, a data analyzer would use their choice of tool on the implemented database. AWS EMR will simply provide us a proof of concept that analysis is viable. When implementing these tests for functionality, I will work with a data analyzer to ensure that our database structure allows deeper analysis.

4.2 Measuring Performance Metrics

Performance metrics for database functionality are important to ensure the usability of our implemented solution. We will assess performance from typical database operations: data inserts, updates, and reads. From this we can obtain a baseline for the database performance, and examine how varying data and query loads compare to the baseline. We will

use AWS Cloudwatch to implement custom metrics in order to measure the performance.

Due to the subjective nature of whether or not the performance of the solution is acceptable, we will meet with current data analyzers and our client to determine if it meets expectations. We plan to provide a model for realistic usage. To create this model, we will need differing sizes of sample data to load into the database and query to obtain an accurate trend for performance, as the database size increases.

4.3 Database Security

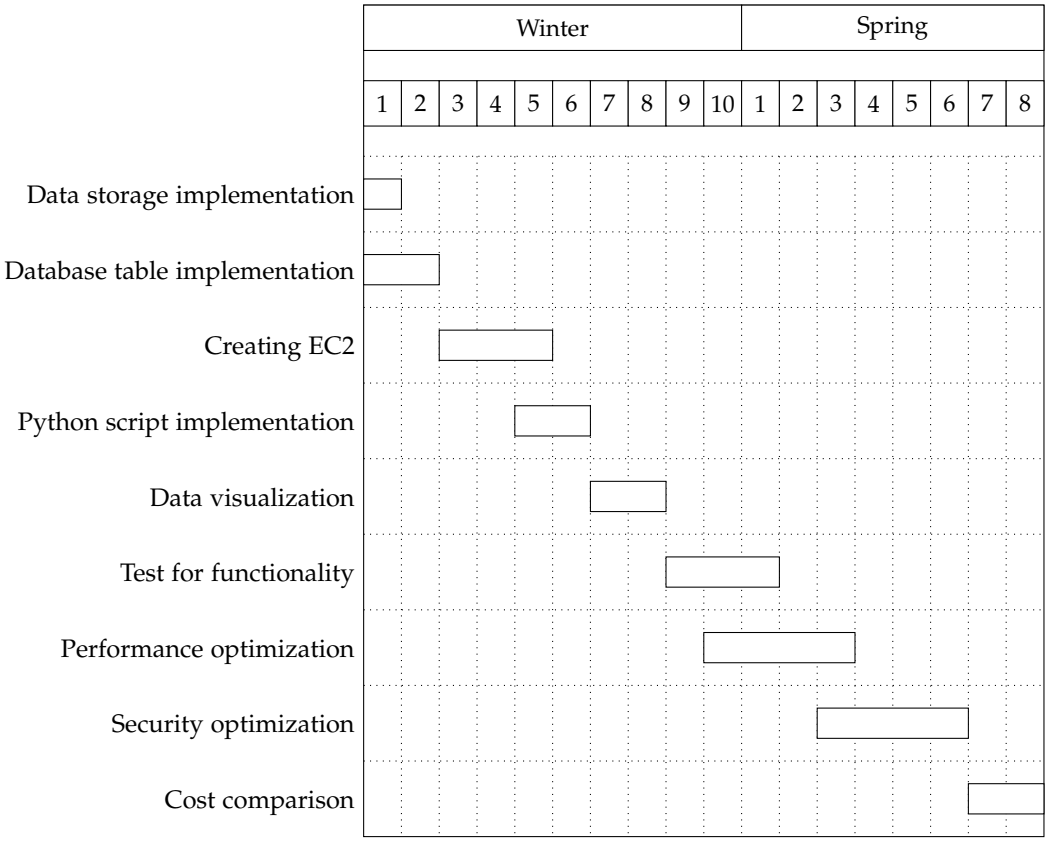
For database security, unfortunately NoSQL databases do not support external encryption tools. Therefore we must use a combination of methods to ensure security of our solution, by restricting access and preventing malicious utilization of the data. The options we chose are AWSs user authentication policies, encrypting sensitive data fields, and using sufficient input validation to avoid injection attacks. Because of the lack of external tool support by NoSQL databases, we must resort to using modular security methods.

AWSs user authentication policies, as well as encrypting sensitive data fields are extremely important to our database security. However, they can only be implemented by the AWS administrator; we, as developers, dont have access. We can utilize sufficient input validation to avoid injection attacks. A benefit of implementing our NoSQL database with AWS DynamoDB is that it prevents common injection attacks by not allowing multiple operations with a single command. However, there do exist some lesser known vulnerabilities, and we will have to thoroughly research and test our solution.

4.4 Cost Comparison

The final step after the completion of our project prototype is a comparison with a locally hosted solution, in order to see if our prototype will be adopted for use by OSU information services. They will be seeing if the cost to value ratio of our cloud prototype is greater than the value provided by local hardware. In order to fully represent the capabilities of our project, we will need to present a thorough, detailed model of the performance as the workload increases. As we get closer to the end, we will need to talk to our client in order to see what else we need for him to do a comprehensive comparison.

5 TIMELINE


Winter
Spring
1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8
Data storage implementation
Database table implementation
Creating EC2
Python script implementation
Data visualization
Test for functionality
Performance optimization
Security optimization
Cost comparison