Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

ML19/20-5.1. Implementation of Geo-Spatial Encoder in Microsoft Azure Cloud

Gouthaami Sukadore Ramesh Lakshmi Alekya Chittem Ravikumar Solanki
gouthaami.sukardoreramesh@stud.fra-uas.de, lakshmi.chittem@stud.fra-uas.de, ravikumar.solanki@stud.fra-uas.de

***Abstract*— This paper explains how the geospatial encoder experiment performed with different types of services offered by Microsoft Azure platform using Cloud computing. Cloud computing is the significant demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. Microsoft Azure cloud platform is used due to its flexibility, advanced site recovery, and built-in integration. The aim of the project to store the data securely and make easily accessible from all around in the world over HTTP or HTTPS using the Azure platform. In this, it described what kind of services are used to deploy the data in Azure and how to create the various storages such as blob storage, table storage and queue storage in detail —discussed about docker creation and code implementation with the final output.**

Keywords—Microsoft Azure, Cloud computing, Scalar Encoder, Cloud Services, Blob storage, Queue storage, Table storage.

I. INTRODUCTION

Cloud computing gives a picture of everything, such as the point of offering a foundation and a functioning application stage, which is adaptable and practical regarding cost for clients. Cloud administrations are very dynamic without a lot of overall characterized qualities, and as indicated by client necessities, there are a few arrangements accessible available of cloud computing. Microsoft Azure is one of the cloud specialist co-ops. The aim goal of this undertaking is to get acquainted with the Microsoft azure condition, stockpiles, what's more, administrations to send and run the software designing task on Cloud. The project is to realize geospatial encoder in Azure services. A similar code is utilized in this task to be deployed on Azure Cloud, and the output will show on table storage.

SE project overview:
Hierarchical temporal memory (HTM) provides a flexible and biologically accurate framework for solving the prediction, classification, and anomaly detection problems of various data types [1]HTM systems require data input are the form of distributed sparse representation (SDR) [2].HTM systems require data entry in the form of Distributed Sparse Representations (SDRs). SDRs differ significantly from standard computer representations such as ASCII for text in the sense that they are encoded directly into the image. SDR consists of a wide array of bits, most of which are 0 and a few are 1s. Each bit has some semantic meaning, so if two SDRs have more than a few overlapping one-bit bits, the two SDRs have a similar purpose. Any data that can be converted into SDRs can be used in a wide range of HTM applications. Therefore, the first step to using the HTM system is to convert the data source to Use what we call an encoder SDR. The encoder converts the native format the data into SDR, which can then be input into the HTM-system. The encoder is responsible for determining which output-file bits are should be one and which output bits should be zero with given input value, thereby capturing the essential semantic characteristics of the data. Similar input-file values should produce highly overlapping SDRs.

II. GENERAL METHODOLOGY

There are various services offered by Azure platforms, such as Azure DevOps, Virtual machine, Azure Cosmos DB, Azure storage and more. To implement our project, we have opted for the Azure storage platform because the data in Azure Storage is accessible from around the world over HTTP or HTTPS. The Azure storage platform includes more data services such as Azure Blobs, Azure Files, Azure Tables, Azure Queues and Azure Disks. We have used Docker tool because its more straightforward to create, deploy, and run applications by using containers.

Creation of the container using docker tool
Docker gives the capacity to package and run an application in an approximately confined condition called a holder. The seclusion and security permit you to run numerous container all the while on a given host. The container is lightweight since they needn't bother with the additional heap of a hypervisor, however, run legitimately inside the host machine's part. This implies you can run a more significant number of containers on a given equipment blend than if you were utilizing virtual-machines. You can even run to docker container inside host machines that are virtual machines. [3]
Docker gives as tooling and a stage to deal with a life-cycle of your compartments:

- Built application and its supporting parts are utilizing the container for code.
- The compartment turns into the unit for conveying and testing your application.
- At the point when it is prepared, send your application into your creation condition, as a holder or an arranged assistance. Works similar to whether your creation condition is a nearby server farm, a cloud supplier, or a crossover of the two.

Usage of Azure Storage:
Azure Storage is a cloud service managed by Microsoft that provides highly available, secure, durable, scalable and redundant storage. Whether it is images-file, audio, video, logs, configuration files, or sensor data from an IoT array, the data needs to be stored in an easily accessible way for analysis, and Azure Storage provides every possible option use case. It as so many core services like Blob Storage, Table Storage, Azure Files and Queue Storage. [4]

Usage of Azure Blob Service:
Blob storage is a service of Microsoft Azure that is used to store binary large objects or BLOBs, which are usually composed of unstructured data (such as text, images, videos and their metadata). The blobs are stored in a class directory structure called "containers". [5]

Blob services include:
1. Blob is any type of data object.
2. Container to pack multiple spots together.
3. Azure storage account, which contains all your Azure storage data objects.

Usage of Azure Table Storage:

Azure Table Storage is a scalable NoSQL key-value data storage system that can be used to store vast amounts of data-file in the Cloud. The storage product uses a schema-less design, and each table has rows composed of Paris of value-key. Microsoft describes it as a solution ideal for storing structured and non-relational file, covering use cases ranging from storing terabytes of structured data serving web applications to retaining data sets that can be accessed without using complex joins or foreign keys Structured data. NET library. [6]

Azure table storage components:

a) A storage account that contains all your tables.
b) A table consists of a collection of "entities":
c) An entity is a set of attributes, like a database row. An entity can be expanded up to 1MB.
d) Attribute: The most detailed element in the list. Features consist of name/value pairs. An entity can pack up to 252 points to store data, and each entity contains three system attributes, which define its partition key, row key, and timestamp.

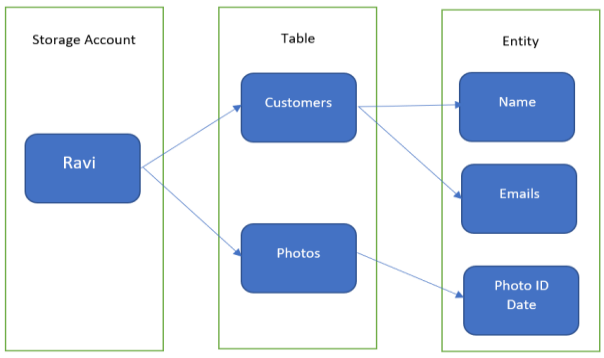


Fig.1: Hierarchy of Table Storage.

Azure table storage allows users to build cloud applications without worrying about architectural lock-in quickly. When developers want to store several terabytes of data while reducing storage costs-when, the stored data does not rely on complex server-side connections or other logic; developers should consider using Azure table storage. Other use cases include disaster recovery solutions or storage of up to 500 TB of data without implementing sharing logic.

Usage of Azure Queue Storage:
Queues have been around for a long time-the simple FIFO (first in first out) architecture of queues makes queues a universal solution for storing messages that do not need be arranged in a specific order. In short, Azure Queue Storage is a service that allows users to place large numbers of messages, process them asynchronously and use them when needed while using a pay-per-use pricing model to reduce costs. [7]
The Azure storage queue is composed of the following component:

a) Storage account, which as all your storage services.
b) A queue is composed of a group of messages.
c) *Message:* A message can contain any type of information. For example, the message can be a text message that should trigger an event on the application, or it can be information about an event that occurred on the website. Mail-in any format can only be up to 64KB, and the maximum time that the mail can remain the Queue is seven days. However, a single line can hold up to 200 GB of messages. The message can be a text string or a byte array, which contains any information such as XML, CSV and other formats.

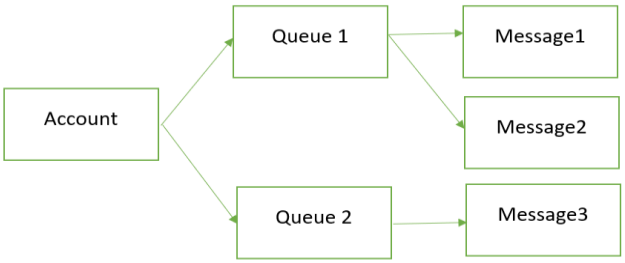


Fig.2: Hierarchy of Queue Storage.

III. IMPLEMENTATION DETAILS

Implementation of Geospatial encoder in Microsoft Azure platform

This Azure project aims to run preview implemented Geospatial encoder project using Azure Cloud Services and Docker. The following is based on the workflow described above to achieve the project's goals.

a) First Step: The geospatial encoder code is packaged as a docker image in a docker container. Then, the image building process will be executed automatically. This docker image has been uploaded to Azure. Then, you can use the container registry and the application to deploy using azure container instance service.

b) Second Step: The user makes an input file is uploaded to blob storage to run the application.
c) Third-Step: Is to trigger the application, for that user make message is send to Queue to start the application.
d) Four-Step: When the Queue received the message, it will perform the deserialization process, because it transforms and changes the data organization to linear the format used for storage.
e) Five-Step: When the Queue received the message, and then it will start the downloading, and it will extract the file.
f) Six-Step: The main unit-test file of SE-project will use the data from the file, which was downloaded from the Blob and will start the execution.
g) Seven Step: After execution output of SE-project, will upload to blob and table storage and then given queue message will delete automatically.

IV. CODE

Execute the Geospatial encoder is implemented according to the workflow; the class Experiment.cs implement the above explained. It is located in the MyExperiment directory of the project folder and is called by the main Program.cs for implementation.

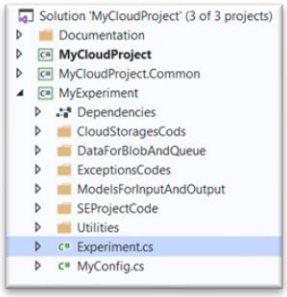
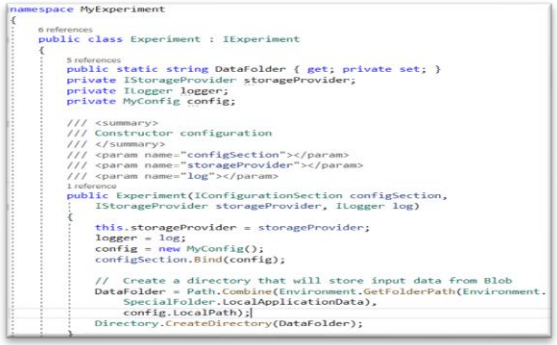


Fig.3: Experiment.cs class location in MyCloudProject.

The purpose of the MyExperiment function is to create a folder. The local path to download the file from the Blob Storage, the data existing in the downloaded file is to Execute and upload back to Blob & Table. Then the program will be executed until the cancel signal is issued to cancel the token. The result of the procedure will follow and then uploaded into the storage blob and table. This stream Several methods discussed below illustrate these actions.



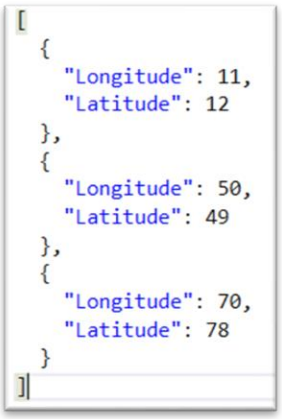
```
namespace MyExperiment
{
    6 references
    public class Experiment : IExperiment
    {
        5 references
        public static string DataFolder { get; private set; }
        private IStorageProvider storageProvider;
        private ILogger logger;
        private MyConfig config;

        /// <summary>
        /// Constructor configuration
        /// </summary>
        /// <param name="configSection"></param>
        /// <param name="storageProvider"></param>
        /// <param name="log"></param>
        1 reference
        public Experiment(IConfigurationSection configSection,
            IStorageProvider storageProvider, ILogger log)
        {
            this.storageProvider = storageProvider;
            logger = log;
            config = new MyConfig();
            configSection.Bind(config);

            //  Create a directory that will store input data from Blob
            DataFolder = Path.Combine(Environment.GetFolderPath(Environment.
                SpecialFolder.LocalApplicationData),
                config.LocalPath);
            Directory.CreateDirectory(DataFolder);
        }
```

Fig.4: Reference of Experiment.cs.

Objects taken in below fig:4 here Experiment.cs code uses the following:

a) Storage provider: concentrated on Blob storage, that is, in Blob and Download files from it.
b) Logger: Recording is the act of keeping logs. Many operating systems, software frameworks, and programs include logging systems.
c) Config: it is used to extract information from the app-config.

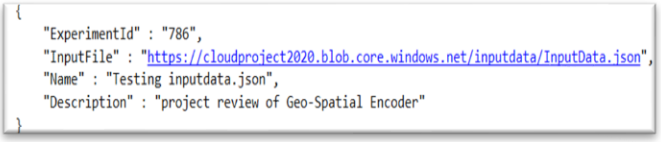
In the Following fig:5 shows the InputFile.json file, which uses for upload to Blob and executes the SE-Project code. I design as per my Geo-Spatial project because, in that project, we need to give an input of Longitude and Latitude.



```
[
  {
    "Longitude": 11,
    "Latitude": 12
  },
  {
    "Longitude": 50,
    "Latitude": 49
  },
  {
    "Longitude": 70,
    "Latitude": 78
  }
]
```

Fig.5.: InputData.json.

In the Fig:6, it shows the message for Queue; it helps to trigger the code.



```
{
   "ExperimentId" : "786",
   "InputFile" : "https://cloudproject2020.blob.core.windows.net/inputdata/InputData.json",
   "Name" : "Testing inputdata.json",
   "Description" : "project review of Geo-Spatial Encoder"
}
```

Fig.6: QueueMessage.txt.

The Fig:7 below show the appconfig.json, which is helping to Upload the data in Azure Cloud Servers. In that, we need StorageConnectionString to connect the local to Azure. And remaining is for creating the container, for queue message and table storage string.

```
"MyConfig": {
  "GroupId": "Wise2019",
  "StorageConnectionString": "*",
  "TrainingContainer": "training-files",
  "ResultContainer": "result-files",
  "ResultTable": "results",
  "Queue": "trigger-queue",
  "StorageConnectionStringCosmosTable": "*",
  "LocalPath": "mycloudproject-data"
}
```

Fig.7: appconfig.json.

The following Fig.8: shows four class, its name as ArrayUtils, FileUtils, FlexComRowMatrixUtility and StringUtilities, it uses to make the output file in txt files.

Fig.8: It shows the Utilities which use in Experiment.cs

The following Fig.9: show the models for Input and Output, it is used to design the input and output files of Geospatial encoder.

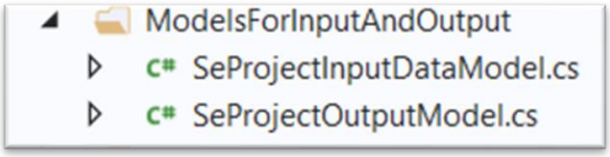


Fig.9: It shows the Models for input and output.

ConfigSection is used to extract information from its appconfig. Use IConfigurationSection and InitHelpers class. After creation, RunQueueListener, the method will be created later. The purpose is to listen to the message queue in the program. The figure shows when the input file is triggered, and the code is run on the Visual Studio console to generate the output file we get:

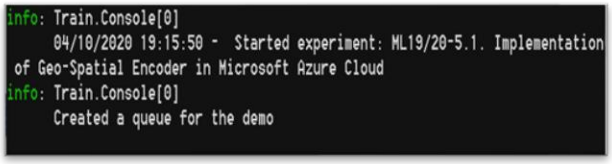


Fig.10. Console output in which the Queue as created and waiting for the message to process forward.

```
// STEP 1. Reading message from Queue and deserializing
var experimentRequestMessage =
    JsonConvert.DeserializeObject<ExerimentRequestMessage>(message.AsString);
logger?.LogInformation(
    $"Received message from the queue with experimentID: " +
    $"{experimentRequestMessage.ExperimentId}, " +
    $"description: {experimentRequestMessage.Description}, " +
    $"name: {experimentRequestMessage.Name}");

// STEP 2. Downloading the input file from the blob storage
var fileToDownload = experimentRequestMessage.InputFile;
var localStorageFilePath = await storageProvider.DownloadInputFile(fileToDownload);

logger?.LogInformation(
    $"File download successful. Downloaded file link: {localStorageFilePath}");

// STEP 3. Running SE experiment with inputs from the input file
var result = await Run(localStorageFilePath);
result.Description = experimentRequestMessage.Description;
var resultAsByte = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(result));

// STEP 4. Uploading result file to blob storage
var uploadedUri =
    await storageProvider.UploadResultFile("ResultFile-" + Guid.NewGuid() + ".txt",
        resultAsByte);
logger?.LogInformation($"Uploaded result file on blob");
result.SeExperimentOutputBlobUrl = Encoding.ASCII.GetString(uploadedUri);

// STEP 5. Uploading result file to table storage
await storageProvider.UploadExperimentResult(result);

// STEP 6. Deleting the message from the queue
await queue.DeleteMessageAsync(message, cancelToken);
```

Fig.11: Step to complete the following project.

The following Step of Code-described in details:

Step-1: which is used to use to read the message from the Queue with the code format, and which is written by the user.
Step-2: The following code will help to download the file from the Blob, which user already upload in the azure input file container.
Step-3: In this following Step, the InputFile, which is download in the Second Step, which will help to run the experiment of SE-project.
Step-4: In this step, it will upload the result file container in Blob Storage, with the help of AzurStorageCode.
Step-5: In the five-step, it will upload the result in Table Storage.
Step-6: In the last step, it will delete the message from the Queue.

The following Fig:12 shows the MainClass for Se-Project, which is used in this project, in the MainClass, I combined my last unit test code into two different functions, one for Longitude and one for Latitude.

```
1 reference
public int[] UnitTestLongitude(double input)
{
    GeoSpatialEncoderTests spactialcase = new GeoSpatialEncoderTests();
    var spacialoutput1 = spactialcase.GermanyToItalyLongitude(input);
    var spacialoutput2 = spactialcase.GermanyToChinaLongitude(input);
    var spacialoutput3 = spactialcase.GermanyToIndialongitude(input);
    return spacialoutput3;


}
1 reference
public int[] UnitTestLatitude(double input)
{
    GeoSpatialEncoderTests spactialcase = new GeoSpatialEncoderTests();
    var spacialoutput4 = spactialcase.GermanyToItalyLatitude(input);
    var spacialoutput5 = spactialcase.GermanyToChinaLatitude(input);
    var spacialoutput6 = spactialcase.GermanyToIndiaLatitude(input);
    return spacialoutput6;


}
```

Fig.12: A primary function which combined the SE-project Unit-Test.

The two following function is called in the Experiment.cs for executing the main code. As shown in fig.13.

```
// Step 1: Run all test cases in a file
foreach (var input in seProjectInputDataList)
{
   GeoSpatialEncoderTests encoderTests = new GeoSpatialEncoderTests();
   var output = encoderTests.UnitTestGermanyToItalyLongitude(input.Longitude);
   var output1 = encoderTests.UnitTestGermanyToItalyLatitude( input.Latitude);

   // Writing output in a file
   FileUtilities.WriteDataInFile("EncodedFileForUnitTest.txt", input.Longitude, input.Latitude,
       output, output1);
}

// Step 2: Uploading output to blob storage
 var uploadedUri =
     await storageProvider.UploadResultFile("EncodedFileForUnitTest.txt",
       null);

logger?.LogInformation(
    $"Test cases output file uploaded successful. Blob URL: {Encoding.ASCII.GetString(uploadedUri)}");
```

Fig.13: This shows the implementation of Unit-test in Experiment.cs.

In the fig:13 according to Step 2: code portrays implementation of test case output file is upload to a blob in azure service using method UploadResultFile.

V. RESULT AND OUTPUT

After executing the program, the result is Upload to Blob storage space with the following methods:UploadResultFile in the AzureBlobOperations.cs class. The file is later uploaded to Blob storage, where the URI of the file Return. The fig:14 shows Run the code and upload the output from it the local path of the Azure storage.

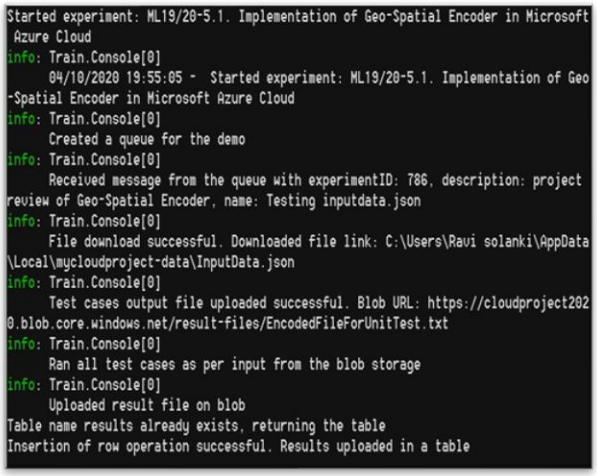


Fig.14: Result from a console at last.

The results obtained will be submitted to Blob and stored in AzureBlobOperations.cs class, and then this process of UploadExperimentResult will start to execute. Results are inserted in the table created by the CreateTableAsync method. The encoded file "EncodedFileForUnitTest.txt" output file is then uploaded from the local path to the Blob storage is As shown in Fig:15 below:

Wise2019	83eaf2f5-c70a-4e1f-bb4b-d1703b1b3269	Sat, 03 Oct 2020 12:50:34 GMT	project review	0000000000000000005	Sat, 03 Oct 2020 12:50
Wise2019	74118ab8-7d40-4b06-b1ba-31b65adb4e66	Sat, 03 Oct 2020 13:39:37 GMT	project review	0000000000000000005	Sat, 03 Oct 2020 13:39
Wise2019	6c1fe4cf-e287-48ef-ad14-c2a2db8ad108	Sat, 03 Oct 2020 14:00:50 GMT	project review	0000000000000000005	Sat, 03 Oct 2020 14:00
Wise2019	f13ff4c8-a516-4482-8de6-411cf62b5089	Sat, 03 Oct 2020 14:13:34 GMT	project review	0000000000000000005	Sat, 03 Oct 2020 14:13

Fig.15: Figure shows the output in table storage.

Fig.16 Displays the encoded file of Geospatial data which latitude and longitude. As shown in the below.

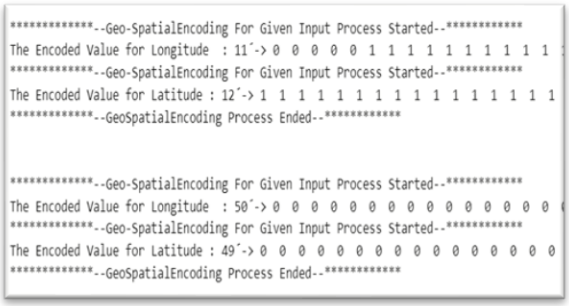


Fig.16: It shows the output of Geo-Spatial Encoder in EncodedFileForUnitTest.txt

VI. REFERENCES

[1] J. H. Subutai Ahmad, Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory, New york: Cornell University, 2015.

[2] Ahmad and Hawkins, "Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex," *Frontiers*, 2016.

[3] H. Subramaniam, "Publishing ASP.NET Core unit test results and code coverage to Azure Devops using Docker Images," 1 march 2019. [Online]. Available: https://medium.com/@harioverhere/running-asp-net-52a6ed92375b.

[4] "Introduction to the core Azure Storage services," [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction?toc=%2fazure%2fstorage%2fblobs%2ftoc.json.

[5] "Introduction to Azure Blob storage," [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.

[6] A. T. storage. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/tables/table-storage-overview.

[7] "Azure queues," [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/queues/storage-queues-introduction.