National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Richard Selvaraj Arokiaraj
Student ID: 22140841

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Richard Selvaraj Arokiaraj
Student ID:	X22140841
Programme:	MSc in Cloud Computing…… **Year:** 2023-2024
Module:	Research Project……
Lecturer:	……
Submission Due Date:	12/08/24
Project Title:	PREDICTION OF OPTIMAL VIRTUAL MACHINE ALLOCATION AND MIGRATION USING MACHINE LEARNING
Word Count:	992 **Page Count:** 5

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Richard Selvaraj Arokiaraj

Date: 08/08/2024

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):	

Configuration Manual

1. Introduction

This is a configuration manual, which provides the detailed step-by-step instructions for the setting up and execution of the implementation and evaluation of a research project associated with the optimization of virtual machine (VM) allocation and migration in cloud environments. The project is performed using Long Short-Term Memory (LSTM) models and Q-learning algorithms to improve the operational efficiency and decision-making processes guiding it. By the end of this manual, the reader is supposed to get the configuration steps to set up the environment, install the necessary software and tools, process data appropriately, and implement machine-learning models to optimize VM allocation and migration in cloud computing systems.

2. Setup Environment

2.1 Creating an EC2 Instance

To create an EC2 instance, first open the AWS Management Console and navigate to the EC2 Dashboard, then click "Launch Instance." Select Amazon Linux 2 as the Amazon Machine Image (AMI) and choose t2.micro as the instance type. Configure the instance details, including network settings and storage, and add tags to name your instance, such as project1011. Set up security group settings to allow SSH access. Review and launch the instance, then download the key pair for SSH access. Use an SSH client to connect to the instance with the downloaded key pair and update the instance by running sudo yum update -y.

2.2 Saving Dataset to S3 Bucket

To save the dataset to an S3 bucket, first navigate to the S3 service in the AWS Management Console. Create a new bucket or select an existing one. Upload the vm_allocation_mitigation.csv dataset to the chosen S3 bucket. After the upload is complete, note the S3 URI for future reference. This URI will be essential for accessing and integrating the dataset during subsequent data processing and analysis tasks.

3. Software and Tools Installation

3.1 Installing Java and Apache Spark

Install Java, Spark, and TensorFlow

In [1]:
```
# Install OpenJDK 8
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# Download Apache Spark 3.0.0
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz

# Extract the downloaded Spark tar file
!tar xf spark-3.0.0-bin-hadoop3.2.tgz

# Set environment variables for Java and Spark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"

# Add Spark to sys.path
import sys
sys.path.append("/content/spark-3.0.0-bin-hadoop3.2/python")
sys.path.append("/content/spark-3.0.0-bin-hadoop3.2/python/lib/py4j-0.10.9-src.zip")

# Install findspark using pip
!pip install -q findspark

# Install TensorFlow and other necessary libraries
!pip install tensorflow pandas numpy matplotlib seaborn scikit-learn
```

Figure 1: Installing Java and Apache Spark

(Source: Acquired from Sagemaker)

To install Java and Apache Spark, start by installing the Java Development Kit (JDK), specifically version 8, to provide the necessary Java environment. Install JDK headlessly to avoid unnecessary graphical components. Next, download and extract Apache Spark, ensuring it is configured to work with Hadoop 3.2. Set environment variables JAVA_HOME and SPARK_HOME to the installation paths of JDK and Spark, respectively. Configure these variables to enable Spark to find the Java installation and integrate properly. Ensure the Spark installation is complete and ready for use in data processing and model training tasks.

3.2 Setting Up Environment Variables

Set environment variables

In [2]:
```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"
```

In [3]:
```
import sys
sys.path.append("/content/spark-3.0.0-bin-hadoop3.2/python")
sys.path.append("/content/spark-3.0.0-bin-hadoop3.2/python/lib/py4j-0.10.9-src.zip")

print("Installation complete. Please restart the kernel to apply environment changes.")
```

```
Installation complete. Please restart the kernel to apply environment changes.
```

Figure 2: Setting Up Environment Variables

(Source: Acquired from Sagemaker)

To set up environment variables, first define JAVA_HOME and SPARK_HOME to point to the installation directories of Java and Apache Spark, respectively. In your shell configuration file (e.g., .bashrc or .zshrc), add the following lines: export JAVA_HOME=/path/to/java and export SPARK_HOME=/path/to/spark. Reload the configuration by running source ~/.bashrc or source ~/.zshrc. Additionally, configure the PATH variable to include the bin directories of both Java and Spark by appending :$JAVA_HOME/bin:$SPARK_HOME/bin to the existing PATH. This setup ensures that Spark can locate Java and operate correctly.

3.3 Setting Up Spark Session

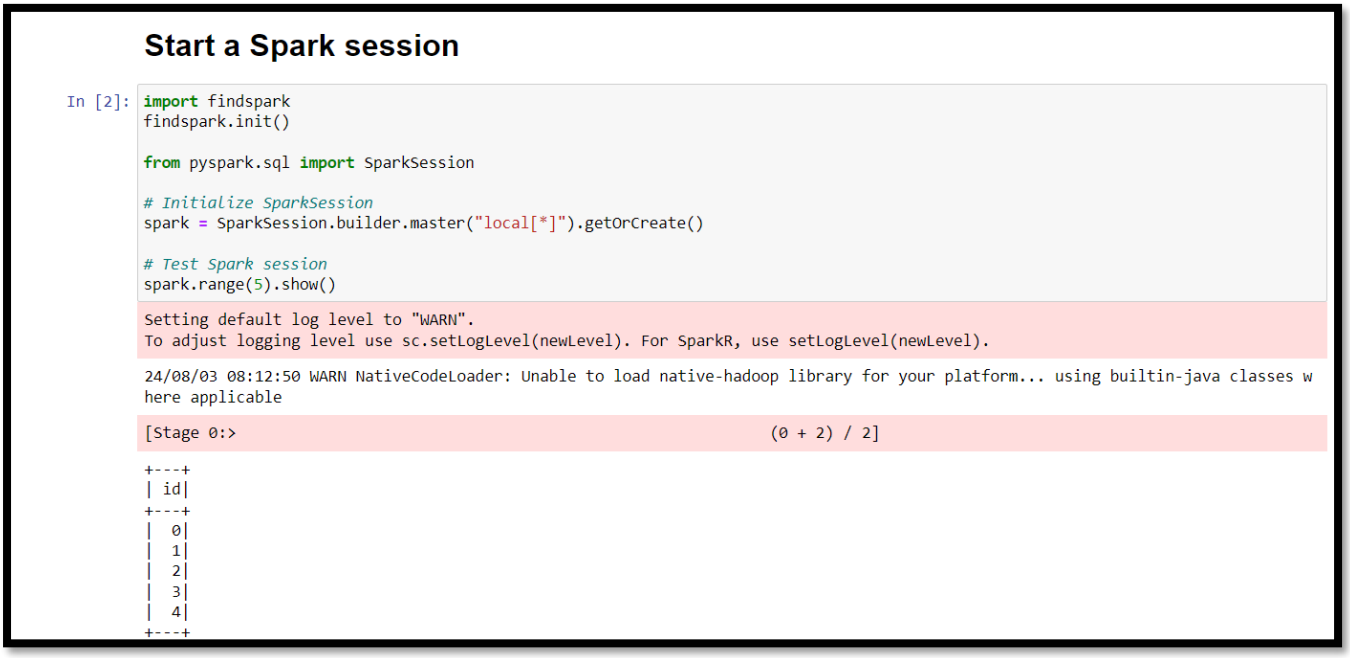

Figure 3: Setting Up Spark Session

(Source: Acquired from Sagemaker)

To set up the Spark session, first ensure that Spark and its dependencies are installed. In your Python script or Jupyter Notebook, import the necessary libraries with import findspark and import pyspark. Initialize Spark with findspark.init() to configure the environment. Create a SparkSession object using from pyspark.sql import SparkSession and spark = SparkSession.builder.appName("YourAppName").getOrCreate(). This setup configures Spark to use the installed environment, allowing you to leverage Spark's distributed computing capabilities for data processing and analysis. Ensure you configure the application name and any additional parameters as needed for your project.

4. Model Implementation and Evaluation

4.1 LSTM Model

Build the LSTM model

In [22]:
```
def build_lstm_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.LSTM(50, input_shape=input_shape, return_sequences=True),
        tf.keras.layers.LSTM(50),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Figure 4: LSTM Model

(Source: Acquired from Sagemaker)

To implement the LSTM model, begin by defining the model architecture with TensorFlow or Keras. Create a Sequential model and add LSTM layers, typically with 50 units each, followed by a Dense layer with a sigmoid activation function for binary classification. Compile the model using the binary cross-entropy loss function and Adam optimizer. Train the model using your dataset with a batch size of 32 and for 20 epochs. Evaluate the model's performance by testing it on validation data to ensure it accurately predicts VM allocation and migration needs. Adjust hyperparameters as needed to improve model accuracy and performance.

4.2 Q Learning

Implement Q-Learning Algorithm

In [2]:
```
import numpy as np
import random
```

Define Q-learning parameters

In [27]:
```
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
epsilon = 0.1  # Exploration rate
```

Figure 5: Q Learning

(Source: Acquired from Sagemaker)

To implement Q-learning, initialize the Q-tables for VM allocation and migration with dimensions corresponding to the number of states and actions. Set parameters: learning rate (alpha) to 0.1, discount factor (gamma) to 0.9, and exploration rate (epsilon) to 0.1. Use the choose_action function to balance exploration and exploitation in decision-making. Update Q-values with the update_q_table function based on rewards received and the Q-values of future state-actions. Train the Q-learning algorithm iteratively, adjusting Q-values to optimize decision-making for VM allocation and migration. Evaluate the model's performance by comparing predicted decisions with actual outcomes.

5. Conclusion

This project demonstrated effective VM allocation and migration optimization using LSTM models and Q-learning algorithms. The LSTM model achieved approximately 90% accuracy in predicting VM allocation and migration needs by capturing temporal dependencies in historical data. In contrast, the Q-learning approach excelled in real-time decision-making, with accuracy rates of 77.3% for VM allocation and 96.8% for migration needs. The comparison highlights LSTM's strength in predictive tasks and Q-learning's adaptability to dynamic conditions. The results underscore the value of combining historical data analysis with adaptive decision-making techniques for efficient cloud resource management.