National College of Ireland

An Exogenous Factor Aware Resource Prediction Model for Auto-Scaling in Cloud

MSc Research Project
Cloud Computing

Priyesh Shah
Student ID: x18207731

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Priyesh Shah
Student ID:	x18207731
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	17/08/2020
Project Title:	An Exogenous Factor Aware Resource Prediction Model for Auto-Scaling in Cloud
Word Count:	6993
Page Count:	21

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:	16th August 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):	

An Exogenous Factor Aware Resource Prediction Model for Auto-Scaling in Cloud

Priyesh Shah
x18207731

Abstract

Auto-scaling is required to handle fluctuating demand in the cloud environment. It helps to add or remove resources based on pre-defined policies without human intervention. Proactive auto-scaling solutions are better than reactive ones to handle fluctuating demand but they mostly use system information and exogenous factors influencing user demand are not considered. In this paper, an exogenous factor aware resource prediction model is presented that works in the Analysis phase of the MAPE control loop. It takes into account exogenous factors to predict the user demand and its corresponding resource requirement. Historical weather and taxi booking data and weather forecast data are used for the analysis of this experiment. Taxi bookings for historical dates having weather conditions i.e. rain similar to the forecast are analyzed to predict the user demand and thereby calculate the resources required. Results show that rain has a notable impact on taxi booking as the number of taxis booked on days having rain is 49% to 61% more and the resources required are 25% to 50% more. Result validation shows that actual vs predicted taxi booking deviation is -12% to 17.5% and the actual vs predicted resource deviation is -17% to 17% which is not significant. The user demand and the resources required can be used to plan an auto-scaling operation for the upcoming days.

1 Introduction

Cloud computing has changed the way organizations spend money on information systems by replacing capital expenses with operational expenses thereby providing greater flexibility and reduced costs. It enabled them to increase and decrease the computational resources as per their need and pay for the resources they use. The National Institute of Standards and Technology (NIST) rightly describes cloud computing as a technological model that is omnipresent, easy to use, available on-demand and in abundance, measured as per the usage, and flexible. *Rapid Elasticity* is one of the 5 essential characteristics of cloud computing identified by NIST (Mell and Grance (2011)). Rapid Elasticity helps users to instantly change the number of resources at the click of a button. Figure 1 shows rapid elasticity in a cloud environment and thus proves the concept of auto-scaling

Estimating the resources required is difficult. Most companies estimate based on expectations but over the years, companies have started considering other factors such as historical demand, upcoming events, etc. However, the estimates often turn out to be incorrect resulting in resource over-provisioning or under-provisioning. Over-provisioning can result in resource wastage and unnecessary costs whereas under-provisioning can lead to performance degradation and unavailability of application (Aslanpour et al. (2017)). Dynamically adjusting the

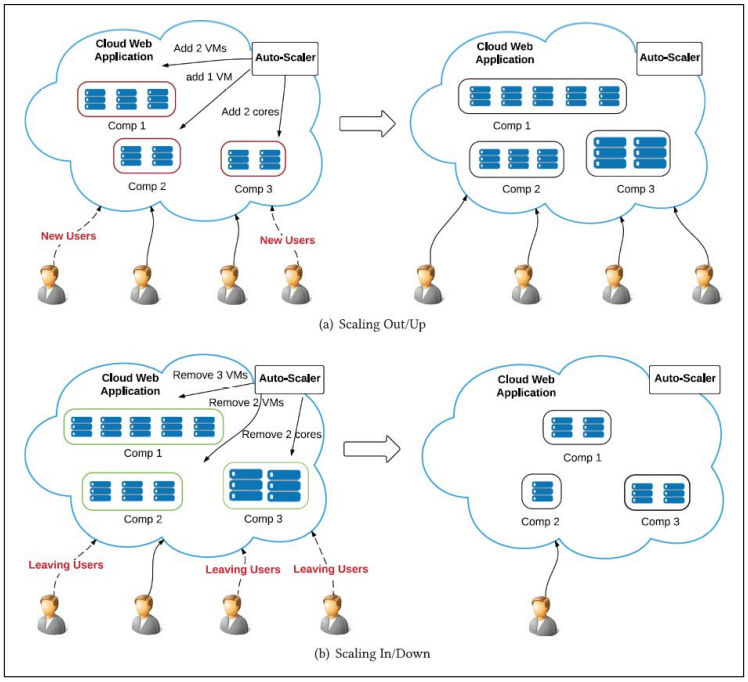


Figure 1: Auto-Scaling Concept (Buyya et al. (2018))

resources is a cumbersome activity as the demand can change frequently. Thus auto-scaling plays a pivotal role in resource management in cloud computing. It allows scaling up and down the resources automatically without any human intervention based on predefined policies. It helps to match the demand and the supply, optimize cost and resource utilization (Smith et al. (2018)) and improve system availability and performance.

Most of the auto-scaling solutions that exist are based on the MAPE control loop which stands for Monitoring, Analysis, Planning, and Execution (Ghobaei-Arani et al. (2018)). Performance metrics used for auto-scaling are monitored in the monitoring phase. The data of the metrics are analyzed in the analysis phase along with workload and resource prediction. The timing when the auto-scaling should be done is determined in the planning phase and the actual execution of auto-scaling is done in the execution phase.

Auto-scaling is mainly categorized into horizontal and vertical scaling. Horizontal scaling means adding or removing additional resources whereas vertical scaling means increasing or decreasing the computational capability of a resource in the system. Auto-scaling solutions can be broadly classified into threshold-based, control and queuing theory-based, time-series based, and reinforcement learning-based auto-scalers (Bauer et al. (2019.1)). Auto-scaling is further divided into reactive and proactive. Reactive auto-scaling means adjusting the number of resources due to an event or an incident in the system. This can be the degradation of response time, breach of threshold values of system parameters, etc. Proactive auto-scaling means identifying or predicting an event or an incident before-hand and adjusting the resources accordingly. Some auto-scalers analyze past workload, resource usage, access logs, social media activity, etc. and adjust the resources accordingly. Making a resource operational takes up to 30 minutes (Smith et al. (2018)). As the reactive auto-scalers react to an anomaly or an event in the system, the possibility of degradation of performance and service level agreement (SLA) violations increases (Iqbal et al. (2018)). The proactive auto-scalers scale the resources based

on the analysis and prediction. However, most of these auto-scaling solutions do not take into account the external factors that can influence the user demand. As suggested (Buyya et al. (2018)), there is a need to find more sophisticated solutions for auto-scaling.

Research Question: Can auto-scaling be planned early based on the number of required resources predicted by analyzing the impact of exogenous factors on user demand?

Through this research, an attempt is made to build a model to predict the number of resources required by analyzing the impact of exogenous factors on user demand. For this project, the weather forecast, historical weather, and taxi booking data are used. The objective of the model is to predict the user demand by analyzing historical and forecast data and thereby calculating the resources that will be required to fulfill the demand. The model developed as part of this project works in the analysis phase of the MAPE loop. It is an independent model that can be integrated with another auto-scaler having planning and execution capabilities.

This paper is divided into various sections. Section 2 discusses auto-scaling taxonomy, related work, and challenges. Section 3 talks about the rationale behind choosing the topic and the process of developing and assessing the model. Section 4 highlights the tools and technologies used, the architecture, and the flowchart. Section 5 mentions the implementation of the model and section 6 describes the evaluation results and its interpretations, result validation, and discussion. Finally, section 7 discusses the conclusion and the future work of the project.

2 Related Work

This section discusses some of the related work in auto-scaling. Section 2.1 talks about the auto-scaling taxonomy and sections 2.2 and 2.3 discuss some of the recent researches done for proactive and hybrid auto-scaling mechanisms. Only reactive auto-scalers have not been discussed as the auto-scaling mechanism implemented in this research project is proactive. Section 2.4 talks about the challenges in auto-scaling and section 2.5 is the conclusion.

2.1 Auto-Scaling: The Taxonomy

Chenhao et al. (2018) have performed a detailed analysis of the recently developed auto-scaling solutions. The researchers have identified the important properties that should be considered while designing an auto-scaling solution. The aspects to be considered while designing auto-scaling mechanisms are displayed in figure 2 and also discussed below:

1. *Application Architecture:* The application architecture that the auto-scaler will handle plays an important role. The architecture can be single-tier, multi-tier, or SOA.

2. *Session Stickness:* Sticky sessions can be a problem as the stateful nature of the web application forces the user to connect to the same server repeatedly which can be a problem in case of scaling of resources. Hence it should be carefully considered.

3. *Adaptivity:* The ability of an application to adapt to the fluctuation in workload, optimize the system performance and resource utilization play a crucial role.

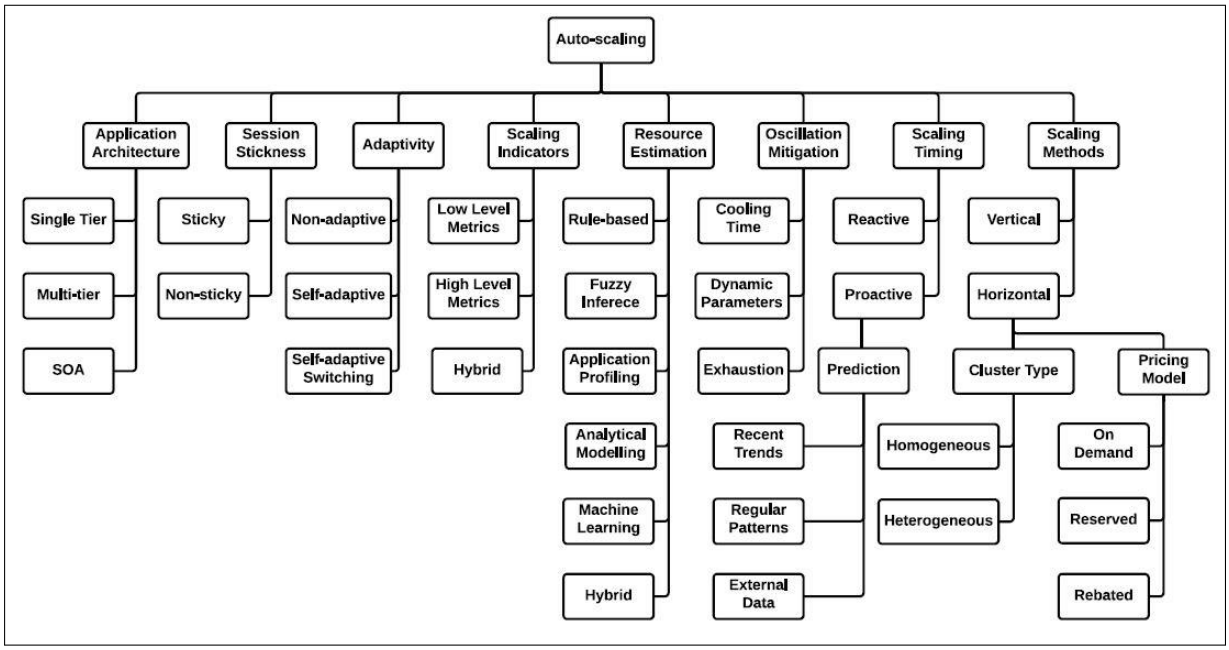


Figure 2: Auto-Scaling: The Taxonomy

4. *Scaling Indicators:* The performance metrics can be low-level metrics observed at hardware-level like CPU and memory utilization, network throughput, etc., high-level metrics observed at application-level such as request and response time, etc., or hybrid.

5. *Resource Estimation:* Estimating the resources required is important to maintain system performance. This can be rule-based, fuzzy, or based on application profiling, analytical modeling, machine learning, or hybrid.

6. *Oscillation Mitigation:* Appropriate provisioning of resources is necessary to avoid over or under-provisioning of resources and also prevents the provisioning or de-provisioning of resources repeatedly.

7. *Scaling Timing:* Reactive auto-scaling works as a corrective action to an event or an incident in the system whereas proactive auto-scaling works proactively based on mathematical evidence to avoid any unwanted event or incident.

8. *Scaling Methods:* Vertical scaling means increasing the computing capacity of the existing resources whereas horizontal scaling means adding additional resources to increase the overall computing capacity.

2.2 Auto-Scaling: Proactive Solutions

Wei et al. (2019) present an auto-scaling model that features a renting plan that considers two keys aspects for self-adaptation, namely, resource pricing and resource type. It is based on Reinforcement Q Learning. It does not require all the information before-hand and dynamically adjusts the resources based on whatever information is available and previous learning. The greedy method and random actions are integrated to generate an ε greedy algorithm. The

experiment was performed using MATLAB and Empirically-Based and Threshold-Based Adjustment algorithms were used for comparison. The results show that the model is unable to match the demand and the supply initially but over a period, with learning from previous decisions, the model fulfills the demand with an adequate supply of resources and the proposed model surpasses the performance of the other two algorithms. However, the model fails to fulfill the demand in the initial days resulting in performance issues.

Smith et al. (2018) have proposed a preemptive auto-scaling solution based on exogenous factors like social media feeds. The researchers aim to improve the latency and the availability of the system while reducing the operational cost to deal with unexpected changes in demand. Specific keywords corresponding to real-world events like traffic conditions, accidents, etc. that can have an impact on the LUAS tram service are monitored on Twitter. It helps in predicting the surge in demand and scale operations proactively. The experiment was performed on AWS using a static website. Dummy tweets were generated on Twitter which was monitored for specific keywords on Twitter feeds. The results show that with an increase in the number of tweets containing the specific keywords, the auto-scaling mechanism is triggered and a server is made operational in 10 minutes. It helps in handling a sudden rise in web traffic and improve the latency and availability of the system. However, the system purely works based on social media hashtags and can give false-positive results as the buzz on social media may not necessarily be a result of a real-world event.

Fe et al. (2017) present a auto-scaling solution that considers the system parameters and the frequency of user requests to evaluate the system throughput, response time, and cost. It uses the Stochastic Petri Nets modeling technique to study the mean response time, type and cost of the virtual machine (VM), their count and job handling capacity, request rate, etc. The experiment was performed on a private cloud using AWS EC2 API using different server configurations. The incoming requests were fired at different intervals and 30 rounds of testing were conducted for each request arrival frequency. The output depicts that t2.micro has a minimum cost but the response time is very bad and t2.small is also similar. Also, a large number of these servers are required to meet the SLA hence increasing the overall cost. Single t2.medium and t2.large can handle a large number of requests, reduce the response time, and meet the SLA thus reducing the overall cost. However, the requests are fired sequentially after a particular time interval which is not an ideal scenario hence the results may not be the same when implemented in real-world applications.

Rizvi and Ramesh (2019) present an auto-scaling framework based on the MAPE loop to dynamically change resources using Fuzzy-Q reinforcement learning. System characteristics like response time and workload are monitored repeatedly. The planning phase uses the Fuzzy-Q method and the analysis phase uses Chebyshev's Inequality principle. It takes the values predicted in the analysis phase and the request rate as input and generates an output which is the decision on the scaling action to be performed. The experiment was performed using the CloudSim toolkit and MATLAB. Real-world traces from NASA are used for the experiment. It is compared with other existing techniques such as LRM. The results depict that the prediction accuracy of Chebyshev's Inequality is better than LRM and helps in reducing the SLA violations and cost by closely matching the supply with the demand. However, the researchers have not taken into consideration the heterogeneity of resources which may have an impact on the prediction results and the overall result may change.

2.2.1 Prediction Based Proactive Solutions

Iqbal et al. (2018) present an unsupervised prediction-based proactive auto-scaling model that determines the workload dynamically scales accordingly. It analyzes a large amount of historical access log of an application to group URIs based on attributes and identifies the access ratio. This data is used to predict the workload for each URI pattern. Neural Networks are used for analysis and prediction. The experiment was conducted using real-world workloads having different request rates like increasing and decreasing linearly, periodic, and random requests. The results indicate that when the amount of access log available for analysis is less, the results are inaccurate and as the size of the access log increases, the accuracy of the workload prediction model increases. However, it requires a huge amount of data to be stored and analyzed repeatedly thus causing high system overhead.

Sahni and Vidyarthi (2016) have proposed a self-adaptive heterogeneity-aware auto-scaling framework to improve resource utilization and minimize costs. The objective is to identify when and how resources should be provisioned to prevent the degradation of Quality of Service (QoS) when there is a change in workload. It does resource profiling and analyzes the historical workload to evaluate the best suitable set of resources. This decision is based on the smallest set of heterogeneous resources having a minimum cost that can meet the QoS. The experiment was conducted in MATLAB using real-world workloads. It is evaluated against ECA-based, Queuing based, and Control theory-based approaches. The results show that the proposed algorithm improves resource utilization by 8% to 25% and the cost reduces by 6% to 23% and it has the least QoS violations. However, significant resource profiling is required to be done as a huge variety of resources are available on any cloud service provider (CSP).

2.3 Auto-Scaling: Hybrid Solutions

Pereira et al. (2019) present a hybrid auto-scaling solution where CPU utilization is the performance metric. The objective is to achieve better QoS. It triggers when the threshold values are breached and applies five different forecasting models. The monitoring module monitors the CPU usage at predefined intervals and writes to a text file. The auto-scaling module transforms the data in a time series and analyses the historical CPU utilization to forecast the usage. It uses the prediction models to predict the usage and the best model is chosen to decide on the scaling operation. The experiment was performed using Apache JMeter to generate a simulation workload. The results indicate that the proposed approach enhanced system throughput by 5.5% to 11.9%. However, it requires continuous monitoring of resources and writing large amounts of data to files which increases the system overhead.

Moghaddam et al. (2019) present an unsupervised hybrid auto-scaling mechanism that detects system anomalies and identifies the cause using the Isolation Trees algorithm. It uses Neural Networks for self-learning. Vertical auto-scaling is triggered when an anomaly is detected for a particular server which does not impact other resources in the system. Horizontal auto-scaling is done when an anomaly that is affecting the entire system is detected. The system information is gathered and analyzed and the auto-scaling method is triggered when an anomaly is predicted. The experiment was performed using CloudSim. The results show that the system triggers the scaling operation as soon as the anomaly is predicted thus ensuring there is no impact on the system performance and no violation occurs. This results in a 20% drop in memory usage which further boosts the system. However, this mechanism requires training the

models and updating them at regular intervals on a large amount of system information thus causing system overhead periodically.

Aslanpour et al. (2017) have proposed a super professional executor (Suprex) model based on the MAPE loop to improve the effectiveness of the execution phase. Resource utilization and response times are monitored for analysis and to plan the scaling operation. During downscaling, the resource is put in quarantine for the remaining billing duration before de-provisioning. For upscaling, it checks resources in quarantine before instantiating a new resource thus reusing resources and reducing cost. The experiment was performed using CloudSim. The model is evaluated against the default and professional executors. The results show quarantining and reusing resources reduces the time and cost to make a resource operational. However, the configuration of the resource is not considered while reusing it so it may prove to be insufficient.

J.V. and Dharma (2018) have proposed a hybrid auto-scaling framework based on the Time Series analysis, Continuous-Time Markov Model (CTMM), and Queuing Model. It calculates the request rate, response time, resource utilization, etc. to accurately predict the resource requirement and avoid resource wastage. The CTMM maintains a balance between the workload and resources allocated. The experiment was performed locally and on AWS using the actual AWS EC2 historical workload traces. The proposed model was evaluated against Queuing Network Model, Threshold-based auto-scaling, RightScale, Autonomic Resource Scaling, vScale, etc. It achieved resource utilization of 94% against the target of 95%. The resource utilization has improved by 10% to 16% whereas the response time has improved by 2.5% to 3% compared to other models. However, the pattern of the EC2 workload traces used for the experiment may not be as diverse as in an actual application so the results may vary.

Bauer et al. (2019) present the Chamulteon framework for horizontal auto-scaling of services. It is an extension of previously designed framework Chameleon (Bauer et al. (2019.1)). It predicts the request arrival rate of the services and triggers the auto-scaler accordingly. It has a cost-aware component Fox and a dynamic request arrival rate forecasting component Telescope. Scaling is done when the request arrival rate breaches the set limits. The experiment was performed using real-world traces. It was compared against Reactive, Hist, Reg, and Adapt models. Results show that Chamulteon has a minimum deviation of 8.9% amongst all. However, it over-provisions the resources for better performance thus increasing cost.

2.4 Auto-Scaling: The Challenges

Chenhao et al. (2018) have highlighted the challenges in designing an auto-scaling solution. The researchers have bifurcated the challenges based on their characteristics as under:

1. *Monitoring Phase:* Identifying the right performance metrics and monitoring intervals is challenging. Monitoring too much can lead to system overhead and monitoring too less can reduce the efficiency of the solution.

2. *Analysis Phase:* Analyzing the data and predicting the workload, incident, or resources is challenging as wrong predictions can lead to false-positive results. This can result in resource wastage and unnecessary costs or performance issues.

3. *Planning Phase:* Deciding the timing of scaling is another challenge. Scaling to soon can again lead to resource wastage and unnecessary costs whereas scaling too late can result in performance issues.

Researchers [Ref.]	Scaling Method	Scaling Timing	Approach	Evaluation Metrics
Wei et al. (2019)	Horizontal	Proactive	Reinforcement Q Based Learning Model	Resource Type & Pricing
Smith et al. (2018)	Horizontal	Proactive	Threshold Based Preemptive Approach	Social Media Feeds
Fe et al. (2017)	Horizontal	Proactive	Stochastic Petri Nets Model	System Throughput, Response Time
Rizvi and Ramesh (2019)	Horizontal	Proactive	Fuzzy Q Based Learning Approach	Response Time, Workload
Iqbal et al. (2018)	Horizontal	Proactive	Dynamic Workload Prediction Model	URI Patterns
Sahni and Vidyarthi (2016)	Horizontal	Proactive	Heterogeneity Aware Adaptive Framework	Resource Profiling, Historical Workload
Pereira et al. (2019)	Horizontal	Hybrid	Threshold and Time Series Approach	CPU Utilization
Moghaddam et al. (2019)	Horizontal	Hybrid	Anomaly Based Cause Aware Framework	CPU & Memory Utilization
Aslanpour et al. (2017)	Horizontal	Hybrid	Cost Aware Super Professional Executor	Resource Utilization & Response Time
J.V. and Dharma (2018)	Horizontal	Hybrid	Hybrid Auto-Scaler	Resource Utilization & Response Time
Bauer et al. (2019)	Horizontal	Hybrid	Coordinated Chamulteon Framework	Request Arrival Rate
Developed Model	Horizontal	Proactive	Exogenous Factor Aware Prediction Model	Exogenous Factors (Weather)

Table 1: Auto-Scaling: Comparative Analysis

4. *Execution Phase:* Every CSP has a unique architecture and hence the auto-scalers need to be designed specifically for CSPs. Designing a generic auto-scaler compatible across the CSPs is a strenuous task.

2.5 Conclusion

The related work indicates that auto-scaling is an area of interest and a significant amount of work is done in this field. Key aspects to be considered while designing any auto-scaling solution are discussed along with the challenges involved. Table 1 summarizes the related work. The evaluation metrics used in the proactive and hybrid solutions are typically confined to system parameters, application-related historical data, and resource type, and pricing. This indicates that there is a need to widen the scope and consider exogenous factors as they can significantly impact the user demand. Exogenous factors like change in weather conditions may lead to a rise or fall in demand. This can lead to resources being under-provisioned or over-provisioned and these circumstances have their disadvantages. Hence, this research attempts to analyze the impact of exogenous factors such as weather on user demand like taxi bookings using weather forecasts and historical weather and taxi booking data. Through this analysis, the resources required will be estimated.

3 Methodology

The related work discussed above indicates that Smith et al. (2018) have considered an external factor but there is a need to include other exogenous factors that can impact the user demand (Buyya et al. (2018)). Through this research, an attempt is made to analyze the impact of external factors on user demand. The exogenous factor considered is the weather and user demand is the taxi bookings. Figure 3 shows the various components of the MAPE loop on which most auto-scaling solutions are developed. The developed model is based on the Analysis phase of the MAPE loop. It works as an analyzer which can be integrated with an auto-scaling mechanism capable of planning and executing the auto-scaling operation or this model can be enhanced in the future to incorporate the planning and execution capabilities.

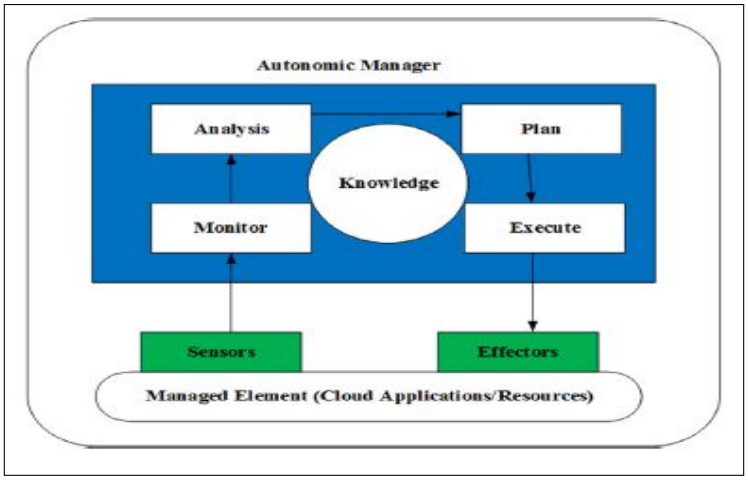


Figure 3: MAPE Control Loop (Ghobaei-Arani et al. (2018))

3.1 Process Flow

The process flow of the model is depicted in figure 4. The historical weather and taxi booking data are downloaded and the weather forecast data is fetched from the OpenWeatherMap API and stored in the MongoDB database. The historical taxi booking data is aggregated to evaluate the number of bookings for each date. The user demand is predicted using the weather forecast and the historical weather and taxi booking data. Based on the predicted value, the resources required are calculated and the result is stored in a CSV file.

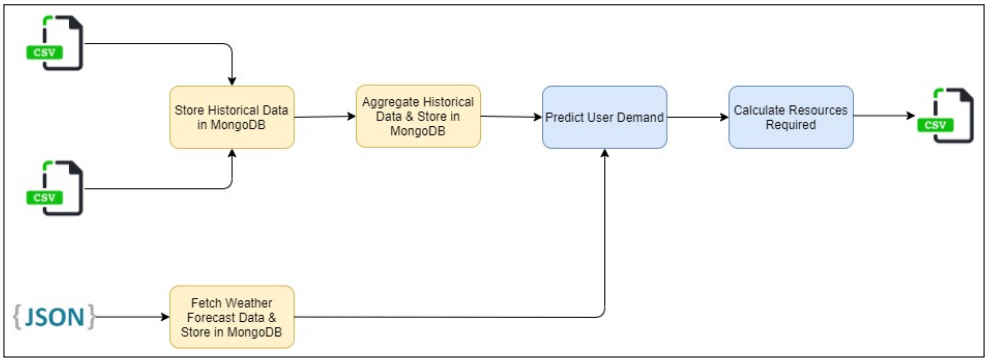


Figure 4: Process Flow

3.2 Tools & Technologies

Python, Apache Spark, MongoDB, and PySpark and PyMongo libraries have been used for developing the project. The decision to use Python programming language, MongoDB database, and Spark is because of the huge size of the data and Python's connectivity with MongoDB and Spark. The project was initially developed and evaluated in a local setup and then evaluated on AWS. A VM with Ubuntu18.04 OS was installed on Oracle VirtualBox on a laptop. All the above-mentioned applications were then installed on the VM. Once the development was completed and the initial evaluation was done, the source code was modified to run the project on AWS. Two AWS EC2 m5.xlarge instances with Ubuntu 18.04 OS are deployed. MongoDB was installed on one instance to use it as a database server. Java, Python, Spark, and the required libraries were installed on the other instance to use it as an application server. Microsoft PowerBI is used for the visualization of the output.

3.3 Data Sources

The data used for this project are collected from multiple sources. For the integrity of the research, trustworthy, authentic, and free data sources are chosen. Due to resource and monetary constraints, data for 2019 is only considered. The data sources of this project are:

1. Historical Taxi Booking Data: The historical taxi booking data is taken from the New York Taxi and Limousine Commission website (TLC (2020)). It is available for free and contains taxi trips details. It does not contain any sensitive user data.

2. Historical Weather Data: The historical weather data is taken from the National Oceanic and Atmospheric Administration (NOAA) website (NOAA (2020)). It is available for free and contains details such as hourly, daily, and monthly weather data. Weather data of the New York Central Park weather station is taken.

3. Weather Forecast: OpenWeatherMap API (OpenWeatherMap (2020)) is chosen to fetch the weather forecast data as it is a popular weather service and some of its services are free. It offers a variety of forecast services. For this project, the daily summary is chosen.

3.4 Assessment

To evaluate the model, two experiments are carried out to ascertain the consistency of the results. The objective is to identify the impact of external factors such as weather on user demand like taxi bookings and calculate the resources required. Also, the week of the day (weekday/weekend) is factored in while analyzing the data for better accuracy. Results are validated against the predicted vs actual taxi bookings and resources required to identify the accuracy of the model and the deviation between the actual vs predicted values.

4 Design Specification

This section discusses the technical specification and the architecture of the model. Section 4.1 talks about the system specification and the rationale behind selecting the tools and technologies. Section 4.2 showcases the detailed architecture and the flowchart of the system.

4.1 System Specification

The performance of an RDBMS system degrades as the amount of data increases hence it is not suitable for data analytics. Apache Hadoop was introduced to process big data as its performance was better than RDBMS, but it is still slow (Yu and Sarwat (2019)). Hence Apache Spark was developed as its speed of processing a very large dataset is better than Hadoop as it comprises in-memory computing. It supports Resilient Distributed Datasets (RDD) which is an abstraction layer for distributed parallel computing (Le Quoc et al. (2019)) and is immutable and fault-tolerant. Python is a popular language amongst the data science community because of the variety of libraries it provides such as NumPy, Pandas, etc. and it is a structured language having simple syntax. PySpark is developed on top of Spark (Le Quoc et al. (2019)) allowing Python scripts to be executed using Spark runtime. It provides a way to connect Spark from Python using the Spark API. The performance and read/write operations of MongoDB in handling large data are better than other databases (Kang et al. (2016)). Also, Python can be easily connected to MongoDB using the connector. Hence Python, Spark, and MongoDB are chosen to implement this project.

The project was initially developed on a local setup. Oracle VirtualBox was installed on a laptop running Windows OS and having Intel i5 8th Gen Processor (4 cores and 8 virtual CPUs). Ubuntu 18.04 OS was installed in the VM and 8GB RAM and 100GB storage was allocated to it. Python, Java, Spark, MongoDB, and PyMongo, PySpark, and other required libraries were installed on the VM. Once the development was completed and the initial evaluation was done, the source code was modified to execute on AWS. Two AWS EC2 m5.xlarge instances with Ubuntu 18.04 OS were deployed. Both these instances had 4 virtual CPUs, 16GB RAM, and 16GB Elastic Block Storage (EBS). MongoDB was installed on one of the EC2 instances and Java, Python, Spark, and PySpark, PyMongo, and other required libraries were installed on another EC2 instance. Microsoft PowerBI is used for the visualization of the output.

4.2 Architecture & Flowchart

The core modules of the model are Historical Taxi Booking Aggregator, User Demand Predictor, and the Resource Calculator. Historical weather and taxi booking data are imported from the CSV files and the weather forecast is fetched using the OpenWeatherMap API and stored in the database. The Historical Taxi Booking Aggregator module aggregates the total taxi bookings for each date. This module prevents aggregation repeatedly while predicting taxi bookings for the forecast days. The User Demand Predictor module is the most important. It extracts the dates from the historical weather data having weather conditions, mainly, rain, similar to the forecast day. This is followed by extracting the taxi bookings for each historical date. The average taxi booking for the forecast day is evaluated by calculating the average of taxi bookings for historical dates. This process is repeated for all forecast days. Finally, the number of resources required is calculated for each forecast day. Figure 5 shows the architecture of the system and figure 6 shows the flowchart. Steps 1, 2, and 4 in the architecture should be executed only when new data is to be uploaded. Flowchart contains details of every operation performed by the model. Actions performed by the model are mentioned below:

1. Import the historical taxi bookings data (CSV files) to MongoDB.
2. Import the historical weather data (CSV files) to MongoDB.
3. Fetch the weather forecast using the OpenWeatherMap API and store it in MongoDB.

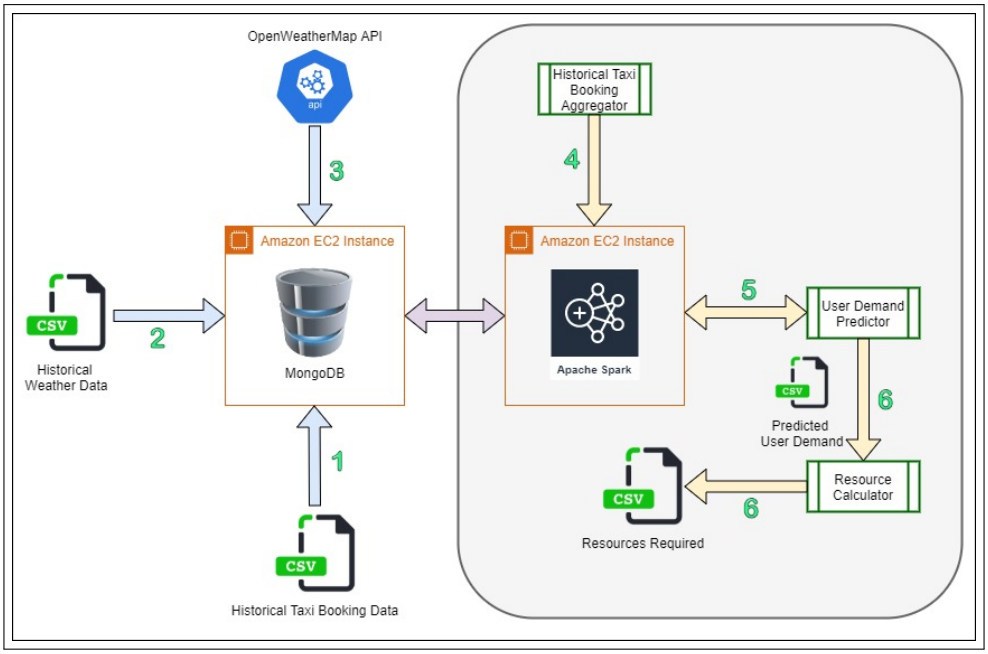


Figure 5: System Architecture

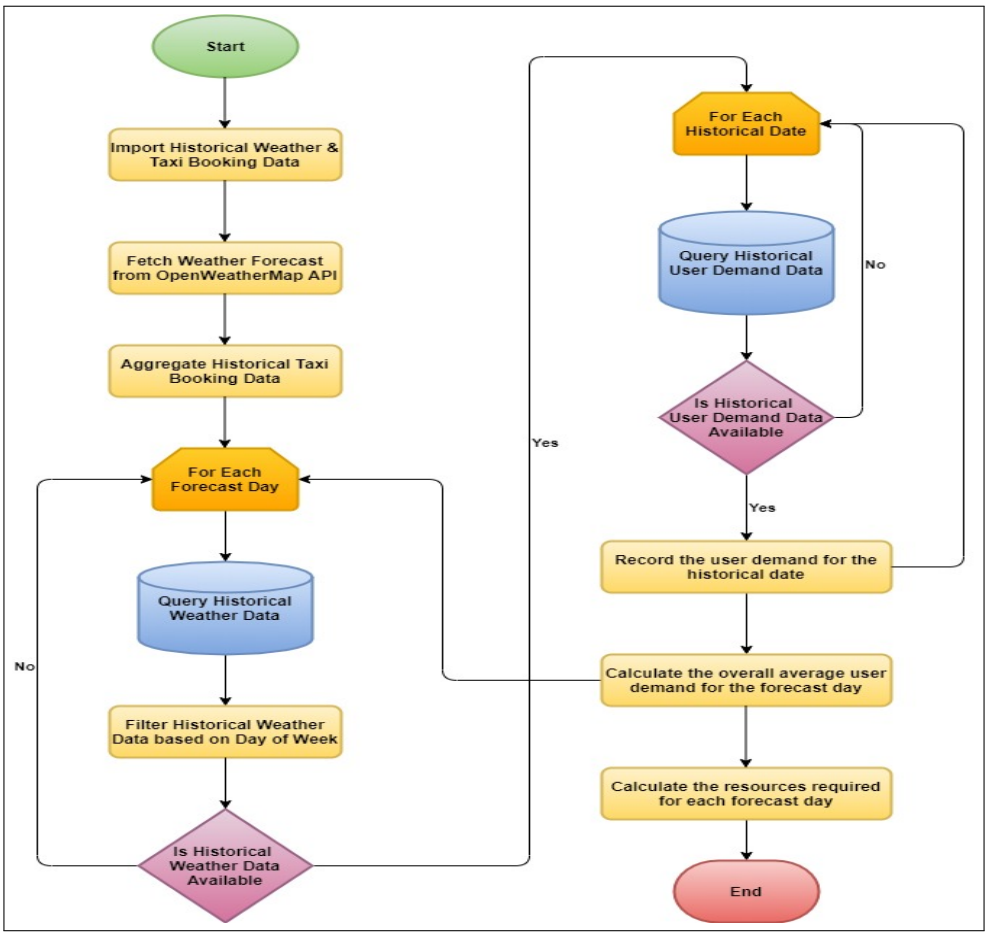


Figure 6: Flowchart

4. Aggregate the taxi bookings for each historical date and store in MongoDB.

5. Predict the user demand for forecast days by evaluating the taxi bookings for historical dates having weather conditions similar to the forecast. Save output in a CSV file.
6. The CSV file generated in the previous step is used to calculate the number of resources required to fulfill the demand. Save output in a CSV file.

5 Implementation

The implementation of the auto-scaling model is discussed in this section. The ETL process involved and its corresponding algorithms are mentioned in section 5.1. The analysis part and its corresponding algorithms are mentioned in section 5.2.

5.1 ETL Process

Data from various sources is gathered in the form of CSV files or JSON response from a RESTful API. This data is cleaned to remove unwanted columns, transformed into required formats, and then stored in MongoDB. Below algorithms highlight the data processing activities:

1. **Algorithm 1** imports historical taxi booking data from CSV files. It parses the CSV file in batches of 100,000 records as each file contains millions of records. All unwanted columns are dropped to reduce the size of the data. It is then converted to JSON format before saving it to a collection in MongoDB.

Algorithm 1: Historical Taxi Booking Data

Input: *CSV File*: Historical Taxi Booking Data CSV File

Open *Historical Taxi Booking Data CSV* File

`// Read CSV file in batches of 100,000 records in each iteration`

for *row in csv_data* **do**

    Drop unwanted columns

    Convert data to JSON format

    Save data in MongoDB collection *historical_taxi_data*

end

2. **Algorithm 2** imports historical weather data from CSV file. All unwanted columns are dropped to reduce the size of the data. The data is filtered to get a daily weather summary. It is converted to JSON format before saving it to a collection in MongoDB.

Algorithm 2: Historical Weather Data

Input: *CSV File*: Historical Weather Data CSV File

Open *Historical Weather Data CSV* File

Drop unwanted columns

$csv\_data \leftarrow csv\_data.filter(TYPE = SOD)$ `// Get Daily Weather Summary data`

Convert data to JSON format

Save data in MongoDB collection *historical_weather_data*

3. **Algorithm 3** fetches the weather forecast for New York. It requires latitude, longitude, and secret key to call the OpenWeatherMap API. The response is parsed and transformed. It is converted to JSON format before saving it to a collection in MongoDB.

Algorithm 3: Fetch Weather Forecast

Input: *api_key, base_url, latitude, longitude*

Form complete URL and request the OpenWeatherMap API
Get the response and convert it to JSON format
Extract the daily summary data from the response
for *i in daily_list* **do**
    $date \leftarrow cast(unix\_datetime)$ `// Convert unix timestamp to date format`
    Save data in MongoDB collection *weather_forecast_data*
end

5.2 Analysis

This is the most critical part of the project. The data is analyzed to predict the user demand and eventually the number of resources required to fulfill the demand. The output of the predicted user demand and the calculated resources required is saved in CSV files for further use.

1. **Algorithm 4** evaluates the number of bookings for each historical date. The timestamp of the booking is converted to date format before aggregating and the results are finally stored in the MongoDB collection.

Algorithm 4: Historical Taxi Booking Aggregator

Input: *historical_taxi_data*: Historical taxi bookings data

Load *historical_taxi_data* `// Load data from MongoDB to PySpark`
$ht\_date\_formatted \leftarrow cast(tpep\_pickup\_datetime)$ `// Cast timestamp to date`
$aggregated\_historical\_taxi\_data \leftarrow groupBy(ht\_date\_formatted)$ `// Aggregate the taxi bookings for each historical date`
Save data in MongoDB collection *aggregated_historical_taxi_data*

2. **Algorithm 5** predicts the taxi bookings for the forecast days based on the weather forecast, historical weather aggregated taxi bookings data. For better prediction, the day of the week (weekday/weekend) is considered. A range is taken for the rain to get historical dates having weather conditions similar to the forecast. The taxi booking is predicted by taking an average of the taxi bookings for the historical dates.

3. **Algorithm 6** calculates the number of resources required to fulfill the predicted user demand. Ocone et al. (2019) had evaluated the number of requests an AWS a1.medium server can handle by using a WordPress site. As per the results, it can handle 30 requests per minute without any failed requests, which meant a 100% success ratio. Hence it is taken as the base to calculate the resources required.

Algorithm 5: User Demand Predictor

Input : *historical_weather_data, aggregated_historical_taxi_data, forecast_data*
Output: *predicted_user_demand_data*: Predicted taxi bookings for forecast days

Load *historical_weather_data*, *aggregated_historical_taxi_data*, *forecast_data* `// Load data from MongoDB to PySpark`
for *forecast_day in forecast_data* **do**
    $forecast\_day\_of\_week \leftarrow cast(forecast\_day.weekday())$ `// Get day of week`
    Convert *forecast_rain* to inches `// To match historical weather data units`
    $forecast\_rain\_upper\_limit \leftarrow (forecast\_rain + (forecast\_rain * 0.15))$
    $forecast\_rain\_lower\_limit \leftarrow (forecast\_rain - (forecast\_rain * 0.15))$
    $historical\_weather\_data.filter(forecast\_rain\_upper\_limit, forecast\_rain\_lower\_limit)$
    if $historical\_weather\_data.count > 0$ **then**
        $historical\_weather\_data.withColumn(day\_of\_week)$ `// Add day of week`
        $join\_data \leftarrow historical\_weather\_data.join(aggregated\_historical\_taxi\_data)$
        if $join\_data.count > 0$ **then**
            if $forecast\_day\_of\_week < 5$ **then**
                $join\_data.filter(day\_of\_week < 5)$
            else
                $join\_data.filter(day\_of\_week >= 5)$
            end
            if $join\_data.count > 0$ **then**
                $predicted\_demand \leftarrow math.ceil(join\_data.groupBy().avg(count))$
                $result\_list.append(forecast\_date, forecast\_rain, predicted\_demand)$
            end
        end
    end
end
Write *result_list* to CSV file

Algorithm 6: Resource Calculator

Input : *CSV File*: User Demand Predictor CSV Output File
Output: *CSV File*: Resource Calculator CSV Output File

Open *User Demand Predictor CSV Output* File
for *row in csv_data* **do**
    Extract *forecast_date, forecast_rain, predicted_demand* from *row*
    $resources\_required \leftarrow math.ceil(predicted\_demand/(30 * 60 * 24))$
    $result\_list.append(forecast\_date, forecast\_rain, predicted\_demand, resources\_required)$
end
Write *result_list* to CSV file

6 Evaluation

This evaluation of the auto-scaling model is discussed in this section. Initially, the evaluation was done in the local set up on a VM and then on AWS. Section 6.1 discusses the evaluation results and its interpretation, section 6.2 highlights the validation of the result, and section 6.3 talks about the detailed findings and the shortfalls of the project.

6.1 Results & Interpretations

The experiments were performed locally as well as on AWS using authentic, free, and publicly available historical weather and taxi bookings, and weather forecast data for New York City. The rain parameter from the historical weather and forecast data is used for evaluation as it can have a notable impact on user demand. Similarly, the day of the week (weekday/weekend) has also been considered to improve the accuracy of user demand prediction and resource calculation. For weekday forecast, only historical weekdays having similar weather conditions are considered and the same logic applies to weekends. 30 requests/minute is taken as the base to calculate the request handling capacity of a server as it has been evaluated by Ocone et al. (2019) through an experiment and documented in an academic research paper. Accordingly, a server can handle 43,200 requests in a day successfully without any error. So, the required resources are calculated by dividing the predicted taxi bookings by 43,200.

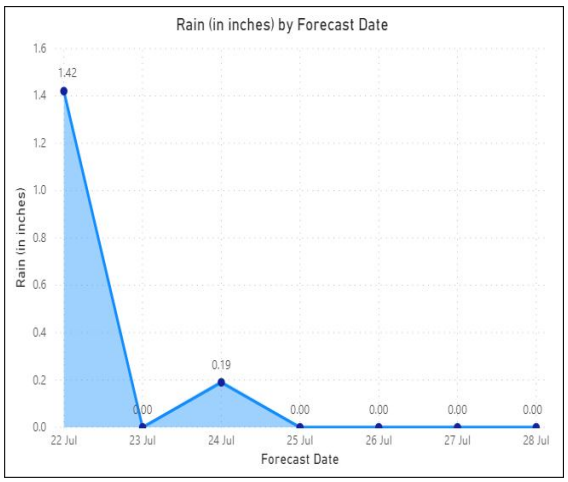
Forecast Date	Rain (in inches)	Predicted Taxi Bookings	Calculated Resources Required
Wednesday, 22 July, 2020	1.42	236842	6
Thursday, 23 July, 2020	0	151412	4
Friday, 24 July, 2020	0.19	243165	6
Saturday, 25 July, 2020	0	182619	5
Sunday, 26 July, 2020	0	182619	5
Monday, 27 July, 2020	0	151412	4
Tuesday, 28 July, 2020	0	151412	4

(a) Experiment 1

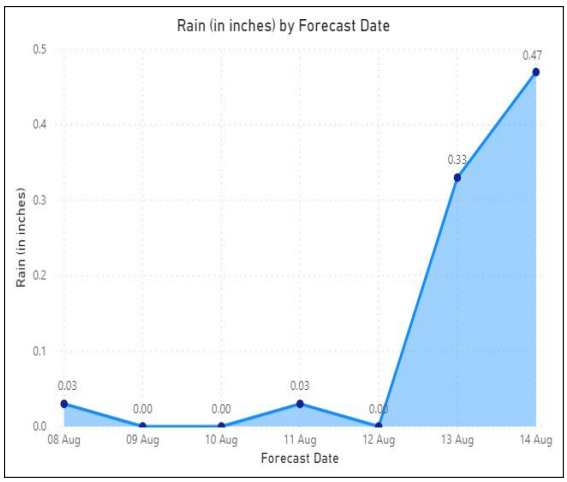
Forecast Date	Rain (in inches)	Predicted Taxi Bookings	Calculated Resources Required
Saturday, 8 August, 2020	0.03	227097	6
Sunday, 9 August, 2020	0	182619	5
Monday, 10 August, 2020	0	151412	4
Tuesday, 11 August, 2020	0.03	234140	6
Wednesday, 12 August, 2020	0	151412	4
Thursday, 13 August, 2020	0.33	243098	6
Friday, 14 August, 2020	0.47	229490	6

(b) Experiment 2

Figure 7: Experiment Results



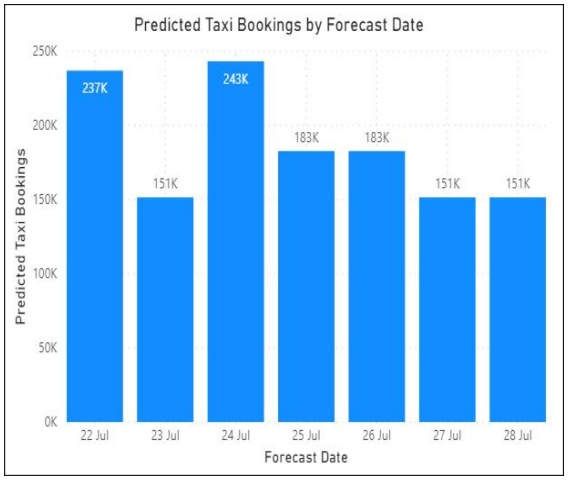
(a) Experiment 1



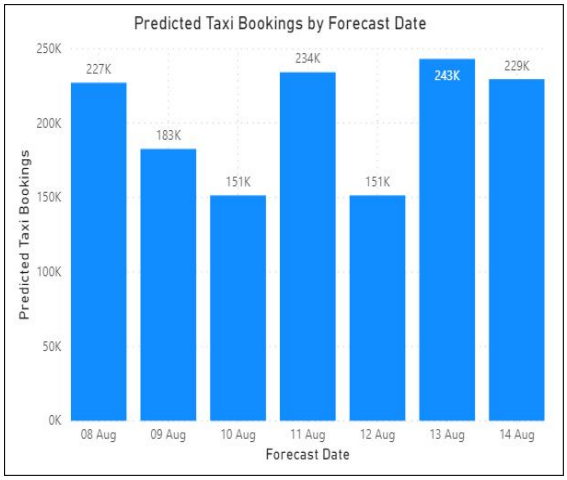
(b) Experiment 2

Figure 8: Rain Forecast Graph

Figure 7 shows the results of the 2 experiments performed on AWS. Figure 7a is of the first experiment whereas figure 7b is of the second experiment. Figures 8, 9, and 10 show the graphical representation of the results. Figures 8a and 8b show the forecast rain, figures 9a and 9b show the predicted taxi bookings, and figures 10a and 10b show the calculated resources required for the two experiments. As seen, it consists of the forecast date, the forecast rain in

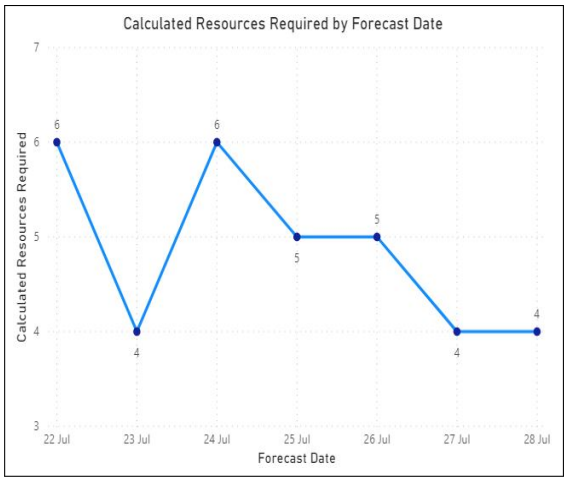


(a) Experiment 1

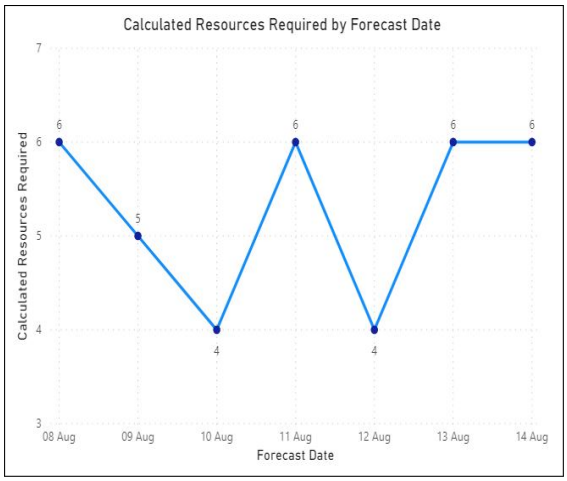


(b) Experiment 2

Figure 9: Predicted Taxi Bookings Graph



(a) Experiment 1



(b) Experiment 2

Figure 10: Calculated Resources Required Graph

inches, predicted taxi bookings count, and the number of resources required. It can be said that the number of taxis booked is related to rainfall. On days when there is no rainfall, there is a decline in the number of taxis booked. The demand for taxis is high when there is rainfall. However, taxi bookings do not increase with an increase in the amount of rain. The number of resources required to fulfill the predicted taxi bookings also varies. On days with no rainfall, the resources required are less compared to the resources required on days having rainfall. A similar pattern is observed on weekdays as well as weekends. The predicted taxi bookings on days having rain increases in the range of 49% to 61% as compared to days without rain and the corresponding resource requirement on days having rain also increase 25% to 50% as compared to days without rain.

6.2 Result Validation

The results obtained above are validated here. As the taxi booking data for 2020 is not available, data from 2019 is used. Dates from 2019 having different amounts of rain are picked and the taxi booking prediction is done for these dates and its corresponding resource requirement is calculated. This is validated against the actual taxi bookings for the respective dates and the resources that would have been required are calculated using the actual taxi booking number. This is shown in figure 11. As seen, the actual vs predicted taxi booking deviation is in the range of -12% to 17.5% and the actual vs predicted resource deviation is in the range of -17% to 17%. Mostly, the deviation is not significant and the value of the predicted resources required either match or exceeds the value of actual resources required value.

Date	Rain (in inches)	Predicted Taxi Bookings	Predicted Resources Required	Actual Taxi Bookings	Actual Resources Required	Actual vs Predictied Taxi Booking Deviation	Actual vs Predictied Resources Deviation
Wednesday, 2 October, 2019	0.03	234140	6	256964	6	9.75	0.00
Saturday, 23 November, 2019	0.03	227097	6	240274	6	5.80	0.00
Friday, 8 February, 2019	0.37	245568	6	288265	7	17.39	16.67
Thursday, 8 August, 2019	0.2	237552	6	209434	5	-11.84	-16.67
Sunday, 28 April, 2019	0.01	222293	6	228245	6	2.68	0.00
Tuesday, 30 July, 2019	0.01	222293	6	224657	6	1.06	0.00
Friday, 30 August, 2019	0	151412	4	162739	4	7.48	0.00
Saturday, 13 July, 2019	0	182619	5	177636	5	-2.73	0.00

Figure 11: Result Validation

6.3 Discussion

Two experiments were performed for different weather forecast dates and it can be observed that the results are similar. The predicted taxi bookings are higher for days having rainfall and lower on days without rainfall. So, the corresponding resource requirement is also higher on days having rainfall and lowed on days without rainfall. This is true for weekdays as well as weekends. Thus it can be said that rain does have an impact on the taxi demand. This calculation of resources required can be used to plan the auto-scaling operation for future dates.

However, the OpenWeatherMap API used for this project offers only a few free services like daily weather forecast summary but does not provide a detailed hourly forecast. Hence, in this research, the taxi booking prediction and the resource calculation is done for an entire day. However, taxi bookings may be higher at the time of rainfall and lower at other times of the day. Due to resource and cost constraints, data for 2019 is only considered as the taxi booking data for 2019 only had over 90 million records. The results may vary when the taxi booking data for a longer duration is used. While fetching historical dates having similar weather conditions, dates from the entire year are fetched. When data of a longer duration is used, the historical dates can be filtered based on months for better accuracy.

7 Conclusion and Future Work

Auto-scaling facilitates rapid elasticity in the cloud environment by enabling businesses to scale up and down the resources automatically based on pre-defined policies thereby avoiding resource wastage and unnecessary costs. However, the auto-scaling policies are largely reactive with some proactive solutions being developed recently. Proactive solutions are better

equipped than reactive ones to avoid performance issues but they generally do not consider external factors. Through this research, an attempt is made to consider external factors that may have an impact and come up with predictions about the user demand and subsequently determine the resources required. This model works in the analysis phase of the MAPE control loop and takes into account historical weather and taxi bookings, and weather forecast data. The taxi bookings on days with rain rose between 49% to 61% and the corresponding resource requirement increased 25% to 50%. The result validation shows that the actual vs predicted taxi booking deviation is between -12% to 17.5% and the actual vs predicted resource deviation is -17% to 17% which is not significant. The user demand prediction and the corresponding resource requirement can be useful in planning the auto-scaling operation for upcoming days. This predicted-based proactive auto-scaling model will help in handling fluctuation in user demand due to exogenous factors. It can be integrated with an auto-scaler that plans and executes the auto-scaling operation based on this analysis.

In the future, the exact time of rain during the day can be considered to predict the user demand more accurately. This will improve the resource utilization further by calculating the resources required during different hours of the day and allocating resources accordingly. The data used for analysis can be extended to cover 10 years or more. Due to this, historical dates from the same month can be used for analysis which will improve the precision of the prediction. Also, different weather conditions such as wind speed, snowfall, temperature, etc can be considered which will refine the prediction. Also, this model can be enhanced in the future to incorporate planning and execution capabilities.

Acknowledgement

I would like to express my sincere gratitude to my mentor Prof. Sean Heeney for going the extra mile and guiding me throughout this project. It wouldn't have been possible without his constant support and motivation. I would also like to thank the Department of Cloud Computing and the Norma Smurfit Library at the National College of Ireland for enhancing our knowledge and making the necessary resources available. Lastly, I would like to express my heartfelt gratitude to my family for having faith in me and persuading me to do my Masters.

References

Aslanpour, M. S., Ghobaei-Arani, M. and Toosi, A. N. (2017). Auto-scaling web applications in clouds: A cost-aware approach, *Journal of Network and Computer Applications* **95**: 26 – 41. JCR Impact Factor: 3.991.
URL: *http://www.sciencedirect.com/science/article/pii/S1084804517302448*

Bauer, A., Herbst, N., Spinner, S., Ali-Eldin, A. and Kounev, S. (2019.1). Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field, *IEEE Transactions on Parallel and Distributed Systems* **30**(4): 800–813. JCR Impact Factor: 3.971.
URL: *https://doi.org/10.1109/TPDS.2018.2870389*

Bauer, A., Lesch, V., Versluis, L., Ilyushkin, A., Herbst, N. and Kounev, S. (2019). Chamulteon: Coordinated auto-scaling of micro-services, *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, (Dallas, TX, USA), pp. 2015–2025. CORE

Ranking: B.
URL: *https://doi.org/10.1109/ICDCS.2019.00199*

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S. and et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade, *ACM Computing Surveys* **51**(5): 1–38. JCR Impact Factor: 5.550.
URL: *https://doi.org/10.1145/3241737*

Chenhao, Q., Calheiros, R. N. and Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey., *ACM Computing Surveys* **51**(4): 1 – 33. JCR Impact Factor: 5.550.
URL: *https://dl.acm.org/doi/10.1145/3148149*

Fe, I., Matos, R., Dantas, J., Melo, C. and Maciel, P. (2017). Stochastic model of performance and cost for auto-scaling planning in public cloud, *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, (Banff, AB, Canada), pp. 2081–2086. CORE Ranking: B.
URL: *https://doi.org/10.1109/SMC.2017.8122926*

Ghobaei-Arani, M., Jabbehdari, S. and Pourmina, M. A. (2018). An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach, *Future Generation Computer Systems* **78**: 191 – 210. JCR Impact Factor: 4.639.
URL: *https://doi.org/10.1016/j.future.2017.02.022*

Iqbal, W., Erradi, A. and Mahmood, A. (2018). Dynamic workload patterns prediction for proactive auto-scaling of web applications, *Journal of Network and Computer Applications* **124**: 94 – 107. JCR Impact Factor: 3.991.
URL: *http://www.sciencedirect.com/science/article/pii/S1084804518303102*

J.V., B. B. and Dharma, D. (2018). Has: Hybrid auto-scaler for resource scaling in cloud environment, *Journal of Parallel and Distributed Computing* **120**: 1 – 15. JCR Impact Factor: 1.185.
URL: *https://doi.org/10.1016/j.jpdc.2018.04.016*

Kang, Y.-S., Park, I.-H. and Youm, S. (2016). Performance prediction of a mongodb-based traceability system in smart factory supply chains, *Sensors* **16**(12): 2126. JCR Impact Factor: 2.475.
URL: *http://dx.doi.org/10.3390/s16122126*

Le Quoc, D., Gregor, F., Singh, J. and Fetzer, C. (2019). Sgx-pyspark: Secure distributed data analytics, *The World Wide Web Conference*, WWW '19, Association for Computing Machinery, New York, NY, USA, p. 3564–3563. CORE Ranking: A*.
URL: *https://doi.org/10.1145/3308558.3314129*

Mell, P. and Grance, T. (2011). *Special Publication 800-145 The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology*.
URL: *https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf*

Moghaddam, S. K., Buyya, R. and Ramamohanarao, K. (2019). ACAS: An anomaly-based cause aware auto-scaling framework for clouds, *Journal of Parallel and Distributed Computing* **126**: 107 – 120. JCR Impact Factor: 1.185.
URL: *http://www.sciencedirect.com/science/article/pii/S0743731518309080*

NOAA, N. (2020). Local climatological data (lcd) — data tools — climate data online (cdo) — national climatic data center (ncdc).
URL: *https://www.ncdc.noaa.gov/cdo-web/datatools/lcd*

Ocone, L., Rak, M. and Villano, U. (2019). Benchmark-based cost analysis of auto scaling web applications in the cloud, *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, (Napoli, Italy), pp. 98–103. CORE Ranking: B.
URL: *https://doi.org/10.1109/WETICE.2019.00027*

OpenWeatherMap (2020). One call api: Weather data for any geographical coordinate - openweathermap.
URL: *https://openweathermap.org/api/one-call-api*

Pereira, P., Araujo, J. and Maciel, P. (2019). A hybrid mechanism of horizontal auto-scaling based on thresholds and time series, *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, (Bari, Italy), pp. 2065–2070. CORE Ranking: B.
URL: *https://doi.org/10.1109/SMC.2019.8914522*

Rizvi, N. and Ramesh, D. (2019). FBQ-LA: Fuzzy based Q-learning approach for elastic workloads in cloud environment., *Journal of Intelligent & Fuzzy Systems* **36**(3): 2715 – 2728. JCR Impact Factor: 1.426.
URL: *https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs18828*

Sahni, J. and Vidyarthi, D. P. (2016). Heterogeneity-aware adaptive auto-scaling heuristic for improved qos and resource usage in cloud environments, *Computing* **99**(4): 351–381. JCR Impact Factor: 1.654.
URL: *https://doi.org/10.1007/s00607-016-0530-9*

Smith, P., González-Vélez, H. and Caton, S. (2018). Social auto-scaling, *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, (Cambridge, UK), pp. 186–195. CORE Ranking: C.
URL: *https://doi.org/10.1109/PDP2018.2018.00033*

TLC, N. (2020). About tlc - tlc.
URL: *https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page*

Wei, Y., Kudenko, D., Liu, S., Pan, L., Wu, L. and Meng, X. (2019). A reinforcement learning based auto-scaling approach for saas providers in dynamic cloud environment, *Mathematical Problems in Engineering* **2019**: 1–11. JCR Impact Factor: 1.145.
URL: *https://doi.org/10.1155/2019/5080647*

Yu, J. and Sarwat, M. (2019). Geospatial data management in Apache Spark: A tutorial, *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, (Macao), pp. 2060–2063. CORE Ranking: A*.
URL: *https://doi.org/10.1109/ICDE.2019.00239*