# HONED RESOURCE SEGREGATION IN CLOUD, FOG AND EDGE COMPUTING USING DATA CONSUMPTION CHURN.

MSc. CLOUD COMPUTING

HARI NARAYANAN SURESH KUMAR
Student ID: 18170625

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | HARI NARAYANAN |
| **Student ID:** | 18170625 |
| **Programme:** | MSc. CLOUD COMPUTING |
| **Year:** | 2019 |
| **Module:** | RESEARCH PROJECT |
| **Supervisor:** | Manuel Tova-Izquierdo |
| **Submission Due Date:** | 12/12/2019 |
| **Project Title:** | HONED RESOURCE SEGREGATION IN CLOUD, FOG AND EDGE COMPUTING USING DATA CONSUMPTION CHURN |
| **Word Count:** | 6287 |
| **Page Count:** | 25 |

| | |
|---|---|
| **Signature:** | |
| **Date:** | 12th December 2019 |

## 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): | |

# HONED RESOURCE SEGREGATION IN CLOUD, FOG AND EDGE COMPUTING USING DATA CONSUMPTION CHURN.

HARI NARAYANAN

**Abstract**

Data in day to day is in upturn trend due to an increase in the usage of technological gadgets. In this avant-garde era, all these data are stored in the cloud and managing these data efficiently in the cloud server is denounce. Even though the cloud servers have paved a way to develop the fog and edge servers, the decisive resource optimization method in all three servers (Cloud, Fog, and Edge) in order to reduce latency in them is still a claiming. I proposed an algorithm that can optimize the resource and hone them by using data analytics methodologies and the algorithm is named as the 'Data churn algorithm' as it hinges on business churn algorithm. This algorithm takes an application logfile from a corresponding application and analyses them based on service time, bytes size, URLs, and etc.. The next step is to make a graph with them to detect a point of variation in the graph to split them into three halves. Each half is then made to fit into Cloud, Fog and Edge servers by an application load balancer that makes the URLs of the application in each instance such as cloud, fog, and edge respectively which is analyzed by the Data churn algorithm. The system performance is calculated by using the tracert command that estimates the latency of the experimented web application. On using the data churn algorithm, the tracert command depicts that there is three-fourth of improvement in latency is observed on comparing with the traditional methods. The following approach successfully reduces latency in the Multilevel servers(Cloud, Fog, and Edge) on comparing with the existing algorithm that they follow.

***Keywords* :** Cloud computing, Data churn Algorithm, Data analytics, Fog computing, Edge computing.

## 1 Introduction

Cloud technologies are reaching their utmost demand due to its on-demand self-service, Instant elasticity, and consistent service. Similar to Cloud servers, the Internet Of Things (IoT) also paves a vital growth in the emergence of technology. All these IoT devices which include mobile devices, sensors, demand new requirements that the existing Cloud technology could not satisfy sufficiently. All these requirements do include a low level of latency, security, mobility support, and location awareness. To suffice all these exactions, The research community has introduced two major technologies such as Fog and Edge computing servers (Shirazi et al.; 2017). Firdhous et al. (2014) describes that the Fog servers are designed in a way to afford the resources that are in Cloud in an omnipresent manner. The reason for coining this name as 'Fog' is that it is closer to the ground level. In a similar way, Fog servers are those that are near in comparing with the end-user devices and the cloud data centers. Fog computing has been designed in a way to meet all the hindrances in the Cloud such as to provide better Quality Of Service, reduced data traffic, low latency and etc. There are few minor drawbacks in the Fog nodes that include high power consumption on comparing with centralized cloud servers, Data consistency, Authentication, and trust issues. Among all these issues, latency in the multilevel servers and resource scheduling between Fog nodes with the cloud servers are major concerns. Algorithms that are used for encryption and security policies makes difficult for devices such as Cloud and Fog to transfer data within themselves and Lack of strong security policies or encryption may lead to data exposure. Edge computing has its own uniqueness on comparing with Fog servers such as increasing the efficiency of infrastructures through its low latency. The absence of an edge server increases the processing time of data thus makes the total process slow in general. Positive aspects of Edge nodes include its agility, reliable connection and quality bandwidth that results in lowering the cost of any organization. The growth of Edge nodes is directly proportional to the IoT devices. It too has some kinds of drawbacks such as requiring additional hardware for IoT devices, only can use a part of the complete data and several security issues. A most common problem faced in both Fog and Edge is which data to place where irrespective of its performance and other concerns.

In 2009, the term cloudlet was introduced that constantly focuses on latency. The author Satyanarayanan et al. (2009) proposes a concept that has a two-tier architecture. The first level architecture is about Cloud nodes and the second is about the Cloudlets also known as Fog servers. The Fog servers are described as widely spread infrastructure components that have their own resources for storage and compute. The storage and compute resources are influenced by nearby mobile computers. The Fog servers act as cache storage that stores only a minimal amount of data that is essentially needed for mobile devices.

The Evolution of these data centers started in the year 1990 when the local physical servers were used to deliver content to the organization. Speech recognition and Cyber foraging emerged in the 20th century that played a considerable role in technological advancements by that time. Peer to Peer computing was developed in 2001 and Cloudlets that are known as fog computing was developed in 2009 by CISCO.

The process of data storage and management in the Cloud, Edge, and Fog servers are distinct. The data in the cloud are stored in physical servers that are owned by a service provider also logical storage pools that help these servers to store and utilize data effectively. Storage types such as block storage or object storage are placed in multiple or single servers. Unlike the cloud nodes, Fog and Edge computing servers do not process with the whole application data hence there is a performance deduction that is observed on latency. Though several resource allocation and scheduling methods (mentioned in Section 2) are already used in these Multilevel servers such as Cloud, Fog and Edge, all of them do not focus on latency reduction. There isn't a common methodology to allocate these resources in an optimized manner which also reduces latency.

In this paper, a common methodology for resource optimization has been proposed through the data input churn. The data that are generated by various domains are broadening in past years. Even though Cloud manages all of them at its best, there is a need to manage them more adroitly and economically as the size of these data increases day by day. All these operations may increases in latency as sometimes it takes more time to deliver the content to the client-server as the functionality increases. The contemporary algorithm known as Data churn Algorithm which is proposed in this paper can be used to classify this monstrous amount of data on the basis of their frequency rate that desires to reduce the latency in Multilevel servers respectively. The categorization of these data based upon its various characteristics can be done using a deep learning model (Churn Analysis) which is also a subset of machine learning. The following methods can help to reduce Latency in Multilevel servers.

**Research Question:**
**Can optimization of resources in Cloud, Fog and Edge servers using a common churn algorithm reduce Latency in them on comparing with existing methodologies?**

The research aims to improve the efficient resource optimization in Cloud, Fog and Edge computing based upon their response time and to allocate them using Data analytic methodologies along with churn analysis. This is undertaken by getting the application logs from the cloud and analyzing them based on various categories as mentioned in section 3 and map the resources to Multilevel servers by using the Data churn algorithm. The performance of the system is evaluated by comparing the application in its initial phase which is before implementing the algorithm and after implementing the algorithm. This focuses on the various computational nodes to use the resources with minimal latency at its utmost efficiency.

This paper is divided into the corresponding sections that give an elaborate understanding of the entire system which is processed by using the data churn algorithm, usage of the system and evaluation of the following approach. Section 2 gives the insights about the various resource optimisation systems process. Section 3 gives a perception about the data churn algorithm process and the way it is handles in the system. Section 4 describes the architecture of the proposed system and its various factors. Section 5 describes about the implementation of the entire systems process. Section 6 evaluates the system and analyse the results of the system and section 7 describes the conclusion and future work.

## 2 Related Work

### 2.1 Cloud Computing Resource Optimization:

Cloud computing comprises four main deployment models such as Software as a service, Infrastructure as a service and Platform as a service. The resource optimization methodology, in general, is focused on Infrastructure as a service. The various kinds of resource optimization techniques in Cloud are listed below

### 2.1.1 Machine learning methods

To maintain the level of latency reduced and to improve the performance of Fog nodes. Zhang et al. (2018) developed numerous sensor streams using correlation co-efficient. This approach comprises of a Dynamic time wrapping algorithm that detects the lag in the sensor streams. To identify the physical location of the node, the event routing algorithm is used, then the existing correlation results are combined. Similarly, The authors Kedia and Lunawat (2018) describes that Input-Output Operation Per Second of the storage stream may be effectively utilized by using the Pearson coefficient and also through supervised and unsupervised learning. These anomaly detection methods help to find the IOPS of every port in order to find the port that transmits data slowly.The architecture mentioned by Kumar and Palaniswami (2012) works as a master-worker plan. The jobs here work on a directed acyclic graph.Rengasamy and Chidambaram (2019) proposed a predictive novel resource allocation framework for cloud servers using an algorithm named Random algorithm that associates cloudlets and servers by appointing a random number to each server. In order to eliminate the requirement of a centralized broker, the list of the server is managed by the client to provide a better resource allocation methodology.

### 2.1.2 Deep Learning resource optimization in Multilevel servers:

Methodologies like Convolution Neural network which is a part of deep learning unifies all computing networks such as Cloud, Fog and Edge. Concepts of Local receptive Field, Activation, shared weights, and Pooling comes under the CNN. It separates all the data that are required for the Edge nodes with that criteria. After that, The RNN processes the data to proceed with predictive analysis in order to optimize edge resources.

### 2.1.3 Resource Mapping:

The optimal load balancing and mapping focus on to provide optimal cloud storage issues where the files are partitioned and stored. Partial replication algorithm is used and this is done to ensure reliability and availability of the data (Al Nuaimi et al.; 2015). The object mapping algorithm proposed by Gan et al. (2010) balances the objects and resources based on a load of storage services. The virtual network resource mapping framework proposed in (Yu et al.; 2008) formulates a path splitting and migration process in order to reduce the computational time.

Multiple nodes such as Cloud, Fog and Edge, an algorithm named polyVine which is a distributed process. This methodology (Samuel et al.; 2013) that sends a single request to an individual provider that allocates resources by giving priority to the unserved requests. This runs sequentially unless every resource task is completed. While in this paper, The author proposed an overall single methodology that segregates the request using Max-flow and Min-cut process. Dipakbhai (2017).

### 2.1.4 Virtual Machine Resource Optimization Techniques:

The Resource optimization techniques based on virtual machine consist of a distributed process for data storage, communication of network and also works on a three-dimensional optimization problem. VM placement places the incoming new request into the hosts and then optimizes the old resources using BFD Algorithm (Best Fit Decreasing Algorithm). In this paper, the author (Younge et al.; 2010) states the resource optimization techniques can be completed by using correct placement, scheduling, and management of the virtual machines. (Sonkar and Kharat; 2016) combines the various workload types that increase the overall server utilization of server resources. The proposed system in this paper focuses on the load balancing algorithm that works with the criteria such as scalability, migration, response time and performance.

### 2.1.5 Traditional Resource Optimization in Cloud:

In order to increase the utilization rate and latency of the Cloud resources, the authorGokilavani et al. (2013a) wants the resources that specify the cloud servers to be in demand. Problems in Resource optimization are scalability, task scheduling, reliability, performance and reallocation of the resources dynamically. Gossip protocol: The nature of this protocol is to perform resource allocation function in the Cloud servers using a distributed middleware architecture. The algorithm distributes the cloud resources that are time-dependant to a group of nodes (Gokilavani et al.; 2013a). The methodology followed in this paper scales with the number of the system but does not scale the number of applications. While executing this protocol, a single node selects the set of other nodes to communicate with each other using the selection function certainly.

Navimipour (2015) compares the Bee's algorithm as a process of how the bees catch their scouts. By applying the same method for scheduling of resources, the data are allocated by using three main functions such as Select, Fitness, and Waggle. The processor here plays the role of scout bees respectively. The fitness function monitors the performance of each task activity over the allocated resources comparing to the same task executing in the group. Bin packing algorithm packs all the objects in a limited size to a bin of specific capacity. The size of the bin should not be less than the greater or less than objects or resources. The weight of the resources and the bin size are the main two factors taken into consideration (Gouda et al.; 2013).

$$\text{Wj =Weight of the item j, C = capacity of each bin}$$

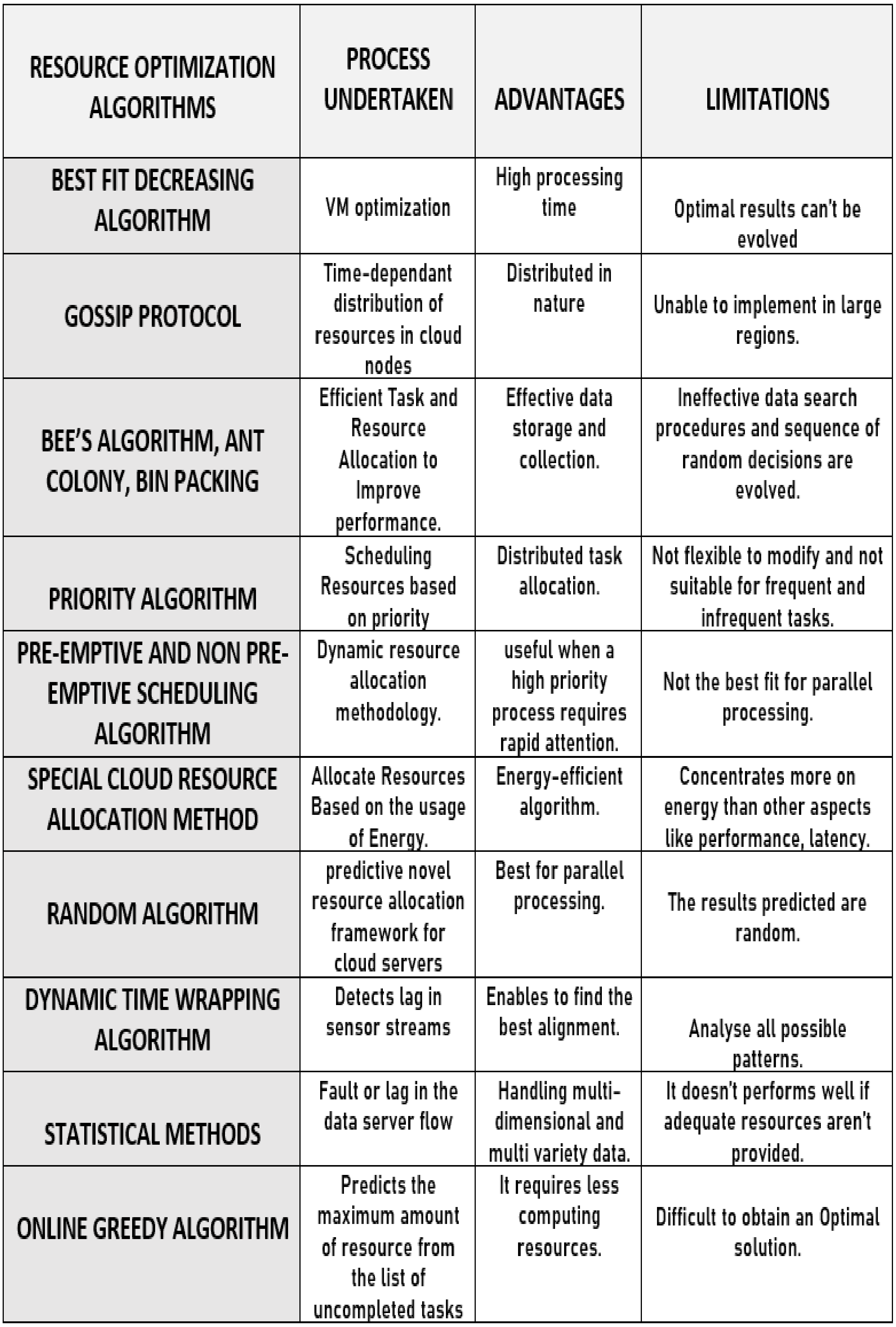
Priority algorithm is used when a job arrives with a cloud scheduler. This scheduler segregates the tasks and these separated tasks are considered to achieve its work based on priorities. Then these resources are allocated to the tasks to perform them in the order of the list (Gokilavani et al.; 2013b).

Nishio et al. (2013) says that Cloud Resource Optimization can be processed ecaciously as it has a wider path for experimentation. The Increase in the storage resources is directly proportional to the data increase. Unlike Cloud, Edge computing does not have vast storage nor data computing power. Researches focus mainly on the ubiquity of fog node. To maintain the widespreadness of Fog computing latency plays a vital role.

### 2.1.6 Fog Resource Optimization:

In this 21st century, The IoT devices are in increasing trend and the data from these devices are growing enormously, this demands in its storage, performance, bandwidth, and latency to an apex level (Santos et al.; 2019). The challenges in Fog nodes as described by Santos et.al are scalability, Bandwidth, energy efficiency and decentralized management.

The Online greedy Algorithm Bi et al. (2019) works with the Fog data centers by taking bandwidth and resource endpoints into consideration. Flexible greedy Algorithm commonly known as FlexOG predicts the maximum amount of resources from the list



| RESOURCE OPTIMIZATION ALGORITHMS | PROCESS UNDERTAKEN | ADVANTAGES | LIMITATIONS |
|---|---|---|---|
| BEST FIT DECREASING ALGORITHM | VM optimization | High processing time | Optimal results can't be evolved |
| GOSSIP PROTOCOL | Time-dependant distribution of resources in cloud nodes | Distributed in nature | Unable to implement in large regions. |
| BEE'S ALGORITHM, ANT COLONY, BIN PACKING | Efficient Task and Resource Allocation to Improve performance. | Effective data storage and collection. | Ineffective data search procedures and sequence of random decisions are evolved. |
| PRIORITY ALGORITHM | Scheduling Resources based on priority | Distributed task allocation. | Not flexible to modify and not suitable for frequent and infrequent tasks. |
| PRE-EMPTIVE AND NON PRE-EMPTIVE SCHEDULING ALGORITHM | Dynamic resource allocation methodology. | useful when a high priority process requires rapid attention. | Not the best fit for parallel processing. |
| SPECIAL CLOUD RESOURCE ALLOCATION METHOD | Allocate Resources Based on the usage of Energy. | Energy-efficient algorithm. | Concentrates more on energy than other aspects like performance, latency. |
| RANDOM ALGORITHM | predictive novel resource allocation framework for cloud servers | Best for parallel processing. | The results predicted are random. |
| DYNAMIC TIME WRAPPING ALGORITHM | Detects lag in sensor streams | Enables to find the best alignment. | Analyse all possible patterns. |
| STATISTICAL METHODS | Fault or lag in the data server flow | Handling multi-dimensional and multi variety data. | It doesn't performs well if adequate resources aren't provided. |
| ONLINE GREEDY ALGORITHM | Predicts the maximum amount of resource from the list of uncompleted tasks | It requires less computing resources. | Difficult to obtain an Optimal solution. |

Figure 1: Represents the interpretation observed from the state of Art methodologies.

of uncompleted tasks by considering the computing time as ‘ti’ for a specific task ‘I’. The FlexOG algorithm derived a result of 90% on comparing with the Social Welfare Maximization Online Auction algorithm. The function of the FlexOG algorithm is to rearrange uncompleted task steps for newly visited tasks.

#### 2.1.7 Edge computing resource Optimization:

The resources that are located in the edge servers functions mainly to enhance the performance and availability of the resources. The Hungarian method (Giang et al.; 2015) used in assignment algorithm enables to optimize the data or resources that are present in the Edge nodes. (Ni et al.; 2017) that uses the cost function to process many features of the system which include Resources that are processed, Latency and prices. This is done by offloading the end devices in order to maximize their life span and also to minimize latency (Zaharia et al.; 2013). On comparing with Fog and Edge computing in multilevel architecture, the priority for edge computing has been high and the static resources that are processed from these two algorithms.

# 3 Methodology

The methodology is exhilarated from the process of Churn rate analysis. The Churn rate can be expressed as a method to calculate the outcome of a processor to calculate the possibility of a specific or collective item in an entity. This Churn rate analysis is commonly undertaken to analyze the customer’s incoming or outgoing rate from a specific product or business. On comparing the same with these multilevel servers, several key performance indicators which can be also known as affinities from the normal process is observed. This correlation between the business churn and server churn rate is described below which the performance of the latency can be determined.

- Application data and its usage: The data that are extracted from the backend of the application that is stored in the database.
- Access of the server data in a Specific time: This specifies the time in which the server experience maximum traffic.
- Response time: Response time from each method (GET/POST) enables to find how long does it take the data packets to travel from server to the client (or) vice-versa.
- User application: The entire workload depends on how the deployed application is created on the basis of its feature (i.e) Larger the application results in a greater server workload.
- Response size: The data packets that can be represented in the form of bytes describes the quantity of data transfer from the server to the client. This is also dependent on Response time.
- Amount of Connections (Request): The amount of connections directly or indirectly describes the activeness of a specific instance.

By the method of Churn data rate analysis in Cloud computing, a new methodology to optimize the Multilevel servers can be evolved and this process also can result in reducing the latency of all the Cloud, Fog and Edge servers.

Data analytics methodologies that is implemented by Mutlag et al. (2019) to reduce latency in Fog computing in order to reduce latency and the Franciska and Swaminathan (2017) and Kawale et al. (2009) described a platform that enables an innuendo for the entire process respectively.All these methodologies combine to form a contemporary algorithm which is named as Data churn Algorithm. The Figure 3 represents the dependence of the key factors of the Churn analysis and the Data churn Algorithm.

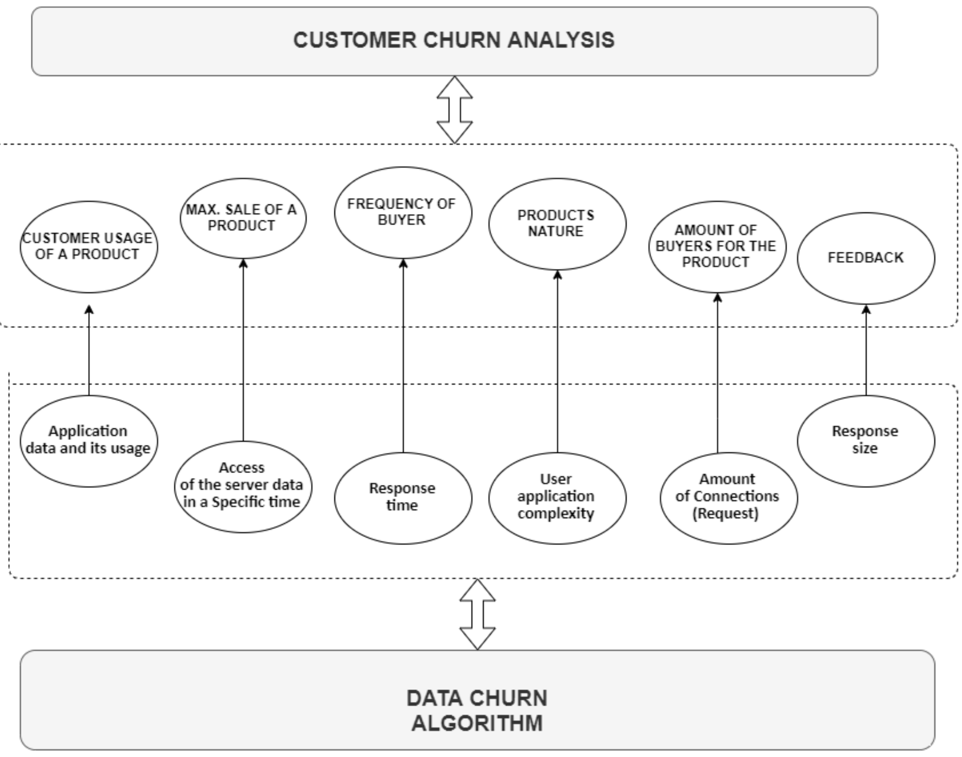

CUSTOMER CHURN ANALYSIS
CUSTOMER USAGE OF A PRODUCT
MAX. SALE OF A PRODUCT
FREQUENCY OF BUYER
PRODUCTS NATURE
AMOUNT OF BUYERS FOR THE PRODUCT
FEEDBACK
Application data and its usage
Access of the server data in a Specific time
Response time
User application complexity
Amount of Connections (Request)
Response size
DATA CHURN ALGORITHM

Figure 2: Relationship representation of Customer/Business churn Analysis and Data churn Algorithm

This process is an antecedent of its own and on comparing with the other resource optimization algorithm. All these algorithms which comprise of similar characteristics only focus on a single type of node. Improving latency in one type of server (either on Cloud, Fog or Edge) does not revamp the latency of the entire application. Henceforth, This algorithm might be a suitable one to reduce the service time. This algorithm is enforced by using a Django framework which is a python REST API framework. The reason for choosing a backend framework as python is because all the data analytics process of the algorithm is occurred by using python.

The main reason for choosing python for this whole process is that the language does have more libraries that are required to perform data analytics operations which I do in data churn algorithm. Python libraries such as pandas, plotly, math and numpy are used in this whole process. This framework gets in a log file of a specific application in a Comma separated file format and then analysis them using the algorithm which is further segregates URL of the application into three main parts to fit into Fog, Cloud and Edge. These processes are explained in the forthcoming sections of 4,5 and 6.

## 4 Design Specification

In order to design the entire system, there are certain components that are required to implement them. The requirements of the system are described below

- A log file for a web application which should be for at least 2 weeks.
- Python

  Libraries in python such as pandas, plotly, math and numpy are used in this whole process.

  Data cleaning and processing is done by using python where all the log files content are taken into CSV format and unwanted details that aren't needed for the research are excluded.
- The algorithm is implemented through Django backend framework to map the log file with the data resources in Multilevel servers.

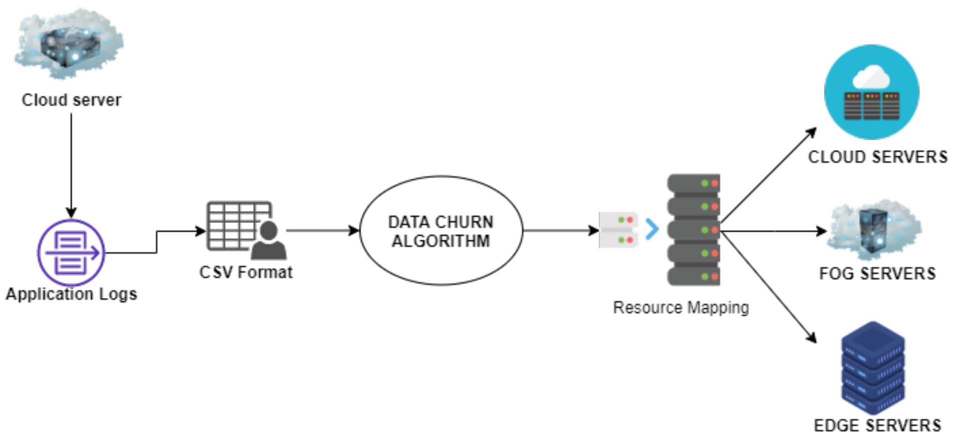


Figure 3: Architecture of the proposed system

The whole process starts from the abstraction of data to the evaluation of results. The tag line of the work flow of each process are described in the Figure 4.

This research is divided into specific stages based on its workflow. The initial level of the project starts with the extraction of application logs from the cloud which is considered experimental data. After that, the data are made into a file of comma-separated value. Then, Various characteristics are analyzed from the converted file and each value is defined by a determining factor in order to categorize. These resources are then classified into a suitable format that can fit into Cloud, Fog and edge servers based the past steps. Then the resource fitting servers are configured such a way that the appropriate resources can be placed in them. The results are then evaluated to prove that this experiment is fit for the given problem.

## 5 Implementation

The implementation of the whole process is described below which comprises of five main phases. The second, third and fourth phase such as resource refinement, logfile encoding, Resource categorization are processed in python language with the help of inbuilt libraries such as pandas, matplotlib, numpy,math and plotly. The process of resource mapping done by using is through the application load balancer (AWS) and each phase is described in detail in Figure 4 and below

1. Resource Extraction
2. Resource refinement
3. Log file encoding
4. Resource categorisation
5. Resource Mapping

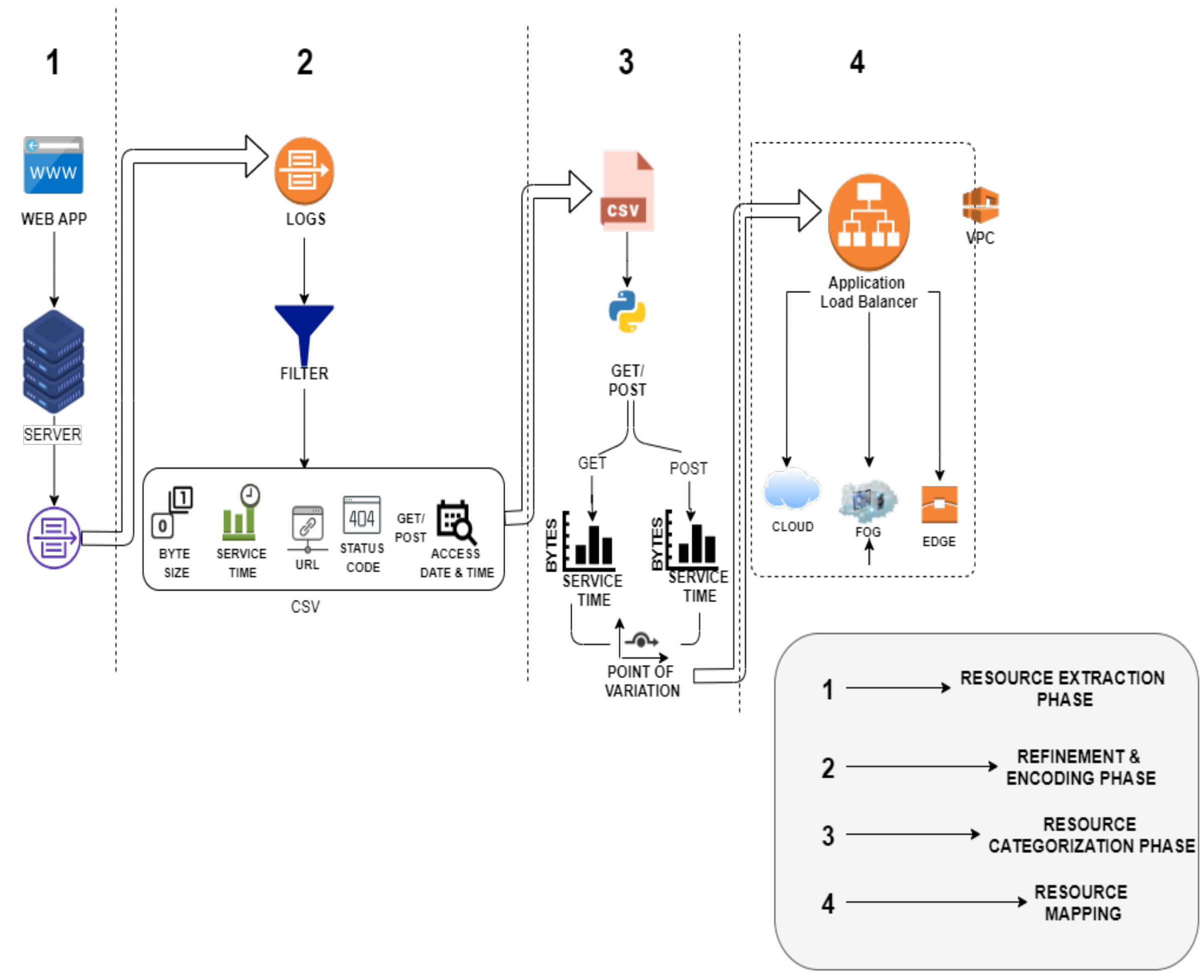


Figure 4: Implementation Workflow of the proposed Architecture

## 5.1 Resource extraction:

The experimental data that are extracted are in the form of logs from a web application. These logs should comprise various information such as IP address, status code, URL that has GET/POST request, request time and etc..

## 5.2 Resource refinement:

In order to convert the generated log file into an accessible format (i.e) into a dataset. The main factors like status code, bytes, date and time of accessing the application are considered on the initial stage of the process then are converted into columns of the dataset by using 'pandas' which is a python library. The start-up and shutdown logs are removed from the dataset which is irrelevant to the research and the dataset can be further denoted as weblogs and router logs.

## 5.3 Logfile encoding:

The accuracy of an algorithm depends on the number of numerical values it persists. Henceforth, the values of these generated data sets are encoded numerically. The columns such as Bytes, Methods, and URL are encoded as below.

### 5.3.1 Bytes Size:

The bytes size is encoded, as there is a region of size starting from the minimal weight to the maximum 5 digit size. Henceforth, To plot the graph with nonchalance, the graph values are encoded as per each 1000 KB.

### 5.3.2 URL encoding:

Each and every URL navigation part are encoded in a way such that there isn't any split that can be observed in the experimented application. This is done by using 'get_dummies' function in python. For example, the second part of the URL link is taken a single numerical value even it is followed by much other navigation further.

## 5.4 Resource categorization:

The categorization of the process is done by using the following Data churn Algorithm in the flow chart and in the the algorithmic representation below

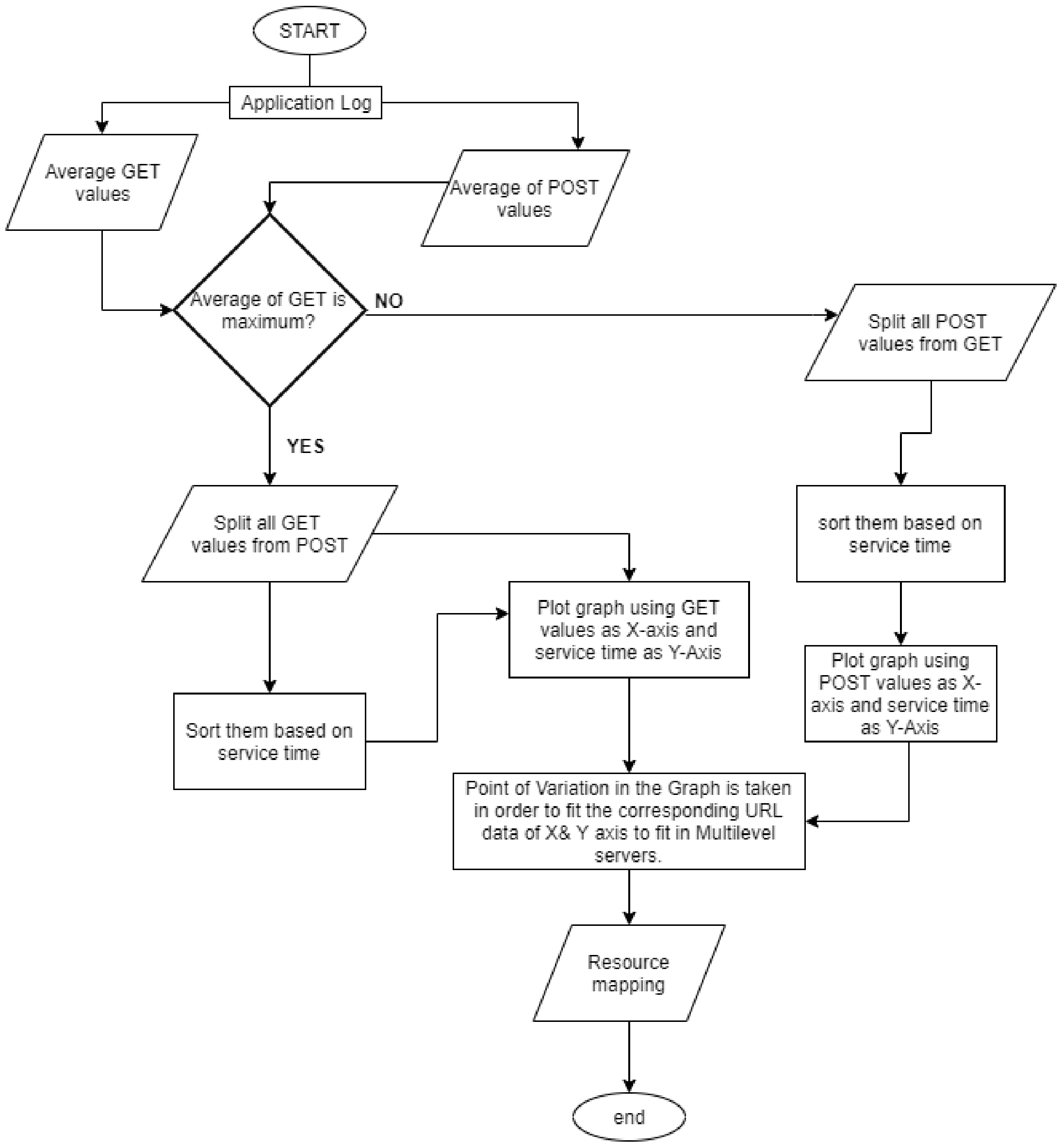

START
Application Log
Average GET values
Average of POST values
Average of GET is maximum?
NO
YES
Split all POST values from GET
sort them based on service time
Plot graph using POST values as X-axis and service time as Y-Axis
Split all GET values from POST
Plot graph using GET values as X-axis and service time as Y-Axis
Sort them based on service time
Point of Variation in the Graph is taken in order to fit the corresponding URL data of X& Y axis to fit in Multilevel servers.
Resource mapping
end

Figure 5: Flow of the Data churn Algorithm

**Algorithm 1** Data churn Algorithm

```
Step1 : Average of all the Bytes values of GET are taken.
Step2 : Average of the Bytes values of POST are taken.
If Average value of GET > Average of POST then follow Step3.
Step3 : Split all the rows that has 'GET' from 'POST'.
Step4 : Take the separated values of GET and Sort them based on their 'service time'.
Step 5: Plot graph using Matplotlib by making
• Service time as X-axis.
• Bytes size as Y-axis.
Firstly, Observe the first POV- Point Of Variation (sudden rise/fall of the pattern) in the graph. Secondly, consider the POV as a splitting factor by undertaking the following two conditions mentioned below.
1 - The peak value observed after or before the POV in the graph goes to cloud.
2 - Remaining values go to FOG and EDGE nodes.
Step6 : Now, to analyze the perfect data fit for Fog and Edge nodes, The process of step 5 is taken in order to split them into Cloud and Fog.
If Average value of POST > Average of GET.
Step7 : Separate the rows that has 'POST' from 'GET'.
Step8 : Take the separated values of POST and Sort them based on their 'service time'.
Step9 : Follow the process of Step5 and 6.
Step10 : The data is decomposed into three major splits and on the basis on its URL column for each split, the whole data is categorized to place into Cloud, Fog and Edge node using a mapping separate algorithm that suits the process respectively. In our case, After following the specified algorithm that is represented above, the following observations are made.
```

## 5.5 Resource Mapping:

The URL columns that are obtained from the Resource categorization phase are to be mapped to the Cloud, Fog and Edge server to evaluate the efficiency of the algorithm. The resource mapping is done once after further refining of the log data so that only accurate data can be placed in the Multilevel servers. The status code such as 200 is not taken into consideration as it states that the request is a success. There are several status codes that are analyzed manually based upon their characteristics and considered to place then in the cloud as it will be better not to place them among one in Cloud, Fog, and Edge.

The reason for opting 304 requests URL to be placed in all the nodes because it gets access to the request but not modified which can be transferred to other servers in case of any failures in any one (i.e) Cloud, Fog and Edge. The Internal server error also known as 500 can be transferred to all the nodes so that it can balance the request failure. The code 503 is also placed in all the three nodes because if one service is unavailable at a cloud, the request can be passed to Fog or Edge to balance the server traffic.

Then once after segregation of the components into data set as described in the steps above, we can now place the URL components to the Multilevel servers. This can be done once after decoding the URL link to the actual one. The web application that is used for experimentation is to be deployed into the Multilevel servers.

The application is now deployed in the cloud by using a load balancer. The chosen load balancer type for this process is application load balancer. This is because it makes an easier way to deploy the application in the EC2 instances and route them into the target groups. In order to connect the application with

3 Autoscaling groups – Each Autoscaling group for an instance where one instance is considered as Cloud, Fog, and Edge in our case. 3 Target groups – Each target group has made to enter three sets of the defined paths of a website. In our case, the divided target splits into three as we have three sets of URLs to store in Cloud, Fog and Edge in the Resource categorization phase. The Target rules are set in the format as in the Rule ID described below

- IF Path is URL SET 1 THEN forward to 1st Autoscaling Group (i.e.) Instance 1
- IF Path is URL SET 2 THEN forward to 2nd Autoscaling Group (i.e.) Instance 2
- IF Path is URL SET 3 THEN forward to 3rd Autoscaling Group (i.e.) Instance3

# 6 Evaluation

## 6.1 Experiment / Case Study 1

The system is experimented by using an apartment management system web application.

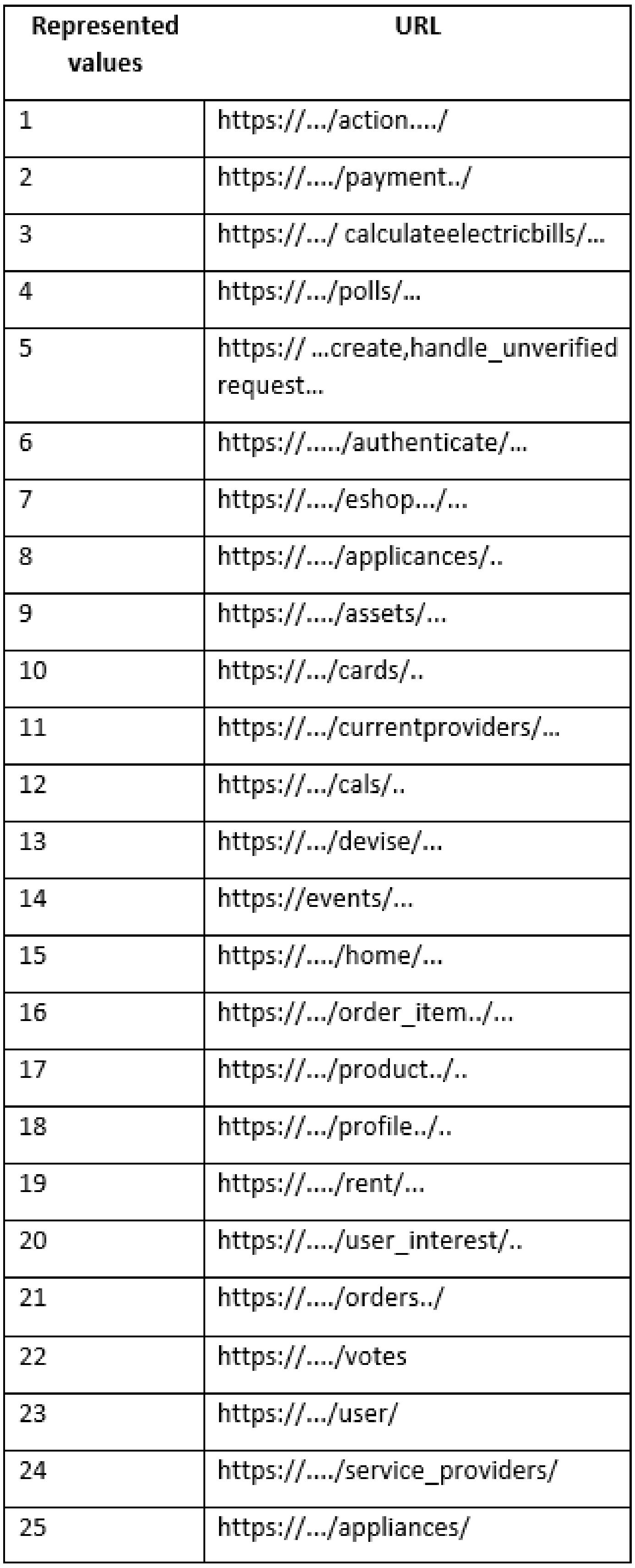
This application comprises various navigation pages starting from the e-commerce page, polling page which is to consider the opinion of the various flatmates, e-bills estimator, and request for a utility in a specific apartment and etc.. that are needed for an apartment to manage.

The application is accessed within around 50 User profiles and the logs of the web application are taken for the experimentation. The deployment of the application is in Heroku. The Bytes size, URL, Status code, Method, Service time and the Date and time of the access are taken considered as the main information from the logs.

The different types of variables are encoded from the CSV file. The Bytes are categorized from 1 to 12 as per their size, Status codes and service time which are in milliseconds remain unaltered since they are already in quantitative measures, URL is encoded in a way such that the application can be classified into parts based upon its module and functionalities. There are around Twenty five main URL parts that are evolved in this apartment management system web application. These URLs are encoded based on their navigation in the application starting from the e-commerce page until the service request as per the Figure 7 below.

Once after obtaining the useful information from the application logs, The refined data are now cleaned to form as a CSV format for further analysis. The algorithm is now implemented to the CSV file but splitting up the GET and POST values. Then the graph is plotted by using a matplotlib library in python and after that, the POV is observed in given the graph in the Figure 10 and 11 and the tabluar observation of the graph from which the result had been obtained is also represented table as Figure 8 and 9.

In Figure 10 of the POST values, Firstly, The service time before 0.37ms and the values of the bytes above 5(encoded) are taken and placed into Edge server by referring

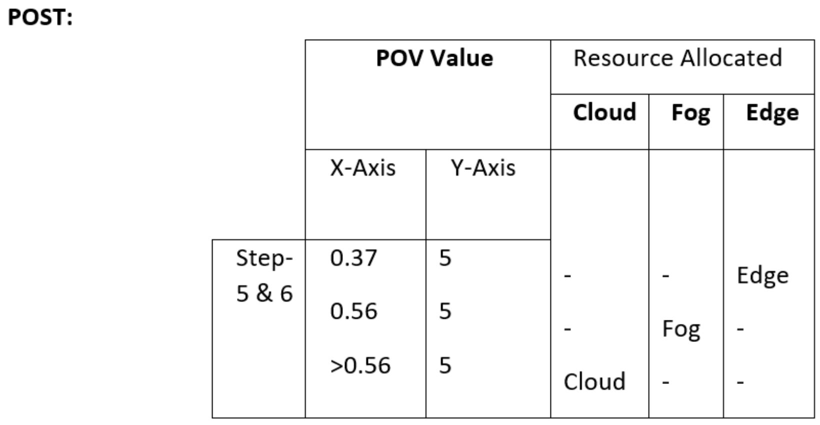


| Represented values | URL |
|---|---|
| 1 | https://.../action..../ |
| 2 | https://..../payment../ |
| 3 | https://.../ calculateelectricbills/... |
| 4 | https://.../polls/... |
| 5 | https:// ...create,handle_unverified request... |
| 6 | https://...../authenticate/... |
| 7 | https://..../eshop.../... |
| 8 | https://..../applicances/.. |
| 9 | https://..../assets/... |
| 10 | https://.../cards/.. |
| 11 | https://.../currentproviders/... |
| 12 | https://.../cals/.. |
| 13 | https://.../devise/... |
| 14 | https://events/... |
| 15 | https://..../home/... |
| 16 | https://.../order_item../... |
| 17 | https://.../product../.. |
| 18 | https://.../profile../.. |
| 19 | https://..../rent/... |
| 20 | https://..../user_interest/.. |
| 21 | https://..../orders../ |
| 22 | https://..../votes |
| 23 | https://.../user/ |
| 24 | https://..../service_providers/ |
| 25 | https://.../appliances/ |

Figure 6: Encoded deatils of URLs

the URL of the corresponding rows and Secondly, The service time values starting from 0.56ms and bytes of 5(encoded) are placed on to the Fog servers. Finally, the last values on the table that are greater than 0.56ms are placed to the Cloud servers respectively which are noted in the table (Figure 7).

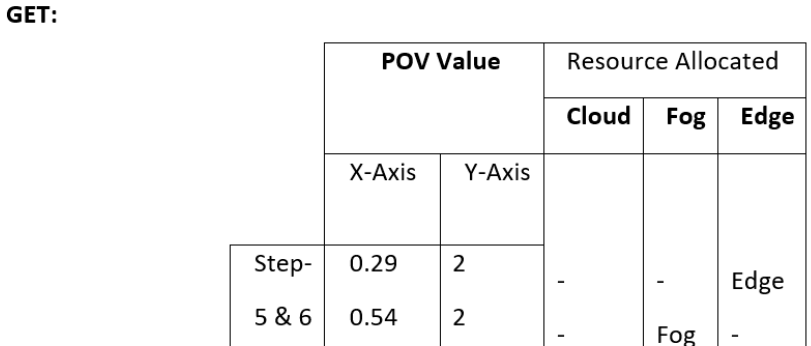
In Figure 9 of the GET values, Firstly, The service time before 0.29ms and the values of the bytes above 2(encoded value) is taken and placed into Edge server by referring the URL of the corresponding rows and Secondly, The service time values starting from 0.54ms and bytes of 2 (encoded) are placed on to the Fog servers. Finally, the last values on the table that are greater than 0.54ms are placed to the Cloud servers correspondingly which are noted in the table (Figure 8).



**POST:**

| | **POV Value** | | Resource Allocated | | |
|---|---|---|---|---|---|
| | | | **Cloud** | **Fog** | **Edge** |
| | X-Axis | Y-Axis | | | |
| Step-5 & 6 | 0.37 | 5 | - | - | Edge |
| | 0.56 | 5 | - | Fog | - |
| | >0.56 | 5 | Cloud | - | - |

Figure 7: Observation of POV in POST Method



**GET:**

| | **POV Value** | | Resource Allocated | | |
|---|---|---|---|---|---|
| | | | **Cloud** | **Fog** | **Edge** |
| | X-Axis | Y-Axis | | | |
| Step-5 & 6 | 0.29 | 2 | - | - | Edge |
| | 0.54 | 2 | - | Fog | - |
| | >0.54 | 2 | Cloud | - | - |

Figure 8: Observation of POV in GET data Method

After splitting up of the tuples in the data sets into three halves, the URL column now plays a vital place. The URL columns which are ranged from 1 to 25 are decoded and these parts of three are taken and place in Cloud, Fog and Edge server that is described in the next paragraph.

The first set of URLs that is to be placed into Cloud (as per the Figure 7) are placed into an EC2 instance of Amazon Web service Northern Virginia location and the Fog and Edge are placed into two separate instances such as London and in Dublin respectively.

Then the Load balancer part of the application has three autoscaling groups and the URL which have divided into three main parts are placed consecutively in three target groups (autoscaling groups). Three target groups have three EC2 instances in general and we also know that one autoscaling group has one instance. After this step, the system is taken into the Performance Evaluation phase.

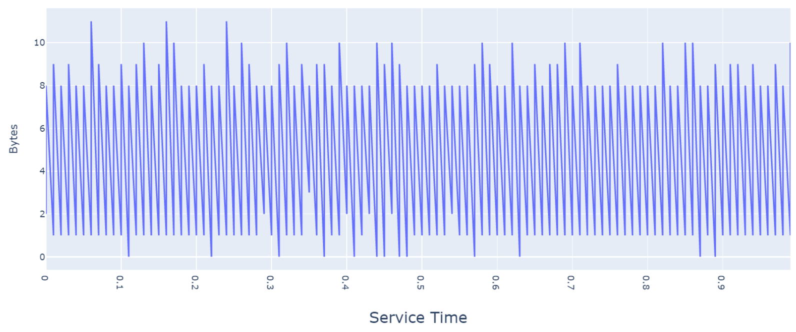


Figure 9: Graphical representation of GET using Data churn Algorithm.

To test the Latency of the system for performance evaluation, The web application is deployed in Heroku and AWS. The system is evaluated by using tracert command used for measuring the number of hops and the Round Trip Time (RTT) needed for the application to reach the client to server.

## 6.2 Result and Interpretation:

- **Phase 1 (Before Implementation):** The web application without the algorithm is deployed into AWS . Then, the application is tested by using 'tracert' command. This is done to evaluate the number of hops the website possess in order to measure the latency. The performance of latency is calculated and plotted as a graph in Figure 11.

- **Phase 2 (After Implementation):** Once after the implementing the algorithm in the entire setup, the performance evaluation of the system is made by the same 'tracert'command. The hops have been calculated and then the latency from the Phase 1 is reduced on comparing with the phase two. The performance of latency is calculated and plotted as a graph in Figure 11.

From the above interpretation, It is evident that the latency is reduced to **three-fourth** in phase-2 on comparing with phase-1 in the experimented web application (**Figure 12**). Henceforth,we can prove that the latency of Multilevel servers can be reduced by using the Data churn Algorithm. This Data churn algorithm also acts as a common algorithm to optimize resources in all three types of multilevel server nodes which motives in latency cutback.

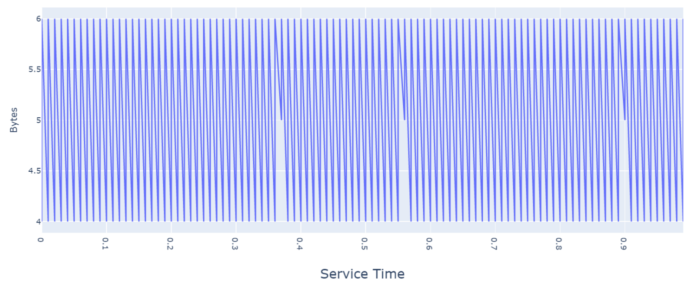


Figure 10: Graphical representation of POST using Data churn Algorithm.

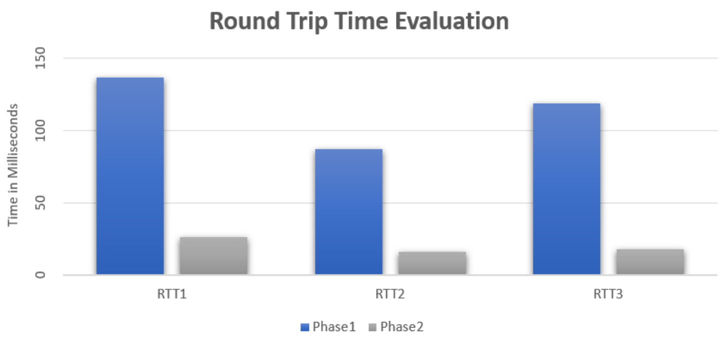


Figure 11: Comparison of latency using tracert at Phase-1 and Phase-2.

## 6.3 Discussion

The algorithm is barred to weighed only on a narrowed space. This Data churn algorithm can be further evaluated into a large application that has higher functionalities. The server technologies used here are not the explicit Fog and Edge servers as it is not feasible to experiment one in the finite source. Instead of testing with a Fog server, a normal instance is used also that the data that are stored in the specific instance supports the similar characteristics that a Fog server has. The edge server is configured by using a cloud front, which is a service of AWS. Even though, the algorithm is evaluated in a small type of application this can be a continual process by using continuous integration and deployment respectively.

## 7 Conclusion and Future Work

Data churn algorithm has drafted in a way to reduce the latency in the Multilevel servers such as Cloud, Fog and Edge. This algorithm can be a potential prevalent one that can be used in all types of server nodes. Currently, the present standards of resource optimization and honing methods intensify only on a single multilevel server type node. Even though this process cannot be an instantaneous process, it can be exploited once after the server starts receiving the data or information from the application. This is in order to receive the logs initially then to make an efficient analysis with the application.

In the future, Once this algorithm is carried out to any high functionality application, this process can also be done in a timely manner by using anomaly detection methodologies. Due to several deteriorations in the accessibility of the Fog server in the system, the replica of the server that has the similar characteristics are used for deployment. Log analysis can be done in a different way apart from using the service time and bytes by using several deep learning methods for various purposes of analysis of text and videos that are present in the server nodes to make them fit in an optimized way.

## 8 Acknowledgement

I would like to thank my project supervisor Prof. Manuel Tova-Izquierdo, without whom this work would not be possible and also would like to thank the department of cloud computing for their strong academic support. Also, I would like to thank the National College of Ireland for making me gain knowledge and supporting me for this research.

## References

Al Nuaimi, K., Mohamed, N., Al Nuaimi, M. and Al-Jaroodi, J. (2015). Partial storage optimization and load control strategy of cloud data centers, *The Scientific World Journal* **2015**.

Bi, F., Stein, S., Gerding, E., Jennings, N. and La Porta, T. (2019). A truthful online mechanism for allocating fog computing resources, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1829–1831.

Dipakbhai, P. J. (2017). Network virtualization based framework for smart grid communication.

Firdhous, M., Ghazali, O. and Hassan, S. (2014). Fog computing: Will it be the future of cloud computing?, The Third International Conference on Informatics & Applications (ICIA2014).

Franciska, I. and Swaminathan, B. (2017). Churn prediction analysis using various clustering algorithms in knime analytics platform, *2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS)*, IEEE, pp. 166–170.

Gan, Y., Han, S., Chen, G. and He, Z. (2010). A research of object mapping algorithm based on cloud storage, *5th International Conference on Pervasive Computing and Applications*, IEEE, pp. 228–231.

Giang, N. K., Blackstock, M., Lea, R. and Leung, V. C. (2015). Developing iot applications in the fog: A distributed dataflow approach, *2015 5th International Conference on the Internet of Things (IOT)*, IEEE, pp. 155–162.

Gokilavani, M., Selvi, S. and Udhayakumar, C. (2013a). A survey on resource allocation and task scheduling algorithms in cloud environment, *International Journal of Engineering and Innovative Technology (IJEIT)* **3**(4).

Gokilavani, M., Selvi, S. and Udhayakumar, C. (2013b). A survey on resource allocation and task scheduling algorithms in cloud environment, *International Journal of Engineering and Innovative Technology (IJEIT)* **3**(4).

Gouda, K., Radhika, T., Akshatha, M. et al. (2013). Priority based resource allocation model for cloud computing, *International Journal of Science, Engineering and Technology Research (IJSETR)* **2**(1): 215–219.

Kawale, J., Pal, A. and Srivastava, J. (2009). Churn prediction in mmorpgs: A social influence based approach, *2009 International Conference on Computational Science and Engineering*, Vol. 4, IEEE, pp. 423–428.

Kedia, R. and Lunawat, A. (2018). Artificial intelligence based storage management architecture, *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, IEEE, pp. 110–114.

Kumar, V. V. and Palaniswami, S. (2012). A dynamic resource allocation method for parallel dataprocessing in cloud computing, *Journal of computer science* **8**(5): 780.

Mutlag, A. A., Ghani, M. K. A., Arunkumar, N. a., Mohamed, M. A. and Mohd, O. (2019). Enabling technologies for fog computing in healthcare iot systems, *Future Generation Computer Systems* **90**: 62–78.

Navimipour, N. J. (2015). Task scheduling in the cloud environments based on an artificial bee colony algorithm, *International Conference on Image Processing*, pp. 38–44.

Ni, L., Zhang, J., Jiang, C., Yan, C. and Yu, K. (2017). Resource allocation strategy in fog computing based on priced timed petri nets, *ieee internet of things journal* **4**(5): 1216–1228.

Nishio, T., Shinkuma, R., Takahashi, T. and Mandayam, N. B. (2013). Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud, *Proceedings of the first international workshop on Mobile cloud computing & networking*, ACM, pp. 19–26.

Rengasamy, R. and Chidambaram, M. (2019). A novel predictive resource allocation framework for cloud computing, *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, IEEE, pp. 118–122.

Samuel, F., Chowdhury, M. and Boutaba, R. (2013). Polyvine: policy-based virtual network embedding across multiple domains, *Journal of Internet Services and Applications* **4**(1): 6.

Santos, J., Wauters, T., Volckaert, B. and De Turck, F. (2019). Resource provisioning in fog computing: From theory to practice, *Sensors* **19**(10): 2238.

Satyanarayanan, M., Bahl, V., Caceres, R. and Davies, N. (2009). The case for vm-based cloudlets in mobile computing, *IEEE pervasive Computing* .

Shirazi, S. N., Gouglidis, A., Farshad, A. and Hutchison, D. (2017). The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective, *IEEE Journal on Selected Areas in Communications* **35**(11): 2586–2595.

Sonkar, S. and Kharat, M. (2016). A review on resource allocation and vm scheduling techniques and a model for efficient resource management in cloud computing environment, *2016 International Conference on ICT in Business Industry & Government (ICTBIG)*, IEEE, pp. 1–7.

Younge, A. J., Von Laszewski, G., Wang, L., Lopez-Alarcon, S. and Carithers, W. (2010). Efficient resource management for cloud computing environments, *International Conference on Green Computing*, IEEE, pp. 357–364.

Yu, M., Yi, Y., Rexford, J. and Chiang, M. (2008). Rethinking virtual network embedding: substrate support for path splitting and migration, *ACM SIGCOMM Computer Communication Review* **38**(2): 17–29.

Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S. and Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale, *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, ACM, pp. 423–438.

Zhang, Z., Liu, C., Zhang, S., Li, X. and Han, Y. (2018). A service-based method for multiple sensor streams aggregation in fog computing, *Wireless Communications and Mobile Computing* **2018**.

# 9 Appendix

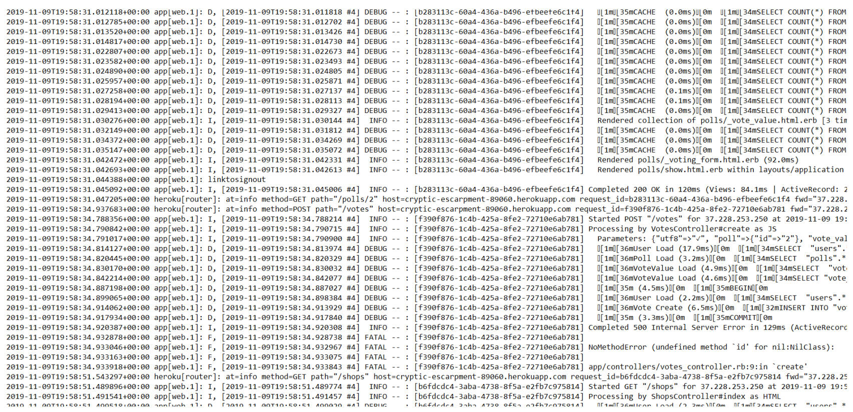
## 9.1 Experimented web application:

The web application that I experimented with for the whole system is made up of ruby and the application is deployed in Heroku. The application is made to manage an apartment easily with its features.

- Web Application link: **Residential community connect**

## 9.2 Logfile:

A logfile is obtained from the application which is used for around 2 weeks by around 50 user accounts. The pictorical image of the logfile is represented below

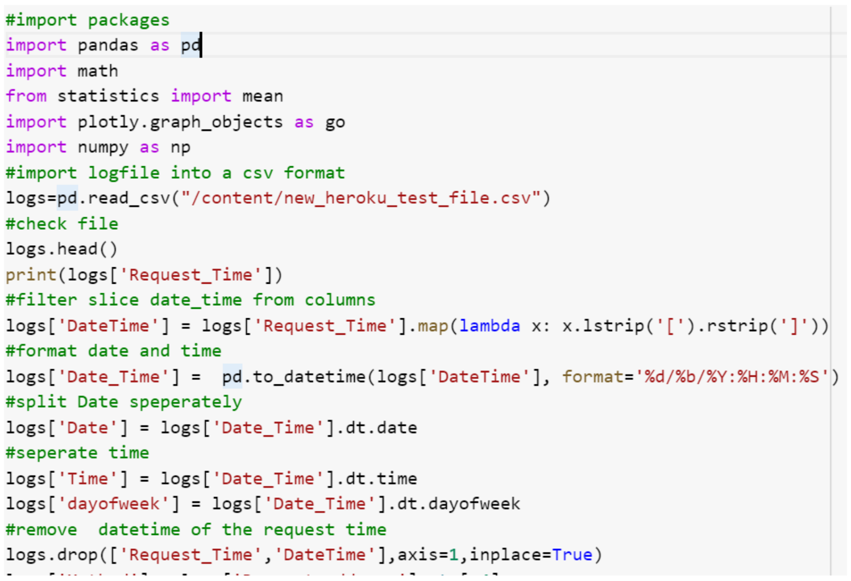


```
2019-11-09T19:58:31.012118+00:00 app[web.1]: D, [2019-11-09T19:58:31.011818 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.012785+00:00 app[web.1]: D, [2019-11-09T19:58:31.012702 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.013520+00:00 app[web.1]: D, [2019-11-09T19:58:31.013426 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.014817+00:00 app[web.1]: D, [2019-11-09T19:58:31.014730 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.022807+00:00 app[web.1]: D, [2019-11-09T19:58:31.022673 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.023582+00:00 app[web.1]: D, [2019-11-09T19:58:31.023493 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.024890+00:00 app[web.1]: D, [2019-11-09T19:58:31.024805 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.025957+00:00 app[web.1]: D, [2019-11-09T19:58:31.025871 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.027258+00:00 app[web.1]: D, [2019-11-09T19:58:31.027137 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.1ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.028194+00:00 app[web.1]: D, [2019-11-09T19:58:31.028113 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.1ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.029413+00:00 app[web.1]: D, [2019-11-09T19:58:31.029327 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.030276+00:00 app[web.1]: I, [2019-11-09T19:58:31.030144 #4]  INFO -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   Rendered collection of polls/_vote_value.html.erb [3 tim
2019-11-09T19:58:31.032149+00:00 app[web.1]: D, [2019-11-09T19:58:31.031812 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.034372+00:00 app[web.1]: D, [2019-11-09T19:58:31.034269 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.035147+00:00 app[web.1]: D, [2019-11-09T19:58:31.035072 #4] DEBUG -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   ▯[1m▯[35mCACHE  (0.0ms)▯[0m  ▯[1m▯[34mSELECT COUNT(*) FROM
2019-11-09T19:58:31.042472+00:00 app[web.1]: I, [2019-11-09T19:58:31.042331 #4]  INFO -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   Rendered polls/_voting_form.html.erb (92.0ms)
2019-11-09T19:58:31.042693+00:00 app[web.1]: I, [2019-11-09T19:58:31.042613 #4]  INFO -- : [b283113c-60a4-436a-b496-efbeefe6c1f4]   Rendered polls/show.html.erb within layouts/application
2019-11-09T19:58:31.044388+00:00 app[web.1]: linktosignout
2019-11-09T19:58:31.045092+00:00 app[web.1]: I, [2019-11-09T19:58:31.045006 #4]  INFO -- : [b283113c-60a4-436a-b496-efbeefe6c1f4] Completed 200 OK in 120ms (Views: 84.1ms | ActiveRecord: 2
2019-11-09T19:58:31.047205+00:00 heroku[router]: at=info method=GET path="/polls/2" host=cryptic-escarpment-89060.herokuapp.com request_id=b283113c-60a4-436a-b496-efbeefe6c1f4 fwd="37.228.
2019-11-09T19:58:34.937683+00:00 heroku[router]: at=info method=POST path="/votes" host=cryptic-escarpment-89060.herokuapp.com request_id=f390f876-1c4b-425a-8fe2-72710e6ab781 fwd="37.228.2
2019-11-09T19:58:34.788356+00:00 app[web.1]: I, [2019-11-09T19:58:34.788214 #4]  INFO -- : [f390f876-1c4b-425a-8fe2-72710e6ab781] Started POST "/votes" for 37.228.253.250 at 2019-11-09 19:
2019-11-09T19:58:34.790842+00:00 app[web.1]: I, [2019-11-09T19:58:34.790715 #4]  INFO -- : [f390f876-1c4b-425a-8fe2-72710e6ab781] Processing by VotesController#create as JS
2019-11-09T19:58:34.791017+00:00 app[web.1]: I, [2019-11-09T19:58:34.790900 #4]  INFO -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   Parameters: {"utf8"=>"✓", "poll"=>{"id"=>"2"}, "vote_val
2019-11-09T19:58:34.814127+00:00 app[web.1]: D, [2019-11-09T19:58:34.813974 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[36mUser Load (17.9ms)▯[0m  ▯[1m▯[34mSELECT  "users".
2019-11-09T19:58:34.820445+00:00 app[web.1]: D, [2019-11-09T19:58:34.820329 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[36mPoll Load (3.2ms)▯[0m  ▯[1m▯[34mSELECT  "polls".*
2019-11-09T19:58:34.830170+00:00 app[web.1]: D, [2019-11-09T19:58:34.830032 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[36mVoteValue Load (4.9ms)▯[0m  ▯[1m▯[34mSELECT  "vot
2019-11-09T19:58:34.842214+00:00 app[web.1]: D, [2019-11-09T19:58:34.842077 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[36mVoteValue Load (4.6ms)▯[0m  ▯[1m▯[34mSELECT "vote_
2019-11-09T19:58:34.887198+00:00 app[web.1]: D, [2019-11-09T19:58:34.887027 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[35m (4.5ms)▯[0m  ▯[1m▯[35mBEGIN▯[0m
2019-11-09T19:58:34.899065+00:00 app[web.1]: D, [2019-11-09T19:58:34.898384 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[36mUser Load (2.2ms)▯[0m  ▯[1m▯[34mSELECT  "users".*
2019-11-09T19:58:34.914062+00:00 app[web.1]: D, [2019-11-09T19:58:34.913929 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[36mVote Create (6.5ms)▯[0m  ▯[1m▯[32mINSERT INTO "vo
2019-11-09T19:58:34.917934+00:00 app[web.1]: D, [2019-11-09T19:58:34.917840 #4] DEBUG -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]   ▯[1m▯[35m (3.3ms)▯[0m  ▯[1m▯[35mCOMMIT▯[0m
2019-11-09T19:58:34.920387+00:00 app[web.1]: I, [2019-11-09T19:58:34.920308 #4]  INFO -- : [f390f876-1c4b-425a-8fe2-72710e6ab781] Completed 500 Internal Server Error in 129ms (ActiveRecorc
2019-11-09T19:58:34.932878+00:00 app[web.1]: F, [2019-11-09T19:58:34.928738 #4] FATAL -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]
2019-11-09T19:58:34.933046+00:00 app[web.1]: F, [2019-11-09T19:58:34.932967 #4] FATAL -- : [f390f876-1c4b-425a-8fe2-72710e6ab781] NoMethodError (undefined method `id' for nil:NilClass):
2019-11-09T19:58:34.933163+00:00 app[web.1]: F, [2019-11-09T19:58:34.933075 #4] FATAL -- : [f390f876-1c4b-425a-8fe2-72710e6ab781]
2019-11-09T19:58:34.933918+00:00 app[web.1]: F, [2019-11-09T19:58:34.933843 #4] FATAL -- : [f390f876-1c4b-425a-8fe2-72710e6ab781] app/controllers/votes_controller.rb:9:in `create'
2019-11-09T19:58:51.543297+00:00 heroku[router]: at=info method=GET path="/shops" host=cryptic-escarpment-89060.herokuapp.com request_id=b6fdcdc4-3aba-4738-8f5a-e2fb7c975814 fwd="37.228.25
2019-11-09T19:58:51.489896+00:00 app[web.1]: I, [2019-11-09T19:58:51.489774 #4]  INFO -- : [b6fdcdc4-3aba-4738-8f5a-e2fb7c975814] Started GET "/shops" for 37.228.253.250 at 2019-11-09 19:5
2019-11-09T19:58:51.491541+00:00 app[web.1]: I, [2019-11-09T19:58:51.491457 #4]  INFO -- : [b6fdcdc4-3aba-4738-8f5a-e2fb7c975814] Processing by ShopsController#index as HTML
2019-11-09T19:58:51.499518+00:00 app[web.1]: D, [2019-11-09T19:58:51.499029 #4] DEBUG -- : [b6fdcdc4-3aba-4738-8f5a-e2fb7c975814]   ▯[1m▯[36mUser Load (2.3ms)▯[0m  ▯[1m▯[34mSELECT  "users".*
```

Figure 12: Application log file used for experimentation

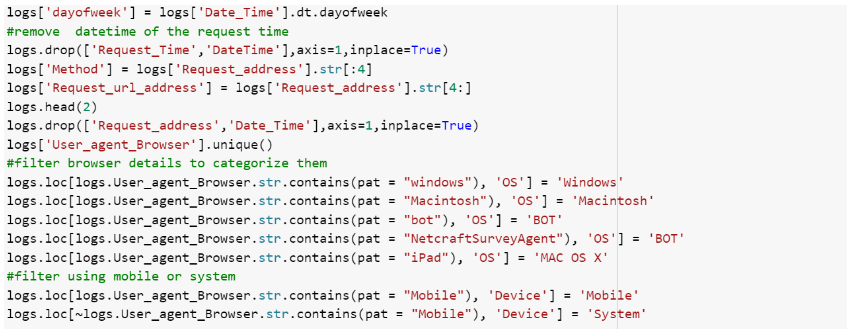
### 9.2.1 The process of log refinement:

The extracted log file is further refined into a data set and the data cleaning is done in python. The process of data cleaning is attached below which imports all the package needed to refine the log file CSV. Splitting the date and time from the data set, categorizing the status codes, separating them according to through the browser functionalities which is represented in Figures 13 and 14.



```
#import packages
import pandas as pd
import math
from statistics import mean
import plotly.graph_objects as go
import numpy as np
#import logfile into a csv format
logs=pd.read_csv("/content/new_heroku_test_file.csv")
#check file
logs.head()
print(logs['Request_Time'])
#filter slice date_time from columns
logs['DateTime'] = logs['Request_Time'].map(lambda x: x.lstrip('[').rstrip(']'))
#format date and time
logs['Date_Time'] =  pd.to_datetime(logs['DateTime'], format='%d/%b/%Y:%H:%M:%S')
#split Date speperately
logs['Date'] = logs['Date_Time'].dt.date
#seperate time
logs['Time'] = logs['Date_Time'].dt.time
logs['dayofweek'] = logs['Date_Time'].dt.dayofweek
#remove  datetime of the request time
logs.drop(['Request_Time','DateTime'],axis=1,inplace=True)
```

Figure 13: Python code used for refining the log file dataset.

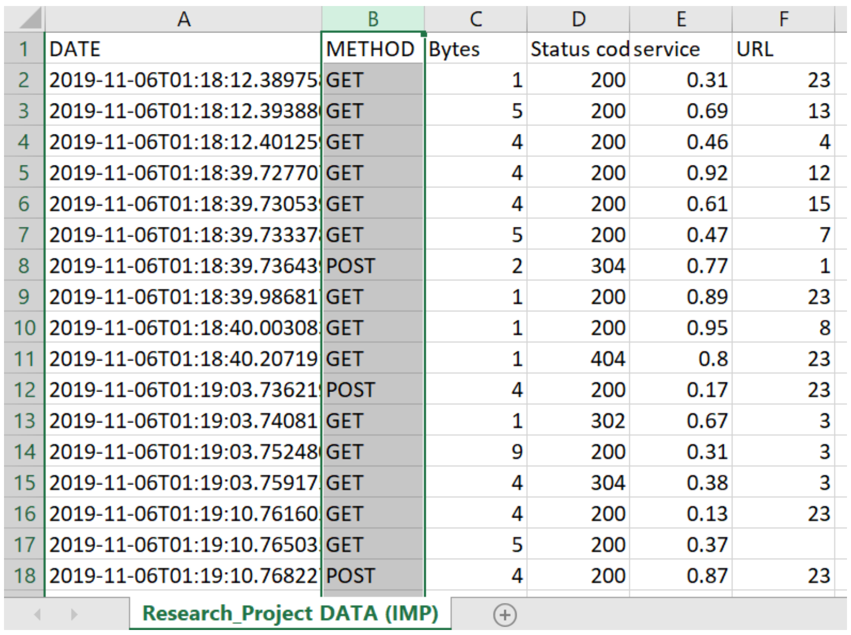


```
logs['dayofweek'] = logs['Date_Time'].dt.dayofweek
#remove  datetime of the request time
logs.drop(['Request_Time','DateTime'],axis=1,inplace=True)
logs['Method'] = logs['Request_address'].str[:4]
logs['Request_url_address'] = logs['Request_address'].str[4:]
logs.head(2)
logs.drop(['Request_address','Date_Time'],axis=1,inplace=True)
logs['User_agent_Browser'].unique()
#filter browser details to categorize them
logs.loc[logs.User_agent_Browser.str.contains(pat = "windows"), 'OS'] = 'Windows'
logs.loc[logs.User_agent_Browser.str.contains(pat = "Macintosh"), 'OS'] = 'Macintosh'
logs.loc[logs.User_agent_Browser.str.contains(pat = "bot"), 'OS'] = 'BOT'
logs.loc[logs.User_agent_Browser.str.contains(pat = "NetcraftSurveyAgent"), 'OS'] = 'BOT'
logs.loc[logs.User_agent_Browser.str.contains(pat = "iPad"), 'OS'] = 'MAC OS X'
#filter using mobile or system
logs.loc[logs.User_agent_Browser.str.contains(pat = "Mobile"), 'Device'] = 'Mobile'
logs.loc[~logs.User_agent_Browser.str.contains(pat = "Mobile"), 'Device'] = 'System'
```

Figure 14: Python code used for refining the log file dataset.

## 9.3 Experimented Data set:

The log file once refined is converted into a proper clean dataset that is fit for experimenting with the data churn algorithm. The cleaned data set comprises of six main columns that comprise of six main columns that are represented in the Figure no. 15.

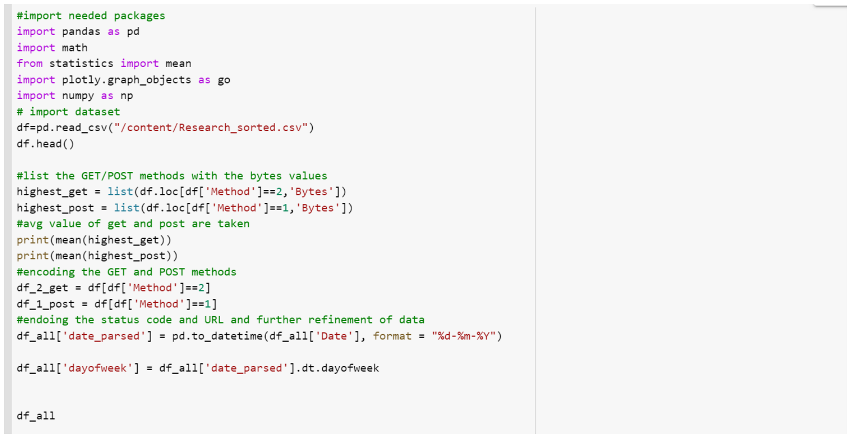


| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | DATE | METHOD | Bytes | Status cod | service | URL |
| 2 | 2019-11-06T01:18:12.38975 | GET | 1 | 200 | 0.31 | 23 |
| 3 | 2019-11-06T01:18:12.39388 | GET | 5 | 200 | 0.69 | 13 |
| 4 | 2019-11-06T01:18:12.40125 | GET | 4 | 200 | 0.46 | 4 |
| 5 | 2019-11-06T01:18:39.72770 | GET | 4 | 200 | 0.92 | 12 |
| 6 | 2019-11-06T01:18:39.73053 | GET | 4 | 200 | 0.61 | 15 |
| 7 | 2019-11-06T01:18:39.73337 | GET | 5 | 200 | 0.47 | 7 |
| 8 | 2019-11-06T01:18:39.73643 | POST | 2 | 304 | 0.77 | 1 |
| 9 | 2019-11-06T01:18:39.98681 | GET | 1 | 200 | 0.89 | 23 |
| 10 | 2019-11-06T01:18:40.00308 | GET | 1 | 200 | 0.95 | 8 |
| 11 | 2019-11-06T01:18:40.20719 | GET | 1 | 404 | 0.8 | 23 |
| 12 | 2019-11-06T01:19:03.73621 | POST | 4 | 200 | 0.17 | 23 |
| 13 | 2019-11-06T01:19:03.74081 | GET | 1 | 302 | 0.67 | 3 |
| 14 | 2019-11-06T01:19:03.75248 | GET | 9 | 200 | 0.31 | 3 |
| 15 | 2019-11-06T01:19:03.75917 | GET | 4 | 304 | 0.38 | 3 |
| 16 | 2019-11-06T01:19:10.76160 | GET | 4 | 200 | 0.13 | 23 |
| 17 | 2019-11-06T01:19:10.76503 | GET | 5 | 200 | 0.37 | |
| 18 | 2019-11-06T01:19:10.76822 | POST | 4 | 200 | 0.87 | 23 |

Figure 15: Data set after the process of cleaning.

## 9.4 Data encoding and processing:

The data is encoded to numerical values which are easy to analyze as plot graphs by using the Data churn algorithm. This is done by using python and code is given below in the Figure 16, 17 and 18.

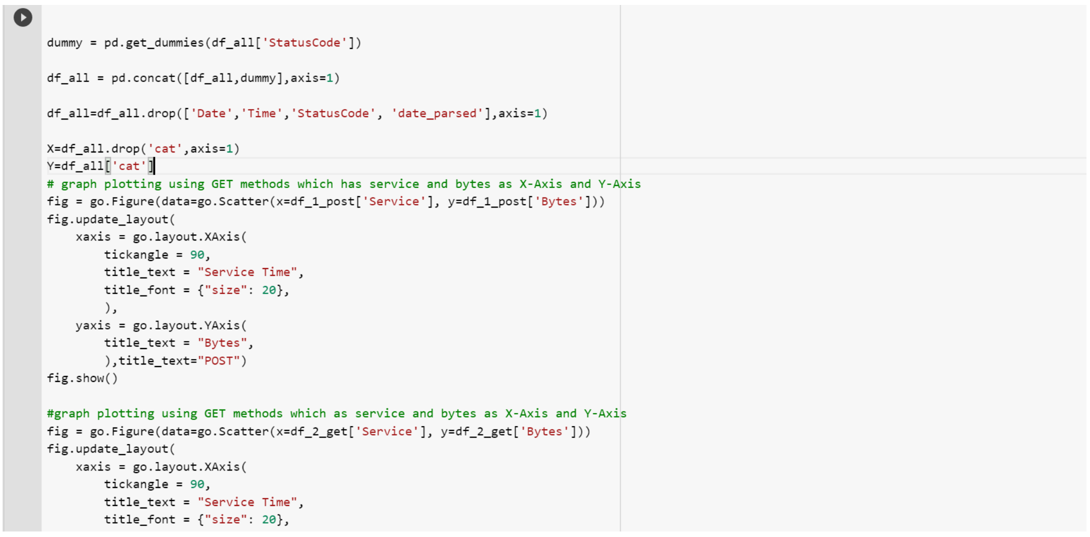


```
#import needed packages
import pandas as pd
import math
from statistics import mean
import plotly.graph_objects as go
import numpy as np
# import dataset
df=pd.read_csv("/content/Research_sorted.csv")
df.head()

#list the GET/POST methods with the bytes values
highest_get = list(df.loc[df['Method']==2,'Bytes'])
highest_post = list(df.loc[df['Method']==1,'Bytes'])
#avg value of get and post are taken
print(mean(highest_get))
print(mean(highest_post))
#encoding the GET and POST methods
df_2_get = df[df['Method']==2]
df_1_post = df[df['Method']==1]
#endoing the status code and URL and further refinement of data
df_all['date_parsed'] = pd.to_datetime(df_all['Date'], format = "%d-%m-%Y")

df_all['dayofweek'] = df_all['date_parsed'].dt.dayofweek


df_all
```

Figure 16: Python code for data encoding and processing data churn algorithm

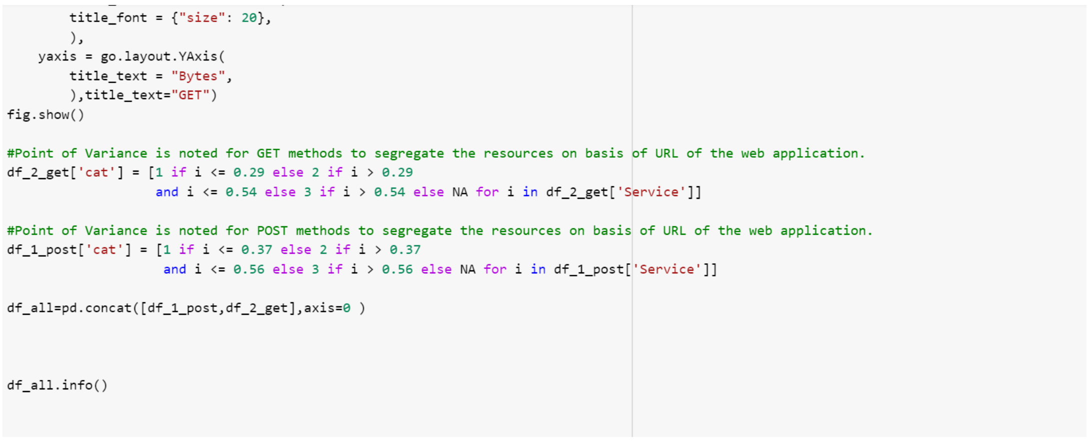
## 9.5 Django-API:

The following file is placed as an API using DJANGO which a python backend framework and the API code is mentioned below which takes all the data columns and stores in the SQL DB to process them into further. After the process, the data churn algorithm is placed into views.py file in Django to process the whole API which is represented in Figure 19.



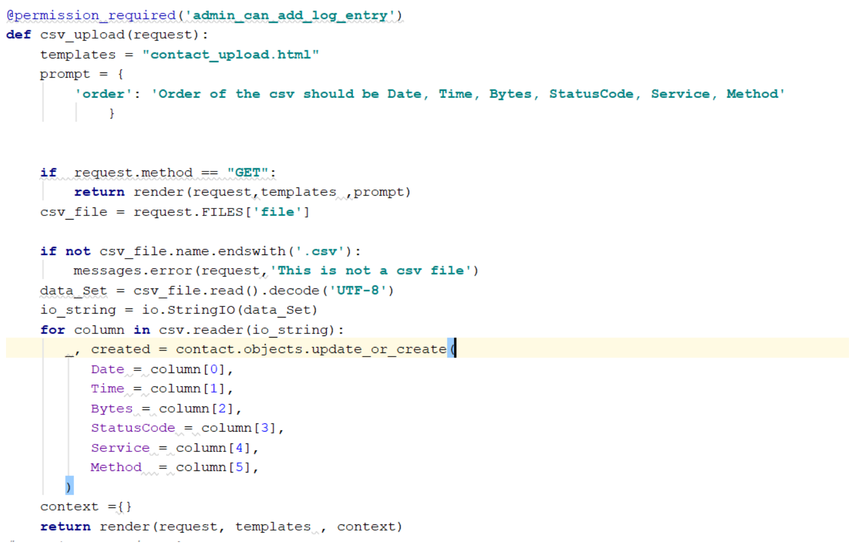
```
dummy = pd.get_dummies(df_all['StatusCode'])

df_all = pd.concat([df_all,dummy],axis=1)

df_all=df_all.drop(['Date','Time','StatusCode', 'date_parsed'],axis=1)

X=df_all.drop('cat',axis=1)
Y=df_all['cat']
# graph plotting using GET methods which has service and bytes as X-Axis and Y-Axis
fig = go.Figure(data=go.Scatter(x=df_1_post['Service'], y=df_1_post['Bytes']))
fig.update_layout(
    xaxis = go.layout.XAxis(
        tickangle = 90,
        title_text = "Service Time",
        title_font = {"size": 20},
        ),
    yaxis = go.layout.YAxis(
        title_text = "Bytes",
        ),title_text="POST")
fig.show()

#graph plotting using GET methods which as service and bytes as X-Axis and Y-Axis
fig = go.Figure(data=go.Scatter(x=df_2_get['Service'], y=df_2_get['Bytes']))
fig.update_layout(
    xaxis = go.layout.XAxis(
        tickangle = 90,
        title_text = "Service Time",
        title_font = {"size": 20},
```

Figure 17: Python code for data encoding and processing data churn algorithm.



```
        title_font = {"size": 20},
        ),
    yaxis = go.layout.YAxis(
        title_text = "Bytes",
        ),title_text="GET")
fig.show()

#Point of Variance is noted for GET methods to segregate the resources on basis of URL of the web application.
df_2_get['cat'] = [1 if i <= 0.29 else 2 if i > 0.29
                   and i <= 0.54 else 3 if i > 0.54 else NA for i in df_2_get['Service']]

#Point of Variance is noted for POST methods to segregate the resources on basis of URL of the web application.
df_1_post['cat'] = [1 if i <= 0.37 else 2 if i > 0.37
                    and i <= 0.56 else 3 if i > 0.56 else NA for i in df_1_post['Service']]

df_all=pd.concat([df_1_post,df_2_get],axis=0 )


df_all.info()
```

Figure 18: Python code for data encoding and processing data churn algorithm.



```python
@permission_required('admin_can_add_log_entry')
def csv_upload(request):
    templates = "contact_upload.html"
    prompt = {
        'order': 'Order of the csv should be Date, Time, Bytes, StatusCode, Service, Method'
            }


    if  request.method == "GET":
        return render(request,templates ,prompt)
    csv_file = request.FILES['file']

    if not csv_file.name.endswith('.csv'):
        messages.error(request,'This is not a csv file')
    data_Set = csv_file.read().decode('UTF-8')
    io_string = io.StringIO(data_Set)
    for column in csv.reader(io_string):
        _, created = contact.objects.update_or_create(
            Date = column[0],
            Time = column[1],
            Bytes = column[2],
            StatusCode = column[3],
            Service = column[4],
            Method  = column[5],
        )
    context ={}
    return render(request, templates , context)
```

Figure 19: Data set after the process of cleaning.

### 9.5.1 Latency evaluation:

**TRACERT:** This windows command shows the time of the data packets that take to travel from the client to the server machine. The following command which is mentioned below helps to attain the details of latency from a specific application.

**tracert ”ip address” (or) “www.web application.com”.**

### 9.5.2 Other commands:

To run a django application

- python manage.py migrate : To migrate db.
- python manage.py createsuperuser : To create admin.
- python manage.py runserver : To run the server.

To run the ruby application that is used for specific experimentation.

- ‘rails db:migrate’: Used to run all the database migration.
- ‘rails server’ is a command used to run a rails application.
- If the application is in production “rails server -e production -p 4000”.