A Web Application that estimates Pi(π) Value and the cost incured in cloud service using Monte-Carlo Method

Sarath, Asok Kumar
6670724, sa02844@surrey.ac.uk

Abstract—**The transition from having software's which are installed on the local computers to cloud level computing is simply getting going to another level. The objective of this paper is to develop a web application which provides the user to specify the values which will generate the estimated Pi values by computing parallelly using Monte-Carlo method. The results of the experiment are described with the help of a graph and the cost estimation for the process to calculate the Pi values is found.**

Keywords—Monte-Carlo, Lambda, EC2, S3 Bucket, GAE, Cost estimation.

I. Introduction

The implementation of an on-request cloud system to estimate the value of Pi and to calculate the cost of using the cloud has been built using different orchestration in the cloud environment. The services are available at any place on the planet, the user can interact with the application with proper guidance provided at the frontend i.e. the user can calculate the value of pi and get the results in the output page. Here the computation of pi value is made faster by running parallelly with the value of resources that user provided, additionally the value of pi is matched with digits that user given in the input page. The backend framework provides a great flexibility to enhance the computation with the use of AWS EC2 and Lambda function with resources when required. The frontend framework used here is Google app engine, a combination of the system is deployed over the public cloud service platform.

A. *Developer*

From the perspective of a developer, the framework provides on-request viability to a supply of resources for making the system flexible and scalable. The enhancement and the choices of resources are the most evaluative element for building a system with maximum efficiency moreover the cloud providers keep an eye on the usage of resources and monitored it. The computation of Pi value is done at the backend and they are divided into two systems and they are:

1. Function as a Service (FaaS) is the backend for computing the value of Pi. It allows the developers to develop, execute and to control the application functionalities without the help of any complex infrastructures which needs to me maintained [1].
2. Infrastructure as a Service (IaaS) is the other backend for computing the value of Pi, which provides hardware and the required softwares. It means that the cloud providers are responsible for networking, storage, server and virtualization [2].

The frontend part of the system is where the user gives the inputs for the computation of Pi value is just a HTML page. It is a platform as a service (PaaS) part of cloud computing where the application can be developed and deployed. The entire system was conveyed under the public cloud models utilising the blend of the cloud providers. The developer can pick out the best benefits from the best services which are used here.

B. *User*

From the viewpoint of the user, although being a complex system, it shows up to be basic and simple one. The system is an on-request self-service application which gives the output very quickly. The application is made available through the network with the help of web browser. The system provides the user the capacity of flexible scaling the resources to make the computation faster and efficiently. The estimated cost of using the cloud platform for the calculation of Pi value and the previous user inputs and its results which is stored in the cloud can be viewed at the output web page.

II. Final Architecture

A. *Major System Components*

The system comprises of five unique components and they are categorised as back-end and front-end components:

1. Back-end Components

a) *Amazon API Gateway*

Amazon API Gateway empowers the system to add, publish, maintain, and secure APIs at any scale. The communication between the client and the server in real-time is enabled by this gateway specifically REST API. The intermediary part between the client and server is maintained by HTTP by its request, integration and response flow. Here, three Representational State Transfer Application Programming Interface (REST APIs) are created for lambda function, S3 and EC2 [3].

b) *AWS Lambda*

Lambda function is a serverless computing platform which is event-driven provided by Amazon Web Service [4]. Here the lambda function runs the code provided by the developer without managing the infrastructure. REST API are used for the functionalities to compute the code. Two lambda functions are implemented in the making of this system.

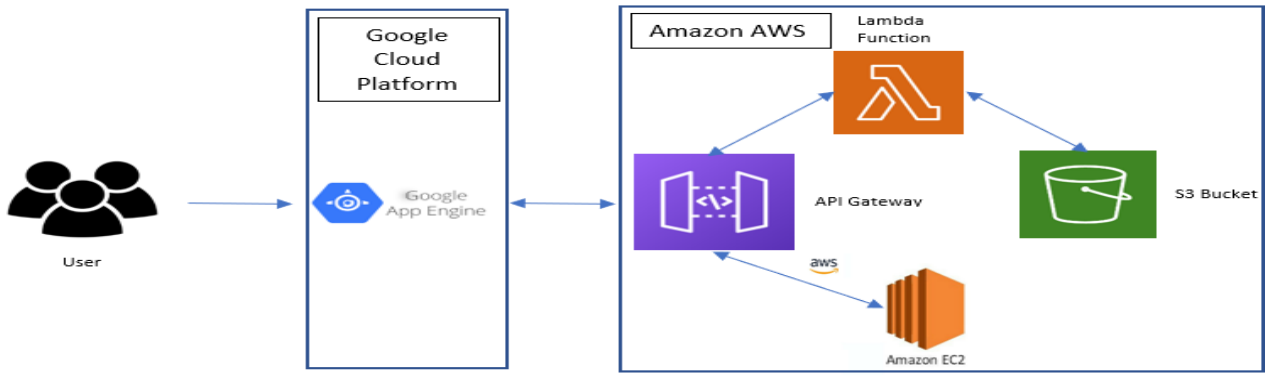


Figure 1. High-level diagram of the major components of the cloud system

c) Amazon Elastic Compute Sytem (Amazon EC2)

EC2 permits the developers to construct an application that can automate the scaling according to the changes needed or periodic peaks [5]. It allows to make developer easy to deploy and manage storage which reduces the need in spending money for hardware. ECS and EMR are dropped because of their lack in development process, i.e. EC2 has got a fast advancement in the process to which the functionality is deployed, including the execution of instance of software bundles. Additionally, EC2 provides a quick and reliable cloud server which is needed by the developer. In this system, EC2 involves in creating Amazon Machine Image (AMI) which will create instances to execute the parallel computation.

d) Amazon S3

Amazon S3 is feature offered by AWS that gives object stockpiling through a web interface with objective to make the web-scaling process easier for the developer [6]. This service enables to save and recover any measure of data whenever from any part of the web. It helps the system to execute the right functionalities from its all stored data. Instead of using Amazon RDS, Amazon Aurora, or Amazon Dynamo over Amazon S3 is because of its effortlessness for gaining and storing data, infrastructure management is not required and cheaper cost of usage. The history data which is saved as a json file in the s3 bucket is retrieved and displayed on the history page of this application. The history includes the number of shots, resources used, reporting rate, matching digits, estimated pi value and the estimated cost of running on the cloud.

2. The front-end Components

a) Google App Engine

It is a cloud computing platform used for creating and facilitation the web application [7]. It is a serverless platform which provides the interaction between REST API and host which has the web application. It allows the user to execute the functionalities examined beforehand.

B. *System Component Interactions*

The interaction between the components begins from GAE to Lambda function or EC2, the homepage will be loaded with choices between the cloud service i.e. between lambda function and EC2. The user can pick any of the two option to load the next page where the user should give the inputs. Here, the user has to give the number of shots, at what rate the estimate pi values should be generated, how many numbers of resources needed for the scalability and up to how many digits the Pi value should be matched. Once the user submits the input in loaded HTML page, the request has been sent through a REST API and the code which does the Monte-Carlo simulation is written in the lambda function (or EC2) will execute and hence returns the values which are required for further calculations. The incircle values and estimated pi values are generated at every reporting rate, later, the shots are divided by the number of resources that user gave at the input. The pi value generated will be compared with math.py value which under a threshold to prevent code from running forever. A plot which shows the real pi value and the estimated pi values at the given reporting rate is generated at the output page, also a table which shows the list of incircle values and the reporting rate is generated.

In addition to that, another lambda function is made to create a history table with previous users inputs by the help of S3 bucket. The access to read and write by the lambda function is given by Identity and Access Management (IAM). The lambda function will send request through REST API whenever the user submits the inputs. Along with input values, the output values are also stored in the S3 bucket. When the history page has to be viewed by the user at the output page, the lambda function reads the json file from the S3 bucket and provide a detailed table with the details of previous history.

When the user request computation in EC2 service, the created instance start running, the number of instances created depends on the number of resources given by the user. When the instance is launched a copy of AMI is running as a server in the cloud. Multiple instances are launched from an AMI for parallelly running the computation for fast response.

III. Implementation

Appspot Link https://pi-calculation-314711.ew.r.appspot.com

A. *Implemtation process*

The general cloud system was predominantly executed with Python. The development started from the backend to the frontend manner.

Initially the backend was developed, it includes the generation of incircle values and the reporting rate from Lambda. The random number for generating the incircle was done using the python library called Random. The code for the computation of the so-called Monte-Carlo simulation was initially tested in the local environment like Jupyter notebook, then it is dropped in the Lambda function. Using the REST API gateway, the lambda function has been invoked. The output from the Lambda function has been received in getpage() function and processed using a function called processor() which process the output lambda function to a form where the calculation can be done easily and it is then passed to another function for further calculation. Here the total number of shots is divided for running parallelly with the number of resources, for paralleling another function was written and execution was done with the help of ThreadPoolExecuter() which helped computation more faster and more importantly, scalability. The further calculation of the estimated Pi values is done logically by carefully running through and extracting the values from the list of incirles and reporting rating which is added separately and divided and then multiplied with 4. A list of estimated pi value is then compared with Math.py where the limit of the matching digits is specified by the user. The status saying whether a matching pi value is also generated during the comparison. A cost estimation code has also added. The above functions were created in parallel_lambda.py file. Development of EC2 is the next task, which was hard, the code which is used in the lambda function was used here, the EC2 instances was then developed. Apache server and WSGI was implemented and configured with help of Flask framework. The security group has been configured and launched. An AMI was created with the main EC2, AMI helps to create number of instances (multiple instances) which depends upon the users input. The creation of instances is limited to prevent from creating more instances. The remaining part is same as that of Lambda function, a function which create and terminate instances were also written. Since, the cred file expires after every 3hrs, it was difficult to test These codes are written in another .py named ec2_process.py file. An index.py file, where inputs are taken, binds together parallel_lambda.py and ec2_run.py. For the implementation of S3 bucket, another lambda function was created and configured its access from IAM. Each time when the user submits the inputs, along with output values a json file will be created and stored in the S3 bucket. Since lambda is connected, the write and read functions are invoked through REST API, which will store or retrieve data when required. When the user wanted to see the history from the output web page, the read function will call and an history table with the previously stored inputs and output will be showed. The backend system is hosted in N.Virginia region in AWS. When looking into the setup of lambda functions, with the python native libraries, pandas, boto3, json and numpy were used [8,9,10]. For coding support, StackOver Flow posts were used and gone through AWS documentation.

The frontend is mainly developed using HTML. The codes from Lab 1 and Lab 3 were used [11,12]. The first page provides an option to select the amazon service i.e. Lambda or EC2, Once the user selects any of these options the next page gets loaded and provides the space for input values which includes the Shots (S), Reporting rate (Q), Matching digits (D) and Resources (R) from the user. A history button is also provided in this web page to view the last users input and output with cost estimation of the service used in the AWS platform. When the user submits the input values another webpage is loaded which shows the output along with user's inputs. This page contains the status which says whether a matching Pi value is found or not, estimated Pi value and also the graph which shows the plot between real Pi value and the estimated Pi values at the given reporting rates as been generated with the help of Plotly [13]. For the framework, Flask was also used along with native python libraries [14]. CSS styling is done in the webpages with the help of HTML templates which are readily available in w3school.com [15].

B. *Testing of Code*

Testing is divided into Backend tests and Front-end tests.

a) Back-end tests

The testing of code was carried out locally. Lambda function code was initially tested locally and then again tested on lambda console which is presented in the AWS. API Gateway is also tested with the help of console to check with all the functionalities is working. For testing both Lambda and API Gateway, JSON events was given as the arguments. HTTP clients were used to run API.

b) Front-end tests

Here also the test was carried out locally by opening the web page manually and it is then deployed in GAE.

IV. RESULTS

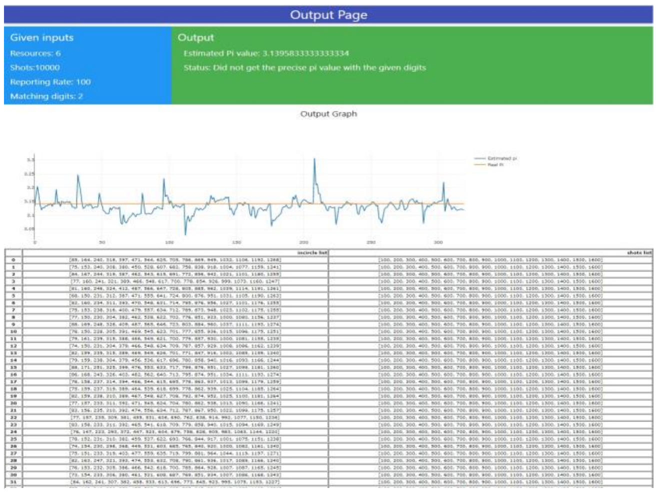
Experimentation with the values of shots and rate has been done for both the services. Also changed the matching digit value and resources used. The chance of getting a matching Pi value is high when the number of shots are more and reporting rate are less. It is also noticeable that, when the number of resources are increased the time taken for the process is less. When the matching digits value is given more, the chance for getting a matched Pi value is very less.

Below is the table showing the estimated Pi values when the parameters are changed.

TABLE I. ESTIMATION OF PI VALUE

Shots	*Rate*	*Digits*	*Resources*	*Pi_value estimated*
100000	1000	2	6	3.145875
120000	1000	3	6	3.13903333333333 3
12000	1000	4	4	3.15633333333333 4
80000	1000	4	3	3.14364102564102 54
100000	100	2	2	3.14428
12000	100	2	6	3.16033333333333 4
50000	100	3	5	3.1356

Below is the output page after using Lambda service, from the output page, the input values are shown along with output values. Here after submitting the inputs, an estimated Pi values has been generated and the graph is also plotted. A table showing Incircle list and shots list is also shown below the graph.


Output Page

Given inputs

Resources: 6

Shots:10000

Reporting Rate: 100

Matching digits: 2

Output

Estimated Pi value: 3.1395833333333334

Status: Did not get the precise pi value with the given digits

Output Graph

	incircle list	shots list
0	[85, 164, 240, 318, 397, 471, 544, 625, 705, 786, 869, 949, 1032, 1106, 1192, 1268]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
1	[75, 153, 240, 308, 380, 450, 529, 607, 682, 758, 838, 918, 1004, 1077, 1159, 1241]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
2	[84, 167, 244, 319, 387, 462, 543, 615, 691, 772, 856, 942, 1021, 1101, 1180, 1259]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
3	[77, 160, 241, 321, 389, 466, 548, 617, 700, 779, 854, 926, 999, 1073, 1160, 1247]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
4	[81, 160, 246, 324, 412, 487, 566, 647, 728, 805, 885, 962, 1039, 1114, 1191, 1261]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
5	[68, 150, 231, 312, 387, 471, 555, 641, 724, 800, 876, 951, 1031, 1105, 1190, 1263]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
6	[82, 160, 234, 311, 393, 470, 548, 621, 714, 795, 876, 956, 1027, 1101, 1176, 1255]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
7	[75, 153, 238, 316, 400, 479, 557, 634, 712, 789, 873, 948, 1023, 1102, 1175, 1258]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
8	[77, 150, 230, 304, 382, 462, 538, 622, 702, 776, 851, 923, 1000, 1080, 1156, 1237]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
9	[88, 169, 248, 328, 409, 487, 565, 646, 723, 803, 884, 960, 1037, 1111, 1193, 1274]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
10	[76, 150, 228, 305, 391, 469, 545, 623, 701, 777, 855, 936, 1015, 1096, 1175, 1251]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
11	[79, 161, 239, 315, 388, 468, 549, 621, 700, 779, 857, 930, 1000, 1081, 1158, 1238]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
12	[74, 150, 231, 304, 379, 466, 548, 624, 709, 787, 857, 929, 1008, 1086, 1162, 1239]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
13	[82, 159, 235, 315, 389, 469, 549, 626, 701, 771, 847, 916, 1002, 1085, 1159, 1240]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
14	[79, 159, 238, 304, 379, 456, 536, 617, 696, 780, 858, 940, 1016, 1093, 1166, 1244]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
15	[88, 171, 251, 325, 399, 476, 553, 633, 717, 798, 876, 951, 1027, 1098, 1181, 1260]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
16	[86, 168, 243, 326, 403, 482, 562, 640, 713, 795, 874, 951, 1034, 1111, 1193, 1274]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
17	[76, 158, 237, 314, 394, 466, 544, 615, 695, 776, 863, 937, 1012, 1099, 1179, 1259]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
18	[75, 159, 237, 319, 389, 464, 539, 618, 699, 778, 862, 939, 1025, 1104, 1185, 1264]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
19	[82, 159, 238, 310, 389, 467, 546, 627, 708, 792, 874, 952, 1025, 1100, 1181, 1264]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
20	[77, 157, 233, 311, 392, 471, 548, 624, 704, 780, 862, 938, 1013, 1090, 1168, 1241]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
21	[83, 156, 235, 310, 392, 474, 556, 634, 712, 787, 867, 950, 1022, 1099, 1175, 1257]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
22	[77, 157, 235, 309, 381, 455, 531, 606, 690, 762, 838, 914, 992, 1077, 1150, 1236]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
23	[83, 158, 233, 311, 392, 465, 541, 618, 709, 779, 858, 940, 1015, 1094, 1169, 1249]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
24	[76, 147, 223, 293, 372, 447, 523, 604, 679, 758, 828, 905, 983, 1063, 1144, 1220]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
25	[76, 152, 231, 310, 382, 459, 537, 622, 693, 766, 844, 917, 1001, 1075, 1151, 1236]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
26	[74, 154, 230, 298, 368, 449, 531, 603, 685, 765, 840, 920, 1000, 1082, 1161, 1240]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
27	[75, 151, 233, 315, 403, 477, 559, 635, 715, 799, 881, 964, 1044, 1115, 1197, 1271]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
28	[82, 163, 247, 321, 393, 474, 553, 632, 708, 790, 861, 936, 1017, 1089, 1166, 1240]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
29	[76, 153, 232, 305, 386, 466, 542, 618, 700, 785, 864, 928, 1007, 1087, 1165, 1245]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
30	[73, 154, 233, 306, 380, 461, 531, 609, 687, 769, 851, 934, 1007, 1086, 1168, 1243]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]
31	[84, 162, 241, 307, 382, 458, 533, 613, 696, 773, 848, 923, 995, 1075, 1153, 1227]	[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600]

Figure 2 Output Page

V. SATISFACTION OF REQUIREMENTS

TABLE II. REQUIREMENTS

#	*C*	*Description*
i.	M	All the services are used i.e. GAE, AWS Lambda and AWS EC2.
ii.	M	Frontend for user to provide inputs for the number of resources to be used, number of shots, running rate and matching digits.
iii.	M	Scalable services are used to run the Pi estimation code and results made available in the frontend.
iv. a	P	The user can provide the inputs S, R, Q and D where any one of the scalable services are used for calculation, but provision for "warm up" not done.
iv. b	M	A way to provide D, S which will be divided across the R, way to provide Q is provided
iv. c	M	A graph showing the estimated pi values with real pi values, table with shots and incircles, a final estimate for Pi value and a history table which is available in the history page
iv. d	N	A reset option to provide a 'zero' analysis without warm up the resources.
iv. e	M	An option for the user to terminate the EC2 instances.

VI. COSTS

The cost for the AWS services are listed in Table III. The run time informations are used to calculate the scalable services. Here, us-east-1 was used for AWS pricing.

Cost calculation for Lambda function was found by taking the run time, GB/s of memory used and per request fee [16].

Cost calculation for EC2 is also calculated by using run time and the amount of data in and out [17].

As an analysis, it is found that, when the number of shots are high the cost are also high, also when the threshold value is kept high, same effect will occur for cost estimation. Cost is less when the number of resources are increased but it all depends on the matching of estimated Pi value with math.pi.

Table III Cost Results

Resources	Shots	Service	Duration Billed (s)	Cost in $
2	100000	Lambda	23.6568	0.00409005
5	50000	Lambda	10.2646	0.00138621
6	12000	Lambda	6.8721	0.000465512
2	20000	EC2	21.689	0.00072152
6	120000	EC2	30.5264	0.000248647
3	15000	EC2	12.0586	0.0001325

REFERENCES

[1] L. Gillam, "Faas (Paas) overview", Week 4/5, COMM034- Cloud Computing, 2021-2021, University of Surrey.
[2] Tim Grance, Peter Mell (2011), "the NIST Definition of Cloud Computing", NIST Special Publication (SP 800-145).
[3] Amazon Web Services, "API Gateway", Developer Guide, Amazon API Gateway, AWS Documentation, 2021
[4] Amazon Web Services, "AWS Lambda", Developer Guide, Amazon API Gateway, AWS Documentation, 2021
[5] Amazon Web Services, "Amazon EC2", Developer Guide, Amazon API Gateway, AWS Documentation, 2021
[6] Amazon Web Services, "Getting started with Amazon Simple Storage Service", Developer Guild, Amazon API Gateway, AWS Documentation, 2021
[7] Google Cloud, "Google App Engine Documentation", Documentation, App Engine, Serverless computing, Google cloud, 2021
[8] The Pandas development team, "Pandas documentation", Documentation, pandas, 2021.
[9] Amazon Web Services, "Boto3 documentation", Boto3 Docs, 2021
[10] The Scipy community, "NumPy v1.19 Manual", Documentation, NumPy, 2020
[11] L. Gillam, "Templates, Forms and External Services", Week 1 Lab, COMM034- Cloud Computing, 2021-2021, University of Surrey.
[12] L. Gillam, "Comparing Serial and Parallel Use", Week 3 Lab, COMM034- Cloud Computing, 2021-2021, University of Surrey.
[13] Plotly Technologies Inc, "Line Charts in JavaScript", Basic Charts, JavaScript, Plotly, 2021
[14] Pallets, "User's Guide" Flask Documentation, 2.0.x, Flask Project, 2010
[15] Refsnes Data, "HTML Tutorial" and "CSS Tutorial", HTML and CSS, w3schools.com, 2021
[16] Amazon Web Service, "AWS Lambda Pricing", AWS Lambda, AWS, 2021
[17] Amazon Web Service, "AWS EC2 Pricing", AWS EC2, AWS, 2021