Analysis of Textsimilarity on cloud

An Experiment of Cloud Computing

Yan Huang
Department of Computing
University of surrey
Guildford
yh00079@surrey.ac.uk

***Abstract*—In this report, the task is to find out the plagiarisms between suspicious files and source files on cloud. The plan to finish the task is across three different cloud systems. The object is an auto scaling system and proper data communication. This report involves the architecture and implementation of the system.**

Keywords—cloud computing; Google App Engine; EC2;Ayto Scaling; OpenStack; plagarism detection

I. INTRODUCTION

This report demonstrates a plagiarism analysis system based on cloud using Goole App Engine (the acronym GAE), Amazon Elastic Computing Cloud (the acronym EC2) and OpenStack (the acronym OS). The following sections describe the plagiarism analysis method, the architecture and description of the system, data flow, testing and result

II. PLAGIARISM ANALYSIS METHOD

To detect the plagiarism, an inverted index of source files is constructed. The original texts in source files are transformed to lowercase and get rid of the punctuation. Then, the texts are divided into lines whose length is called “window size”. There are N overlapped words between neighbouring lines. Here N is called “overlap size”. Both “window size” and “overlap size” can be specified. Moreover, each line is converted to an MD5 hash value as the index’s fingerprint. The index contains the fingerprints and their location (the name of source file where a fingerprint comes from).

Once the suspicious file is received, the suspicious file is transformed to hash values by the same method. By comparing the suspicious file’s and the index’s fingerprint, plagiarism can be detected.

III. HIGH-LEVEL ARCHITECTURE

In this system, three different cloud services are applied to deal with the different tasks. The data are communicated between EC2 and OS as well as GAE and OS in the REST way.

A. GAE

GAE is a platform as a service (PaaS) cloud computing platform. I conduct Software as a service (SaaS) from it. Actually, in this case, it is a web application.

GAE is responsible to build an interface for user to interact with the whole system. Users can upload suspicious files, check their usage report, and specify window size, overlap size, how many instances they want. After users submit a task, GAE conducts the task to be ready to be collected by OS. When GAE receives an analysed result comes from OS, it will be shown by Google chart.

B. OS

The analysis is implemented in OS. There are two kinds of nodes built in OS. The main node keeps detecting the task queue in GAE. If a task is detected, the main node requests the suspicious file and parameters (window size, overlap size and number of instances) from. Then the main code distributes the task to activate children nodes according to the number of instances (map step). After that, the children nodes will ask EC2 for the index of source file accoring to the received task. Afterwards, the children nodes will analyse the plagiarism and send the result to main node. The main node will integrate the results from chidren nodes to get the final result (reduce step).

C. EC2

EC2’s mission is to build the indices according to the parameters from children nodes in OS when it receives the request and deliver them. It is elastic. The Elastic Load

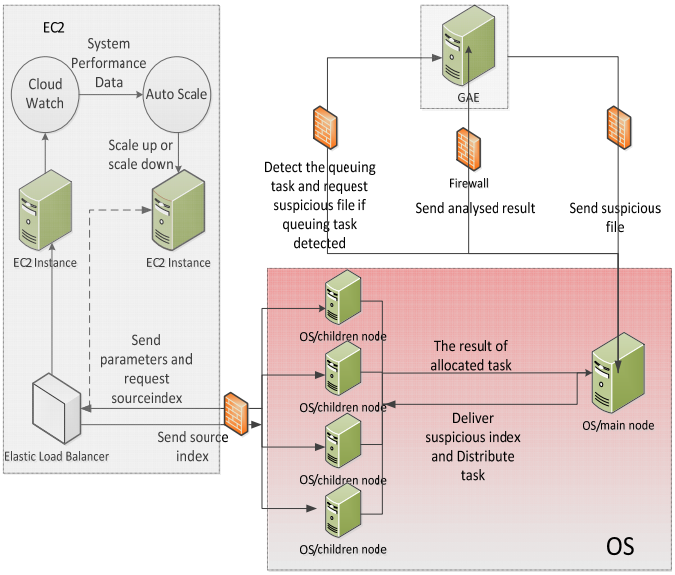


Fig. 1. High-level architecture

Balancer (for short ELB) can allocate the request to a less busy instance to make EC2 fully utilised. Additionally, the total number of EC2 instances can be increased or decreased by auto scaling according to the system performance.

IV. Implementation And Data Flow

A. GAE

Handler

GAE is designed as the unique interface for user in this task. The uploaded file is stored in Blobstore. The results of analysis, users' usage log and the queuing tasks are stored in Datastore. The GAE's handlers and mapped URL are written in Table 1. Other server can use these handlers by opening the correct URL.

Datastore

There are 4 tables in Datastrore. UserFile is the relationship between files and users. Userlog contains the user' usage histories for calculating the users' charge in future work. The Queue stores the queuing tasks and pending tasks. If the status_code=0, the tasks of queuing, once the OS get the task but no result returned yet, the status_code turns to 1. When its result is received, the task will be deleted from Queue table. The Result table contains the analysed result. Show result handler can show the result by query this table. Table 2 describes the structures of these tables

TABLE II. GAE's Handlers

Handler	URL
MainHandler	/
Handle the main page include users' registration, sign in and out. Create the interface to upload file, check result and usage log, place tasks.	
UploadHandler	'/upload'
Upload file and Create a blob key automatically. The file is stored in blobstore and bond it to user by updating UserFile table in datastore	
ServeHandler	/serve/([^/]+)?
Download a file by its blob key, for example /serve/YkCKAZDRniPbl6tWMyKwjw==	
QueueHandler	/queue/([^/]+)?
Put a task in Queue table with the blob key and the parameters. The format is /queue/blob key?window_size=&overlap_size=&number_of_instances=	
GetQueueHandler	/get_queue'
OS get the queuing task by open this url. It resposes the blob key and parameters of a queuingtask, or blank page if the queue is empty	
ResultHandler	/result/([^/]+)?
OS posts result via this handler. The result is encoded in the form of json. GAE can decode it to a dictionary	
ShowResultHandler	/show_result/([^/]+)?
Show result by Google chart. The format is /show_result/blob key/result encoded in json	

TABLE I. The Strctures of Data Tables

Table	Schema
Queue	file_key=BlobReferenceProperty window_size, overlap_size, number_of_instances, status_code=IntegerProperty
Result	file_key=BlobReferenceProperty result = StringProperty
UserFile	user_id = StringProperty file_key = BlobReferenceProperty
Userlog	user_id = StringProperty file_key =BlobReferenceProperty window_size, overlap_size, number_of_instances = IntegerProperty time=DateTimeProperty

B. OS

Get Data from GAE

OS takes the responsibility to analyse the suspicious file. Because the OS is a private and hides behind the firewall, it cannot receive the request from outside server. Therefore, it has to initiate a request in order to get data from GAE. The main node keeps query the queue of tasks in GAE. If the queue is empty, it idles for 10 seconds. Otherwise it gets the task as well as the according parameters and suspicious file. Then the file is transformed to hash values.

Break Task and Distribution

If an OS instance gets all the 500 source files' index from EC2, out of memory error is raised sometimes. Additionally, it is too slow (it takes around 4 minutes). Therefore, the source index has to be broken into several parts as well as the task according to the specified number of instances by user.

The main node handles the task distribution. Denote the number of instances as N. The task is divided into N parts evenly. Each children node receives a task with 4 parameters – "window size", "overlap size", "start" and "end" (for i-th children node, $start[i] = (i-1) * Ceil(\frac{500}{n}) + 1, end[i] = start[i] + Ceil\left(\frac{500}{n}\right)$). After accepting the task, each children node compares the suspicious file to 500/N source files, whose sequence number between "start" and "end", instead of all the 500 files. As a result, each children node only deals with 1/N of the whole source index. Accordingly, he consumption of time and memory reduce roughly by N times

All the children nodes' have to implement their tasks in parallel or the consumption would not reduce. In order to do this, the main node builds a process pool which contains N processes and uses map function built in python to run them in parallel.

Each children node requests index from EC2 only contains the source files whose sequence number between its own “start” and “end”. Afterwards, it matches the suspicious index to its partial source index and sends its result to main node.

In practise, the main node uses ssh protocol (use sshpass to avoid the password prompt) to send the suspicious index and parameters to children nodes, activate children node to complete their task and get the results. The children nodes get the required index from EC2 by constructing URL with parameters.

Reduce and response

After receiving all the results from children nodes, the main node integrates the results to get the final result and send it to GAE in the form of json. The final result includes the length o f suspicious text, the sequence number of the source files which the plagiarisms are found in and the according number of plagiarisms. At last, the suspicious index files will be deleted.

C. EC2

Build Web Application

In order to allow OS’s children nodes to get the index by URL, I build a web application in EC2 based on apache2 and mod_python. For example, to get the index built by the source files whose sequence number between 1 and 125 where window size=8 and overlap size=2, the URL is /build_index?window_size=8&overlap_size=2&start=1&end= 125 (Omit the ELB’s address). The output is encoded in the form of json, so the children nodes in OS can convert it into dictionary conveniently.

ELB and Auto Scaling

In order to make the EC2 instances elastic, ELB and Auto Scaling are applied. All the incoming application traffic to the ELB’s public DNS name will be distributed automatically among multiple EC2 instances. ELB keeps pinging EC2 instances to check their healthy and reroutes traffic to healthy instances. It protects the unhealthy instances from be overloaded and maximise the healthy instances utilisation. Auto scaling can scale the EC2 instances’ capacity automatically, it works with an ELB and CloudWatch. CloudWatch provides monitoring for EC2 instances and transfer the system performance data to Auto Scaling. Auto Scaling scales the EC2 according to the policy.

In order to apply Auto Scaling, an Auto Scaling Group and an EBL are necessary. After setting the Auto Scaling Group and bonding the EBL, the policies have to be set. These things can be done via Auto Scaling Command Line Tool. Afterwards, CloudWatch must be configured and bond with the policy. It is done via CloudWatch API.

In this case, I create a group called MyGroup whose maximum number of instances is 4, minimum is 1 and instance type is m1.large. Then I create ScaleUpPolicy which increases the number of instances by 1 and ScaleDownPolicy decreases the number by 1. Afterwards, I create two CloudWatch alarm which are CPUUtilisation<40 for 10 minutes and CPUUtilisation>80 for 1 minute. Finally, I bond these two alarms to ScaleUpPolicy and ScaleDownPolicy separately.

For testing the availability of Auto Scaling, I use several machines to run Siege to give the EC2 instances the number of concurrent connections for a period of time and it does work.

No S3 Bucket

I do not use S3 bucket in this task, because both the speed of connection between S3 and EC2 instances and the connection between S3 and OS are unstable. Sometimes, it is extremely slow. I do not know it is a temporary phenomenon or usual. I give it up anyway.

The Choice between EC2 and Elastic MapReduce (EMR)

Different from the choice in my proposal, I choose EC2 instead. In the term of configuration, EC2 is the same as a personal computer. I can customise it as my wish. Obviously, comparing to EMR, it is more flexible. Additionally, it Is easier to debug.

As mentioned in the generic feedback, Using EMR forces me to use an unnecessary service.

V. TESING

I use suspicious-document11081 to test the performance of my system. It contains around 170,000 words. With window size=8, overlap size=2 number of instances=4, the consuming time is 62.19 seconds. As may be imagined, it will cost more than 4 minutes if the task is not broken and distributed to multiple children nodes in OS or the distributed tasks are not completed in parallel..

The result of suspicious-document11081 is shown by Google chart in Fig.2. and Fig.3. shows the first 10 most plagiarised source file after matching all the suspicious files.

VI. SECURITY

Instead of the plain text, the data from GAE is encoded by base64 before being transmitted. The main node in OS will decode the encrypted text before building the suspicious index.

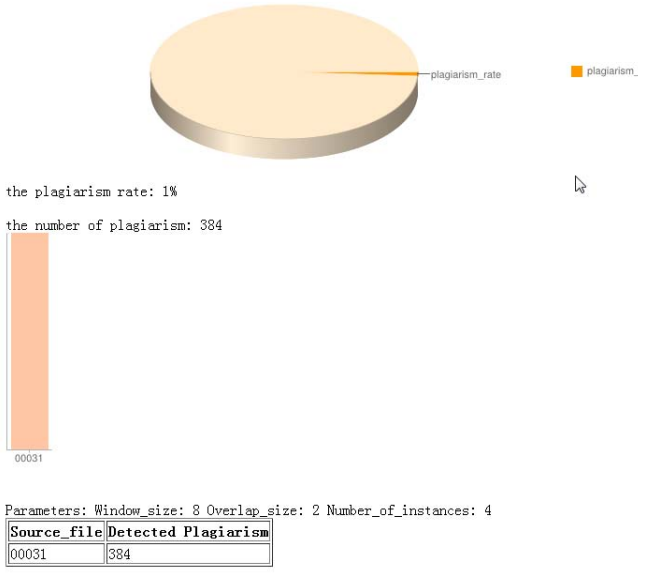
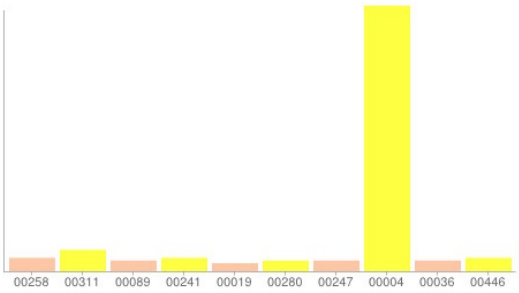


Fig. 2. A testing result (suspicious-document11081)



Parameters: Window_size: 8 Overlap_size: 2 Number_of_instances: 4

Source_file	Detected Plagiarism
00258	5
00311	8
00089	4
00241	5
00019	3
00280	4
00247	4
00004	99
00036	4
00446	5

Fig. 4. The first 10 most plagiarised source file.

VII. The Scalability of System

A. *The Result shown in GAE*

The plagiarisms are seldom in given suspicious files actually. My result's output style is adaptable. It can show the result via Google chart correctly even if the plagiarisms come from multiple source files. (For example, the Fig.4. shows the suspicious-document07141.txt's analysed result whose plagiarism texts come from different source files)

B. *Construction of Source Index in EC2*

In my system, the source index in calculated when EC2 receives the request rather than prepared source index. It increases the scalability of system markedly. Actually, the spending time to calculate the source index is very small. The most of the consuming time is incurred by data transmission. For example, with window size=8, overlap size=2. EC2 takes around 47 seconds to constructs the index of all the 500 source files. But it spends around 220 seconds to transmit the index from EC2 to OS. So it hardly increases the consuming time. (As mentioned before, the connection to S3 is unstable. The costing time would increase sometimes) However, it makes the index very flexible, especially when the parameters are not fixed (just like the situation in this task). Furthermore, in the case of adding or (deleting) a new (existed) source file, I just do it in EC2 instance rather than rebuild all the index with different parameters. In addition, using the real-time index can save a large chunk of storage space in S3 which is equivalent to save money.

C. *The Architecture of OS*

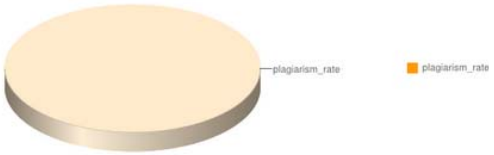
I use 2-level architecture in OS, which increase the scalability of the system greatly. The main node controls the children nodes and integrates the result in MapReduce way. As mentioned above, it reduces the time consumption evidently. With the increase of the number of source files, the size of source index will raise. If the index is dealt with by only one instance, the memory error will be raised (Even the 500 source files index can incur memory error in an m1.small OS instance) and the whole system will be corrupted. It not only reduces the time consumption but also avoid memory error to divide the task into several parts.

VIII. Chanllendge and Future Work

The security of my system is insufficient. Any other server can get the queuing tasks by request /get_queue. I plan to build a session key to avoid the unauthorised server to get the tasks.

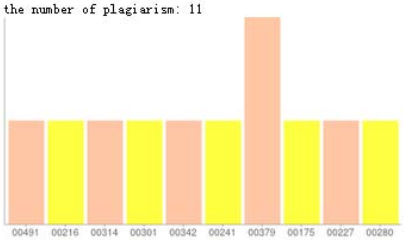
In addition, the system cannot show the original plagiarism text. I will build a dictionary whose key are the hash values to save the original text in the main node in OS. It allows the main node to query the original plagiarism text by hash value and deliver them to GAE to show them in a proper way.

Users are not supposed to change the source files. Otherwise, they can spoil the system. For instance, a user can add an unpublished paper into source files and the author would be judged as a plagiarism person. But the administrator should be able to amend the source files. There is no interface for administrators in GAE. It would better build an interface in GAE or an API to allow the administrators to interactive with the source files in EC2. Anyway it is a complicated mission because, after the changing the source file, a new AMI has to be build and the Auto Scaling Group must be updated due to the changed AMI Id.



the plagiarism rate: 0.02796%

the number of plagiarism: 11



Parameters: Window_size: 8 Overlap_size: 2 Number_of_instances: 4

Source_file	Detected Plagiarism
00491	1
00216	1
00314	1
00301	1
00342	1
00241	1
00379	2
00175	1
00227	1
00280	1

Fig. 3. Another testing result (suspicious-document07189)