CLOUD COMPUTING COURSEWORK

Austin Agbo

MSc Information Systems

University of Surrey, Guildford

6346574

*Abstract*—This document demonstrates an understanding of Cloud principles, and discuses the work to be done in building plagiarisms detection software using the Google App Engine, Open Stack and EMR.

cloudcomputing; aws; ec2; s3;

# Introduction

As the interest in cloud computing spikes up, by proof of the high number of companies investing into cloud services. It is reputably the savior to those who do not want to engage in “heavy lifting”. Normally, occurs with Internal IT infrastructure or outsourcing to an Internet Service Provider (organizations, who would carry out managing of your system on their servers).

After costs have been weighed and you decide cloud computing is the way forward for your plagiarisms detection software, you want a model that will enable a convenient on-demand network access, shared pool of configurable computing resources that can be quickly provisioned and released with minimal service provider interaction.

Attempting to employ these various services, we most first understand what these specific cloud services essentially are and how they can help us achieve our goal.

Google App Engine can be classified as a Public Cloud, it is owned by a well-known company and you can use some of it, some of the time for free. This is very beneficial for my system as a view/front end of the application. Why?

It provides services as a platform whereas a consumer does not manage the core infrastructure i.e. network, servers, OS, or storage but you have control over the deployed application and maybe, provide ability to host its environment configurations.

Not bad, but for the system we are attempting to build. An

Infrastructure as a Service is required – as the consumer in this case, I am allowed to process, store, and handle networking. Also, resource to deploy and run software, this includes operating systems and applications (e.g. host firewalls). This Infrastructure is what AWS and Open Stack provides to me, only difference is AWS can be classified as public while Open Stack is more private. So in my system’s entirety I will be building Software as a Service, which would offer the user a service that uses an application running on a cloud infrastructure. In this case, having working software as a service, it can then be described as a Hybrid Cloud.

Now knowing what our system can be described and classified as, I am aware these different organizations can be used advantageously to the system’s benefit, while the Private

Cloud as a student of the University of Surrey – open stack is free and provides scalability of servers, consideration of usage of servers on EMR would be more critical as this would cost us.

An advantage here is even though EMR would cost us we would only pay for as many servers used, and won’t completely pay for unused cycles compared to a subscription fee when dealing with an ISP. This benefits my system in general to have more of the application/functionality on open stack so things are cheaper and more flexible.

# DESCRIPTION OF SYSTEM

The system would have a front end that would allow the user to select text document, pick some options that they would like to be used for the process of detecting plagiarism.

It would make sense for this part of the system to live on GAE, as this would demand the least resources of the system.

This frontend will be developed in Python; the program will allow instruction entered by user to update a link of information that will be sent to an Open Stack instance.

This program deployed to GAE cloud. The engine’s features include Automatic scaling and load balancing. It would scale servers to meet traffic demands and I will only be charged if I go above the limit i.e. 1 GB of data storage and traffic. I would only need to enable billing when my traffic is close to the peak of storage I am allowed to utilize. The application will just need to scale to accommodate the amount of requests made to the deployed site.

All documents would be stored in the Amazon Simple Storage Service (s3). Specify buckets that will hold un-indexed files, indexed files and matched files. As whole software should be able to detect plagiarism by indexing, matching, EMR/Open Stack or both should handle parts of the system.

S3 provides accessibility in real time via a web server API from anywhere on the Internet. So when a job is completed and results are in S3 bucket, this can help in moving or copying data using the S3’s application program interface.

Also, using the S3 bucket to back up the Hadoop log files generated by the cluster so inspecting that the program is carrying out the job expected. Data can be monitored using the cloud watcher.

The software should allow use of the Amazon Elastic Map Reduce (EMR) to provide a managed framework where a user would instruct and monitor Hadoop Clusters with Amazon Elastic Compute Cloud (EC2) instances. This would provide easy scalability of servers instructed by user and help increase job process speed. Using the EMR, there are EC2 AMI’s that are set for the workflow and can easily be used in starting the job process. An image of a physical Linux system would be hosted, as the Open Stack aspect, it will serve as the controller of the system, regulating the process and job flow of the system. It would fetch instructions from the front end and then from there, decide whether to start EMR or a hadoop cluster on the running instance. For this cluster to be possible, hadoop should already be installed and using the S3 API to download files that will be worked on. Checking the process to fire a job should be inline with the running workflow, so conditions (switch/if statements) should be set to keep the process automated and running correctly. Now, a job can be fired on the instance by programming the hadoop commands by bash scripting but to fire a job on EMR would require an API. Boto is an API that can be used to achieve this; make a request to AWS using key values and monitor the process.

NON-CLOUD DATA ANALYSIS

Working with source and suspicious documents, to analyze these documents, it would benefit the system to go through each document, word to word so a small pattern of words can be formed then the imperative comparable compared against each other. E.g. For a string “I hope these words are remembered and memorized”, becomes “I\_hope\_these\_words\_are\_remembered 1

hope\_these\_words\_are\_remembered\_and 1

these\_words\_are\_remembered\_and\_memorized 1”.

For a 2nd string “I hope these words are remembered and memorized for ever you see” becomes “I\_hope\_these\_words\_are\_remebered 1

and\_memorized\_for\_ever\_you\_see 1

are\_remebered\_and\_memorized\_for\_ever 1

hope\_these\_words\_are\_remebered\_and 1

remebered\_and\_memorized\_for\_ever\_you 1

these\_words\_are\_remebered\_and\_memorized 1

words\_are\_remebered\_and\_memorized\_for 1”.

Appending ‘1’ to each line is so when these patterns are compared against each other, a variable is incremented when it recognizes a duplicate pattern. This variable will end up being the number of patterns repeating in between these two strings.

Organizing these patterns is the process of producing an index of words in the document (Map program). The number of patterns = number of words presently. Incrementing and storing duplicates also occurs as part of the indexing (Reduce program).

If x is patterns count = word count and y is number of duplicates. Y/x of 100 = percentage of similarity. Now we can easily produce results from just indexing all the files together but the 1st problem with this is we can’t locate the exact position where duplicates exist, also our result can only be described as an approximate since we can’t yet account for pattern repeats in a source document (This added count would be a miscalculation of similarities). What needs to be done is index separately, account for duplicates in both Source and Suspicious documents - In that case, it is better to store start and end positions so we can easily match patterns that are the same to discard duplicates.

The mapper’s job is to process the input data; the output is then passed to the reducer. The reducer processes the mapper’s output - creating new output values. Therefore, stabilizing and grouping data into key – value pairs that can be passed on for matching.

Matching – Here, the map program should sort these index by document name, and then start positions looking like this (document00010 | 145 | 150 |161\_index\_i ). Therefore it will be easier to Merge and develop minimal representation of the whole document. Reducing documents would help us identify duplicates based on count of same key value pairs that exist in both suspicious and source documents.

Map-reduce groups data by type, splitting data into different nodes. Processing its portions simultaneously. It allows several nodes to process data independently and then combine it together. Combines the data so that a single node can process it at once. The way the Hadoop cluster works (can range from 10 to thousands of nodes), all connected by a network. A task is broken into smaller parts. The smaller parts are usually based on the amount and size of data been processed.

The Mapper on a node operates on that smaller part. It takes the mapper’s data from each node and brings it together on nodes all around the cluster. The Reducer runs the node and knows it has access to everything with the same key.

Suspicious Data Sample presented as Index Indicating Pattern, Start Position, End Position, 1, and Document Name.

16\_our\_and 96 101 1 suspicious-document00010

179\_the\_of 147 152 1 suspicious-document00010

1\_ii\_charleroi 90 95 1 suspicious-document00010

37\_iv\_return 102 107 1 suspicious-document00010

.

##### References

CLOUD COMPUTING (COMM034) Lecture Notes

<https://www.youtube.com/watch?v=5zBxl6HUFA4>

<https://www.youtube.com/watch?v=Hhj3fOdt7zo>

<http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html>

<https://www.youtube.com/watch?v=bcjSe0xCHbE>